Abstraction of the Supervisory Control Solution to Deal With Planning Problems in Manufacturing Systems

Patrícia N. Pena, Juliana N. Vilela, Michel R. C. Alves, and Gustavo C. Rafael

***Abstract*—In industry, the performance has to be optimized to allow its competitiveness. The goal then is to use the resources at their maximal capacity, reduce the production time, and be flexible to adapt to the customers requests. The closed-loop behavior of a system under the supervisory control theory (SCT) guarantees nonblockingness and safety requirements and, thus, it can be used as the search universe for a planning problem. SCT suffers with the "curse of dimensionality" when systems become bigger and more complex. This article presents a set of sufficient conditions that allow to work with abstractions of the closed-loop behavior (supremal controllable sublanguage), instead of the closed-loop behavior itself, as the search universe to solve a planning problem. Such abstraction is the natural projection of the supervisor into the set of controllable events satisfying the observer property. This process leads to a reduction of the search space.**

***Index Terms*—Discrete event systems, manufacturing, optimization, supervisory control.**

I. Introduction

The industry paradigm has evolved from mass production to mass customization and nowadays to an emerging paradigm of personalization [18]. The machinery of factories changed to be flexible enough to support the changes of production. However, the industries still seek to produce more, faster, and using all the available resources.

Strategies that define how to comply with such demands are subject of study in many research areas, such as the operational research area. This problem is often called the *scheduling problem*.

In other words, the aim of a *scheduling problem* is to organize a set of tasks over a set of resources in such a way that it optimizes some criterion measure. Two temporal metrics that may be used are the makespan (execution time of a batch) and the throughput (amount of work completed during a fixed period of time). As manufacturing systems evolve to a more flexible production, minimization of makespan is likely to become more important than throughput [31].

Manuscript received May 29, 2020; revised October 8, 2020 and December 20, 2020; accepted January 15, 2021. Date of publication January 20, 2021; date of current version December 29, 2021. This work was supported in part by the National Council for Scientific and Technological Development - CNPq under grant 443656/2018-5, in part by the CAPES, Brazil, in part by the Fapemig, and in part by the PRPq-UFMG. Recommended by Associate Editor K. Cai. *(Corresponding author: Patrícia N. Pena.)*

Patrícia N. Pena is with the Department of Electronics Engineering, Universidade Federal de Minas Gerais, Belo Horizonte 31270-901, Brazil (e-mail: ppena@ufmg.br).

Juliana N. Vilela is with the Department of Electrical and Computer Engineering, University of Detroit Mercy, Detroit, MI 48221 USA (e-mail: juliananvilela@gmail.com).

Michel R. C. Alves and Gustavo C. Rafael are with the Graduate Program in Electrical Engineering, Universidade Federal de Minas Gerais, Belo Horizonte 31270-901, Brazil (e-mail: michel.chagasalves@hotmail.com; gustavo.caetanorafael@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TAC.2021.3053228.

Digital Object Identifier 10.1109/TAC.2021.3053228

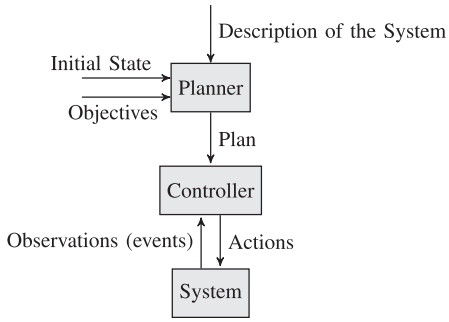

Fig. 1. Conceptual model for planning (adapted from [14]).

The scheduling problem is a combinatorial problem and, as such, may be classified as NP-hard. Thus, it is reasonable to expect that an exact solution may not be reached and heuristic methods are probably a good way to go. Heuristic methods provide suboptimal solutions with modest computational requirement [2].

A scheduling problem with resource allocation is a type of *planning problem* [14]. It takes as inputs the actions to be carried out together with resource constraints and an optimization criteria and returns a temporal organization and resource allocation for the actions to meet constraints and optimize the criteria.

A conceptual model for the solution of a planning problem has three main layers: the planner, the controller, and the system, as shown in Fig. 1, adapted from [14]. The planner receives the system model, the initial state, and the objectives to be achieved. At this layer, the best path is found and transferred to the controller as a *Plan*. The *Controller* executes the plan at the *System*, observing its dynamics through the *events* and operating over the system through the *actions*. The so-called *actions* can be controlled (disabled, forced) by the *Controller*, unlike the *events* that just happen. The relation with the model for SCT is clear. There are commands that originate in the control system—controllable events (under SCT approach) or "actions" (in Fig. 1)—and responses that originate in the system—uncontrollable events (under SCT) or "observations (events)" (in Fig. 1).

Considering the use of heuristics, the problem of representing the constraints inherent to the manufacturing system is very difficult, as recognized by [3], [5], [13], [35].

We use the classical untimed supervisory control theory (SCT) [24] to model the constraints. The structure provided by SCT represents the search space at a cost of being somehow conservative (an operation does not start if it is not safe to do so). Deterministic finite automata and SCT have been used to model the system's behavior, as seen in [1], [17], [19], [21], and [23]; or to solve the whole problem, as seen in [10], [15], and [26]–[28].

Because SCT models the logically legal behavior, it can be used as the search space of an optimization problem. However, the combinatorial nature of SCT causes the search to have high computational cost.

Abstractions have been used in the literature to generate the search space [15], [17], [30] of optimization problems. These approaches rely on the use of a new class of equivalence in their abstraction or on the use of tick automata to deal with the passage of time. All of them apply Dijkstra's algorithm or a modified version of it.

In this article, we extend the work developed in [29], where an abstraction of the closed-loop behavior obtained through natural projection is used for makespan analysis, by formulating an optimization problem, developing a heuristic, and illustrating the ideas in a manufacturing system of the literature. We propose to implement a version of the conceptual model of Fig. 1 [14], where the *Description of the System* is an abstracted version of the closed-loop behavior. The *Planner* runs a heuristic to generate a *Plan* that is composed only by *Actions* (controllable events). The *Objectives* are related to the size of the batch to be produced and the *Controller* implements a sublanguage of the closed-loop behavior that enforces the *Plan*.

The article is organized as follows. Section II introduces the necessary background. Section III presents the ideas of the article followed by, in Section IV-A, the main results. The optimization formulation is presented in Section IV-B and a case study is presented in Section V. Finally, concluding remarks are given in Section VI.

II. Preliminaries

This article is set in the SCT framework. The reader is referred to [4] for a detailed introduction to the SCT. Behaviors of a DES are modeled using strings of events taken from a finite event set Σ. Σ^* is the set of all finite strings of events in Σ, including the empty string ε. 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 $su = t$. A subset $L \subseteq \Sigma^*$ is called a language. The prefix-closure $\overline{L}$ of a language $L \subseteq \Sigma^*$ is the set of all prefixes of strings in L, i.e., $\overline{L} = \{s \in \Sigma^* |\ s \leq t \text{ for some } t \in L\}$. Finite state automata are used to recognize languages.

Definition 1: A finite-state automaton is a tuple $G = (Q, \Sigma, \rightarrow, q^0, Q^m)$, where Q is a finite set of states, Σ is a finite set of events, $\rightarrow \subseteq Q \times \Sigma \times Q$ is the state transition relation, $q^0 \in Q$ is the initial state, and $Q^m \subseteq Q$ is a finite set of marked states.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, meaning that from state x with event σ state y is reached, and is extended to strings in Σ^* by letting $x \xrightarrow{\varepsilon} x$ for all $x \in Q$, and $x \xrightarrow{s\sigma} z$ if $x \xrightarrow{s} y$ and $y \xrightarrow{\sigma} z$ for some $y \in Q$. Furthermore, $x \xrightarrow{s}$ means $x \xrightarrow{s} y$ for some $s \in \Sigma^*$, $x, y \in Q$. Also, if G is an automaton, then $G \xrightarrow{s}$ and $G \xrightarrow{s} X$ mean $q^0 \xrightarrow{s} x$, $x \in Q$ and $q^0 \xrightarrow{s} x$, $x \in X \subseteq Q$, respectively. Finally, the generated language of the automaton G is $\mathcal{L}(G) = \{s \in \Sigma^* \,|\, G \xrightarrow{s}\}$ and the marked language is $\mathcal{L}_m(G) = \{s \in \Sigma^* \,|\, G \xrightarrow{s} Q^m\}$. The active event function, defined by $\Gamma : Q \rightarrow 2^{\Sigma}$, is given by $\Gamma(q) = \{\sigma \in \Sigma | q \xrightarrow{\sigma}\}$. In other words, $\Gamma(q)$, $q \in Q$, returns the set of events enabled in such state. G is deterministic, if $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ always implies $y_1 = y_2$.

Given an automaton $G = (Q, \Sigma, \rightarrow, q^0, Q^m)$, a state $x \in Q$ is called accessible if $G \xrightarrow{t} x$, for some $t \in \mathcal{L}(G)$, and co-accessible if $x \xrightarrow{t} y$ for some $y \in Q^m$ and $t \in \Sigma^*$. An automaton G is called accessible if every state $x \in Q$ is accessible, and nonblocking if every accessible state $x \in Q$ is co-accessible, what corresponds to $\mathcal{L}(G) = \overline{\mathcal{L}_m(G)}$. The elimination of nonaccessible states yields an accessible component $G_{ac} = Ac(G)$.

We introduce a special type of automaton, the cyclic automata, an automaton that has a single marked state coinciding with the initial state, that does not contain self-loop transitions and is nonblocking.

Definition 2: [25] An automaton $G = (Q, \Sigma, \rightarrow, q^0, Q^m)$ is cyclic if G is nonblocking, $Q^m = \{q^0\}$, and for all events $\sigma \in \Sigma$ and for any $q_1, q_2 \in Q$ such that $q_1 \xrightarrow{\sigma} q_2$, it holds that $q_1 \neq q_2$.

The natural projection $\theta : \Sigma^* \rightarrow \Sigma_i^*$ is an operation over languages that maps strings in Σ^* to strings in Σ_i^*, $\Sigma_i \subseteq \Sigma$, by erasing all events not contained in Σ_i. This operation is defined for strings as $\theta(\varepsilon) = \varepsilon$ and $\theta(s\sigma) = \theta(s)$ if $s \in \Sigma^*, \sigma \notin \Sigma_i$ and $\theta(s\sigma) = \theta(s)\sigma$ if $s \in \Sigma^*, \sigma \in \Sigma_i$. The concept is extended to languages by defining $\theta(L) = \{t \in \Sigma_i^* | t = \theta(s) \text{ for some } s \in L\}$. The inverse projection maps the sequence taken from Σ_i^* into sequences in Σ^*, $\Sigma_i \subseteq \Sigma$. It is defined as $\theta^{-1}(t) = \{s \in \Sigma^* \,|\, \theta(s) = t\}$.

A natural projection may have the property known as the observer property [32], which is defined as follows.

Definition 3: [32] Let $L \subseteq \Sigma^*$ be a language, $\Sigma_1 \subseteq \Sigma$ be an event set, and $\theta : \Sigma^* \rightarrow \Sigma_1^*$ be the natural projection of strings in Σ^* to strings in Σ_1^*. If

$$(\forall a \in \overline{L})(\forall b \in \Sigma_1^*)\theta(a)b \in \theta(L) \Rightarrow (\exists c \in \Sigma^*)$$
$$\theta(ac) = \theta(a)b \text{ and } ac \in L$$

then, the natural projection $\theta(L)$ has the observer property.

If the observer property is satisfied, given an automaton and a natural projection, then the latter is "observation equivalent" to the former, which means that all branching in the automaton remains visible in its projection [33]. Furthermore, the projection will be smaller or at most of the same size of the original automaton (in case $\Sigma_i = \Sigma$). A projection with the observer property is called an OP-abstraction.

The parallel composition of two automata $G_1 = (Q_1, \Sigma_1, \rightarrow_1, q^{01}, Q^{m1})$ and $G_2 = (Q_2, \Sigma_2, \rightarrow_2, q^{02}, Q^{m2})$ is given by $G_1||G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \rightarrow, (q^{01}, q^{02}), \quad Q^{m1} \times Q^{m2})$, where $(x_1, y_1) \xrightarrow{\sigma} =$

$$= \begin{cases} (x_1 \xrightarrow{\sigma}_1, y_1 \xrightarrow{\sigma}_2) \text{ if } x_1 \xrightarrow{\sigma}_1 \text{ and } y_1 \xrightarrow{\sigma}_2 \text{ are defined} \\ (x_1 \xrightarrow{\sigma}_1, y_1) \text{ if } x_1 \xrightarrow{\sigma}_1 \text{ is defined and } \sigma \in \Sigma_1 \setminus \Sigma_2 \\ (x_1, y_1 \xrightarrow{\sigma}_2) \text{ if } y_1 \xrightarrow{\sigma}_2 \text{ is defined and } \sigma \in \Sigma_2 \setminus \Sigma_1 \\ \text{undefined otherwise.} \end{cases}$$

The parallel composition of two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ is calculated as

$$L_1||L_2 = \theta^{-1}_{\Sigma \rightarrow \Sigma_1}(L_1) \cap \theta^{-1}_{\Sigma \rightarrow \Sigma_2}(L_2)$$

with $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\theta_{\Sigma \rightarrow \Sigma_i} : \Sigma^* \rightarrow \Sigma_i^*$, $i = \{1, 2\}$.

A. Supervisory Control Theory

The set of events used to model a system G is partitioned into two subsets, $\Sigma = \Sigma_c \cup \Sigma_u$, where Σ_c is the subset of controllable events, and Σ_u the subset of uncontrollable events. The supervisor acts over the controllable events, restricting the plant's behavior to a subset of $\mathcal{L}(G)$. The desired language K is obtained by composing specifications E with $\mathcal{L}_m(G)$. Sometimes the implementation of K is not possible because uncontrollable events need to be prevented in order to attain such behavior, or K may be blocking. Blocking is corrected by taking the co-accessible component of the automaton that implements K. Before solving the other mentioned problem, the controllability of a language is formalized in Definition 4.

Definition 4: [24] A language $K \subseteq \Sigma^*$ is controllable with respect to a given plant G if

$$\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K}$$

where $\overline{K}\Sigma_u$ are strings in $\overline{K}$ concatenated with events from Σ_u.

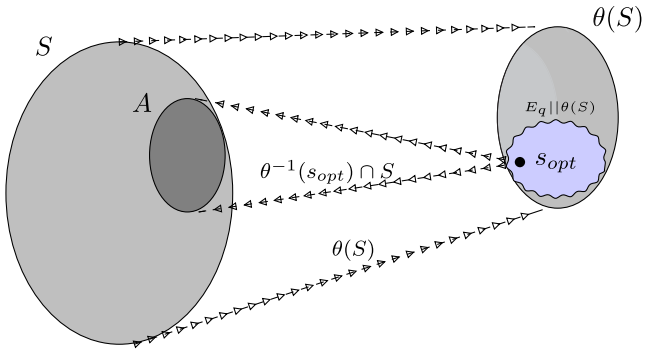

Fig. 2. Conceptual idea, start from S, obtain $\theta(S)$ and $E_q||\theta(S)$, pick $s_{\text{opt}} \in E_q||\theta(S)$.

If a language $K \subseteq \mathcal{L}_m(G)$ is controllable, then there exists a supervisor (control agent) that implements K. If K is not controllable, then there exists a supremal controllable sublanguage, S, that implements the most permissive and nonblocking behavior that does not violate the behavior imposed by the specifications. The supervisor, $\mathcal{S}$, also called monolithic supervisor, implements $S = Sup\mathcal{C}(K, \mathcal{L}(G))$.

Among the properties of controllability, it is important to highlight that if two given languages K_1 and K_2 are controllable, then $K_1 \cup K_2$ is controllable, but $K_1 \cap K_2$ need not be controllable.

III. Conceptual Idea

The SCT provides supervision to a system, without ever defining the best trajectory to be taken among the legal ones. So, at each state of the system, there may be more than one allowed continuation that can be executed. In a production system, it makes sense to search for the best trajectory, in order to minimize some cost; for instance, makespan or energy consumption.

We conceive the solution to a planning problem to be composed only of controllable events, namely, the sequence of commands that should be executed in order to minimize makespan. This idea is coherent with the conceptual model presented in Fig. 1 [14].

Let S be the supremal controllable sublanguage of the system, the monolithic supervisor, and $\theta(S)$ the supervisor abstraction, with $\theta : \Sigma^* \to \Sigma_c^*$. We aim to find a string, s_{opt}, of controllable events that produce a batch of size k. A solution, to be chosen as the best one, has to be compared only with other solutions that generate batches of the same size. In general, a supervisor and its abstraction are cyclic automata that contain all the strings of events that lead to the production of batches of all sizes. A language E_q is defined in such a way that, when composed with the supervisor language, collects the subset of strings that produce a number of products (and are to be compared to each other) and its automaton is acyclic. If two products are to be produced and the last controllable event in the production is α_3, then $E_q = \{\alpha_3\alpha_3\}$. Composing E_q with the abstraction, the language is restricted to the strings that produce such batch. E_q is formally defined later.

An optimal string s_{opt} is picked among all legal strings of controllable events, $s_{\text{opt}} \in E_q||\theta(S) \subseteq \theta(S) \subseteq \Sigma_c^*$. To establish that s_{opt} is a solution to the planning problem, it is necessary to make sure that such string can be implemented in the real system. Then, we build the sublanguage $A = \theta^{-1}(s_{\text{opt}}) \cap S \subseteq S$, that projects into s_{opt}. Such idea is illustrated in Fig. 2.

This language A can be seen as a new supervisor (the controller of Fig. 1) that implements a smaller language, enforcing the execution of the sequence of controllable events in s_{opt}, interleaved with the uncontrollable events as they wish to appear in the system.

The next step is the optimization. We associate durations to the operations, such that an uncontrollable event can only happen a number of time units later than its controllable counterpart. When we consider time feasibility, among all sequences in A, there is only one that is considered in the optimization algorithm, the one that has the controllable events being executed as soon as possible, and the uncontrollable interleaved whenever they are condition to the execution of the next controllable event in the plan s_{opt}. This is achieved by associating a zero duration to all controllable events. The uncontrollable events are allowed only after the duration of the operation that they conclude has passed.

In this article, we present conditions under which A is controllable. If such is true, we can do the search in $E_q||\theta(S)$. Then, we formulate an optimization problem and propose a heuristic to solve it.

IV. Main Results

This section is divided into four subsections starting from the theoretical result that is the foundation to the optimization and development of a heuristic to find a sequence in the abstraction that minimizes makespan.

A. Theoretical Results

We intend to use the abstraction, natural projection of the supremal controllable language (referred to as $\theta(S)$), as the search space of an optimization problem. As shown in the previous section, if the reconstructed language A is controllable, then the plan used to build it is executable to the end in the system. Theorem 1 establishes that if $\theta(S)$ has the observer property, then A is controllable.

Theorem 1: [29] Let G be a system, S be the monolithic supervisor implementing the supremal controllable sublanguage contained in a desired language K, and $\theta : \Sigma^* \to \Sigma_c^*$ be the natural projection. For all $s \in \theta(S)$ and $A = \theta^{-1}(s) \cap S$:

$\theta(S)$ has the observer property $\Longrightarrow A$ is controllable with respect to $\mathcal{L}(G)$.

Proof: To show that A is controllable with respect to $\mathcal{L}(G)$, $\forall a \in \overline{A}$ and $\forall \sigma \in \Sigma_u$ with $a\sigma \in \mathcal{L}(G)$, we must show that $a\sigma \in \overline{A}$. Pick any string $s \in \theta(S) \subseteq \Sigma_c^*$. Let $a \in \overline{S}$ be such that $\theta(a) \leq s$ and $a\sigma \in \mathcal{L}(G)$, with $\sigma \in \Sigma_u$. By construction, we also know that $a \in \overline{A}$.

Because S is controllable, we have

$$\forall \sigma \in \Sigma_u, a\sigma \in \mathcal{L}(G) \Rightarrow a\sigma \in \overline{S}. \tag{1}$$

From (1), we have that $\theta(a\sigma) \in \theta(\overline{S})$. Because of the initial assumption, $\theta(a) \leq s$, we know that $\exists b \in \Sigma_c^*$ such that

$$\theta(a)b = \theta(a\sigma)b = s \in \theta(S). \tag{2}$$

Since $\theta(S)$ has the observer property, Definition 3, $\exists t \in \Sigma^*$ such that

$$\theta(a\sigma t) = \theta(a\sigma)b \tag{3}$$

and

$$a\sigma t \in S. \tag{4}$$

From (2) and (3), we have that $\theta(a\sigma t) = s$. Thus

$$a\sigma t \in \theta^{-1}(s). \tag{5}$$

Using $A = \theta^{-1}(s) \cap S$, (4) and (5), we can conclude that $a\sigma t \in A$, and thus $a\sigma \in \overline{A}$. ◇

The result presented in Theorem 1 allows us to pick any $s \in \theta(S)$ and use it as a plan in the system, if $\theta(S)$ is an OP-abstraction.

The verification of the observer property can be done using different approaches: the OP-verifier [22], and algorithms in [11] and [34].

The language $\theta(S)$ is composed of the set of all possible (legal) sequences of controllable events that respect the specifications. However, it does not make any sense to compare different size batches. So, we use a sublanguage composed of all legal behaviors that produce the same amount of products as the search space of the optimization problem and that is done by using a specification E_q.

Definition 5: A language E_q is a *quantity specification*, for the production of a batch of k products, if it has the following characteristics:

1) $\Sigma_{E_q} = \{\sigma\}$, where σ is the last controllable event of the string that leads to the production of 1 product;
2) $E_q = \{v\} \subseteq \Sigma^*_{E_q}$, where $|v| = k$.

In more flexible systems, where more than one product can be produced, the language E_q should be adapted accordingly by building, for instance, E_q with more than one trace, formed with more events (from a larger alphabet), reflecting the mixed batch. Corollary 1 is presented, restricting Theorem 1 to the sublanguage A constructed from $s \in E_q||\theta(S) \subseteq \theta(S)$.

Corollary 1: Let S be the monolithic supervisor implementing the closed-loop behavior of a controlled system, E_q be the quantity specification according to Definition 5, and $\theta : \Sigma^* \to \Sigma_c^*$ be the natural projection and $\theta(S) \subseteq \Sigma_c^*$ be an OP-abstraction. Then

$s \in E_q||\theta(S) \Longrightarrow A = \theta^{-1}(s) \cap S$ is controllable.

Proof: As $\Sigma_{E_q} \subseteq \Sigma_c$, it follows that $E_q||\theta(S) \subseteq \theta(S)$. Then, $\forall s \in E_q||\theta(S)$, $s \in \theta(S)$, and thus, we can apply Theorem 1 and conclude that $A = \theta^{-1}(s) \cap S$ is controllable. ◇

Next, we formulate the optimization problem and present the heuristic proposed.

B. Formulation of the Optimization Problem

The optimization problem is to minimize the total production time (makespan) required for a batch of products in a manufacturing system. The problem is formally presented below, adapted from [23].

- Let Σ be the set of events associated to a plant, $\Sigma = \Sigma_c \cup \Sigma_u$.
- Let $\theta : \Sigma^* \to \Sigma_c^*$ be the natural projection, S be the supremal controllable sublanguage of K, $\theta(S)$ an OP-abstraction, and $E_q||\theta(S)$ the sublanguage of $\theta(S)$ that collects the traces that produce the same batch.
- Let T_s denote the time elapsed while the plant processes the plan $s \in E_q||\theta(S)$.

The scheduling optimization problem can be stated as

$$s_{opt} = \underset{s \in E_q||\theta(S)}{\operatorname{argmin}} \; T_s.$$

To obtain T_s, an individual $s \in E_q||\theta(S)$ is executed in a computational program that relies on simulation,[1] of DES, and implements the complete model S. The individual s is composed only by controllable events, that are set to have 0 duration. In order to execute such individual, the computational program interleaves available uncontrollable events when they are necessary to allow the continuation of the controllable sequence and updates T_s. This procedure is done in a way that respects the duration between each controllable event and its counterpart (see Table II). From this process, survives a sequence in S, with controllable events executed in the same order as they appear in the individual s, and that is able to produce a batch, along with its individual makespan (final value of T_s). It may be the case that more than one sequence survives, that may happen if more than one uncontrollable event is available with the same schedule. In any case, the heuristic deals with one sequence, that is the output of the computational program.

[1]Following the guidelines of [4] chapter 10.

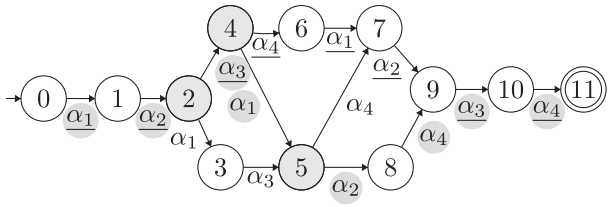

Fig. 3. Automaton M.

In the optimization process, a group of individuals is generated using the proposed heuristics, and the makespan is the metric used to evaluate them. Individuals with smaller makespans are classified as better compared to others with bigger makespans. Next, we present the heuristics proposed to generate new individuals from some original sequence.

C. Heuristic

This section proposes a methodology that uses a sequence and an acyclic automaton that runs that sequence to generate candidates to be evaluated and used to solve the optimization problem of Section IV-B. In this article, $\mathcal{L}_m(M) = E_q||\theta(S)$.

First, two new definitions regarding the transitions in and out of a given state of an automaton are presented.

Definition 6: Let $M = (Q, \_, \_, \_, \_)$ be a deterministic automaton and $\Gamma(q)$ its corresponding active event function. A state $q \in Q$ 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 DSs.

In other words, a state is classified as a DS if the number of active events is greater than one. Q_D is the set of all DSs.

Example 1: Let M be the automaton in Fig. 3 . The set of all DSs is $Q_D = \{2, 4, 5\}$, shaded, since $\Gamma(2) = \{\alpha_1, \alpha_3\}$, $\Gamma(4) = \{\alpha_1, \alpha_4\}$ e $\Gamma(5) = \{\alpha_2, \alpha_4\}$.

The set of DSs visited by a string s is named $Q_{DS}(s)$.

Definition 7: Consider $s \in \mathcal{L}(M)$, then $Q_{DS}(s) = \{q \in Q_D \mid q^0 \xrightarrow{x} q,\ x \leq s\}$ is the ordered set of DSs visited while executing s.

Definition 7 establishes a relationship between a string $s \in \mathcal{L}(M)$ and the DSs along its path.

Example 2: Let M be the automaton in Fig. 3 and let $s_1 = \alpha_1\alpha_2\alpha_3\alpha_4\alpha_1\alpha_2\alpha_3\alpha_4 \in \mathcal{L}_m(M)$, underlined in the figure. The set of all DSs in the path of s_1 is $Q_{DS}(s_1) = \{2, 4\}$.

The heuristic method consists on creating a new individual from an initial sequence $s \in \mathcal{L}_m(M)$, by changing the suffix of the string from a DS on. A parameter $0 \leq \lambda \leq 1$ is used to represent the amount of the sequence that is going to be modified. Specifically, λ represents the percentage of DSs (Q_{DS}), that are going to be kept unchanged in the sequence. If λ is 1 (or close to 1), it means that we will keep all (almost all) states in Q_{DS} and will obtain offspring that are similar to the parent. Now, when $\lambda = 0$ (or close to 0), we would not necessarily keep any of the states in Q_{DS}, and each new individual s' would differ much more from its parent. A pseudocode is presented in Algorithm 1.

The input of the algorithm is the string $s \in \mathcal{L}_m(M)$, the automaton tuple (M), the active event function Γ and λ. The output is a new string (s_{new}), built from s.

In Algorithm 1, line 1, function $Select_{DS}(Q_{DS}(s), \lambda))$ adds the initial $\lfloor \lambda \cdot |Q_{DS}(s)| \rfloor$ ($\lfloor x \rfloor$ gives as an output the greatest integer less than or equal to x) states to the ordered set $Q_{DS_{new}}$, such that $Q_{DS_{new}} \subseteq Q_{DS}(s)$. The last state of $Q_{DS_{new}}$ becomes the current state q (line 2) and $s_{new} \leq s$ is the string that runs from the initial state

Algorithm 1: Individual Generator.

Input : $s, M = (\_, \_, \rightarrow_M, q^0, Q^m), \Gamma, \lambda$
Output: s_{new}
1 $Q_{DS_{new}} \leftarrow Select_{DS}(Q_{DS}(s), \lambda)$
2 $q \leftarrow Last(Q_{DS_{new}})$
3 $s_{new} \leftarrow t : t \leq s \wedge q^0 \xrightarrow{t}_M q$
4 **while** $q \notin Q^m$ **do**
5 $\quad \sigma \leftarrow Random(\Gamma(q))$
6 $\quad s_{new} \leftarrow s_{new}\sigma$
7 $\quad q \leftarrow q_d : q \xrightarrow{\sigma}_M q_d$
8 **end**

to q, in line 3. String s_{new} is then iteratively augmented by randomly picking a continuation (lines 4–8) until the marked state is reached, what will eventually happen since there are no loops in M.

Example 3: Let M be the automaton in Fig. 3. Consider an initial sequence $s_2 = \alpha_1\alpha_2\alpha_3\alpha_1\alpha_2\alpha_4\alpha_3\alpha_4$ (shaded), with $Q_{DS}(s_2) = \{2, 4, 5\}$. Suppose a $\lambda = 0.7$, meaning that 70% of the DSs in $Q_{DS}(s_2)$ are kept in the new sequence. The DSs 2 and 4 in the ordered set $Q_{DS}(s_2)$ are kept (2 out of 3) and the string from the initial state that visits states 2 and 4 is $s'_{new} = \alpha_1\alpha_2\alpha_3 \in \mathcal{L}(M)$. The continuation from state $4 \in Q_{DS}$ is picked randomly. Suppose that event α_4 is picked, then $s''_{new} = \alpha_1\alpha_2\alpha_3\alpha_4$ and $M \xrightarrow{s''_{new}} 6$. From state 6 on, no DSs are reached, so the complete sequence is going to be $s_{new} = \alpha_1\alpha_2\alpha_3\alpha_4\alpha_1\alpha_2\alpha_3\alpha_4$. In Fig. 3, the resulting sequence s_{new} is underlined, and $Q_{DS}(s_{new}) = \{2, 4\}$.

As illustrated, the heuristic is used to build individuals from an initial sequence s_2 (in the example) of controllable events $s_2 \in \mathcal{L}_m(M) = E_q||\theta(S)$, changing a certain amount of the DSs. A percentage of $Q_{DS}(s)$ is kept, given by parameter λ.

This technique represents an efficient way to produce individuals to be evaluated in an optimization problem.

In the next section, we illustrate how the theoretical results are used in the solution of a planning problem. To do so, a flexible manufacturing system (FMS) is selected, the heuristic is applied in the context of a clonal selection algorithm, and the results are discussed.

V. Case Study

This section is divided into two subsections. First, we present the FMS and illustrate the evaluation of a sequence. Then, we discuss the results. The models and specifications have been presented in other papers such as [20] and [23] and will be omitted here.

A. Flexible Manufacturing System

In the FMS [9], two types of products can be manufactured, a product A, which is a block with a conical pin on the top, and product B, 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), and they are represented by rectangles (Fig. 4). These machines are connected through unitary buffers (B_1 to B_8), the circles. The events are represented by arrows, being the controllable events labeled with odd numbers while uncontrollable events are labeled with even numbers. The control objective is to avoid underflow and overflow of the buffers, and the correct flow of the pieces in the system. The monolithic supervisor $S = Sup\mathcal{C}(K, \mathcal{L}(G))$ is implemented by an automaton with 45 504 states and 200 124 transitions. The automaton that implements the OP-abstraction $\theta(S)$ has 1920 states and 7040 transitions.

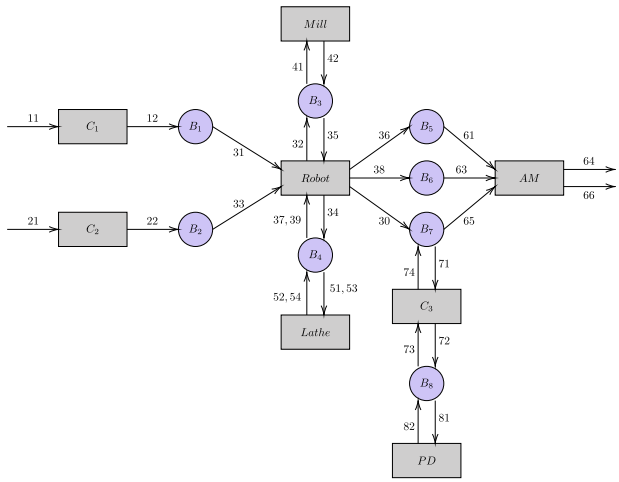

Fig. 4. Flexible manufacturing system.

TABLE I
FMS Time Interval (Expressed in Time Units, t.u.) Between Controllable and Uncontrollable Events

Machine	Events	Time (t.u.)	Machine	Events	Time (t.u.)
C_1	11/12	25	C_2	21/22	25
Robot	31/32	21	Mill	41/42	30
	33/34	19	C_3	71/72	25
	35/36	16		73/74	25
	37/38	24	AM	61	15
	39/30	20		63/64	25
Lathe	51/52	38		65/66	25
	53/54	32	PD	81/82	24

The production of each product (A and B) can be expressed by a sequence of controllable events (commands from the control system to the plant) that need 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 sequences of controllable events representing these three subproducts are as follows.

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

Those sequences can be interleaved with each other in order to generate batches with different makespans. As an example, we mention a string to produce one product A and one product B, in sequence. This is one of the legal behaviors, $s \in \theta(S) = \theta(Sup\mathcal{C}(K, \mathcal{L}(G))$, although it is probably not a good option in terms of makespan: $s = 11 - 31 - 41 - 35 - 61 - 21 - 33 - 51 - 37 - 63 - 11 - 31 - 41 - 35 - 61 - 21 - 33 - 53 - 39 - 71 - 81 - 73 - 65$ because it does not explore the potential to produce the two products, A and B, in parallel.

The computational program that implements the supervised system (automata for the subsystems and reduced supervisor), such as in [23], is used to lift the sequence of controllable events to a complete sequence, filling the uncontrollable events when they are timely feasible.

If we take a sequence to produce a batch composed of one product A, $s \in \theta(S)||\{63\}$, for instance: $s_1 = 11 - 21 - 31 - 33 - 41 - 35 - 61 - 51 - 37 - 63$, the makespan of such sequence, using Table I, is calculated by running such sequence in the closed-loop behavior (reduced supervisor and subsystems). Notice, from Table I, that all machines but the assembly machine have a corresponding uncontrollable event for each controllable event. In the assembly machine, events 63

TABLE II
EVALUATION OF A SEQUENCE s_1 FOR MAKESPAN

i	$s_1(i)$	Makespan $\overline{s_1}$	
1	**11**	0	
2	**21**	0	25
	12	25	
3	**31**	25	25
	22	25	
	32	46	21
4	**33**	46	
5	**41**	46	19
	34	65	30
	42	76	
6	**35**	76	16
	36	92	
7	**61**	92	
8	**51**	92	38
	52	130	max(92 + 15,makespan)
9	**37**	130	
	38	154	24
10	**63**	154	
	64	**179**	25

and 65 are conditioned by the controllable event 61 (there is no event 62). All the other controllable events are executed as soon as they appear in the sequence, with 0 duration.

Then, the execution of s_1 yields the lifted sequence (from Table II): $11-21-12-31-22-32-33-41-34-42-35-36-61-51-52-37-38-63-64$; with makespan 179 t.u.

For the batch (1,0), one product A, the number of possible solutions of the optimal sequence is searched in S is 6774. By searching in the abstraction, the number is reduced to 126 sequences.

We use the heuristic to generate individuals for the populations of a clonal selection algorithm (CSA) [7], an algorithm that is inspired in the principles of the immunological system of mammals.

In order to explore the search space, we start with 2*N* individuals and choose the best N individuals to compose the initial population. In the CSA, such individuals are cloned and mutated using the heuristic presented in Section IV-C, while the parents are left untouched. The mutation combines the parameter λ and the 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 fitness function used is based on the ranking of the solutions, which in this article is the makespan. Thus, the solutions with the smallest makespan are selected for the next generation. Details on the CSA can be found in [7].

The size of the population N is defined experimentally. A larger population (or a high number of clones) will cause the optimization to take longer to finish. A closer look may show that it has converged in the first or second generation. One way of mitigating the long run is to define a stop criteria related to the number of generations without improvements, rather than to fix a number of generations. Another way to mitigate is to reduce the size of the population. The tuning used for larger problems (larger batches) is going to work for the small problems, so an individual tuning is of interest when runtime is a concern.

In this article, we used $N = 25$ and $\lambda = 0.15$. Then, in each subsequent generation, for each individual, 15 clones were generated and mutated. We used a stop criteria of 11 generations without improvements and maximum of 30 generations.

B. Results

Regarding the makespan, the results obtained using the CSA with DS mutation can be seen in Table III. To compare the findings, the results from formal verification (FV) using model checking [20] are presented.

TABLE III
MAKESPAN AND EXECUTION TIME FOR SOME OF THE BATCHES, COMPARED TO THE FORMAL VERIFICATION METHOD—SMALLEST MAKESPAN IN BOLD FACE

	Formal Verification [20]		CSA-DS Mutation	
(nA,nB)	Runtime	Makespan	Runtime	Makespan
(1, 1)	0.6 min	**238**	0.5 min	**238**
(3, 3)	5.0 min	**552**	1.7 min	**552**
(5, 5)	10.2 min	**866**	3.5 min	**866**
(7, 7)	17.5 min	**1,180**	5.8 min	**1,180**
(10, 10)	30.2 min	**1,651**	9.5 min	**1,651**
(13, 13)	45.1 min	**2,122**	7.42min	2,155
(15, 15)	61.7 min	**2,436**	8.55 min	2,469
(17, 17)	-	-	10.06 min	**2,798**
(20, 20)	-	-	14.39 min	**3,308**

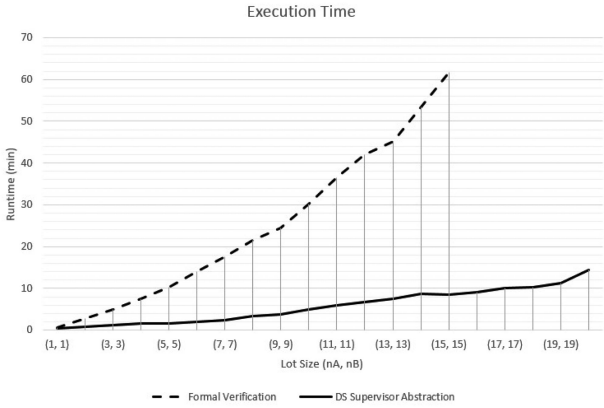

Fig. 5. Execution time of the two methods, from batch (1,1) to (20,20).

In Table III, the number of products A and B is identified as nA and nB. The size of the batch to be produced is increased from (1,1) to (20,20) (we show a subset of the batches in Table III). For each method, two columns were created, one for the runtime (total simulation time) and another for the makespan. The emphasized numbers in the third column of Table III represent the true optimal makespan, reached by using FV. It can be noticed that with the heuristic proposed, the same makespan is obtained for almost all batches (sequences emphasized in the last column), in a much shorter amount of time.

The FV approach was executed in the discrete event systems tool *Supremica* [20], making use of binary decision diagrams [6] to represent state sets symbolically, which greatly reduces memory consumption. Extended automata are used to model the system in Supremica, where the guards model the duration of the operations. Additional extended automata are created to capture the optimization goal. The method outputs a counterexample that is the sequence with the smallest makespan possible. The computation was done on a standard PC with a 2.8 GHz CPU and 16 GiB of RAM. The exact solution was found for batches up to the size (15,15). Such batch is computed in 61.7 min. There was not enough memory to store the reachable state-space when the size of the batch grew above (15,15).

The CSA-DS reaches the optimal solution (same makespan) up to size (11,11). Although from this batch on, suboptimal solutions are obtained, the increase in the execution time is much slower when compared to the FV approach, what can be seen in Fig. 5, confirming the capability of this approach to solve bigger problems.

VI. CONCLUSION

The SCT is a very interesting approach to implement safety restrictions, guaranteeing nonblockingness and permissiveness. However, it

does not care about performance. The closed-loop behavior implements the set of all legal behaviors of a system. Thus, it can be used as the search universe of a planning problem. This article shows under what conditions an abstraction can be used as the search universe instead. Such abstraction introduces great reduction on the state-space of the search universe problem and has the good property that any sequence picked from it can be executed in the system.

With fewer states, the optimization problem becomes easier to be solved and the solution more general. It is more general, because it is composed of the sequence of commands to be applied to the system but it translates into the language with the uncontrollable events, without forcing any order on them.

Any method can be used to search for the optimal sequence. We develop a heuristic to be used together with a CSA in order to find a solution to the planning problem. The case study illustrates the approach that solves a planning problem taking advantage of the concepts of the SCT.

As future work, we intend to use other optimization methods to try to find the solution faster. Also, we intend to extend the result to deal with local modular supervisors [8] and compositional supervisory control [12] to avoid the state-space explosion caused by the use of monolithic supervisory control.

References

[1] Y. Abdedda and O. Maler, "Job-shop scheduling using timed automata," in *Proc. Comput. Aided Verification*, 2001, pp. 478–492.

[2] K. R. Baker and D. Trietsch, *Principles of Sequencing and Scheduling*, 1st ed. Hoboken, NJ, USA: Wiley, 2009.

[3] R. L. Becerra and C. A. C. Coello, "Cultural algorithm for solving the job shop scheduling problem," in *Knowledge Incorporation in Evolutionary Computation*. Berlin, Germany: Springer, 2005, pp. 37–55.

[4] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. , NY, USA: Springer, 2008.

[5] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms I: Representation," *Comput. Ind. Eng.*, vol. 30,no. 4, pp. 983–997, 1996.

[6] E. M. Clarke Jr, O. Grumberg, and D. A. Peled. *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[7] L. N. de Castro and F. J. Von Zuben, "Learning and optimization using the clonal selection principle," *IEEE Trans. Evol. Comput.*, vol. 6, no. 3, pp. 239–251, Jun. 2002.

[8] M. H. de Queiroz and J. E. R. Cury, "Modular control of composed systems," in *Proc. Amer. Control Conf*, 2000, pp. 4051–4055.

[9] M. H. de Queiroz, J. E. R. Cury, and W. M. Wonham, "Multitasking supervisory control of discrete-event systems," *Discrete Event Dyn. Syst.*, vol. 15, no. 4, pp. 375–395, 2005.

[10] E. Fabre and L. Jezequel, "Distributed optimal planning: An approach by weighted automata calculus," in *Proc. IEEE Conf. Decis. Control 28th Chin. Control Conf.*, 2009, pp. 211–216.

[11] L. Feng and W. M. Wonham, "On the Computation of Natural Observers in Discrete-Event Systems," *Discrete Event Dyn. Syst.*, vol. 20, pp. 63–102, 2010.

[12] H. Flordal, R. Malik, M. Fabian, and K. Akesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discrete Event Dyn. Syst.*, vol. 17, pp. 475–504, 2007.

[13] M. Gen, R. Cheng, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms II: Hybrid genetic search strategies," *Comput. Ind. Eng.*, vol. 36, pp. 343–364, 1999.

[14] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning - Theory and Practice*. New York, NY, USA: Elsevier, 2004.

[15] F. Hagebring and B. Lennartson, "Compositional optimization of discrete event systems," in *Proc. IEEE 14th Int. Conf. Automat. Sci. Eng*., 2018, pp. 849–856.

[16] R. C. Hill and S. Lafortune, "Planning under abstraction within a supervisory control context," in *Proc. IEEE 55th Conf. Decis. Control*, 2016, pp. 4770–4777.

[17] S. Jack Hu, "Evolving paradigms of manufacturing: From mass production to mass customization and personalization," *Procedia CIRP*, vol. 7, pp. 3–8, 2013.

[18] A. Kobetski and M. Fabian, "Scheduling of discrete event systems using mixed integer linear programming," in *Proc. 8th Int. Workshop Discrete Event Syst.*, 2006, pp. 76–81.

[19] R. Malik and P. N. Pena, "Optimal task scheduling in a flexible manufacturing system using model checking," in *Proc. 14th Int. Workshop Discrete Event Syst.*, 2018, pp. 241–246.

[20] S. Panek, O. Stursberg, and S. Engell, "Job-shop scheduling by combining reachability analysis with linear programming," in *Proc. 7th Int. Workshop Discrete Event Syst.*, 2004, pp. 199–204.

[21] P. N. Pena, Hugo J. Bravo, A. E. Carrilho da Cunha, R. Malik, S. Lafortune, and J. E. R. Cury, "Verification of the observer property in discrete event systems," *IEEE Trans. Autom. Control*, vol. 59, no. 8, pp. 2176–2181, Aug. 2014.

[22] P. N. Pena, T. A. Costa, R. S. Silva, and R. H. C. Takahashi, "Control of flexible manufacturing systems under model uncertainty using supervisory control theory and evolutionary computation schedule synthesis," *Inf. Sci.*, vol. 329, pp. 491–502, 2016.

[23] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[24] K. W. Schmidt, "Optimal supervisory control of discrete event systems: Cyclicity and interleaving of tasks," *SIAM J. Control Optim.*, vol. 53, no. 3, pp. 1425–1439, 2015.

[25] R. Su, "Abstraction-based synthesis of timed supervisors for time-weighted systems," in *Proc. 11th Int. Workshop Discrete Event Syst.*, 2012, pp. 128–134.

[26] R. Su, J. H. van Schuppen, and J. E. Rooda, "The synthesis of time optimal supervisors by using heaps-of-pieces," *IEEE Trans. Autom. Control*, vol. 57, no. 1, pp. 105–118, Jan. 2012.

[27] R. Su and G. Woeginger, "String execution time for finite languages: Max is easy, min is hard," *Automatica*, vol. 47, pp. 2326–2329, 2011.

[28] J. N. Vilela and P. N. Pena, "Supervisor abstraction to deal with planning problems in manufacturing systems," in *Proc. 13th Int. Workshop Discrete Event Syst.*, 2016, pp. 117–122.

[29] S. Ware and R. Su, "Time optimal synthesis based upon sequential abstraction and its application to cluster tools," *IEEE Trans. Automat. Sci. Eng.*, vol. 14, no. 2, pp. 772–784, Apr. 2017.

[30] S. Ware and R. Su, "Time optimal synthesis based upon sequential abstraction and maximizing parallelism," in *Proc. IEEE 13th Conf. Automat. Sci. Eng*., 2017, pp. 926–931.

[31] K. C. Wong, J. G. Thistle, R. P. Malhame, and H. H. Hoang, " Supervisory control of distributed systems: Conflict resolution," in *Proc. 37th IEEE Conf. Decis. Control*, 1998, pp. 3275–3280.

[32] K. C. Wong and W. M. Wonham, "Hierarchical control of timed discrete-event systems," *Discrete Event Dyn. Syst.*, vol. 6, no. 3, pp. 275–306, Jul. 1996.

[33] K. C. Wong and W. M. Wonham, "On the computation of observers in discrete-event systems," *Discrete Event Dyn. Syst.*, vol. 8, pp. 247–297, 2004.

[34] R. Zhang and C. Wu, "A simulated annealing algorithm based on block properties for the job shop scheduling problem with total weighted tardiness objective," *Comput. Operations Res.*, vol. 38, pp. 854–867, 2011.