Planning on Discrete Event Systems Using Parallelism Maximization

Lucas V. R. Alves, Patrícia N. Pena, Ricardo H. C. Takahashi

Abstract—This work aims to present solutions for a production planning problem in Discrete Event Systems. The application of Supervisory Control Theory limits the search universe to safe production sequences, so it makes sense to use the supervisor structure (closed loop behavior) to determine the best sequences. Two efficient solutions to the problem are proposed, and both seek to maximize the parallelism as a way to minimize the makespan.

Keywords—Discrete Event Systems, Supervisory Control Theory, Planning.

I. INTRODUCTION

Most of the actual industrial systems may be, at some point, modeled as a Discrete Event System (DES), where the states change given the occurrence of events. In this paper, we model Discrete Event Systems using automata and languages, allowing a distinction on the system to be controlled (plant) and the controller (supervisor).

In industry, the most valuable resource is time so, choosing good operational sequences, such as those which minimize makespan, is of great importance in manufacturing. As a result, the research and application of efficient planning and task scheduling techniques are the key to answer to the demand for efficiency.

Task scheduling refers to the allocation, over time, of finite resources to tasks in the production process, using some optimization criterion. This problem can be divided into two classes, the Deterministic Scheduling Problem (also called model-predictive scheduling) and the Stochastic Scheduling Problem. A scheduling is deterministic when the system is so predictable that a model can be used and the result will match the behavior of the system with negligible error and, on the other hand, a scheduling is stochastic when the system is subject to unpredictable disturbances, rendering the system states predictable only in a statistical sense. There are also systems that are mostly deterministic but not completely predictable, as the deterministic schedule techniques would require.

Over the years, several formalisms to address the scheduling problem emerged in the literature in the context of mathematical programming, Petri Nets, Timed Automata, Verification Models, among others.

In the context of Supervisory Control Theory (SCT), of Discrete Event Systems (DES), there are many approaches. Usually these techniques are focused on minimizing the makespan, the total production time of a batch of products, or maximizing the throughput in the continuous production. The main advantage on representing DES using automata and languages is the Supervisory Control Theory (SCT), a framework in which plants and specifications are used to generate an optimal supervisor, in a sense of being minimally restrictive. There are also constructive approaches, using Prioritized Planning and using sequential language projection.

A minimally restrictive supervisor guarantees that any sequence that is allowed by the controller is safe, but different sequences may have different performances on execution time.

Choosing the best sequence, in time sense, is very difficult, usually being impractical for real life problems. The main problem when working with makespan is that the general case is non polynomial, making many industrial problems intractable. To address these kind of problems the most common approaches in the literature are the heuristics.

When the task scheduling problem does not take into consideration the constraints on the resources to be used in the operation, and only uses temporal and precedence constraints, it can be called a planning problem. Using automata and languages to model Discrete Event Systems (DES) and using Supervisory Control Theory to obtain a supervisor which guarantees a safe operation already gives the precedence constraints so, our problem can be classified as a planning problem and we propose to turn it into a path finding problem.

In this work, we change the optimization problem, extending the work in [21], to address the temporal correctness of the resulting sequences. Instead of minimizing the makespan, we maximize the number of active parallel tasks during the production. The maximum parallel sequence is a sub-optimal solution in the time sense and this paper shows a procedure to compute such sequence in linear time.

We treat the problem as a deterministic scheduling problem given that the execution time of each component of the system is known and deterministic. Therefore, the presented algorithms fix the position of, both, controllable and uncontrollable events since the occurrence of uncontrollable events can be predicted.

In [21], we deal with the logical behavior given by the monolithic supervisor. The procedure establishes an order to the whole sequence from the initial state to the marked state reached when the production is complete. In the SCT, all logical behaviors that are correct are accepted by the supervisor, but the logic correctness does not ensure time cor-

Lucas V. R. Alves is with Technical College and Graduate Program in Electrical Engineering, Universidade Federal de Minas Gerais lucasvra@ufmg.br
Patrícia N. Pena is with Department of Electronics Engineering, Universidade Federal de Minas Gerais ppena@ufmg.br
Ricardo H. C. Takahashi is with Department of Mathematics, Universidade Federal de Minas Gerais taka@ufmg.br
This work has been supported by the Brazilian agency CAPES, CNPq and Fapemig.
rectness. If such sequence is run in the real system, without the information of time, it may have some inconsistencies related to the uncontrollable events. This work solves this unfeasibility problem, by including the duration of the operations of each machine in the optimization.

This paper is structured in the following way. In Section II we show the preliminaries concepts and main definitions. In Section III we present the main ideas supporting the parallel maximization, in Section IV the algorithm of parallelism maximization taking time into consideration is presented. In Section V a heuristic algorithm is presented, the algorithm of string generation and marked language are, respectively, $Q_s$ as language of strings.

Section VII contains an illustrative example of the application of the algorithms. The conclusions and final comments are in Section VIII.

II. Preliminaries

In this section, we summarize some fundamental concepts and results of the SCT [10] that are needed for the theoretical development of the paper.

Let $\Sigma$ be a finite non-empty set of events, referred to as an alphabet. Behaviors of DES are modeled by finite strings over $\Sigma$. The Kleene closure $\Sigma^*$ is the set of all strings on $\Sigma$, including the empty string $\epsilon$. A subset $L \subseteq \Sigma^*$ is called a language. The concatenation of strings $s, t \in \Sigma^*$ is written as $st$. A string $s \in \Sigma^*$ is called a prefix of $t \in \Sigma^*$, written $s \leq t$, if there exists $u \in \Sigma^*$ such that $stu = t$. The prefix-closure $L$ of a language $L \subseteq \Sigma^*$ is the set of all prefixes of strings in $L$, i.e., $L = \{ s \in \Sigma^* \mid s \leq t \text{ for some } t \in L \}$. 

Definition 1: A deterministic finite automata is a 5-tuple $G = (Q, \Sigma, \delta, q_0, Q_m)$, where $Q$ is a finite set of states, $\Sigma$ is an alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function, $q_0 \in Q$ is the initial state and $Q_m \subseteq Q$ is the set of marked states.

The transition function can be extended to recognize words over $\Sigma^*$ as $\delta(q, s) = q'$ if $\delta(q, \sigma) = x$ and $\delta(x, s) = q'$. The generated and marked language are, respectively, $\mathcal{L}(G) = \{ s \in \Sigma^* \mid \delta(q_0, s) = q' \land q' \in Q_m \}$ and $\mathcal{L}(S) \subseteq \mathcal{L}(G)$. The active function, defined by $\Gamma : Q \to 2^\Sigma^*$, is given a state $q$, the set of events $\sigma \in \Sigma$ for which $\delta(q, \sigma)$ is defined.

Definition 2: Let $G_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, Q_{m1})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, Q_{m2})$. The synchronous product of $G_1$ and $G_2$ is:

$G_{1\|2} = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{12}, (q_{01}, q_{02}), Q_{m1} \times Q_{m2})$

where

$\delta((q_1, q_2), \sigma) = \begin{cases} 
\delta_1(q_1, \sigma), & \text{if } \sigma \in \Gamma_1(q_1) \cap \Gamma_2(q_2) \\
\delta_2(q_2, \sigma), & \text{if } \sigma \in \Gamma_1(q_1) \cap \Sigma_2 \\
(q_1, \delta_2(q_2, \sigma)), & \text{if } \sigma \in \Gamma_2(q_2) \cap \Sigma_1 \\
\text{undefined}, & \text{otherwise}
\end{cases}$

and $\Gamma_{1\|2}(q_1, q_2) = [\Gamma_1(q_1) \cap \Gamma_2(q_2)] \cup [\Gamma_1(q_1) \cap \Sigma_2] \cup [\Gamma_2(q_2) \cap \Sigma_1].$

The Supervisory Control Theory is a formal method, based on language and automata theory, to the systematic calculus of supervisors. The system to be controlled is called plant, the controller agent is called supervisor and the control problem is to find a supervisor which enforces the specifications in a minimally restrictive way. The plant is modeled by an automaton $G = (Q, \Sigma, \delta, q_0, Q_m)$ and $S = \Sigma_1 \cup \Sigma_2$ where $\Sigma_1$ is the set of controllable events, which can be disabled by an external agent, and $\Sigma_2$ is the set of uncontrollable events, which cannot be disabled by an external agent. The plant represents the logical model of the DES, the system behavior under no control action. The supervisor’s $S$ role is to regulate the plant’s behavior to meet a desired behavior $K$ disabling controllable events.

Let $E$ be an automaton that represents the specification imposed on $G$. We say that $K = \mathcal{L}_m(G \parallel E) \subseteq \mathcal{L}_m(G)$ is controllable with respect to $G$ if $K \Sigma_m \cap \mathcal{L}(G) \subseteq K$. There exists a nonblocking supervisor $V$ for $G$ such that $\mathcal{L}_m(V \parallel G) = K$ if and only if $K$ is controllable with respect to $G$. If $K$ does not satisfy the condition, then the supremal controllable and nonblocking sublanguage $\sup\mathcal{L}(K, G)$ can be synthesized. It represents the least restrictive nonblocking supervisor.

The generated and marked language of a plant $G$ under the action of a supervisor $S$ are, respectively, $\mathcal{L}(S) \subseteq \mathcal{L}(G)$ and $\mathcal{L}(S \parallel G) \subseteq \mathcal{L}(S \parallel G)$. A supervisor $S$ is called non-blocking when $\mathcal{L}_m(S \parallel G) = \mathcal{L}(S \parallel G)$.

III. Modelling Parallelism

The idea of a task is introduced as a property of a state of a deterministic automaton. A state may have zero or more tasks being executed. If one wants only to maximize the number of machines working, the number of tasks associated to a state should be 0 if it is an idle state or 1 if it is a working state. If the machine has parallelism on itself, as a processor with multiple cores, the number of tasks on each state may be any non negative integer. In order to establish the number of tasks performed in each state, we define the active tasks function.

Definition 3: Let $G = (Q, \Sigma, \delta, q_0, Q_m)$ be a deterministic automaton. The active tasks function, $f_{at} : Q \to \mathbb{Z}^+$, is a function that, for every state $q \in Q$, assigns a non negative integer that represents the number of active tasks.

Usually, specification automata do not perform tasks themselves. In order to keep coherence, we may define an active tasks function that assigns zero tasks for all states of the specifications. The same maneuver should be used for plant automata which are not interesting for the optimization process.

The active tasks function of a composed automaton is defined.

Definition 4: Let $f_{at_1}$ and $f_{at_2}$ be the active tasks functions of $G_1$ and $G_2$, respectively. The active tasks functions of $G_{1\|2} = G_1 \parallel G_2$ is:

$f_{at_{1\|2}}(q_1, q_2) = f_{at_1}(q_1) + f_{at_2}(q_2).$

The expansion to multiple automata is straightforward. In order to illustrate the main ideas of the paper, the small factory [22] is going to be used, along with the main definitions.

Example 1: The small factory consists of two machines and an unitary buffer, as show in Fig. [1].
The plant and specification automata for the example are presented in Fig. 2. The active tasks functions \( f_{at} \) for \( M_1, M_2 \) and \( E \) are presented in TABLE I.

**TABLE I: Example** Tasks on each state of components of the small factory.

| Automaton | State \( q \) | \( f_{at}(q) \) |
|-----------|--------------|----------------|
| \( M_1 \) | \( I \)       | 0              |
|           | \( W \)      | 1              |
| \( M_2 \) | \( I \)       | 0              |
|           | \( W \)      | 1              |
| \( E \)   | \( E \)      | 0              |
|           | \( F \)      | 0              |

The composition of the two machines, \( M = M_1 \| M_2 \) is shown in Fig. 3 and the active tasks function applied over it is presented in TABLE II. As we may see, the state \( WI \) represents the machine \( M_1 \) in state \( W \), with one active task, and machine \( M_2 \) on state \( I \) with zero active tasks, so, the state \( WI \) has one active task.

For the plant \( M \) and the specification \( E \), the monolithic supervisor \( S \) is presented in Fig. 3(b) and the number of active tasks in each state is given by TABLE II. As the supervisor only disables events on the system, the set of states of \( S \) is a subset of the set of states of \( M \| E \), so the states of \( S \) inherit the active task function of \( M \| E \).

In order to evaluate the parallelism of a string \( \sigma s \in \mathcal{L}(G) \) starting on a state \( q \in Q \), we define the cumulative active tasks function:

**Definition 5:** Let \( G = (Q, \Sigma, \delta, q_0, Q_m) \) be an automaton subject to \( f_{at} \). The cumulative active tasks function \( F_{at} : Q \times \Sigma^* \rightarrow \mathbb{Z}^+ \) is:

\[
F_{at}(q, \varepsilon) = f_{at}(q) \\
F_{at}(q, \sigma s) = F_{at}(q) + F_{at}(\delta(q, \sigma), s).
\]

**Example 2:** Consider the problem of Example 1 and two strings \( s_2 \) and \( s_3 \), the cumulative active tasks function evaluates as \( F_{at}(\delta \Sigma E, s_2) = F_{at}(\delta \Sigma E, s_3) = 4 \) and \( F_{at}(\delta \Sigma E, s_1) = 6 \). Then, \( s_3 \) has more parallelism than \( s_2 \).

In the following, we formalize the optimization problem and present algorithms to solve it.

**IV. TIME CONSTRAINED MAXIMUM PARALLELISM**

As the pure logical maximum parallelism algorithm uses no time information we cannot ensure the maximum parallel sequence can be executed without any modification since we fix the position of uncontrollable events without knowing when they will happen. So, we extended the algorithm inserting time information in order to force the resulting sequence to be time coherent. This modification results in a non polynomial algorithm and, to keep polynomial complexity, we use a heuristic step.

**A. Parallelism Maximization Problem**

When working with time, some sequences of events, logically feasible are unfeasible in time, for example, suppose we have two processes, from example 1 defined by the sequences \( a_1b_1 \) and \( a_2b_2 \), the events occur instantaneously but there is a delay of 5 t.u. from event \( a_1 \) to event \( b_1 \) and a delay of 10 t.u. from event \( a_2 \) to event \( b_2 \). Given the deterministic timing, a sequence \( a_1a_2b_2b_1 \), even when allowed by the supervisor, is not timing feasible.

In order to evaluate the time until an event happens in a supervisor, given the events already occurred we can define a temporal function:

**Definition 6:** Let \( f_T : \Sigma^* \times \Sigma \rightarrow \mathbb{R}^+ \) be the temporal function of the closed loop system \( S \). Given an event \( \sigma \in \Sigma \) and a
sequence \( s \in \mathcal{L}(S/G), f_T(s, \sigma) = t \), where \( t \) is the time until the event \( \sigma \) occurs given the sequence \( s \) already occurred. If \( \delta(\delta(q_0, s), \sigma) \) is not defined, then \( f_T(s, \sigma) = \infty \).

Usually, the temporal function is implemented as an event scheduler, similar to those used on discrete event systems simulation [24]. Another useful measurement is the amount of time we need to execute a complete sequence of events. For this purpose we can expand the temporal function to give the time of a sequence.

Definition 7: Let \( f_T : \Sigma^* \rightarrow \mathbb{R}^* \) be the extended temporal function, defined as:

\[
\begin{align*}
    f_T(\epsilon) &= 0 \\
    f_T(s\sigma) &= f_T(s) + f_T(s, \sigma)
\end{align*}
\]

Let \( S \) be a supervisor for a production system \( G = \bigcup_{k=0}^{N} G_k \), where \( G_k, k \in \{0\ldots N\} \) is the set of subplants of the system, and let \( f_{at} \) be the active tasks function associated to the automaton that implements the closed loop behavior, \( S/G \). Let \( n \) be the number of events needed to produce a batch of products and let the search universe be the language \( L = \{ s \in \mathcal{L}_n(S/G) : n = |s| \land f_T(s) \neq \infty \} \). The discrete event systems planning problem can be defined as an optimization problem:

\[
s^* = \operatorname{argmax}_{s \in L} F_{at}(s)
\]

where \( s^* \) is one of the sequences that maximize the number of tasks occurring in parallel on the system and, that, is time correct in the time function \( f_T \).

B. Parallelism Maximization Algorithm

The optimization problem can be solved as a longest path problem, where the weight of a transition is the number of active tasks on the destination state. The existence of cycles in the supervisor prevents us from using a direct approach. We must turn the automaton into an acyclic graph first, growth until the cardinality of the solution is reached.

The desired acyclic graph is given by the composition of the supervisor automaton \( S \) with an unwind automaton \( G_a \) where \( \mathcal{L}_n(G_a) = \{ s \in \Sigma^* : |s| = n \} \). \( \Sigma \) is the event set of \( S \) and \( n \) is the number of events in a batch.

On the resulting acyclic graph, a vertex is represented as a pair \((q,k)\) where \( q \) is the original state of the automaton and \( k \) is the number of events occurred to reach the state \( q \). Starting from the initial state \( q_0 \), we can travel on the graph in topological order, and so, a maximum path algorithm can be executed in linear time.

Example 3: Consider the small factory of Example 1. The unwind automaton for \( n = 8 \) is shown in Fig. 4.

The supervisor, when composed with an unwind automaton for the production of two products (eight events), generates the acyclic graph shown in Fig. 5. Now, it is possible to apply a longest path algorithm, in order to obtain a maximum parallel sequence.

Find an \( s^* \), when taking time into consideration, is almost as hard as finding the sequence which minimizes the makespan, so, in order to take advantage of the maximum parallelism we will use a heuristic branch and bound approach.

The inputs for the algorithm are the set of states of the supervisor \( (Q) \), the transition function \( (\delta) \), the active event function \( (\Gamma) \), the initial state \( (q_0) \) and the search depth \( \text{(depth)} \). As a result, the algorithm fills the structure \text{path} which holds the path from the initial vertex \((q_0,0)\) to each vertex reached on the search. The composition of the supervisor with the unwind automaton is done on-the-fly during the execution of the algorithm.

From line 1 to 13 the structures are initialized. From line 15 to 45 a while loop is executed until the queue \( F \) is empty. On line 31 the time is calculated and, on line 32, if the time to the vertex is \( \infty \), the path is not timing reachable, then the vertex is ignored.

Also, knowing that, usually, the execution of a controllable event should increase the number of tasks, we postpone the execution of uncontrollable events, and instead of visiting events in \( \Gamma(q) \) we only visit controllable events unless there are no controllable events active, when we visit the uncontrollable events with less time to occur (lines 19 to 28).

From line 39 to 43 there is an conditional if to evaluate if the path using event \( \sigma \) is better than the previous path. As the future possible paths starting from vertex \((v,i+1)\) depend on the path from the initial vertex to \((v,i+1)\), we should keep all paths to \((v,i+1)\) because, maybe, they are not good at this point but may be far better in the future. In order to maintain the algorithm polynomial in complexity, we take a greedy step and keep only one of the best paths.

It is important to note that in order to the algorithm converge, we have to limit the number of times each controllable event may occur, this makes sense since we know the size of

\[
\begin{align*}
    0 \rightarrow \Sigma \rightarrow 1 \rightarrow \Sigma \rightarrow 2 \rightarrow \Sigma \rightarrow 3 = \Sigma \rightarrow 4 = \Sigma \rightarrow 5 \rightarrow \Sigma \rightarrow 6 \rightarrow \Sigma \rightarrow 7 \rightarrow \Sigma \rightarrow 8 \rightarrow \Sigma
\end{align*}
\]
the batch we intend to produce and the recipe to produce it.

The step by step execution of Algorithm 1 for the Small Factory of Example 1 is presented in Fig. 6.

The complexity of the algorithm is the same of a breadth-first search, \( \mathcal{O}(v + a) \) where \( v \) is the number of vertices and \( a \) is the number of edges. In this algorithm, a vertex is a state in determined depth, so for a depth of \( n \) events, the number of vertices is \( v = (n + 1)Q \) and the number of edges is \( a = n \) →, \( Q \), where \( Q \) and → are, respectively, the set of states and the set of transitions of the supervisor, so the complexity is, in the worse scenario, \( \mathcal{O}(nW + n) \).

**Example 4:** Consider the small factory, presented in Example 2. The execution of Algorithm 1 is presented in Fig. 6. In this approach, the duration of the operation of the machines is part of the optimization, so a time interval of 10 t.u. is considered between events \( a_1 \) and \( b_1 \) and 5 t.u. between events \( a_2 \) and \( b_2 \).

In each step, starting in the initial state, the algorithm travels the closed loop system accumulating the number of tasks in the path and updating a schedule with the time until the occurrence on each uncontrollable event. For instance, at the initial step, Fig. 6 (a), \( T(b_1) = T(b_2) = \infty \) since such events are now allowed in the supervisor. When state \( WIE1 \) is reached, \( T(b_1) = 10 \), since \( a_1 \) has occurred, and \( T(b_2) = \infty \), in Figures 6 (a) (b) (c) there is only one event to execute but when the state \( WIE3 \) is reached (Figure 6 (d)) the events \( b_2 \) and \( a_1 \) could be executed, but only event \( a_1 \) is visited because the algorithm always executes only controllable events when possible. In state WWE4 (Figure 6 (e)), events \( b_1 \) and \( b_2 \) are logically possible to occur but \( b_2 \) occurrence is time unfeasible. In Figures 6 (f) (g) (h) (i) the algorithm has only one path to follow reaching the final state III8.

The algorithm finishes when it reaches a marked state, after executing 8 events. The resulting sequence in this example is \( s^* = a_1 \ b_1 \ a_2 \ b_1 \ b_1 \ a_2 \ b_2 \), the parallelism \( F_{\text{PAR}}(s^*) = 6 \) and \( f_T(s^*) = 25 \ t.u. \).

V. PARALLELISM BASED HEURISTIC SOLUTION

Following the idea of parallelism maximization and that controllable events should increase the number of tasks being executed we proposed a time-oriented heuristic that consists on applying the same delay of uncontrollable events used in the Algorithm 1 which seems to increase parallelism and reduce the branch-factor, but instead of maximizing parallelism, we minimize the makespan.

A. Makespan Minimization Problem

Let \( S \) be a supervisor for a production system \( G = \prod_{k=0}^{N} G_k \), where \( G_k \), \( k \in \{0, \ldots, N\} \) is the set of subplants of the system, and let \( f_T \) be the time function associated to the automaton that implements the closed loop behavior, \( S / G \). Let \( n \) be the number of events needed to produce a batch of products and let the search universe be the language \( L = \{ s \in \mathcal{Z}^n/\Sigma / G \}; n = |s| \land f_T(s) \neq \infty \}. \) The discrete event systems planning problem can be defined as an optimization problem:

\[
\forall \delta \in \mathcal{L} \quad f_T(s)
\]

where \( s^* \) is one of the sequences that minimizes the makespan of the production batch.

B. Heuristic Makespan Minimization Algorithm

The algorithm follows the same logic of an exact algorithm, so, a state, when reached by different path, with different schedulers are maintained duplicated to the next iteration. A path is only discarded when there is another path that reaches the same state with less makespan. As in Algorithm 1 for the algorithm to converge, we have to limit the number of times each controllable event may occur.

To execute the algorithm, the state set of the supervisor \( Q \), the event set of the supervisor \( \Sigma = \Sigma_m \cup \Sigma_d \), the transition function \( \delta \), the active events function \( \Gamma \), the initial state \( q_0 \) and the search depth \( n \) are necessary. The structure \( a \) is an event scheduler such \( a(\sigma) \) in a state \( (q,i) \) is equivalent.
(a) IIE0:
\[ f_T(path(IIE0)b_1) = \infty, \]
\[ f_T(path(IIE0)b_2) = \infty, \]
\[ Fat = 0 \]

(b) WIE1:
\[ f_T(path(WIE1)b_1) = 10, \]
\[ f_T(path(WIE1)b_2) = \infty, \]
\[ Fat = 1 \]

(c) IIF2:
\[ f_T(path(IIF2)b_1) = \infty, \]
\[ f_T(path(IIF2)b_2) = \infty, \]
\[ Fat = 1 \]

(d) IWE3:
\[ f_T(path(IWE3)b_1) = \infty, \]
\[ f_T(path(IWE3)b_2) = 5, \]
\[ Fat = 2 \]

(e) WWE4:
\[ f_T(path(WWE4)b_1) = 10, \]
\[ f_T(path(WWE4)b_2) = 5, \]
\[ Fat = 4; \]

(f) WIE5:
\[ f_T(path(WIE5)b_1) = 5, \]
\[ f_T(path(WIE5)b_2) = \infty, \]
\[ Fat = 5 \]

(g) IIF6:
\[ f_T(path(IIF6)b_1) = \infty, \]
\[ f_T(path(IIF6)b_2) = \infty, \]
\[ Fat = 5 \]

(h) IWE7:
\[ f_T(path(IWE7)b_1) = \infty, \]
\[ f_T(path(IWE7)b_2) = 5, \]
\[ Fat = 6 \]

(i) IIE8:
\[ f_T(path(IIE8)b_1) = \infty, \]
\[ f_T(path(IIE8)b_2) = \infty, \]
\[ Fat = 6 \]

Fig. 6: Example of the execution of the Parallelism Maximization with Time Restrictions Algorithm for the Small Factory Problem producing two products.
Algorithm 2: Heuristic Makespan Minimization Algorithm (HMM)

input : \( Q, \delta, \Sigma = \Sigma_i \cup \Sigma_e, \Gamma, q_0, \text{depth} \)
output: \( \text{path}, \text{time} \)
1 \( \text{path}(q,a,0) \leftarrow \varepsilon \)
2 \( \text{time}(q,a,0) \leftarrow 0 \)
3 \( F \leftarrow (q_0, a, 0) \)
4 while \( F \) is not empty do
5 \( \text{if } \exists \sigma \in (T(q) \cap \Sigma_i \cap \Sigma_e) : \text{time}(\text{path}(q,a,i) | \sigma) < \infty \) then
6 \( t_{\text{new}} \leftarrow \min \{ t_{\text{new}} : \text{time}(\text{path}(q,a,i) | \sigma) \} \)
7 \( \text{if } \exists \sigma \in (T(q) \cap \Sigma_i \cap \Sigma_e) : \text{time}(\text{path}(q,a,i) | \sigma) \leq t_{\text{new}} \) then
8 \( E \leftarrow \Gamma(q) \cap \{ \sigma : \sigma_i \in \Sigma \land \text{time}(\text{path}(q,a,i) | \sigma) \leq t_{\text{new}} \} \)
9 \( \text{else} \)
10 \( E \leftarrow \Gamma(q) \cap \{ \sigma : \sigma_i \in \Sigma \land \text{time}(\text{path}(q,a,i) | \sigma) \leq t_{\text{new}} \} \)
11 \( \text{end} \)
12 \( \text{end} \)
13 foreach event \( \delta \) in \( E \) do
14 \( H \leftarrow \delta(q, \sigma) \)
15 \( t \leftarrow \text{time}(\text{path}(q,a,i) | \sigma) \)
16 \( a_s \leftarrow \text{update}(a, \sigma) \)
17 \( \text{if } t = \infty \) then continue
18 \( \text{if } F \) does not contain \( (v,a_s,i+1) \) then
19 \( F \leftarrow (v,a_s,i+1) \)
20 \( t_{\text{new}} \leftarrow \text{time}(q,a,i) + t \)
21 \( \text{if } t_{\text{new}} \leq \text{time}(\text{path}(v,a_s,i+1)) \) then
22 \( \text{path}(v,a_s,i+1) \leftarrow \text{path}(q,a,i) | \sigma \)
23 \( \text{end} \)
24 \( \text{end} \)
25 \( \text{end} \)

\( \Gamma(q) \) is the set of all states that \( q \) can reach.

The algorithm initializes the path to the initial state as the empty sequence \( \text{path}(q,a,0) \leftarrow \varepsilon \) and the initial time as \( \text{time}(q,a,0) \leftarrow 0 \).

It is important to note that, in this algorithm a vertex has the form \((q,a,i)\) where \( q \) is an state of the supervisor, \( a \) is the event schedule and \( i \) is the depth.

The initial state is inserted into the queue \( F \) and, while the \( F \) has items, the first item is removed from the queue. The heuristic part consists of giving priority to the execution of controllable events over uncontrollable events. The algorithm verifies if there are transitions triggered by controllable events and if these transitions do not increase the timer (the time before the occurrence of the event is equal to the time after the occurrence of the event). If these transitions exist, the events which trigger them are inserted in the set of events to be evaluated \( E \). If there are no controllable transitions or if they increase the timer, the set of events to be evaluated, only the events (controllable or not) which implies in the less increase in time are executed \( t_{\text{min}} \).

For each event in the set \( E \), the algorithm calculates the increase in time for the transition, updates the event scheduler and verifies if the obtained sequence is time feasible. If the destination state was not evaluated yet, it is inserted in the queue \( F \). When the transitions leads to a state with a shorter production time, the path is taken as the best to that state. As a vertex visited by the algorithm is represented by \((q,a,i)\), a state \( q \), in the same depth \( i \), is treated as different vertices when they have different event schedulers.

The Heuristic Makespan Minimization Algorithm (HMM) presents a non-polynomial complexity because it duplicates states when the path converges to the same state with different event schedulers, but the heuristic reduces the branching factor, improving the performance of the algorithm even in larger problems.

VI. Experiments

In order to compare the algorithms, we use greedy algorithms that, at each step, choose the event which increases less the makespan.

The test example is the Flexible Manufacturing System (FMS) [24], composed of eight machines, three conveyors \((C_1, C_2 \text{ and } C_3)\), a mill, a lathe, a robot, a painting device \( (PD) \) and an assembly machine \( (AM) \). In each plant, the number of active tasks is represented on the state name so, for a state \((q,i)\), we have \( fa(t_q,i) = i \). The plants models are shown on Figure 7.

As we can see, the initial state of each plant is an idle state, so there are no active tasks and the other states have one active task each. Controllable events are represented by odd numbers and the uncontrollable ones are represented by even numbers.

The FMS produces two kinds of products, a Product A and a Product B. Both products share the same base, given by the following sequence of pairs (the controllable events must obey
the order in the sequence, but the uncontrollable events may occur in any order allowed in the supervisor):

$$b = \{(11, 12), (31, 32), (41, 42), (35, 36), (61)\}.$$ 

To produce the pin of a Product A, the pairs to be executed are:

$$p_a = \{(21, 22), (33, 34), (51, 52), (37, 38), (63, 64)\}.$$ 

Finally, to produce the pin of a Product B, the pairs are:

$$p_b = \{(21, 22), (33, 34), (53, 54), (39, 30), (71, 72), (81, 82), (73, 74), (65, 66)\}.$$ 

The monolithic supervisor of the FMS, using the Supervisory Control Theory, has 45,504 states and 200,124 transitions. In the tests, a batch that produces one Product A and one Product B is considered a batch of size one, so, a batch of size \(N\) produces \(N\) Products A and \(n\) Products B. Each pair of products (one A and one B) is represented by a sequence of 44 events, so a batch of size \(N\) is represented by \(44 \times n\) events.

In order to use time, we have to define the time interval between related events, as shown on Table III. The FMS has a peculiarity, the controllable events 63 and 65 occurs at least 15 time units after the event 61.

**TABLE III: Time interval between related events**

| Controllable Events | Uncontrollable Events | Time Interval [t.u.] |
|---------------------|-----------------------|----------------------|
| 11                  | 12                    | 26                   |
| 31                  | 32                    | 22                   |
| 33                  | 34                    | 20                   |
| 35                  | 36                    | 17                   |
| 37                  | 38                    | 25                   |
| 39                  | 30                    | 21                   |
| 41                  | 42                    | 31                   |
| 51                  | 52                    | 39                   |
| 53                  | 54                    | 33                   |
| 63                  | 64                    | 27                   |
| 65                  | 66                    | 27                   |
| 71                  | 72                    | 26                   |
| 81                  | 82                    | 25                   |

Using the time intervals, three algorithms were executed, the Parallelism Maximization with Time Restrictions (PMT), the Heuristic Makespan Minimization (HMM) and a timed greedy algorithm (GT), which, in each state, takes the transition that minimizes the makespan.

Each algorithm was executed once for batch sizes of one pair of products to 1000 pairs of products and the results are shown on Figure 8. The computations were performed in a computer with CPU Intel Xeon E5-2667 2.90 GHz and 64GB of RAM memory.

Fig. 8 solved the same problem using Model Checking for \(N \leq 15\). In the paper, the authors also provide a formula to calculate the makespan for any size of batch,

$$T(N) = 157N + 81. \quad (1)$$

This makespan is shown on Table IV as the optimal value, referred to as FV, from Formal Verification. It is important to notice that the ability to find the makespan for any size of batch does not correspond to finding the sequence that will provide such makespan. Using model checking the sequence could be found only to batches up to \(N = 15\). To sizes greater than \(N = 15\) the execution ran out of memory. In Table IV the time execution of the model checking to find a solution is presented up to 15.
As we can see in Fig. 8 (a), the fastest algorithm is the greedy one, but the Parallelism Maximization with Time Restrictions (PMT) is also very fast, finding a sequence to produce 1000 pairs of products in less than one minute. The Heuristic Makespan Minimization (HMM) is slower, but has a good execution time.

Both algorithms, the Parallelism Maximization with Time Restrictions (PMT) and the Heuristic Makespan Minimization (HMM), give good results in the sense of makespan, but in all situations the HMM gives a better makespan, at the cost of a higher execution time. Another important fact is that the HMM hits the lower bound [9], in all situations that the optimal sequence is known. Also, we are able to find a sequence with algorithm HMM that has the makespan value predicted by [1].

| Batch Size | PMT | HMM | FV |
|------------|-----|-----|----|
| 1          | 228 | 238 | 228 |
| 5          | 878 | 866 | 866 |
| 10         | 1663| 1651| 1651|
| 15         | 2448| 2436| 2436|
| 50         | 7943| 7931| 7931|
| 100        | 15793| 15781| 15781|
| 500        | 78593| 78581| 78581|
| 750        | 117843| 117831| 117831|
| 1000       | 157193| 157081| 157081|

The PMT gives a slightly bigger cumulative parallelism, as shown on Table V and Fig. 8 (c), which indicates that parallelism is a good measurement to performance of a process.

TABLE V: Cumulative Parallelism obtained using the algorithms PMT, HMM and Formal Verification (FV) and their corresponding execution time (FV was executed on a standard PC with a 2.8 GHz CPU and 16 GB of RAM).

| Batch Size | PMT | HMM | FV |
|------------|-----|-----|----|
| 1          | 93  | 93  |    |
| 5          | 713 | 635 |    |
| 10         | 1488| 1315|    |
| 15         | 2263| 1995|    |
| 50         | 7688| 6755|    |
| 100        | 15438| 13555|    |
| 500        | 77438| 67955|    |
| 750        | 116188| 101555|    |
| 1000       | 154938| 135955|    |

VII. CONCLUSION

This paper presents two efficient algorithms based on the idea of maximizing the parallelism among equipment to minimize makespan. Both algorithms take heuristic steps, what allow the convergence even for very large batches of products. Both algorithms are heuristics and the main difference among them is the fact that the PMT is a parallelism-oriented heuristic with polynomial complexity and the HMM is a time-oriented heuristic with a non-polynomial complexity. The algorithms were illustrated in a small example and then a experiment with a much more complex system of the literature was presented.

The Parallelism Maximization with Time Restrictions gives good results, being a polynomial algorithm and slightly faster than the Heuristic Makespan Minimization, the PMT seems a good compromise between execution time and results. The HMM algorithm allows a enormous reduction on the branching factor of an exact algorithm so, even being non-polynomial, presents a reasonable execution time. Also, HMM hits the optimal makespan for all batch sizes that it is known and was also able to find a solution with the makespan predicted in [9] for batches up to 2000 products.

Finally, we believe the results show that use the parallelism is a good strategy to solve scheduling problems. In fact, our results show that increasing parallelism while respecting time restrictions is a good way to increase production performance, even though, there are multiple sequences with the same makespan but different levels of parallelism.

The same idea of parallelism maximization can be used as an optimization criterion for other optimization approaches. Sequences obtained using parallelism oriented algorithms may be used as starting sequences for local search and evolutionary algorithms or with branch and bound based techniques.

Adaptations could be made in order to deal with stochastic scheduling problems, applying the idea of parallelism maximization in online algorithms, estimating the intervals and re-planning when necessary.

The same development presented in this paper can be easily extended to work with decentralized strategies, such as the Local Modular Supervisory Control Theory.

ACKNOWLEDGMENT

This work has been supported by the Brazilian agencies CAPES, CNPq and Fapemig.

REFERENCES

[1] W. Wang, C. Yuan, and L. Xiaobing, “A fuzzy approach to multi-product mixed production job shop scheduling algorithm,” in Fuzzy Systems and Knowledge Discovery, 2008. FSKD ’08. Fifth International Conference on, vol. 1, Oct 2008, pp. 95–99.
[2] M. L. Pinedo, Scheduling: Theory, Algorithms, and Systems, 3rd ed. Springer Publishing Company, Incorporated, 2012.
[3] H. Aytug, M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy, “Executing production schedules in the face of uncertainties: A review and some future directions,” European Journal of Operational Research, vol. 161, no. 1, pp. 86 – 110, 2005, iEPM: Focus on Scheduling. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221703005307
[4] Y. Song, M. T. Zhang, J. Yi, L. Zhang, and L. Zheng, “Bottleneck station scheduling in semiconductor assembly and test manufacturing using ant colony optimization,” IEEE Transactions on Automation Science and Engineering, vol. 4, no. 4, pp. 569–578, 2007.
[5] A. Schrijver, Theory of Linear and Integer Programming. New York, NY, USA: John Wiley & Sons, Inc., 1986.
[6] E. López-Mellado, N. Villanueva-Paredes, and H. Almeida-Canepa, “Modelling of batch production systems using petri nets with dynamic tokens,” Mathematics and Computers in Simulation, vol. 67, no. 6, pp. 541 – 558, 2003.
[7] Y. Abdeddaïm, E. Asarin, and O. Maler, “Scheduling with Timed Automata,” Theoretical Computer Science, vol. 354, no. 2, pp. 272 – 300, 2006.

[8] A. Herzig, M. V. de Menezes, L. N. de Barros, and R. Wassermann, “On the revision of planning tasks,” in ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic – Including Prestigious Applications of Intelligent Systems (PAIS 2014), 2014, pp. 435–440.

[9] R. Malik and P. N. Pena, “Optimal task scheduling in a flexible manufacturing system using model checking,” in 2018 14th International Workshop on Discrete Event Systems (WODES), June 2018, pp. 241–246.

[10] P. J. G. Ramadge and W. M. Wonham, “The Control of Discrete Event Systems,” Proc. of the IEEE, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[11] A. Kobetski and M. Fabian, “Scheduling of discrete event systems using mixed integer linear programming,” in Discrete Event Systems, 2006 8th International Workshop on, July 2006, pp. 76–81.

[12] D. Pinha, M. de Queiroz, and J. Cury, “Optimal scheduling of a repair shipyard based on supervisory control theory,” in Automation Science and Engineering (CASE), 2011 IEEE Conference on, Aug 2011, pp. 39–44.

[13] R. Su, J. van Schuppen, and J. Rooda, “The synthesis of time optimal supervisors by using heaps-of-pieces,” Automatic Control, IEEE Transactions on, vol. 57, no. 1, pp. 105–118, Jan 2012.

[14] S. Ware and R. Su, “Incremental scheduling of discrete event systems,” in 2016 13th International Workshop on Discrete Event Systems (WODES), May 2016, pp. 147–152.

[15] ——, “Time optimal synthesis based upon sequential abstraction and its application to cluster tools,” IEEE Transactions on Automation Science and Engineering, vol. 14, no. 2, pp. 772–784, April 2017.

[16] ——, “Time optimal synthesis based upon sequential abstraction and maximizing parallelism,” in 2017 13th IEEE Conference on Automation Science and Engineering (CASE), Aug 2017, pp. 926–931.

[17] M. R. Garey and D. S. Johnson, Computers and intractability. W.H. Freeman, 1979.

[18] P. N. Pena, T. A. Costa, R. S. Silva, and R. H. Takahashi, “Control of flexible manufacturing systems under model uncertainty using supervisory control theory and evolutionary computation schedule synthesis,” Information Sciences, vol. 329, pp. 491 – 502, 2016, special issue on Discovery Science.

[19] C. Almeder and L. Mönch, “Metaheuristics for scheduling jobs with incompatible families on parallel batching machines,” Journal of the Operational Research Society, vol. 62, no. 12, pp. 2083–2096, 2011.

[20] M. Gilhab, D. Nau, and P. Traverso, Automated Planning Theory and Practice. Elsevier.

[21] L. V. R. Alves, H. J. Bravo, P. N. Pena, and R. H. C. Takahashi, “Planning on discrete events systems: A logical approach,” in 2016 IEEE International Conference on Automation Science and Engineering (CASE), Aug 2016, pp. 1055–1060.

[22] W. M. Wonham, Supervisory Control of Discrete-Event Systems. Toronto, Canada: Systems Control Group, Department of Electrical & Computer Engineering, University of Toronto, 2014.

[23] C. Cassandras and S. Lafortune, Introduction to Discrete Event Systems, 2nd ed. Springer, 2008.

[24] M. H. de Queiroz, J. E. R. Cury, and W. M. Wonham, “Multitasking supervisory control of discrete-event systems,” Discrete Event Dynamic Systems, vol. 15, no. 4, pp. 375–395, Dec 2005. [Online]. Available: [https://doi.org/10.1007/s10626-005-4058-y]