2021 60th IEEE Conference on Decision and Control (CDC)
December 13-15, 2021. Austin, Texas

Supervisory Control for Stabilization under Multiple Local Average Payoff Constraints

Yiding Ji, Xiang Yin, Wei Xiao

Abstract— **This work investigates stabilization of discrete event systems via supervisory control under a series of quantitative constraints. Every event in the system model is weighted by a vector which represents payoffs of quantitative variables associated by the event. Multidimensional weight flows are generated when events occur successively. The supervisor aims to drive the generated strings to a reach set of target states to stabilize the system. Meanwhile, the supervisor is also responsible for regulating the weight flows so as to guarantee that at each dimension, the average weight every a certain number of events does not fall below a given threshold. Next, the formulated supervisory control problem is transformed to a two-player game between the supervisor and the environment on a specially defined game structure. Then we specify the objective of the game and synthesize game winning supervisors, which turn out to provably solve the proposed problem.**

Index Terms— **discrete event systems, automata, supervisory control, stabilization, local average payoffs**

I. Introduction

Supervisory control is a central theme in the context of discrete event systems (DES). There is a plant under control and a specification is given to model the desired behavior of the plant. The supervisor issues control actions to restrict the behavior of the plant within the specification [3], [27].

Supervisory control theory has been comprehensively discussed in various DES models, among which automata and Petri nets are most commonly used. Many mechanisms of supervisory control have been developed so far: control under partial observation [31], [32], networked control [24], [35], decentralized control [13], [30], control of timed DES [21], [34], learning based control [28], [33], compositional control [6], [18], robust control [1], [17], online control [15], [22], supervisory control for DES with nondeterministic specifications [25], [29], to name a few. The conventional framework of qualitative supervisory control is also extended to quantitative settings, where supervisors are designed to achieve some measures defined over states and transitions. This topic has drawn considerable attention and been investigated under various frameworks, see, e.g., [7], [8], [10], [16], [19], [20], [26] for some recent advances.

Requirement for *stability* is ubiquitous in control engineering applications. As a motivating example, we consider a heating, ventilation and air-conditioning (HVAC) system which is designed to convey a heat transfer fluid to multiple air handlers, providing cooling or heating loads in a building. Ultimately, the system should be controlled to operate stably. The primary goals of controllers in an HVAC system are two fold: to drive/stabilize the system to a well functioning mode and regulate some flows, e.g., air flow and water flow, so as to maintain the temperature, moisture and air quality in the conditioned spaces. After the initial phase of fast cooling or heating, flows should maintain a relatively *stable/steady* rate without much fluctuation as operation continues, which is essential for the safe and smooth operation of the system. Here the flow rate is defined as the *average payload* every a certain number of time unites. The rate should not go beyond a predefined range so that the HVAC system may operate safely and smoothly. When there is a potential risk of violating the range, the controllers should issue commands to regulate the flows and prevent the violation in advance.

Inspired by the above situation, we investigate supervisory control for DES stabilization under multiple average payoff constraints in this paper. We consider a weighted automaton model where each event is associated with a weight vector to represent all types of *resources* or *payloads*. A predefined set of *target states* indicate completion of important tasks and a system is *stable* if all its generated strings visit target states after a finite number of transitions. In addition, a system is *live* if it does not terminate after any given number of events. Supervisors are employed to enforce these properties when necessary. Moreover, the major task of supervisors is to guarantee that the average weight (payoff) over a fixed number of transitions be above a given threshold at each dimension of the system, thus the flows are also "stable". Intuitively, the horizon to evaluate the mean weight may be interpreted as a "window" that rolls when new events occur.

Next we formulate the problem of *supervisory control for stabilization under stable-flow constraints* and solve it in sequence: the qualitative issues are addressed first, followed by quantitative ones. At the first phase, we introduce *window information states* to compactly encompass necessary information on states and local payoffs under control decisions and observations. Based on this concept, *window bipartite transition system (WBTS)* is defined, allowing us to transform the automaton model to a two-player game where the supervisor plays against the environment. After introducing the generic definition, we construct the largest WBTS that only contains stabilizing and live supervisors. At the second phase, we formulate a Büchi manner game [2] on the largest WBTS, by taking stable-flow constraints into consideration. In the end, we leverage well established algorithms to syn-

Second author's research supported in part by the National Natural Science Foundation of China, under grants 61803259, 61833012.

Yiding Ji is with Systems Hub, Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China, also with Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Kowloon, Hong Kong, China. {jiyiding}@ust.hk

Xiang Yin is with Department of Automation, Shanghai Jiao Tong University, Shanghai, China. {yinxiang}@sjtu.edu.cn

Wei Xiao is with Division of Systems Engineering, Boston University, Boston, Massachusetts, United States. {xiaowei}@bu.edu

thesize winning supervisors from the game and completely solve the original supervisory control problem.

This work generalizes the framework of stabilization in DES [14], [23] and is an extension of our prior works [11], [12] which explore supervisory control under single local average payoff constraint. In contrast to [11], we propose a distinct game structure based on window information states and apply a different game theoretic approach to solve the problem. The setting of this work also differs from the one in [12] where supervisory control is studied under partial observation. Moreover, the problem in [12] is solved under another game framework, namely, safety game. Therefore, the method in this work is significantly different and generally incomparable with the ones in [11], [12]. More recently, [9] solves an optimal stabilization problem for stochastic DES under a different framework. Our approach is also inspired by results in algorithmic game theory, especially mean payoff games [2], [4]. As far as we know, this work is the first one to investigate supervisory control for DES stabilization under multiple local average payoff constraints.

The following sections are organized as follows. Section II introduces the DES model and some preliminaries of this work. Section III formulates the stable-flow stabilization problem. In Section IV, a game theoretic approach is developed to solve the supervisory control problem of Section III. Finally we conclude the work in Section V.

II. System Model

We consider a quantitative discrete event system modeled by a weighted finite-state automaton:

$$G = (X, E, f, x_0, \omega)$$

where X is the finite state space, E is the finite set of events, $f : X \times E \to X$ is the partial transition function, $x_0 \in X$ is the initial state and $\omega : E \to \mathbb{Z}^k$ is the k-dimensional weight function that assigns an integer vector to each event. The vector reflects payloads or payoffs of quantitative variables associated with the event. Additionally, we say that G is of dimension k if ω is a k-dimensional function. Given $e \in E$, we let $\omega^{(i)}(e)$ be the i-th element of $\omega(e)$. The domain of f can be extended to $X \times E^*$ in the standard manner [3] and we still denote the extended function by f. The language generated by G is $\mathscr{L}(G) = \{s \in E^* : f(x_0, s)!\}$ where ! means "is defined". The function ω is additive and its domain can be extended to E^* by letting $\omega(\varepsilon) = 0$, $\omega(se) = \omega(s) + \omega(e)$ for all $s \in E^*$ and $e \in E$. We denote by W the maximum absolute value of event weights in G, i.e., $W = \max_{e \in E} \max_{1 \le i \le k} |\omega^{(i)}(e)|$.

For simplicity, we write $x_1 \xrightarrow{e} x_2$ if $f(x_1, e) = x_2$ for some $x_1, x_2 \in X$ and $e \in E$. A *run* in G is a finite or infinite sequence of alternating states and events in the form: $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_n} x_{n+1}$. A run is called *initial* if it starts from the initial state of G. We denote by $Run(G)$ and $Run_{inf}(G)$ the set of runs and infinite runs in G, respectively. For index $1 \le i \le n$, we call $x_i \xrightarrow{e_i} \cdots \xrightarrow{e_n} x_{n+1}$ a *suffix* of r and $x_1 \xrightarrow{e_1} \cdots \xrightarrow{e_i} x_{i+1}$ a *prefix* of r. In addition, for indexes j and m such that $1 \le j < m \le n$, we call $x_j \xrightarrow{e_j} x_{j+1} \xrightarrow{e_{j+1}} \cdots \xrightarrow{e_m} x_{m+1}$ a *fragment* of r, which is a run by itself. Furthermore, we let $r(j, m)$ stand for the run fragment starting from x_j and ending in x_{m+1}.

A set of states $X_T \subseteq X$ is of particular interests and is termed as the *target state* set. We call the system *stable* if all feasible strings will eventually reach target states. We also consider the (weak) *liveness* property: G is live if $\forall s \in \mathscr{L}(G)$, $\exists u \in E$, s.t. $su \in \mathscr{L}(G)$, i.e., there is a transition defined out of any state in G so that every finite run may be extended to be infinite. Weak liveness is without loss of generality as it can be relaxed by adding observable self-loops at terminal states where no active events are defined. When it comes to "liveness" in the following context, we mean weak liveness.

The system is controlled by a *supervisor* which dynamically enables and disables events to achieve the given specification [3]. Formally, a supervisor is a function $S : \mathscr{L}(G) \to \Gamma$. The event set E is partitioned as $E = E_c \cup E_{uc}$, where E_c and E_{uc} are the set of controllable and uncontrollable events, respectively. A control decision $\gamma \in 2^E$ is *admissible* if $E_{uc} \subseteq \gamma$, i.e., no uncontrollable event is disabled. We only consider admissible control decisions and let all events be *observable* in this work. We use S/G to represent the controlled system under S. Accordingly, we denote by $\mathscr{L}(S/G)$ the language generated in S/G and $Run(S/G)$ the set of runs in S/G. We call a supervisor S live if the controlled system S/G is live.

In this work, supervisors are also employed to *stabilize* the system. A *stabilizing supervisor* drives every string in the supervised system to reach target states eventually.

Definition 1 (Stabilizing Supervisor): Given G and target states X_T, a supervisor S is stabilizing if $\forall s \in \mathscr{L}(S/G)$, $\exists t \in E^*$ and $M_t \in \mathbb{N}^+$ such that $|t| \ge M_t \Rightarrow f(x_0, st) \in X_T$.

Apart from stabilization and liveness, we also consider quantitative properties of generated *weight flows* when events occur consecutively in G. Specifically, we discuss the *stability* of weight flows via *local* average weights (payoffs) per a limited number of transitions, which differs from the asymptotic mean weights in [9], [20]. Given a run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \xrightarrow{e_n} x_{n+1}$ in G, its *total payoff* is $\sum_{i=1}^{n} \omega(e_i)$ and *average payoff* is $\frac{1}{n} \sum_{i=1}^{n} \omega(e_i)$. Both types of payoffs are vectors when G is multidimensional. Given a vector $v \in \mathbb{Z}^k$, we also let $v(i)$ be the i-th element in v for $1 \le i \le k$. We may imagine that local average payoffs are evaluated within a "window". The weight flows have less fluctuation and are therefore more stable when the local average payoffs always remain in an appropriate range. The window size is *fixed* as a positive integer N in the following discussion.

Definition 2 (Stable-Flow Windows): Given G of dimension k, window size $N \in \mathbb{N}^+$ and threshold $v \in \mathbb{Z}^k$, a run $x_1 \xrightarrow{e_1} x_2 \cdots \xrightarrow{e_N} x_{N+1}$ in G forms an N-step stable-flow window if $\forall 1 \le j \le k$, $\exists 1 \le \ell \le N$ such that $\frac{1}{\ell} \sum_{i=1}^{\ell} \omega^{(j)}(e_i) \ge v(j)$.

A run is an (N-step) stable-flow window if at any dimension of G, the local average payoff turns to be no less than a threshold within at most N events. In other words, if the local mean payoff at dimension j is below $v(j)$, it should be compensated before the window reaches its end. The inequality in Definition 2 is the same as $\frac{1}{\ell} \sum_{i=1}^{i=\ell} (\omega^{(j)}(e_i) - v(j)) \ge 0$, i.e., we may subtract vector v from each event weight vector and equivalently evaluate whether the mean payoff is above 0 at every dimension. Without loss of generality, we assume that v be *a zero vector* throughout the remainder of the work.

The windows are "sliding" when new events occur. It is reasonable to require that stable-flow windows appear infinite often so that weight flows remain stable. When the system is live and never terminates, it is tolerable that stable-flow windows begin to appear after the system has been operating for a while. Then we introduce *stable-flow infinite runs*.

Definition 3 (Stable-Flow Infinite Runs): Given G of dimension k with set of target states X_T , window size $N \in \mathbb{N}^+$ and zero vector $v \in \mathbb{Z}^k$, we say that $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots \in Run_{inf}(G)$ is a stable-flow infinite run if $\exists i \geq 1$ such that $\forall j \geq i$, $r(j, j+N)$ forms a stable-flow window.

Note that we only require that stable-flow windows be perpetually formed from certain position x_i (not necessarily the initial state) of the run in Definition 3. That is, a stable-flow infinite run is *independent* of its finite prefixes. When there are non stable-flow infinite runs in the system, we apply supervisory control to address the issue.

III. Problem Formulation

In this section, we formulate the key problem to be studied: supervisory control for DES stabilization under stable-flow constraints, which aims to design a supervisor for state and weight flow stabilization. Both qualitative and quantitative conditions are involved in the problem. Then a useful property is introduced, which will contribute to the final solution of the problem.

Problem 1 (Stabilization under Stable-Flow Constraints): Given system G of dimension k with set of target states X_T and window size $N \in \mathbb{N}^+$, design a supervisor S such that: (i) S/G is both stable and live; (ii) any infinite run in S/G is a stable-flow infinite run.

The scenario in Problem 1 is illustrated in Figure 1, where the window size is $N = 3$. The supervisor's decisions regulate the local mean payoffs in every dimension of the system within the local horizon and drive the system to target states. Within the window, we calculate the average payoff over 1,2 and 3 events to see if the corresponding values are above or equal to the threshold. When an event happens at the current state, the horizon rolls one step ahead to the reached state where the supervisor issues new commands. One open question is whether we need to evaluate local average payoffs at every state of a run to determine stable-flow windows.

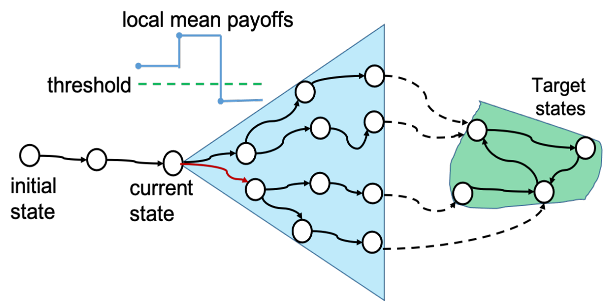

Fig. 1. State stabilization with local mean payoff constraints

Remark 1: Given a run $r = x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} \cdots$, suppose x_j with $j \leq N$ is the first position where a stable-flow window is formed at certain dimension m, i.e., $\sum_{i=1}^{j} \omega^{(m)}(e_i) \geq 0$ and $\sum_{i=1}^{j'} \omega^m a(e_i) < 0$ for all $j' < j$. By some simple derivation, we know that $\sum_{i=j'}^{j} \omega(e_i^{(m)}) \geq 0 > \sum_{i=1}^{j'-1} \omega^{(m)}(e_i)$ holds for any $j' < j$, otherwise it contradicts with x_j being the first place where a stable-flow window is formed. So any run fragment $r(j', j)$ also forms a stable-flow window. This indicates that we are safe to evaluate local average payoffs from position j, thus skip the positions before j. We call this observation as *inductive property* of stable-flow windows, which is leveraged in the next section to solve Problem 1.

Example 1: We consider a two-dimensional system G in Figure 2. The event set is partitioned as $E_c = \{a, b, c\}$ and $E_{uc} = \{u_1, u_2, u_3, u_4, u_5\}$. The set of target states is $X_T = \{x_3, x_6\}$ which are marked in blue. The event weight vectors are drawn next to the corresponding events in the figure. The window size is set to be $N = 3$. Apparently, G is not stable because of the self-loop at x_7, which never reaches X_T. In addition, $x_6 \xrightarrow{u_4} x_1 \xrightarrow{a} x_4 \xrightarrow{b} x_5 \xrightarrow{u_3} x_6 \xrightarrow{u_4} x_1 \xrightarrow{a} x_4 \xrightarrow{b} x_5 \cdots$ is not a stable-flow infinite run since its fragment $x_6 \xrightarrow{u_4} x_1 \xrightarrow{a} x_4 \xrightarrow{b} x_5$ is not a 3-step stable-flow window due to $\omega^{(1)}(u_4) < 0$, $\omega^{(2)}(u_4) < 0$, $\omega^{(1)}(u_4 a) < 0$, $\omega^{(2)}(u_4 a) < 0$, $\omega^{(1)}(u_4 ab) < 0$ and $\omega^{(2)}(u_4 ab) < 0$. Our goal is to solve Problem 1 on G and this example is used throughout the work.

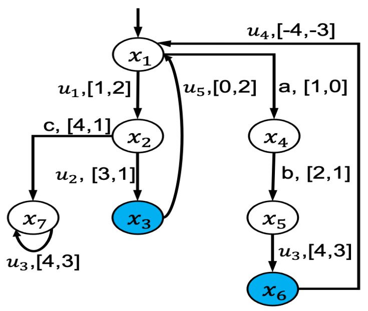

Fig. 2. The two-dimensional weighted automaton G in Example 1

IV. A Game Approach for Supervisor Synthesis

This section presents a game theoretic approach to solve Problem 1. As an intermediate step, the automaton model in Section II is first transformed to a two-player game structure, where the supervisor plays against the system (aka environment). We introduce *window information states* as the building block for the game structure termed *window bipartite transition system (WBTS)*. Then we formulate a game with properly defined objectives on a special WBTS. Finally we analyze the game and synthesize the supervisor's winning strategies which in turn correctly solve Problem 1.

A. Window Bipartite Transition System

The critical challenge ahead is to redefine the requirements of Problem 1 and find a succinct way to incorporate qualitative and quantitative information for the decision making of players on the game. For this purpose, we have Definition 4.

Definition 4 (Window Information States): Given G of dimension k and fixed window size N, a window information state is a tuple: $q^w = (x, (c_1^w, c_1^l), (c_2^w, c_2^l), \cdots, (c_k^w, c_k^l)) \in X \times$

$(\mathbb{Z} \times [0,1,\cdots,N])^k$ where x is a state in G and $\forall 1 \leq i \leq k$, c_i^w is the weight sum counter and c_i^l is the window length counter for dimension i, respectively.

We denote by Q_{wd} the set of window information states whose components include a state from automaton G and N pairs of "counters". Given $q^w \in Q_{wd}$, we use $Sta(q^w)$ to stand for the state from G, i.e., $Sta(q^w) = x$. Note that there are exactly k pairs in q^w, corresponding to the dimensions of G. Specifically, for a pair (c_i^w, c_i^l) in q^w, c_i^w reflects the current weight sum in dimension i when q^w is reached, and c_i^l represents the remaining length of the window to potentially achieve a nonnegative weight sum in dimension i from q^w. We call that c_i^w is *paired* with c_i^l and vise versa. The window length counter c_i^l is no greater than the window size N and we sill specify the range of weight sum counter c_i^w later.

If we augment window information states with control decisions, we denote by $Q_{wd}^a \subseteq Q_{wd} \times \Gamma$ the set of *augmented window information states*. For $q^{aw} = (x,(c_1^w,c_1^l),(c_2^w,c_2^l),\cdots,(c_k^w,c_k^l),\gamma) \in Q_{wd}^a$, we let $I_s(q^{aw})$ and $Ctr(q^{aw})$ be the window information state and control decision in q^{aw}, respectively, so that $q^{aw} = (I_s(q^{ag}), Ctr(q^{aw}))$. With a slight abuse of notation, we also denote by $Sta(q^{aw})$ the state component from G, i.e., $Sta(q^{aw}) = x$.

Next we formulate a two-player game between the supervisor and the environment. For this purpose, a game arena named *window bipartite transition system (WBTS)* is introduced to systematically characterize the update of window information states with control decisions and event occurrences, after the game is initiated. The generic definition is given as follows and a special WBTS is constructed as the basis for solving Problem 1 afterwards.

Definition 5 (Window Bipartite Transition System (WBTS)): A WBTS with respect to G of dimension k and window size N is a tuple $T = (Q_Y, Q_Z, E, \Gamma, f_{yz}, f_{zy}, \omega, y_0)$ where

- $Q_Y \subseteq Q_{wd}$ is the set of window information states;
- $Q_Z \subseteq Q_{wd}^a$ is the set of augmented window information states;
- E is the set of events of G;
- Γ is the set of (admissible) control decisions;
- $f_{yz} : Q_Y \times \Gamma \to Q_Z$ is the transition function from Q_Y states to Q_Z states where for $y \in Q_Y$, $\gamma \in \Gamma$ and $z \in Q_Z$, we have $f_{yz}(y,\gamma) = z \Rightarrow z = (y,\gamma)$;
- $f_{zy} : Q_Z \times E \to Q_Y$ is the transition function from Q_Z states to Q_Y states where for $z^a = (x,(c_1^w,c_1^l),(c_2^w,c_2^l),\cdots,(c_k^w,c_k^l),\gamma) \in Q_Z$, $e \in E$ and $y = (x',(\tilde{c}_1^w,\tilde{c}_1^l),(\tilde{c}_2^w,\tilde{c}_2^l),\cdots,(\tilde{c}_k^w,\tilde{c}_k^l)) \in Q_Y^g$, we have $f_{zy}(z,e) = y$ if and only if $e \in \gamma$, $x' = f(x,e)$ and one of the following conditions hold:

$$\tilde{c}_i^w = 0, \tilde{c}_i^l = N \text{ if } c_i^w + \omega^{(i)}(e) \geq 0$$
$$\tilde{c}_i^w = c_i^w + \omega^{(i)}(e), \tilde{c}_i^l = c_i^l - 1 \text{ if } c_i^w + \omega^{(i)}(e) < 0, c^l \geq 1;$$
$$\tilde{c}_i^w = \omega^{(i)}(e), \tilde{c}_i^l = N-1 \text{ if } c_i^w + \omega^{(i)}(e) < 0, c^l = 0;$$

- ω is the k-dimensional event weight function inherited from G and labels f_{zy} transitions;
- $y_0 = (x_0,(0,N),\cdots(0,N)) \in Q_Y$ is the initial state.

The supervisor plays from Q_Y states (Y-states for short) by making control decisions and the environment plays from Q_Z states (Z-states for short) by executing enabled events. For a Y-state y, we denote by $C_T(y)$ the set of control decisions defined at y. In essence, we characterize control decisions and event occurrences separately by defining these two types of transitions. Here f_{yz} transitions remember the most recent (admissible) decisions of the supervisor .

On the other hand, f_{zy} transitions are more involved, which represent the update of reachable states and accumulative payoffs within windows under event occurrences. Our goal is to keep track of the remaining window length before a nonnegative weight sum is achieved. There are three possible cases when a newly enabled event e occurs. First, once the weight sum turns nonnegative (within at most N events) at dimension i, we reset the i-th sum to 0 and the i-th window length counter to N, respectively. Then we start counting the weight sum of a new sequence of events from the current window information state. Second, if the current weight sum is still negative and the remaining window length is at least 1 at dimension i, we update the i-th weight counter by adding the i-th element of the event weight vector $\omega(e)$ and then reduce the i-th window length counter by 1. This implies that it is still possible to achieve a nonnegative weigh sum within N events. Third, if the current weight sum is negative and the remaining window length is 0 at certain dimension i, we set the i-th weight counter as $\omega^{(i)}(e)$ and set the i-th window length counter to $N-1$, i.e., we restart counting both the weight sum and the remaining window length after the i-th window length counter turns 0 with a negative i-th weight sum counter. Thus, for a pair of counters in a state of a WBTS, the minimum value of the weight counter is $-W \cdot N$ and the maximum value is W where W is the maximum absolute value of event weights mentioned in Section II. Event weight function ω is inherited from G and labels f_{zy} transitions. Finally, the initial state y_0 is $(x_0,(0,N),(0,N))\cdots(0,N))$ where all weight counters are set 0 and all length counters are set N before any event occurs.

For a state in a WBTS T, if one of its weight sum counters is negative while its paired window length counter equals 0, i.e., the window already reaches its maximum length N, we call this state *undesirable* as it implies that an unstable-flow window is formed. Furthermore, if a state in T contains a target state of G, we name it a target state of T.

In a WBTS, we call a Y-state y *terminal* if it has no successor states. When no active events are defined at $Sta(y)$ in G, the supervisor is unable to make any control decision and y is terminal. If a WBTS has no terminal Y-states, we call it *complete*. Additionally, a Z-state is called *terminal* if no transition is defined out of it as the supervisor disables all events. Terminal states should be avoided if we are to solve Problem 1, otherwise the supervised system is not live.

When the supervisor and environment take turns to play, runs are generated in a WBTS T and a run is of the form $r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{\gamma_n} z_n \xrightarrow{e_n} y_{n+1}$. We denote by $Run(T)$ and $Run_{inf}(T)$ the set of runs and infinite runs in T, respectively. We also denote by $Run_y(T)$ (respectively $Run_z(T)$) the set of runs whose last states are Y-states (respectively Z-states).

Example 2: In Figure 3 we present a window bipartite transition system with respect to G in Figure 2. Here we use rectangular and round to represent Y-states and Z-states, respectively. Control decisions from γ_1 to γ_8 label transitions from Y-states to Z-states, and enabled events label transitions from Z-states to Y-states. To simplify the graph, we do not draw event weights and omit pairs of weight and window length counters in each state.

As is seen, the supervisor has two choices γ_1 and γ_2 at Y-state y_0. If γ_1 is chosen, then Z-state z_1 is reached, where the environment has two choices: a and u_1. When event a occurs from z_1, Y-state y_1 is reached. Both weight counters and window length counters are updated when new states are added. The remaining part of the structure is interpreted in a similar manner.

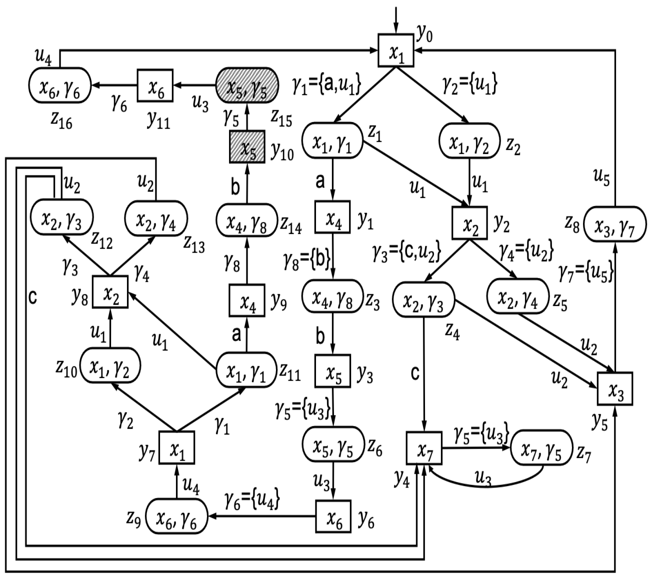

Fig. 3. A WBTS for Example 2

Generally, both players make decisions based on their history of observations and decisions and it is natural to consider their strategies in a WBTS T. A *supervisor's strategy (control strategy)* in T is defined as $\pi_s : Run_y(T) \to \Gamma$ and an *environment's strategy* is defined as $\pi_e : Run_z(T) \to E_o$. Given a control strategy π_s and a Y-state y, we define $Run(\pi_s, y, T) = \{y \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{\gamma_{n-1}} z_{n-1} \xrightarrow{e_{n-1}} y_n : \forall i < n, \gamma_i = \pi_s(y \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{\gamma_{i-1}} z_{i-1} \xrightarrow{e_{i-1}} y_i)\}$ as the set of runs starting from y and *consistent* with π_s, i.e., every control decision in a run $r \in Run(\pi_s, y)$ is specified by π_s. Since a control strategy preserves the mechanism of a standard supervisor in supervisory control theory [3], we will not distinguish between the terms "supervisor" and "supervisor's strategy (control strategy)" in the following discussion. We denote by π_s/G the supervised system under π_s and call π_s stabilizing live if π_s/G is stabilizing/live.

Roughly speaking, a strategy has *memory* if the player makes different decisions when the same state is reached again, otherwise, it is *memoryless*. The readers may refer to [2] for more details concerning the memory of strategies.

It is possible to explicitly "extract" a unique supervisor from a complete WBTS T if a control decision is specified at each Y-state of T. This supervisor is denoted S_T which is *realized* by an automaton $G_T = (Q_Y, E, \xi, y_0)$. We have $\xi : Q_Y \times E \to Q_Y$ as the transition function where $\forall y \in Q_Y$, $\forall e \in E$: $\xi(y, e) = f_{zy}(f_{yz}(y, \gamma), e)$ if γ is chosen at y and $e \in \gamma$. y_0 is the initial Y-state of T. The language of the supervised system is $\mathscr{L}(S_T/G) = \mathscr{L}(S_T \times G)$ where $\times$ is the standard product operation between automata [3].

In a WBTS T, we recursively define the supervisor's *attractor* with respect to a set of states Q in T:

$$\begin{aligned}
&Attr^T_{s,0}(Q) = Q \\
&Attr^T_{s,i+1}(Q) = \{y \in Q_Y \setminus Attr^T_{s,i}(Q) : \exists y \xrightarrow{\gamma} z \text{ s.t. } z \in Attr^T_{s,i}(Q)\} \\
&\cup \{z \in Q_Z \setminus Attr^T_{s,i}(Q) : \forall z \xrightarrow{e} y, y \in Attr^T_{s,i}(Q)\} \\
&Attr^T_s(Q) = \bigcup_{i \geq 0} Attr^T_{s,i}(Q) \qquad (1)
\end{aligned}$$

The above definition indicates that the supervisor is able to take certain actions to reach $Attr^T_{s,i}(Q)$ from $Attr^T_{s,i+1}(Q)$, while the environment is forced to enter $Attr^T_{s,i}(Q)$ from $Attr^T_{s,i+1}(Q)$. Therefore, $Attr^T_s(Q)$ returns the *largest* set of states where Q is reached *for sure* by the supervisor within a finite number of transitions, no matter what strategies taken by the environment. In other words, it is impossible for the supervisor to reach Q from states not in $Attr^T_s(Q)$. The environment's attractor with respect to Q is defined analogously and denoted by $Attr^T_e(Q)$. It is known that the attractor can be computed in linear time provided the game graph is finite [2], thus it takes $O(n(T))$ to compute $Attr^T_s(Q)$ where $n(T)$ denotes the number of transitions in T.

Given two WBTSs $T_1 = (Q^1_Y, Q^1_Z, E, \Gamma, f^1_{yz}, f^1_{zy}, \omega^1, y^1_0)$ and $T_2 = (Q^2_Y, Q^2_Z, E, \Gamma, f^2_{yz}, f^2_{zy}, \omega^2, y^2_0)$, we say T_1 is a *subgame* of T_2, denoted by $T_1 \sqsubseteq T_2$, if $Q^1_Y \subseteq Q^2_Y$, $Q^1_Z \subseteq Q^2_Z$ and for all $y \in Q^1_Y$, $z \in Q^1_Z$, $\gamma \in \Gamma$, $e \in E$, we have $f^1_{yz}(y, \gamma) = z \Rightarrow f^2_{yz}(y, \gamma) = z$ and $f^1_{zy}(z, e) = y \Rightarrow f^2_{zy}(z, e) = y$. Given a WBTS T and a set of states $Q \subseteq Q_Y \cup Q_Z$, we denote by $T' = T \downharpoonright Q$ if $T' \sqsubseteq T$ and Q is the state space of T', i.e., the game on T is restricted to a subgame with states in Q.

Now we are ready to solve Problem 1 in two steps. First we present Algorithm 1 to construct the biggest complete WBTS $T_b = (Q^b_Y, Q^b_Z, E, \Gamma, f^b_{yz}, f^b_{zy}, \omega, y_0)$ that encloses only stabilizing and live supervisors. Here being the biggest is in the graph merging sense, i.e., for any complete WBTS T that encloses only stabilizing and live supervisors, we have $T \sqsubseteq T_b$. By building T_b, we address stabilization and liveness issues of Problem 1. Then we formulate a game on T_b by taking stable-flow constraints into consideration and we postpone the discussion to the next subsection.

Algorithm 1 recursively builds the state space of T_b in a depth-first search manner following Definition 5. It is an extension of the algorithm of building the weighted bipartite transition system in [11], where the major discrepancy lies in the state space and transition functions. We consider window information states here and the way of defining transition functions is different from [11]. In Procedure *Extend*, we iteratively add window information states to track states and accumulative payoffs within windows. We denote by Q_{ta} and Q^a_{ud} the set of target states and undesirable states, respectively. Notice that only non-terminal Z-states

Algorithm 1: Build the WBTS for supervisor synthesis

```
   Input   : $G$, $N$
   Output  : $T_b = (Q_Y^b, Q_Z^b, E, \Gamma, f_{yz}^b, f_{zy}^b, \omega, y_0)$ w.r.t. $G$
1  $Q_Y^b = \{y_0\}$, $Q_Z^b = \emptyset$, $Q_{ud} = \emptyset$, $Q_{ta} = \emptyset$;
2  $Extend(y_0, G, N)$;
3  while there exist $Y$-states without successors do
4      remove all such $Y$-states and their predecessor
       $Z$-states, take the accessible part;
5  denote by $T_{pre}$ the resulting structure and $Q'_{ta}$ the states
   from $Q_{ta}$ that remain in $T_{pre}$, calculate $Attr_s^{T_{pre}}(Q'_{ta})$ ;
6  return $T_b = T_{pre} \downharpoonright Attr_s^{T_{pre}}(Q'_{ta})$;
   Procedure: $Extend(y, G, N)$
7  for $\gamma \in \Gamma$ do
8      $z = f_{yz}^b(y, \gamma)$ by Definition 5;
9      add $y \xrightarrow{\gamma} z$ to $f_{yz}^b$;
10     if $y \in Q_{ud}$ then
11         $Q_{ud} = Q_{ud} \cup \{z\}$;
12     if $z \notin Q_Z$ and $z$ is not terminal then
13         $Q_Z = Q_Z \cup \{z\}$;
14         if $Sta(z) \in X_T$ then
15             $Q_{ta} = Q_{ta} \cup \{z\}$;
16         for $e \in \gamma$ do
17             $y' = f_{zy}^b(z, e)$ by Definition 5, write down
               $y'$ as $(x', (c_1'^w, c_1'^l), (c_2'^w, c_2'^l), \cdots, (c_k'^w, c_k'^l))$
18             add $z \xrightarrow{e} y'$ to $f_{yz}^b$;
19             if $y' \notin Q_Y^b$ then
20                 $Q_Y^b = Q_Y^b \cup \{y'\}$;
21                 if $\exists 1 \leq i \leq k$ s.t. $c_i'^w < 0$ and
                   $c_i'^l = N$ then
22                     $Q_{ud} = Q_{ud} \cup \{y'\}$;
23                 $Extend(y', G, N)$;
```

are added, so the resulting structure after *Extend* may entail terminal Y-states without successors. We prune away such Y-states as well as their preceding Z-states in line 4, so that T_{pre} is complete and supervisors in T_{pre} are live. After removal of states, we calculate the supervisor's attractor with respect to target states remaining in T_{pre} and return T_b as the subgame of T_{pre} restricted to $Attr_s^{T_{pre}}(Q'_{ta})$. In this manner, supervisors in T_b are guaranteed to reach target states Q'_{ta}, thus stabilizing. Note that it is possible that the initial state y_0 is not included in T_b. Then there does not exist a supervisor that stabilizes G from the initial state, so there is no solution to Problem 1. We continue our discussion for supervisor synthesis to resolve Problem 1 if y_0 is included in T_b.

Theorem 1: Given G, a control strategy is stabilizing and live if and only if it is in T_b.

Proof: "Only if": By contradiction. Suppose a control strategy (supervisor) π_s is stabilizing and live, but it is not in T_b. There should be a WBTS T such that π_s is in T, which further implies that there exists a run consistent with π_s such that it is not in T_b. If it is the case, the union of T and T_b is strictly larger than T_b in the graph merging sense, which contradicts with T_b being the biggest WBTS.

"If": Given a control strategy π_s in T_b and for any infinite run $y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \xrightarrow{\gamma_2} z_2 \xrightarrow{e_2} \cdots$ consistent with π_s, there exists $n \geq 1$ such that $y_n \in Q'_{ta}$ and $Sta(y_n) \in X_T$ by Algorithm 1 which involves computing the supervisor's attractor with respect to target states. Thus X_T is eventually visited and π_s is stabilizing. Besides, π_s is also live as every state in T_b has successors so π_s/G never terminates. ■

Remark 2: Given G of dimension k and window size N, the state space of T_b is of size $O(|X| \times (\{-W \cdot N, \cdots, -1, 0, 1, \cdots, W\} \times \{0, 1, \cdots N\})^k) + O(|X| \times 2^{|E|} \times (\{-W \cdot N, \cdots, -1, 0, 1, \cdots, W\} \times \{0, 1, \cdots N\})^k) = O(|X| \cdot 2^{|E|} \cdot N^{2k} \cdot W^k)$. The first term (respectively second term) on the left side of the equation represents the maximal possible number of Y-states (respectively Z-states) in T_b. The range of weight sum counter values is $\{-W \cdot N, \cdots, -1, 0, 1, \cdots, W\}$ and comes from the second paragraph after Definition 5.

Example 3: We continue Example 1 and build T_b with respect to G following Algorithm 1. It turns out that after procedure *Extend* and removal of states by the While loop of Algorithm 1, the intermediate window bipartite transition system T_{pre} is exactly the structure in Figure 3. Two shaded states $y_{10} = (x_5, (-1, 0), (-2, 0))$ and $z_{15} = (x_5, (-1, 0), (-2, 0), \gamma_5)$ in Figure 3 are undesirable since their weigh counters are negative and window length counters are 0 by calculation. Two states y_7 and z_7 contain a non-target state x_7 from G and it is possible for the supervisor to reach these two states from y_0. Then by calculating the supervisor's attractor in line 5 of Algorithm 1 and restricting T_{pre} to its subgame, we obtain T_b in Figure 4 where the supervisor is sure to reach target states of G and stabilize the system. Note that any control strategy in T_b is also live as there are no terminal states in T_b.

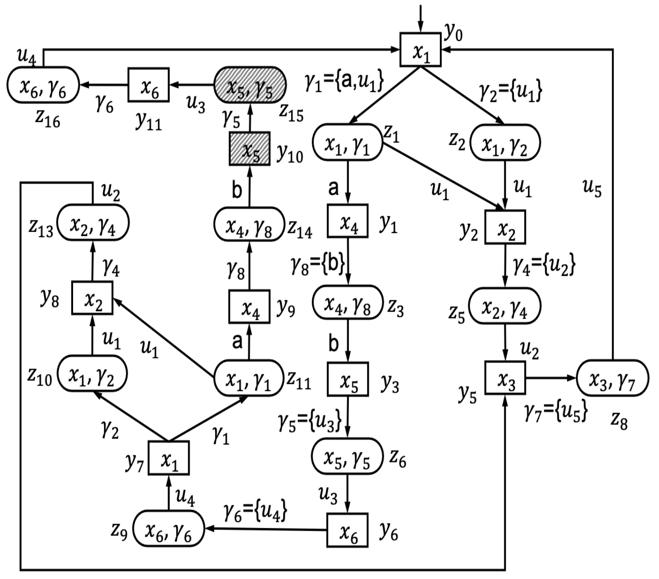

Fig. 4. T_b for Example 3

B. Game Winning Supervisor Synthesis

We have shown that supervisors in T_b are stabilizing and live, thus Problem 1 is partially solved. In what follows, we consider a game on T_b where the supervisor aims to achieve the *stable-flow window objective* and the environment aims to falsify the objective. Then we prove that winning strategies of the supervisor result in solutions to Problem 1.

Given a WBTS T, we define *stable-flow windows* of T in parallel with their counterparts of G:

$$Run_{sf}(T) = \{r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{e_N^a} y_{N+1} \in Run(T) :$$
$$\forall 1 \leq t \leq k, \exists 1 \leq \ell \leq N, \text{ s.t. } \frac{1}{\ell}\sum_{i=1}^{\ell} \omega^{(t)}(e_i) \geq 0\} \tag{2}$$

It is obvious that if $y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \xrightarrow{e_2} \cdots \xrightarrow{e_N} y_{N+1}$ is a stable-flow window in T, then $Sta(y_1) \xrightarrow{e_1} Sta(y_2) \xrightarrow{e_2} \cdots \xrightarrow{e_N^a} Sta(y_{N+1})$ is a stable-flow window in G. On the contrary, $y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{e_N} y_{N+1}$ is called an *unstable-flow window* in T if $\exists 1 \leq t \leq k$, $\forall 1 \leq \ell \leq N$, such that $\sum_{i=1}^{\ell} \omega^{(t)}(e_i) < 0$.

Then we consider infinite runs of T and define *stable-flow window objective* for both players as:

$$W_{sf}(T) = \{r = y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \cdots \in Run_{inf}(T) : \forall 1 \leq t \leq k,$$
$$\exists i \geq 1 \text{ s.t. } \forall j \geq i, \exists 1 \leq \ell \leq N, \frac{1}{\ell}\sum_{p=0}^{\ell-1} \omega(e_{j+p}^{(t)}) \geq 0\} \tag{3}$$

Simple observation shows that if infinite run $y_1 \xrightarrow{\gamma_1} z_1 \xrightarrow{e_1} y_2 \xrightarrow{\gamma_2} \cdots$ is in $W_{sf}(T)$, then $Sta(y_1) \xrightarrow{e_1} Sta(y_2) \xrightarrow{e_2} \cdots$ is a stable-flow infinite run in G. The supervisor is said to *achieve* $W_{sf}(T)$ if there exists a control strategy π_s such that any infinite run consistent with π_s is included in $W_{sf}(T)$.

We claim that if undesirable states in T_b are not visited infinitely often under a control strategy, the strategy achieves $W_{sf}(T_b)$ and leads to a supervisor satisfying the stable-flow constraints. Then we leverage results from co-Büchi games to synthesize supervisors and completely solve Problem 1.

Theorem 2: If a control strategy in T_b does not visit undesirable states infinitely often, then it achieves $W_{sf}(T_b)$ and solves Problem 1.

Proof: Suppose π_s is a control strategy in T_b such that all infinite runs consistent with π_s do not contain infinite undesirable states. Then given such a run r, there exists a Y-state y_i in r such that all Y-states after y_i are not undesirable. Next, we argue that from any Y-state y_j with $j \geq i$ in r, a stable-flow window is formed every N events, i.e., a nonnegative weight sum is achieved within the next N events at every dimension. We apply the inductive property mentioned at the end of Section III to show this argument. Assume that y_j^a is with $c_t^w = 0$ for some dimension $1 \leq t \leq k$ and $y_{j'}^a$ is the next state with $c_t^w = 0$ after y_j^g. From Algorithm 1, we know that weight sum counters of the states in T_b will get reset within at most N event occurrences, otherwise an undesirable state will be reached. Since the reset happens immediately after the weight sum counter turns nonnegative, all suffixes of the run fragment between y_j^a and $y_j^a j'$ has a nonnegative weight sum. Therefore, for all $j < j'' < j'$ and dimension t, a stable-flow window is formed. Consequently, r is in $W_{sf}(T_b)$ and the supervisor achieves the stable-flow window objective. Since π_s is already stabilizing and live by Theorem 1, it solves Problem 1. ■

Theorem 2 implies that if a supervisor achieves $W_{sf}(T_b)$ and solves Problem 1 by "eventually always" visiting desirable states. Clearly, this is a co-Büchi objective, which is the dual of standard Büchi objective [2]. Then we may leverage the *divide and conquer* manner algorithm in Section 3.2 of [2] to calculate the (supervisor's) *winning region*, a set of states from which the supervisor achieves $W_{sf}(T_b)$. The rough idea for the procedure of computing the supervisor's winning region is as follows. We inductively compute increasing over approximations of the supervisor's winning region at each round of iteration, "shrink" the game graph by removing states not in the last round's approximation result before the new iteration, then start a new round of computation. The approximated winning region is refined at each round and guaranteed to converge after a finite number of iterations [2].

To this end, we restrict the game on T_b to the supervisor's winning region, from which we choose a control decision at each Y-state and synthesize a strategy that provably solves Problem 1. Existing algorithms and tools abound for winning strategy synthesis of Büchi manner games, see, e.g., [5]. Technical details are omitted here and the readers may refer to [5] for a more comprehensive argument. Finally we extract a supervisor from the game graph following the argument in subsection IV-A as a unique decision has been specified at each Y-state. Notice that the resulting supervisor may have memory with respect to the state space of G since different control decisions may be issued at the same state of G when local average payoffs are considered for the stable-flow constraints. We end this section with the following example.

Example 4: We continue the discussion in Example 3 to synthesize a supervisor which completely solves Problem 1. Using the platform developed in [5], we obtain a winning strategy of the supervisor, which is indicated by green lines in Figure 5 and achieves the stable-flow window objective $W_{sf}(T)$. If the supervisor plays this strategy, then it will not visit the shaded states infinitely often. Finally we follow the discussion in subsection IV-A and convert the control strategy to a supervisor S shown in Figure 6 (the states are renamed to be consistent with G), which is live and stabilizes the system G in Figure 2 by driving all strings to reach target states. We may also verify that all infinite runs in S/G are stable-flow infinite runs since stable-flow windows are formed every three transitions. The supervisor has memory as it alternates between enabling and disabling b at state x_1.

V. Conclusion

We extended our prior results to consider a stabilization problem by supervisory control, which simultaneously requires multiple local mean payoffs fulfill some given bounds, for the first time in DES. Based on the concept of window information states, the stabilization problem was redefined as a two-player game between the supervisor and the environment on a specially built window bipartite transition system. After analyzing the game and defining proper objectives, we synthesized winning strategies of the supervisor, which were then converted to solve our proposed problem. For future work directions, we will explore more efficient representation

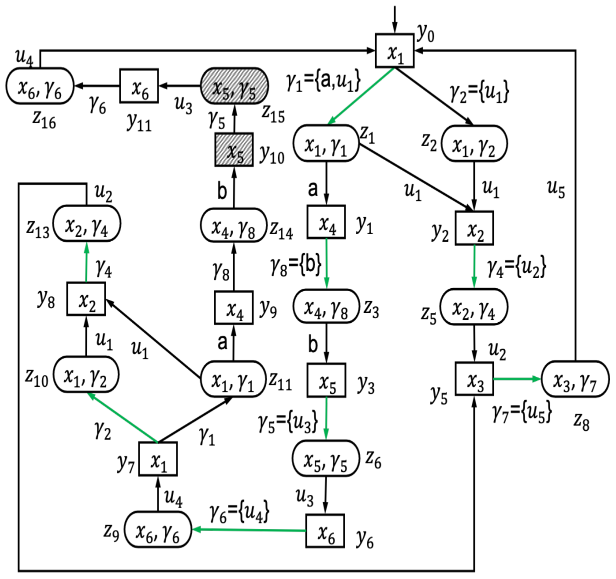

Fig. 5. A winning strategy of the supervisor marked in green lines

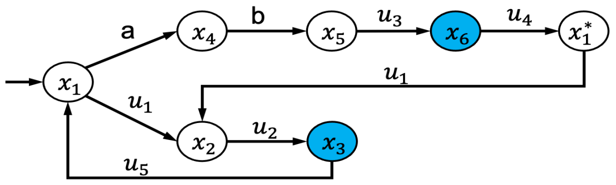

Fig. 6. A supervisor solving Problem 1

of the game structure and extend the framework of local average payoff supervisory control into stochastic settings.

REFERENCES

[1] M. V. S. Alves, J. C. Basilio, A. E. C. da Cunha, L. K. Carvalho, and M. V. Moreira. Robust supervisory control of discrete event systems against intermittent loss of observations. *International Journal of Control*, pages 1–13, 2019.

[2] K. R. Apt and E. Grädel. *Lectures in game theory for computer scientists*. Cambridge University Press, 2011.

[3] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems – 2nd Edition*. Springer, 2008.

[4] K. Chatterjee, L. Doyen, M. Randour, and J.-F. Raskin. Looking at mean-payoff and total-payoff through windows. *Information and Computation*, 242:25–52, 2015.

[5] C.-H. Cheng, A. Knoll, M. Luttenberger, and C. Buckl. Gavs+: An open platform for the research of algorithmic game solving. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 258–261, 2011.

[6] D. A. Ciolek, V. Braberman, N. DIppolito, S. Sardiña, and S. Uchitel. Compositional supervisory control via reactive synthesis and automated planning. *IEEE Transactions on Automatic Control*, 65(8):3502–3516, 2020.

[7] F. Hagebring and B. Lennartson. Time-optimal control of large-scale systems of systems using compositional optimization. *Discrete Event Dynamic Systems:Theory and Applications*, 29(3):411–443, 2019.

[8] X. Han, Z. Chen, and R. Su. Synthesis of minimally restrictive optimal stability-enforcing supervisors for nondeterministic discrete event systems. *Systems & Control Letters*, 123:33–39, 2019.

[9] Y. Ji and X. Yin. Optimal stabilization of discrete event systems with guaranteed worst cost. In *21st IFAC World Congress*, pages 1741–1746, 2020.

[10] Y. Ji, X. Yin, and S. Lafortune. Optimal supervisory control with mean payoff objectives and under partial observation. *Automatica*, 123:109359, 2021.

[11] Y. Ji, X. Yin, and S. Lafortune. Local mean payoff supervisory control for discrete event systems. *IEEE Transactions on Automatic Control*, DOI: 10.1109/TAC.2021.3075186, 2021.

[12] Y. Ji, X. Yin, and W. Xiao. Local mean payoff supervisory control under partial observation. In *15th IFAC International Workshop on Discrete Event Systems*, pages 390–396, 2020.

[13] J. Komenda and T. Masopust. Computation of controllable and coobservable sublanguages in decentralized supervisory control via communication. *Discrete Event Dynamic Systems: Theory and Applications*, 27(4):585–608, 2017.

[14] R. Kumar, V. Garg, and S. I. Marcus. Language stability and stabilizability of discrete event dynamical systems. *SIAM Journal on Control and Optimization*, 31(5):1294–1320, 1993.

[15] Z. Liu, X. Yin, S. Shu, F. Lin, and S. Li. Online supervisory control of networked discrete-event systems with control delays. *IEEE Trans. on Automatic Control*, DOI: 10.1109/TAC.2021.3080495, 2021.

[16] Z. Ma, M. Zou, J. Zhang, and Z. Li. Design of optimal control sequences in petri nets using basis marking analysis. *IEEE Transactions on Automatic Control*, DOI: 10.1109/TAC.2021.3106883, 2021.

[17] R. Meira-Góes, S. Lafortune, and H. Marchand. Synthesis of supervisors robust against sensor deception attacks. *IEEE Transactions on Automatic Control*, DOI: 10.1109/TAC.2021.3051459, 2021.

[18] S. Mohajerani, R. Malik, and M. Fabian. Compositional synthesis of supervisors in the form of state machines and state maps. *Automatica*, 76:277–281, 2017.

[19] V. Pantelic and M. Lawford. Optimal supervisory control of probabilistic discrete event systems. *IEEE Transactions on Automatic Control*, 57(5):1110–1124, 2012.

[20] S. Pruekprasert, T. Ushio, and T. Kanazawa. Quantitative supervisory control game for discrete event systems. *IEEE Transactions on Automatic Control*, 61(10):2987–3000, 2016.

[21] A. Rashidinejad, M. Reniers, and L. Feng. Supervisory control of timed discrete-event systems subject to communication delays and non-FIFO observations. In *14th IFAC International Workshop on Discrete Event Systems.*, pages 456–463, 2018.

[22] A. Sakakibara and T. Ushi. On-line permissive supervisory control of discrete event systems for scLTL specifications. *IEEE Control Systems Letters*, 4(3):530–535, 2020.

[23] K. W. Schmidt and C. Breindl. A framework for state attraction of discrete event systems under partial observation. *Information Sciences*, 281:265–280, 2014.

[24] S. Shu and F. Lin. Supervisor synthesis for networked discrete event systems with communication delays. *IEEE Transactions on Automatic Control*, 60(8):2183–2188, 2015.

[25] S. Takai. Synthesis of maximally permissive supervisors for nondeterministic discrete event systems with nondeterministic specifications. *IEEE Transactions on Automatic Control*, 66(7):3197 – 3204, 2021.

[26] B. J. C. van Putten, B. van der Sanden, M. Reniers, J. Voeten, and R. Schiffelers. Supervisor synthesis and throughput optimization of partially-controllable manufacturing systems. *Discrete Event Dynamic Systems:Theory and Applications*, 31:103–135, 2020.

[27] W. M. Wonham and K. Cai. *Supervisory control of discrete-event systems*. Springer, 2019.

[28] B. Wu, X. Zhang, and H. Lin. Permissive supervisor synthesis for Markov decision processes through learning. *IEEE Transactions on Automatic Control*, 64(8):3332 – 3338, 2019.

[29] X. Yin. Supervisor synthesis for Mealy automata with output functions: A model transformation approach. *IEEE Transactions on Automatic Control*, 62(5):2576–2581, 2017.

[30] X. Yin and S. Lafortune. Decentralized supervisory control with intersection-based architecture. *IEEE Transactions on Automatic Control*, 61(11):3644–3650, 2016.

[31] X. Yin and S. Lafortune. Synthesis of maximally permissive supervisors for partially-observed discrete-event systems. *IEEE Transactions on Automatic Control*, 61(5):1239–1254, 2016.

[32] X. Yin and S. Lafortune. A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems. *IEEE Trans. on Automatic Control*, 61(8):2140–2154, 2016.

[33] H. Zhang, L. Feng, and Z. Li. A learning-based synthesis approach to the supremal nonblocking supervisor of discrete-event systems. *IEEE Transactions on Automatic Control*, 63(10):3345–3360, 2018.

[34] R. Zhang, K. Cai, Y. Gan, Z. Wang, and W. M. Wonham. Supervision localization of timed discrete-event systems. *Automatica*, 49(9):2786–2794, 2013.

[35] Y. Zhu, L. Lin, S. Ware, and R. Su. Supervisor synthesis for networked discrete event systems with communication delays and lossy channels. In *58th IEEE Conf. on Decision and Control*, pages 6730–6735, 2019.