USING AN ABSTRACTION OF THE SUPERVISOR TO SOLVE A PLANNING PROBLEM IN MANUFACTURING SYSTEMS

GUSTAVO CAETANO RAFAEL[*], PATRICIA N. PENA[†]

[*] *Graduate Program in Electrical Engineering - Universidade Federal de Minas Gerais - Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil*

[†] *Department of Electronics Engineering , Universidade Federal de Minas Gerais - Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil*

Abstract— This paper deals with a production-planning problem in the industrial environment from the perspective of Discrete Event Systems. The use of the solution obtained by applying the Supervisory Control Theory (SCT) as a search space for the optimization problem, using evolutionary algorithms, has been described in the literature as the SCO - Supervisory Control and Optimization approach. In the SCO, a string of the closed-loop behavior that minimizes the *makespan* for the production of a batch of products is sought. In the present work, a heuristic that guarantees the generation of feasible individuals is proposed, aside with the use of an abstraction of the closed-loop behavior as the search universe for the optimization. Lastly, a case study is presented, for which the optimal solution in all instances in which it is known was found.

Keywords— Discrete Event Systems, Supervisory Control Theory, Optimization, Evolutionary Algorithms, Production Planning.

Resumo— Este trabalho aborda o problema de planejamento de produção no ambiente industrial sob a perspectiva de Sistemas a Eventos Discretos. O uso da solução obtida pela aplicação da Teoria de Controle Supervisório como espaço de busca para um problema de otimização, usando algoritmos evolucionários, foi descrito da literatura como a abordagem CSO - Controle Supervisório e Otimização. Na CSO busca-se uma cadeia do comportamento em malha fechada que minimiza o *makespan* para produção de um lote de produtos. No presente trabalho, propõe-se uma heurística para geração de indivíduos garantidamente factíveis, além do uso de uma abstração do comportamento em malha fechada como o universo de busca para a otimização. Por fim, apresenta-se um estudo de caso para o qual foi possível encontrar a solução ótima para todas as instâncias em que a mesma é conhecida.

Palavras-chave— Sistemas a Eventos Discretos, Teoria de Controle Supervisório, Otimização, Algoritmos Evolucionários, Planejamento de Produção.

1 Introduction

In times of constant change in the economic scenario, organizations not only should reduce their losses, but also carefully invest their most important asset, time. With that in mind, a well designed production planning technique can be the difference between succeeding or not.

As a way to respond to this ever increasing demand for efficiency, many researchers have applied scheduling and production planning techniques do deal with a wide range of problems in industry, (Wang et al., 2008).

Task scheduling is an optimization problem, where a new solution arises every time resources are allocated. Usually, a finite number of assets is assigned accordingly to a set of tasks, this is made throughout a permutation on this elements (Pinedo, 2016).

Once a new solution is found, it is necessary to classify it as feasible, when it respects all the system's constraints, or as unfeasible, if the solution violates any of the requirements. Usually, in this type of problems, the set of all possible solutions is given by $(N!)^M$, where N represents the number of tasks and M the number of machines (Arisha et al., 2001). Along with the Job Shop Scheduling (JSS), problems that experience an exponential growth are classified as *NP-hard*. This class does not have known methods to find exact solutions in polynomial time (Garey & Johnson, 1980). One way to solve them is to apply dynamic programming techniques to find exactly optimal solutions for this problem (Bellman & Dreyfus, 2015). However, the complexity may be prohibitive for medium size problems.

Due to the highly demanding processing power needed, many researchers take approaches that differ from the traditional analytical and heuristic techniques found in (Arisha et al., 2001).

Within the Discrete Event Systems (DES) literature, many formalisms are used to solve the problem of minimizing the production time (makespan). Among them, there are Petri Nets (López-Mellado et al., 2005), timed automata (Abdeddaïm et al., 2006) and formal verification (Herzig et al., 2014).

In the context of Supervisory Control Theory (SCT) (Ramadge & Wonham, 1989) extensions have been proposed to solve the optimization problem, like in (Su et al., 2012), (Pinha et al., 2011) among others. There are also approaches that use the structure provided by SCT, that implements the restrictions to the safe operation of the system, and run optimization on top of such structure. Among them, we cite (Alves et al., 2016a) that uses a heuristic that favors the total accumulated parallelism of equipment during production, which leads to a sub-optimal solution for the makespan problem. This approach is used to compare with our results.

This paper presents a new heuristic for sequence generation combined with a technique to reduce the search space. Then, they were combined in an evolutionary algorithm, the Clonal Selection Algorithm (CSA), to minimize the makespan in a flexible manufacturing system.

In the following, in Section II, the preliminaries concepts and main definitions are presented. Then, in Section III, the main results are presented. In Section IV, there is a brief introduction to the optimization technique applied in this paper. Section V contains one example of the application, the conclusions is in Section VI.

2 Preliminaries

In this section, some basic ideas that are important for this work are summarized and the problem will be stated. The reader should refer to (Cassandras & Lafortune, 2009) for a detailed explanation.

Let Σ be a finite and nonempty set of events, referred to as an alphabet. Σ^* is the set of all strings on Σ, including the empty string ϵ. A subset $L \subseteq \Sigma^*$ is called a language. The concatenation of strings s, $u \in \Sigma^*$ is written as su. 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 $su = t$. The prefix-closure $\overline{L}$ of a language $L \subseteq \Sigma^*$ is the set of all prefixes of strings in L, i.e., $\overline{L} = \{s \in \Sigma^* | s \leq t$ for some $t \in L\}$.

A finite automaton is a 5-tuple $G = (Q, \Sigma, \delta, q_0, Q_m)$ where Q is a finite set of states, Σ 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 Σ^* as $\delta(q, \sigma s) = q'$ if $\delta(q, \sigma) = x$ and $\delta(x, s) = q'$. The generated and marked language are, respectively, $\mathcal{L}(G) = \{s \in \Sigma^* | \delta(q_0, s) = q' \wedge q' \in Q\}$ e $\mathcal{L}_m(G) = \{s \in \Sigma^* | \delta(q_0, s) = q' \wedge q' \in Q_m\}$. The active event function, $\Gamma : Q \to 2^{\Sigma}$, is a function that, given a state q, outputs the set of events $\sigma \in \Sigma$ for which $\delta(q, \sigma)$ is defined.

The natural projection $P_i : \Sigma^* \to \Sigma_i^*$ is an operation that maps the strings of Σ^* into strings of Σ_i^*, $\Sigma_i^* \subseteq \Sigma^*$, by erasing all the events that are not contained in Σ_i. The inverse projection $P_i^{-1} : \Sigma_i^* \to \Sigma^*$ is defined as $P_i^{-1}(L) = \{s \in \Sigma^* | P_i(s) \in L\}$ and is composed of all traces that, when projected, recovers traces from L.

The natural projection can have a known property called Observer Property (OP) presented in Definition 1.

Definition 1 *(Wong, 1998) Let $L \subseteq \Sigma^*$ be a language, $\Sigma_i \subseteq \Sigma$ an alphabet and $P : \Sigma^* \to \Sigma_i^*$ a natural projection. If $(\forall a \in \overline{L})(\forall b \in \Sigma_i^*)$, $P(a)b \in P(L) \Rightarrow (\exists c \in \Sigma^*) P(ac) = P(a)b$ and $ac \in L$, then $P(L)$ has the observer property.*

2.1 Supervisory Control Theory

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* and is represented by an automaton $G = (Q, \Sigma, \delta, q_0, Q_m)$ and $\Sigma = \Sigma_c \cup \Sigma_u$ where Σ_c is the set of controllable events, which can be disabled by an external agent and Σ_u is the set of uncontrollable events, which cannot be disabled by an external agent. There is also the *supervisor* that has to regulate the plant behavior to meet a desired behavior K disabling only controllable events, in a minimally restrictive way.

Let E be an automaton that represents the specification imposed over G. Language $K = \mathcal{L}_m(G||E) \subseteq \mathcal{L}_m(G)$ is controllable w.r.t G if $\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K}$. A non-blocking supervisor V for G such that $\mathcal{L}_m(V/G) = K$ exists if, and only if, K is controllable with respect to G. If K does not satisfy the condition, then the supremal controllable and non-blocking sub-language $S = sup\mathcal{C}(K, G)$ can be synthesized. S is also used to represent the automaton that implements the supervisor.

2.2 Supervisor Abstraction

In (Vilela & Pena, 2016) a supervisor abstraction, the natural projection of the supervisor to the controllable events, $P_S : \Sigma^* \to \Sigma_c^*$, is shown to keep a good property of the original supervisor. This is a theoretical result that allows to search for optimal solutions on a smaller universe, $P_S(S)$ instead or S. The above mentioned "good property" is that, once a trace $s_{opt} \in P_S(S)$ is picked, that trace, when lifted to the original alphabet, can be executed to the end. This result is presented in Theorem 1.

Theorem 1 *(Vilela & Pena, 2016) Let G be a plant, $S = sup\mathcal{C}(K, G)$ the supervisor and $P_S : \Sigma^* \to \Sigma_c^*$ a natural projection. For all sequences $s_{opt} \in P_S(S)$ and $A = P_S^{-1}(s_{opt}) \cap S$, if $P_S(S)$ has the observer property, then A is controllable with respect to $\mathcal{L}(G)$.*

Theorem 1 establishes that, under certain conditions over the projection, the lifted language A (composed of all traces of S that project to s_{opt}) is controllable. The verification of controllability guarantees that all the interleavings that may arise in the lifted trace, if possible in the plant, are possible when implementing the plan in the system. In other words, to implement the controllable trace it is not necessary to disable uncontrollable events.

2.3 SCO-Supervisory Control and Optimization

This methodology was proposed to address scheduling problems in a manufacturing cell (Silva et al., 2011). In this approach, the Supervisory Control Theory encoded the problem constraints and an evolutionary algorithm performed the optimization. Different optimization techniques were used (Silva et al., 2011; Oliveira et al., 2013; Pena et al., 2016). Among them, we mention the Clonal Selection Algorithm (CSA) (De Castro & Von Zuben, 2002), that is also used in this work. SCO could only find solutions for small batches of products and it was extended to deal with large batches (Costa et al., 2018). The SCO-CONCAT works by concatenating optimized solutions, obtained with the SCO, to compose a larger batch of products. Despite the improvements made along each iteration, this methodology has a low time efficiency. The main reason for that is the high number of unfeasible solutions generated during the optimization process.

2.4 Problem Definition

In this paper, the optimization problem is to minimize the total production time (makespan) required for a batch of products in a manufacturing system. Here, the problem is formally presented as stated in (Pena et al., 2016):

- Let Σ be the set of events associated to a plant (commands and responses), which is divided into controllable events Σ_c and uncontrollable events Σ_u;
- Let A be the set of instances of events from Σ which are associated to the production of the complete batch, and let $A^c = A \cap \Sigma_c$ be the subset of controllable events of A which are associated to the production of the complete batch;
- Let A_k be an ordered set of the events of A, representing a production schedule candidate, $|A| = |A_k|$, and let A^c_k denote the ordered subset of controllable events in A_k;
- Let $N = \{A^c_1, A^c_2, ..., A^c_{|N|}\}$ be the set of all permutations of the elements of A_c, which result in feasible sequences, i.e., all ordered sets composed with the elements of A_c;
- Let T_k denote the time elapsed while the plant processes the sequence A_k. If the sequence is unfeasible, $T_k = \infty$.

The scheduling optimization problem can be stated as:

$$A^* = \underset{k \in \{1,\ldots,|N|\}}{\text{argmin}} \quad T_k$$

3 Main Results

This section is organized in three parts. Initially, some definitions will be introduced, then an example will be used to illustrate its application and finally the proposed solution will be shown.

3.1 Definitions

Two new properties regarding the transitions in and out of a given state are presented.

Definition 2 *Let $G = (Q, \Sigma, \delta, q_0, Q_m)$ be a deterministic automaton. A state q is called a divergent state (DS) if $|\Gamma(q)| > 1$. The set $Q_D = \{q \in Q \mid \Gamma(q) > 1\}$ is the set of all DS states.*

A state is called divergent if the number of *active events* is greater than one.

The relationship between a string of events and the DS states along its path is established in Definition 3.

Definition 3 *Let $s \in \mathcal{L}_m(G)$, then $Q_{DS}(s) = \{q \in Q_D \mid \delta(q_0, s') = q \quad \forall s' \in \overline{s}\}$ is the set of divergent states in relation to s.*

Q_{DS} is the set of divergent states visited by a string s.

3.2 Proposed solution

To find a sequence of events that minimizes the makespan, the supervisor abstraction $P_S(S)$, Definitions 2 and 3 are used. The goal is to start from an initial sequence of controllable events and make small changes in the sequence, then evaluate the impact of such change in the overall makespan.

Example 1 *In Figure 1, the acyclic graph PS_{prod} represents all the sequences of controllable events that creates two products in a given manufacturing system. Lets consider*

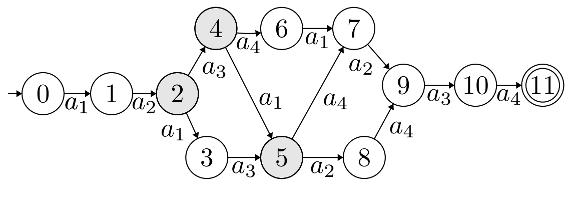

Figure 1: Acyclic graph (PS_{prod}) the Q_{DS} states are filled in gray.

an initial string of controllable events $s = a_1a_2a_3a_4a_1a_2a_3a_4\mathcal{L}_m(PS_{prod})$, $Q_D = \{2, 4, 5\}$

(filled gray). In this scenario, $Q_{DS}(s) = \{2, 4\}$, Figure 1.

The idea is to find a slightly different sequence. We do so by picking, from a divergent state, a different continuation. For instance, we can pick state $4 \in Q_{DS}(s)$ and pick a_1 and complete the sequence until the marked state. For each state in $Q_{DS}(s)$ a decision has to be made.

Since $\mathcal{L}_m(PS_{Prod}) \subseteq P_S(S)$ and $P_S(S)$ has the observer property, then any trace from $\mathcal{L}_m(PS_{Prod})$ can be executed in the complete system (Theorem 1). This idea is formalized in an algorithm in the following section.

3.3 Sequence Generation

The main idea is to take a string $s \in \mathcal{L}_m(PS_{prod})$ and perform a series of modifications on it to generate a new string s_{new} to be evaluated.

Initially, an individual (Ind) is represented by a 2-tuple ($Ind = (Q_{DS}, s)$), where Q_{DS} is the set of divergent states in relation to s and s is the sequence of events.

With the purpose of generating a new individual from Ind one parameter should be defined. This parameter is related to the idea of how much of the sequence is going to be modified. Its implementation in the algorithm is given by the percentage of preserved divergent states (λ) of $Q_{DS}(s)$.

The other inputs are: the active event function (Γ), the set of all states Q and the set of marked states (Q_m). The output solution is a 2-tuple (Q_{DSnew}, s_{new}), where $Q_{DS} \subseteq Q$ is the set of divergent states and s_{new} is the new complete sequence of controllable events.

Algorithm 1: Sequence Generator

```
   Input  : $Q, Q_m, Q_{DS}, s, \lambda, \Gamma$
   Output: $Q_{DSnew}, s_{new}$
1  $Q_{DSnew} \leftarrow Select_{DS}(Q_{DS}, \lambda)$
2  $q \leftarrow Last(Q_{DSnew})$
3  $s_{new} \leftarrow NewSequence(s, q)$
4  while $q \notin Q_m$ do
5      if $|\Gamma(q)| > 1$ then
6          if q $\notin Q_{DSnew}$ then $Q_{DSnew} \cup \{q\}$
7          $\sigma \leftarrow Random(\Gamma(q))$
8      else
9          $\sigma \leftarrow \Gamma(q)$
10     end
11     $s_{new} \leftarrow s_{new}\sigma$
12     $q \leftarrow \delta(q, \sigma)$
13 end
```

In lines 1 to 3, the initial $round(\lambda \cdot |Q_{DS}(s)|)$ states are kept and included in Q_{DSnew} (line 1) and the current state q becomes the last state in Q_{DSnew}(line 2). Also, the prefix of the new sequence is obtained (line 3), by running the automaton from the initial state to state q passing through states of Q_{DSnew}. The sequence s_{new} is then iteratively generated based on the possible continuations from the states reached (lines 4 to 13) until the marked state is reached. If another divergent state is reached in the path (line 5 to 8), the continuation from there is picked randomly (and stored in s_{new}) and the state is added to $Q_{DSnew}(s_{new})$.

Example 2 *Algorithm 1 is applied to Example 1. Let $s = a_1a_2a_3a_1a_2a_4a_3a_4$ and $\lambda = 0.7$. The set $Q_{DS}(s) = \{2, 4, 5\}$ is the set of divergent states that are in the path o string s. With $\lambda = 0.7$, 70% of the DS states in $Q_{DS}(s)$ are added to $Q_{DS}(s_{new}) = \{2, 4\}$. The string s_{new} is initialized as $s_{new} = a_1a_2a_3$, in line 3. At this point, we enter the while loop. Once state $4 \in Q_{DS}$, we run lines 5 to 8, and pick a continuation randomly. Suppose that event a_4 is picked in line 7. Then, $s_{new} = a_1a_2a_3a_4$ and $q \leftarrow 6$. From state 6 on, no divergent states are reached, so the complete sequence is going to be $s_{new} = a_1a_2a_3a_4a_1a_2a_3a_4$, it was obtained by the execution of the "else" in lines 8 and 9 until state $11 \in Q_m$ is reached.*

The greatest achievement in this work, in relation to SCO related approaches, relies on the string generation process that reduces the unfeasibility to zero.

4 Optimization Method

The optimization algorithm is presented and integrated with the ideas presented in Section 3.3.

4.1 Clonal Selection Algorithm

The optimization method used was the Clonal Selection Algorithm (CSA)(De Castro & Von Zuben, 2002), inspired in the principles of the immunological system of mammals. As presented by (Oliveira et al., 2013) the improvement of the solutions works as a metaphor for the immune system of the living organisms. First, several replicas of the current antibodies are made, with the most useful antibodies receiving more replicas. Then, these replicas are randomly mutated, this process generates solutions that are similar to the original, but they differ to some extent.

The pseudo-code for the CSA can be seen in Algorithm 2. There, the inputs for this algorithm are the number of individuals (N), the number of generations ($nGen$), the percentage preserved DS states from the original sequence (λ) and the number of products (nP). In the end, the best solutions (individuals) are returned.

In Algorithm 2 the initial population has $2N$ random individuals (uniform distribution), this is made to allow the exploration of the search space. Then, this population is time evaluated and the N best individuals (the ones with the smaller makespan) are selected for the cloning process.

The number of clones ($nClones$) produced for each individual (Ind) is calculated in terms

Algorithm 2: Clonal Selection Algorithm

```
    Input  : $N, nGen, \lambda, nP$
    Output: Solution
 1  Initial population of 2N individuals ($nP$)
 2  N Best individuals are selected
 3  for $j \leftarrow 0$ to $N$ do
 4      SelectedInd.add( BestIndividuala[j] )
 5  end
 6  while Stop criterion not achieved do
 7      for each Ind ← in Population do
 8          $TestedIndividual.add(Ind)$
 9          for $k \leftarrow 0$ to $Eval(Ind, noClones)$ do
10              $(Q_{DSnew}, S_{new}) \leftarrow$ Seq.Gen(Ind, $\lambda$)
11              $newInd \leftarrow Time.Ev(S_{new}, Q_{DSnew})$
12              $TestedIndividual.Add(newInd)$
13          end
14          $NextGen.add(TestedIndividual.Min())$
15          $TestedIndividual.Clear()$
16      end
17      $Population \leftarrow NextGen$
18      $CheckMakespan(Population, nGen)$
19  end
```

of a function called $Eval(Ind, noClones)$, that guarantees that the best individual generates $(\beta)nClones$ clones while the worst one generates $(1-\beta)nClones$ clones, line 10. The parameter β may assume values between 0 and 1, and the parameter $nClones$ represents the maximum number of clones. The fitness function used is based on the ranking of the solutions. The best solution, the ones with the smallest makespan, are selected for the next generation.

Regarding the mutation aspect, it is performed only on the clones (lines 10 to 14) and the parents are kept untouched, which can be seen in line 9. Besides, its intensity varies with a fitness function such that an individual with higher fitness (smaller makespan), suffers mutations that are less intense, while individuals with lower fitness (meaning higher production time) have a more aggressive mutation. The intensity of the mutation is implemented by the parameter λ. The mutation operation used was described in Section 3.3 and implemented in Algorithm 1 and it is presented in line 11 function Seq.Gen (Sequence Generation).

The stop criterion is accordingly to two conditions: the maximum number of generations ($MaxGen$), or the number of generations without improvements ($nGen$) is reached. In line 19 the function $CheckMakespan()$ takes the current population makespan and $nGen$.

5 Experimental Results

As a test for the proposed technique, the flexible manufacturing system (FMS) problem (de Queiroz et al., 2005) was used. The representation of the FMS can be seen in Figure 2. This production plant has two types of products, A and B. Product A is a block with a conical pin on top and product B is a block with a cylindrical painted pin. There are eight machines in the FMS, three Conveyors (C_1, C_2 and C_3), a Mill, a Lathe, a Robot, a Painting Device (PD) and an Assembly Machine (AM). These devices are connected through unitary buffers (B_1 to B_8). The events are represented by arrows, as can be seen in Figure 2.

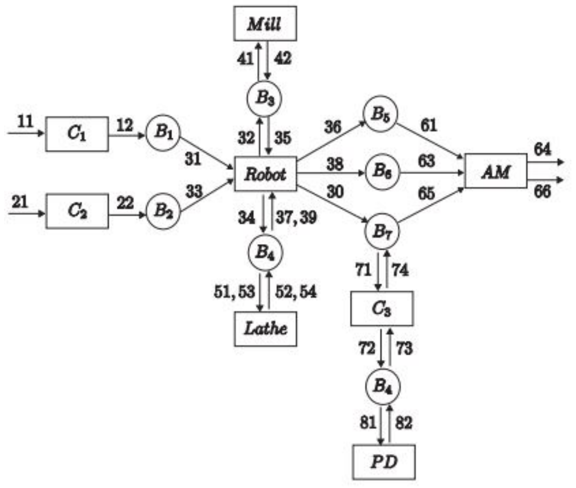

Figure 2: Flexible Manufacturing System.

5.1 The Sequencing Codification

The production of each product (A and B) can be expressed by a sequence of controllable events that needs to be executed in a specific order. To manufacture one unit of product A, it is necessary to combine two parts: one $Base$ and one Pin_A. In a similar way, a product B is composed by the pair $Base$ and Pin_B. The sequence of controllable events representing these three elements are:

- $Base:\ 11-31-41-35-61;$
- $Pin_A:\ 21-33-51-37-63;$
- $Pin_B:\ 21-33-53-39-71-81-73-65.$

Example 3 *Lets start with a sequential production of products A and B.*

- *Product A* $= Base + Pin_A = 11-31-41-35-61-21-33-51-37-63;$
- *Product B* $= Base + Pin_B = 11-31-41-35-61-21-33-53-39-71-81-73-65.$

Those sequences can be interleaved with each other in order to generate batches with different makespans. For each controllable event there is a correspondent uncontrollable event and the time between their occurrences (first the controllable that turns on a machine and then the uncontrollable that turns it off). Each interval represents the amount of time needed for a machine to finish its task. The only exception is the controllable event 61, that belongs to the Assembly Machine

(AM), which does not have a correspondent uncontrollable event. In Table 1, the pairs of events and their time interval (expressed in time units - t.u.) are listed according to the machines.

Table 1: Time interval between controllable and uncontrollable events for the FSM.

Machine	**Control. Events**	**Uncontrol. Events**	**Time Interval (t.u.)**
C1	11	12	26
C2	21	22	26
Robot	31	32	22
	33	34	20
	35	36	17
	37	38	25
	39	30	21
Mill	41	42	31
Lathe	51	52	39
	53	54	33
AM	61	-	15
	63	64	27
	65	66	27
C3	71	72	26
	73	74	26
PD	81	82	25

5.2 *Optimization Results*

In order to solve this problem, a controllable and non-blocking supervisor was synthesized, that resulted in an automata with 45.504 states and 200.124 transitions, in which the supervisor abstraction was applied to be used as the search space.

For this problem, any sequence representing the manufacturing of N_A products A and N_B products B, results in a string with $10N_A + 13N_B$ controllable events. This happens because there is a fixed amount of controllable events necessary to produce products A or B.

The optimization problem consists of producing batches (products A and B) with the same amount of products N_{prod}. In this scenario, N_{prod} was varied from $N_{prod} = 1$ to $N_{prod} = 10$ and for each pair of products (A and B) the makespan was evaluated.

With the aim of finding the best optimization performance, in different sets of products, the parameters of the evolutionary algorithm were tested. The chosen values were: $\lambda = 0.9$, $nGen = 10$, $nClones = 15$, $MaxGen = 60$ and the number of individuals $N = 25$. Lastly, for each batch of products 30 runs of the optimization algorithm were made.

All tests presented in this paper were executed on a notebook with an Intel Core I7-3537U 2.0 GHz processor and 8.0 GB of RAM. In addition, the UltraDES library (Martins et al., 2017) was used to compute the supervisor and its abstraction.

The optimization results using the DS approach can be seen in Table 2. To compare the findings, two other works were used, an implemented version of the Timed Maximum Parallelism (TMP) (Alves et al., 2016b) and the Formal Verification (FV) approach (Malik & Pena, 2018). In Table 2 the number of products A and B are identified as N_A and N_B. For each method two columns were created, one for the Runtime (total simulation time) and another for the Makespan. Under the DS Supervisor Abstraction, the total optimization time was considered, but in the Makespan column, only the best results were presented.

The only method that yields exact results is the Formal Verification method, what justifies its always higher runtime. Although the Timed Maximum Parallelism is the fastest method, its solution is still local optimal. The most surprising results came from the DS approach, which was a combination of the new sequence generation technique and the CSA. This method was not only capable to reach the optimal solution, but also did it three times faster than the verification method in the worst case ($N_A = 10, N_B = 10$).

6 Conclusion

The SCT solution is used as the search universe for the metaheuristics, like in the SCO approach. We present an approach for sequence generation that yields only feasible individuals for an optimization algorithm.

We use a theoretical result that allows to consider, as the search universe, the natural projection (to the alphabet of controllable events) of the supervisor reducing in a great extent the complexity of the problem. With this new sequence generation approach, we reduce the unfeasibility to zero, making the approach much more efficient.

The approach does not guarantee that the optimal solution will always be reached. However, our findings in the case study show that we were able to reach the optimal solution (the same solution obtained by the formal verification method) in all cases that it is known.

Acknowledgements

The authors acknowledge the support by the Brazilian agencies CNPq, CAPES and FAPEMIG.

References

Abdeddaïm, Y., Asarin, E. & Maler, O. (2006). Scheduling with Timed Automata, *Theoretical Computer Science* **354**(2): 272 – 300.

Alves, L. V., Bravo, H. J., Pena, P. N. & Takahashi, R. H. (2016a). Planning on discrete events systems: A logical approach, *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, IEEE, pp. 1055–1060.

Alves, L. V., Pena, P. N. & Takahashi, R. H. (2016b). Planejamento da produção baseado no critério do

Table 2: FMS Makespan optimization with three different methods

	Formal Verification		Maximum Parallelism		DS Supervisor Abstraction	
(N_A,N_B)	Runtime	Makespan	Runtime	Makespan	Runtime	Makespan
(1, 1)	0.6 min	**238**	0.0001 min	272	0.5 min	**238**
(2, 2)	2.7 min	**395**	0.0002 min	414	1.0 min	**395**
(3, 3)	5.0 min	**552**	0.0005 min	571	1.7 min	**552**
(4, 4)	7.5 min	**709**	0.0009 min	728	2.3 min	**709**
(5, 5)	10.2 min	**866**	0.0014 min	885	3.5 min	**866**
(6, 6)	14.0 min	**1023**	0.0021 min	1042	4.2 min	**1023**
(7, 7)	17.5 min	**1180**	0.0028 min	1199	5.2 min	**1180**
(8, 8)	21.5 min	**1337**	0.0037 min	1356	5.8 min	**1337**
(9, 9)	24.5 min	**1494**	0.0048 min	1513	7.4 min	**1494**
(10, 10)	30.2 min	**1651**	0.006 min	1670	7.9 min	**1651**

máximo paralelismo com restrições temporais, *Anais do XXI Congresso Brasileiro de Automática, CBA* .

Arisha, A., Young, P. & El Baradie, M. (2001). Job shop scheduling problem: an overview, *International Conference for Flexible Automation and Intelligent Manufacturing, FAIM 01* pp. 682–693.

Bellman, R. E. & Dreyfus, S. E. (2015). *Applied dynamic programming*, Princeton university press.

Cassandras, C. G. & Lafortune, S. (2009). *Introduction to discrete event systems*, Springer Science & Business Media.

Costa, T. A., Pena, P. N. & Takahashi, R. H. (2018). Scoconcat: a solution to a planning problem in flexible manufacturing systems using supervisory control theory and optimization techniques, *Journal of Control, Automation and Electrical Systems* pp. 1–12.

De Castro, L. N. & Von Zuben, F. J. (2002). Learning and optimization using the clonal selection principle, *IEEE Transactions on Evolutionary Computation* **6**(3): 239–251.

de Queiroz, M. H., Cury, J. E. & Wonham, W. M. (2005). Multitasking supervisory control of discrete-event systems, *Discrete Event Dynamic Systems: Theory and Applications* **15**(4): 375–395.

Garey, M. R. & Johnson, D. S. (1980). Computers and intractability: A guide to the theory of *np*-completeness, *Bulletin (New Series) of the American Mathematical Society* **3**(2): 898–904.

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

López-Mellado, E., Villanueva-Paredes, N. & Almeyda-Canepa, H. (2005). Modelling of batch production systems using petri nets with dynamic tokens, *Mathematics and Computers in Simulation* **67**(6): 541–558.

Malik, R. & Pena, P. N. (2018). Optimal task scheduling in a flexible manufacturing system using model checking, *2018 14th International Workshop on Discrete Event Systems, WODES 2018*.

Martins, L. R. R., Alves, L. V. R. & Pena, P. N. (2017). Ultrades-a library for modeling, analysis and control of discrete event systems, *Proceedings of the 20th World Congress of the International Federation of Automatic Control* **50**(1): 5831–5836.

Oliveira, A. C., Costa, T. A., Pena, P. N. & Takahashi, R. H. (2013). Clonal selection algorithms for task scheduling in a flexible manufacturing cell with supervisory control, *2013 Congress on Evolutionary Computation (CEC)*, IEEE, pp. 982–988.

Pena, P. N., Costa, T. A., Silva, R. S. & Takahashi, R. H. (2016). Control of flexible manufacturing systems under model uncertainty using supervisory control theory and evolutionary computation schedule synthesis, *Information Sciences* **329**: 491–502.

Pinedo, M. L. (2016). *Scheduling: theory, algorithms, and systems*, Springer.

Pinha, D. C., de Queiroz, M. H. & Cury, J. E. (2011). Optimal scheduling of a repair shipyard based on supervisory control theory, *2011 IEEE Conference on Automation Science and Engineering (CASE)*, IEEE, pp. 39–44.

Ramadge, P. J. & Wonham, W. M. (1989). The control of discrete event systems, *Proceedings of the IEEE* **77**(1): 81–98.

Silva, R. S., Oliveira, A. C., Pena, P. N. & Takahashi, R. H. (2011). Algoritmo clonal para job shop scheduling com controle supervisório, *X Simpósio Brasileiro de Automação Inteligente* pp. 1376–1381.

Su, R., Van Schuppen, J. H. & Rooda, J. E. (2012). The synthesis of time optimal supervisors by using heaps-of-pieces, *IEEE Transactions on Automatic Control* **57**(1): 105–118.

Vilela, J. N. & Pena, P. N. (2016). Supervisor abstraction to deal with planning problems in manufacturing systems, *2016 13th International Workshop on Discrete Event Systems, WODES 2016*, pp. 117–122.

Wang, W., Yuan, C. & Xiaobing, L. (2008). A fuzzy approach to multi-product mixed production job shop scheduling algorithm, *Proceedings - 5th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2008*, Vol. 1, IEEE, pp. 95–99.

Wong, K. (1998). On the complexity of projections of discrete-event systems, *Proceedings- 1998 International Workshop on Discrete Event Systems, WODES 1998*, pp. 201–206.