Regret-Optimal Supervisory Control of Partially-Known Discrete-Event Systems

Jianing Zhao†, Bohan Cui†, Dimos V. Dimarogonas, Rupak Majumdar and Xiang Yin

***Abstract*— In this paper, we investigate the optimal supervisory control problem for reachability tasks in *partially-known* discrete-event systems (DES). Specifically, we assume that the supervisor lacks prior knowledge of feasible events in certain states and can only acquire this information by visiting them. To evaluate the performance, we introduce *regret* as a metric, quantifying the gap between the actual cost and the best-response cost the supervisor could have achieved. We formalize this problem setting and develop an efficient algorithm to compute an optimal supervisor—one that ensures reachability while minimizing regret. A case study demonstrates that (i) regret is a meaningful performance measure in supervisory control of partially-known DES and (ii) our approach is both correct and effective.**

I. Introduction

Discrete-Event Systems (DES) are an important class of systems with discrete state spaces and event-triggered dynamics [1]. They play an essential role in modeling, analysis and control of high-level behaviors for engineering cyber-physical systems such as manufacturing systems, embedded software, and communication networks. In the context of DES, one of the central problems is how to enforce the closed-loop properties of the system. Supervisory Control Theory (SCT), initiated by Ramadge and Wonham [2], is a powerful formal framework widely used for the synthesis of DES controllers under some desired specifications, such as safety, liveness and nonblockingness; see, e.g., the textbook [3] and some recent works [4]–[12].

One important branch in SCT is the synthesis of an optimal supervisor with respect to a given performance measure. This problem, known as the optimal supervisory control problem, has been widely investigated in the literature; see, e.g., [13]–[19]. Particularly, in [16], the authors proposed an optimal supervisory control framework considering both the occurrence cost and disablement cost. The supervisor's objective is to reach marked states optimally in terms of the worst-case total accumulated cost. Following this framework, in [13], a liveness task and the worst-case average cost are considered. In [20], the authors further extended the specification to the cyclic task in which the supervisor is required to visit marked states infinitely.

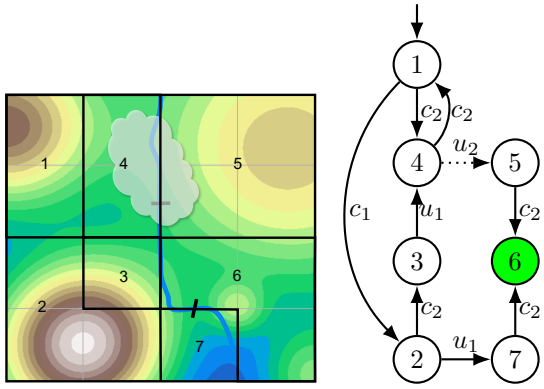
This work was supported by the National Natural Science Foundation of China (62061136004, 62173226, 61803259).

J. Zhao, B. Cui and X. Yin are with the Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: {jnzhao,bohan_cui,yinxiang}@sjtu.edu.cn.

D. V. Dimarogonas is with the School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm SE 10044, Sweden. E-mail: dimos@kth.se.

R. Majumdar is with the Max-Planck Institute for Software Systems, Kaiserslautern 67663, Germany. E-mail: rupak@mpi-sws.org.

† indicates the equal contribution.



(a) Robot Workspace (b) PK-DES

Fig. 1. A motivating example, where a robot needs to reach region 6 from region 0 with partially-known environment information.

In most of the existing works in SCT, uncertainties are modeled as uncontrollable events. These models, while suitable for describing disturbances, fall short in capturing the scenarios where the system structure is inherently unknown and the supervisor is required to actively explore the system to uncover the actual configuration. Relying solely on the worst-case analysis may lead to a highly conservative supervisor. To be more specific, let us consider a simple motivating example where a robot moves in a partially-known workspace as shown in Fig 1(a), the corresponding DFA model is as shown in Fig 1(b). We assume that the robot knows the terrain and the bridge between regions 6 and 7, but is uncertain about the bridge between regions 4 and 5 due to a cloud. The goal is to reach region 6 from region 1 with minimal energy cost. However, traversing mountainous terrain incurs significantly higher energy consumption compared to the plains. A worst-case strategy follows path $1 \to 2 \to 7 \to 6$ to prevent unnecessary backtracking. However, this may lead to regret if a bridge does exist at region 4, as the robot would have missed a shortcut. A more adaptive approach is to first check region 4. If a bridge is present, the robot takes the shortcut and saves energy; otherwise, it backtracks to 1. While this strategy may have a slightly higher worst-case cost, it takes the potential huge advantage of exploring the unknown regions.

The above example illustrates that even in a fully-controllable and deterministic setting, worst-case performance may not always be suitable for evaluating a planning strategy. A systematic approach is needed to quantitatively balance the trade-off between exploration and conservatism.

Motivated by this, in this paper, we formulate and solve a novel optimal supervisory control problem by considering scenarios where the supervisor operates in a *partially-known* DES: the supervisor knows all system states but lacks prior information about feasible transitions from certain states unless explored. Different from nondeterministic DES, where outcomes are random and may vary even under identical conditions, a partially-known DES involves initial uncertainty about the system's structure but remains deterministic. Building upon the above setting, the main results of this paper are summarized as follows:

- We first propose a formal model for the partially-known DES that encompasses all possible actual environments. Building on this model, we define the evolution of the supervisor's knowledge within the partially-known DES and formulate the regret-optimal supervisory control problem for which both the reachability specification and the regret evaluation are introduced.
- Next, we define a novel the tripartite transition system to capture all potential knowledge evolution paths under different control decisions and actual environments.
- Finally, we develop a game-theoretical approach to solve the proposed optimal control problem. The solution is divided into two parts: the first focuses on deriving the optimal micro-strategy for each unknown state, while the second addresses the design of a macro-strategy across different unknown states. By integrating these micro- and macro-strategies, we obtain an optimal solution that minimizes regret.

Our work is closely related to graph games with quantitative objectives [21], [22], particularly the regret minimization problem [23]–[25]. However, their setting assumes the environment-player's strategy is unrestricted, allowing it to change decisions freely each time it revisits the same state. This assumption does not align with the partially-known environment scenario considered in our work, where the environment is fixed and the environment-player must act consistently when revisiting the same state.

II. Supervisory Control of Fully-Known DES

A. System Model

Let Σ be a finite set of events. A *string* is a finite sequence of events and we denote by Σ^* the set of all strings over Σ including the empty string ϵ. For integer n, we denote by Σ^n the set of strings with length n. For string $s \in \Sigma^*$, the length of s is denoted by $|s|$ with $|\epsilon| = 0$. A language $L \subseteq \Sigma^*$ is a set of strings. The prefix-closure of L is denote by $\bar{L}$, i.e., $\bar{L} = \{u \in \Sigma^* : \exists v \in \Sigma^* \text{ s.t. } uv \in L\}$.

We consider a *fully-known* discrete-event system modeled by a deterministic finite-state automaton (DFA)

$$G = (X, \Sigma, \delta_G, x_0, X_m)$$

where X is the finite set of states, Σ is the finite set of events, $\delta_G : X \times \Sigma \to X$ is the partial transition function, $x_0 \in X$ is the initial state, and $X_m \subseteq X$ is the set of marked states. The transition function δ_G can also be extended to $\delta_G : X \times \Sigma^* \to X$ in the usual manner. The language generated by G from state x is defined by $\mathcal{L}(G, x) = \{s \in \Sigma^* : \delta_G(x, s)!\}$, where "!" means "is defined". We also define $\mathcal{L}(G, Q) := \bigcup_{x \in Q} \mathcal{L}(G, x)$ as the language generated from a set of states $Q \subseteq X$, define the language generated by G is $\mathcal{L}(G) := \mathcal{L}(G, x_0)$, and the language marked by G is $\mathcal{L}_m(G) = \{s \in \mathcal{L}(G) : \delta_G(s) \in X_m\}$. For any $s \in \mathcal{L}(G)$, we write $\delta_G(x_0, s)$ simply as $\delta_G(s)$. For any $x \in X$, we define $\Lambda_G(x) = \{\sigma \in \Sigma : \delta_G(x, \sigma)!\}$ as the set of active events at x. For any $s \in \mathcal{L}(G)$, we also write $\Lambda_G(\delta_G(s))$ as $\Lambda_G(s)$ for simplicity.

In the supervisory control framework, a supervisor can restrict the behavior of the system G by dynamically disabling/enabling some system events. In this setting, the event set Σ is partitioned as $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$, where Σ_c is the set of controllable events and Σ_{uc} is the set of uncontrollable events. A control decision $\gamma \in 2^\Sigma$ is said to be admissible if $\Sigma_{uc} \subseteq \gamma$, namely, uncontrollable events can never be disabled. Particularly, in this paper, we consider the control decisions in which *at most one controllable event* is included. This setting is without loss of generality since it does not influence the controllability of the system. We define

$$\Gamma = \left\{\gamma \in 2^\Sigma : \Sigma_{uc} \subseteq \gamma \wedge |\gamma \setminus \Sigma_{uc}| \leq 1\right\}$$

as the set of admissible control decisions. Then, a supervisor is a mapping

$$S : \mathcal{L}(G) \to \Gamma \tag{1}$$

We denote by $\Psi(G)$ the set of all supervisors in G. Moreover, we use the notation S/G to represent the controlled system and the language generated by S/G, denoted by $\mathcal{L}(S/G)$, is defined recursively in the following manners:

i) $\epsilon \in \mathcal{L}(S/G)$; and
ii) for any $s \in \Sigma^*$, $\sigma \in \Sigma$ we have $s\sigma \in \mathcal{L}(S/G)$ iff $s\sigma \in \mathcal{L}(G)$, $s \in \mathcal{L}(S/G)$, and $\sigma \in S(s)$.

The language marked by S/G is denoted by $\mathcal{L}_m(S/G) = \mathcal{L}(S/G) \cap \mathcal{L}_m(G)$.

In this paper, we aim to synthesis a supervisor enforcing *reachability objective*, which requires that the controlled DES will eventually visit the marked states, i.e.,

$$\begin{aligned} &\forall s \in \mathcal{L}(S/G), \exists n \in \mathbb{N}, \forall t \in \mathcal{L}(S/G)/s : \\ &|t| \geq n \Rightarrow \overline{\{st\}} \cap \mathcal{L}_m(G) \neq \emptyset \end{aligned} \tag{2}$$

For convenience, we define

$$\Psi_W(G) = \{S \in \Psi(G) : S \text{ satisfies (2)}\}$$

as the set of all winning supervisors for the reachability.

In this paper, we consider the optimal supervisory control in a quantitative manner. To this end, for each event, we define the cost function $w : \Sigma \to \mathbb{R}_{\geq 0}$ describing the cost incurred whenever the DES generates an event. Given a finite string $t = \sigma_1 \cdots \sigma_n \in \mathcal{L}(G)$, we define $\mathsf{cost}(t) = \sum_{i=1}^n w(\sigma_i)$ as the total cost along the string s. For a string $s \in \mathcal{L}_m(G)$, we define its cost as the total cost when it first reaches the set of marked states X_m, i.e.,

$$\begin{aligned} \mathsf{cost}(s) &= \mathsf{cost}(t), \\ t \in \overline{\{s\}} \cap \mathcal{L}_m(G) &\wedge t' \notin \mathcal{L}_m(G), \forall t' \in \overline{\{t\}} \backslash \{t\} \end{aligned} \tag{3}$$

Algorithm 1: Min-Max Game (`SolveMinMax`)

Input: A BTS $\mathcal{B}$ with the set of marked states X_m and a cost function $w_{\mathcal{B}}$

Output: Optimal strategy $\pi^\star : Q_X \to Q_Y$ and the optimal minmax value minmax

foreach $x \in Q_X$ **do**
  if $x \in X_m$ **then**
    $\mathsf{Val}^{(0)}(x) \leftarrow 0$ and $\pi^{(0)}(x) \leftarrow \Lambda_G(x) \cup \Sigma_{uc}$
  else
    $\mathsf{Val}^{(0)}(x) \leftarrow \infty$
repeat
  foreach $y \in Q_Y$ **do**
    $\mathsf{Val}^{(k+1)}(y) \leftarrow \max_{x' \in \mathrm{Succ}(y)} \Big(\mathsf{Val}^{(k)}(x') + w_{\mathcal{B}}(y, x') \Big)$
  foreach $x \in Q_X$ **do**
    $\mathsf{Val}^{(k+1)}(x) \leftarrow \min_{y \in \mathrm{Succ}(x)} \Big(\mathsf{Val}^{(k)}(y) + w_{\mathcal{B}}(x, y) \Big)$
    $\pi^{(k+1)}(x) \leftarrow \arg\min_{y \in \mathrm{Succ}(x)} \Big(\mathsf{Val}^{(k)}(y) + w_{\mathcal{B}}(x, y) \Big)$
  $k \leftarrow k + 1$
until $\forall z \in Q_X \cup Q_Y : \mathsf{Val}^{(k+1)}(z) = \mathsf{Val}^{(k)}(z)$;
return $\pi^{(k)}, \mathsf{Val}^{(k)}(x_0)$

Given a winning supervisor $S \in \Psi_W(G)$, we define the cost of the controlled system S/G as

$$\mathsf{cost}(S/G) = \max_{s \in \mathcal{L}_m(S/G)} \mathsf{cost}(s) \tag{4}$$

Then, we aim to synthesize an optimal supervisor $S^\star$ that minimizes the cost with the reachability objective, i.e.,

$$\mathsf{cost}(S^\star/G) \leq \mathsf{cost}(S/G),\ \forall S \in \Psi_W(G) \tag{5}$$

B. Controller Synthesis

In this subsection, we review the optimal supervisory control synthesis algorithm for reachability based on the following structure of bipartite transition system [26].

Definition 1 (Bipartite Transition System): Given system G, we construct its bipartite transition system (BTS) as

$$\mathcal{B} = (Q_X, Q_Y, x_0, h_{XY}, h_{YX}, \Sigma, \Gamma)$$

where

- $Q_X = X \cup \{d\}$ is the set of X-states, where d is the deadlocked state;
- $Q_Y = X \times \Gamma$ is the set of Y-states;
- $x_0 \in Q_Y$ is the initial X-state;
- $h_{XY} : Q_X \times \Gamma \to Q_Y$ is the transition function from X-states to Y-states define by: for any $x \in Q_X$, $\gamma \in \Gamma$ and $y = (x, \gamma) \in Q_Y$, $y = h_{XY}(x, \gamma)$;
- $h_{YX} : Q_Y \times (\Sigma \cup \{\epsilon\}) \to Q_X$ is the transition function from Y-states to X-states define by: for any $y = (x, \gamma) \in Q_Y$, we have
  - if $\Lambda_G(x) \cap \gamma \neq \emptyset$, we define
  $$h_{YX}(y, \sigma) = \delta(x, \sigma),\ \forall \sigma \in \Lambda_G(x) \cap \gamma \tag{6}$$
  - if $\Lambda_G(x) \cap \gamma = \emptyset$, we define
  $$h_{YX}(y, \epsilon) = d \tag{7}$$
- Σ is the set of events in G;
- Γ is the set of admissible control decisions in G.

Here we note that the marked states in G are implicitly included in X-states of $\mathcal{B}$, i.e., $X_m \subset Q_X$. For the sake of writing convenience, in this paper, we use $\mathrm{Succ}(\cdot)$ to denote the successor-states of a given state in the graph, i.e.,

- for each $x \in Q_X$, we have
$$\mathrm{Succ}(x) = \{h_{XY}(x, \gamma) : \gamma \in \Gamma\}$$
- for each $x \in Q_Y$, we have
$$\mathrm{Succ}(x) = \{h_{YX}(y, \sigma) : \sigma \in \Sigma \cup \{\epsilon\} \text{ s.t. } h_{YX}(y, \sigma)!\}$$

Hereafter, the detailed definition of $\mathrm{Succ}(\cdot)$ for other kinds of graphs will be omitted.

To capture the optimality, we define a cost function for $\mathcal{B}$ as follows:

- for each $x \in Q_X \backslash \{d\}$ and each $y \in h_{XY}(x)$, we define
$$w_{\mathcal{B}}(x, y) = 0 \tag{8}$$
- for each $y \in Q_Y$ and each $x' = h_{YX}(y, \sigma) \in h_{YX}(y)$, we define
$$w_{\mathcal{B}}(y, x') = w(\sigma) \tag{9}$$

Then the optimal supervisor can be obtained by solving a standard *min-max reachability game* in $\mathcal{B}$, i.e., Algorithm 1.

Given the optimal strategy $\pi^\star$ returned by Algorithm 1, in execution, the optimal supervisor $S^\star$ can work as follows: at each instant, $S^\star$ remembers the current X-state x and pick the decision γ such that $h_{XY}(x, \gamma) = \pi^\star(x)$. Then we update the current to Y-state according to γ and wait for the next event to occur. After that, we update the current state again to X-state and so forth.

III. Partially-Known DES

Although uncertainties in environments can be captured by the uncontrollable events in the full-known DES, there are also scenarios where there are initially unknown states in the system that could become known after being visited by system trajectories. We capture such kind of *information uncertainty* by the following partially-known DES.

Definition 2 (Partially-Known DES): A partially-known DES (PK-DES) is a 6-tuple

$$\mathbb{G} = (X, \Sigma, \delta, \Delta, x_0, X_m)$$

where X is the set of states; Σ is the set of events; $\delta : X \times \Sigma \to X$ is the transition function; x_0 is the initial state; X_m is the set of marked states; different from the DFA,

$$\Delta : X \to 2^{2^\Sigma}$$

is the *event-pattern function* that assigns each state a family of activated events.

The intuition of the PK-DES $\mathbb{G}$ is explained as follows. Essentially, PK-DES is used to describe the *possible world* for the system. That is, the system has some prior information regarding the possible events of each unknown state but does not know which ones are activated before the system trajectory actually visits it. Therefore, in the PK-DES $\mathbb{G}$, for each state $x \in X$, we have

$$\Delta(x) = \{o_1, o_2, \ldots, o_{|\Delta(s)|}\}$$

where each $o_i \in 2^\Sigma$ is called an *event-pattern* representing a possible set of activated events at state x. For convenience, we refer each $o_i \in \Delta(x)$ to as an *observation* at state x, since the system "observes" the activated events when visiting state x. Therefore, for each $x \in X$, we say x is a

- *known state* if $|\Delta(x)| = 1$; and
- *unknown state* if $|\Delta(x)| > 1$.

Accordingly, we partition the state space as $X = X_k \dot\cup X_{uk}$ where X_k is the set of known states and X_{uk} is the set of unknown states. We refer the visit to an unknown state to as an *exploration*.

Definition 3 (Compatible DES): Given a PK-DES $\mathbb{G}$, a DFA $G = (X, \Sigma, \delta_G, x_0, X_m)$ is a compatible DES with $\mathbb{G}$, denoted by $G \in \mathbb{G}$, if for any $x \in X$, we have

i) $\Lambda_G(x) \in \Delta(x)$; and
ii) for each $\sigma \in \Lambda_G(x)$, we have $\delta_G(x, \sigma) = \delta(x, \sigma)$.

In the partially-known setting, a supervisor cannot be synthesized only based on the finite string generated by the system. In addition, the observed successor-pattern at each state should also be taken into consideration. To be specific, if the system trajectory visits a known state, then there would not be any useful information about the partially-known DES, since $\Delta(x)$ is already a singleton. Only when the system trajectory visits unknown state, it will gain new information and successor-pattern at this state will become known from then on.

To capture the result of an exploration, we call a tuple $\kappa = (x(\kappa), o(\kappa)) \in X \times 2^\Sigma$ as a *knowledge state*, where $o \in \Delta(x)$ is the set of activated events of x that becomes known to the agent after exploring x. We denote by

$$\mathsf{Kw} = \{\kappa \in X \times 2^\Sigma : o(\kappa) \in \Delta(x(\kappa))\} \tag{10}$$

the set of all possible knowledge states. A *history* in $\mathbb{G}$ is a finite sequence of knowledge states and events

$$\hbar = (x_0, o_0)\sigma_0(x_1, o_1)\sigma_1 \cdots (x_n, o_n) \in (\mathsf{Kw} \cdot \Sigma)^* \mathsf{Kw}$$

such that

i) for any $i = 0, 1, \ldots, n-1$, we have $\sigma_i \in o_i$ and $x_{i+1} = \delta(x_i, \sigma_i)$;
ii) for any $i, j = 1, \ldots, n$, we have $x_i = x_j \Rightarrow o_i = o_j$.

For such history $\hbar$, we call $\sigma_1\sigma_2 \cdots \sigma_{n-1} \in X^*$ its string. We denote by $\mathcal{H}(\mathbb{G})$ and $\mathcal{L}(\mathbb{G})$ the set of all finite histories and paths in PK-DES $\mathbb{G}$, respectively.

Given knowledge on the PK-DES will be accumulated along a history, which is captured by the following concept of *knowledge set*. A knowledge set $\mathcal{K} = \langle \kappa_1, \ldots, \kappa_{|\mathcal{K}|} \rangle$ where $\kappa_i \in \mathcal{K}$ is an *ordered set* of knowledge states such that

$$\forall \kappa, \kappa' \in \mathcal{K} : x(\kappa) = x(\kappa') \Rightarrow o(\kappa) = o(\kappa').$$

We denote by $\mathbb{KW}$ the set of all knowledge sets. With a slight abuse of notation, for each $x \in X$, we write $x \in \mathcal{K}$ if $(x, o) \in \mathcal{K}$ for some observation $o \in \Delta(x)$. We denote by $o_\mathcal{K}(x) \in \Delta(x)$ the unique observation such that $(x, o_\mathcal{K}(x)) \in \mathcal{K}$.

During the system execution, the system maintains a knowledge set to record its exploration history. Once a new unknown state is explored, the knowledge set is updated according to the following function. Given a knowledge set $\mathcal{K} \in \mathbb{KW}$ and a knowledge state $\kappa \in \mathsf{Kw}$, we have

$$\texttt{update}(\mathcal{K}, \kappa) = \begin{cases} \mathcal{K}, & \text{if } x(\kappa) \in \mathcal{K} \\ \langle \kappa_1, \ldots, \kappa_{|\mathcal{K}|}, \kappa \rangle, & \text{otherwise} \end{cases} \tag{11}$$

Finally, given a knowledge set, the system could refine the PK-DES by eliminating uncertainties that have been explored. Given a PK-DES $\mathbb{G}$ and a knowledge set $\mathcal{K} \in \mathbb{KW}$, the refined PK-DES is a new PK-DES

$$\mathbb{G}_\mathcal{K} = (X, \Sigma, \delta, \Delta', x_0, X_m) \tag{12}$$

such that for any $x \in X$, we have

$$\Delta'(x) = \begin{cases} \{o_\mathcal{K}(x)\}, & \text{if } x \in \mathcal{K} \\ \Delta(x), & \text{if } x \notin \mathcal{K} \end{cases}$$

Therefore, under the setting of partially-known DES, it is no longer sufficient to synthesize a supervisor in (1), since the supervisor should decide the control decision based on both of what the system has visited and what it has known. Therefore, we define the notion of *strategic supervisor*.

Definition 4 (Strategic Supervisor): A strategic supervisor is a function

$$\mathbb{S} : \mathcal{H}(\mathbb{G}) \to \Gamma$$

such that for any $\hbar = \kappa_0\sigma_1\kappa_1\sigma_2 \cdots \kappa_n$ where $\kappa_i = (x_i, o_i)$, we have

$$\mathbb{S}(\hbar) = \{\sigma_c\} \cup \Sigma_{uc},\ \sigma_c \in o_n \cap \Sigma_c$$

i.e., the supervisor makes a control decision based on the observed set of activated events at x_n.

We denote by $\Phi(\mathbb{G})$ the set of all strategic supervisors in $\mathbb{G}$.

Given a strategic supervisor $\mathbb{S} \in \Phi(\mathbb{G})$ and an actual DES $G \in \mathbb{G}$, the controlled system, denoted by $\mathbb{S}/G$, whose language $\mathcal{L}(\mathbb{S}/G)$ is recursively defined as:

i) $\epsilon \in \mathcal{L}(\mathbb{S}/G)$;
ii) for any $\hbar = \kappa_0\sigma_1\kappa_1\sigma_2 \cdots \kappa_n$, we have $\sigma_1\sigma_2 \cdots \sigma_n \in \mathcal{L}(\mathbb{S}/G)$ iff $\sigma_1\sigma_2 \cdots \sigma_{n-1} \in \mathcal{L}(\mathbb{S}/G)$ and $\sigma_n \in \mathbb{S}(\hbar)$.

In the partially-known setting, since the actual DES $G \in \mathbb{G}$ is unknown *a priori*, we aim to synthesize a strategic supervisor $\mathbb{S} \in \Phi(\mathbb{G})$ such that

$$\forall G \in \mathbb{G} : \mathcal{L}(\mathbb{S}/G) \text{ satisfies (2)} \tag{13}$$

We denote by $\Phi_W(\mathbb{G}) \subseteq \Phi(\mathbb{G})$ the set of all winning strategic supervisors for the reachability objective.

To evaluate the performance of a strategic supervisor $\mathbb{S}$, a natural approach is to still consider the *worst-case cost* similar to the definition of (4) among all possible actual DES, i.e.,

$$\mathsf{Cost}_{\text{worst}}(\mathbb{S}) := \max_{G \in \mathbb{G}} \mathsf{cost}(\mathbb{S}/G) \tag{14}$$

However, this metric cannot capture the potential benefit obtained from exploring unknown states. To address this issue, we propose to use *regret* to evaluate the performance of a strategic supervisor, which is defined as follows.

Definition 5 (Regret): Given a partially-known DES $\mathbb{G}$, the regret of a supervisor $\mathbb{S} \in \Phi(\mathbb{G})$ is

$$\mathsf{Reg}(\mathbb{S}) = \max_{G\in\mathbb{G}} \left(\mathsf{cost}(\mathbb{S}/G) - \min_{\mathbb{S}'\in\Phi_W(G)} \mathsf{cost}(\mathbb{S}'/G) \right)$$

Finally, we formally formulate the problem that we solve in this paper as follows.

Problem 1 (Regret-Optimal Supervisory Control): Given a partially-known DES $\mathbb{G}$, synthesize a strategic supervisor $\mathbb{S} \in \Phi_W(\mathbb{G})$ such that

$$\forall \mathbb{S}' \in \Phi_W(\mathbb{G}) : \mathsf{Reg}(\mathbb{S}) \leq \mathsf{Reg}(\mathbb{S}'). \tag{15}$$

IV. Main Results

In this section, we first show that the regret-optimal supervisory control problem can be reduced to a quantitative *three-player graph game* by incorporating knowledge into the state space. Then, we propose the algorithm to obtain the regret-optimal supervisor based on two algorithms for micro- and macro-strategy synthesis.

A. Tripartite Transition System

We aim to incorporate the knowledge set into the state space of the game arena and explicitly split the movements of the control decision, the system execution, and the nondeterminism of the environment, which leads to the following notion of *tripartite transition system*.

Definition 6 (Tripartite Transition System): Given a PK-DES $\mathbb{G}$, its tripartite transition system (TTS) is a tuple

$$\mathcal{G} = (V_X, V_Y, V_Z, v_0, f_{XZ}, f_{ZY}, f_{YX}, \Sigma, \Gamma)$$

where

- $V_X = X \times \mathbb{KW}$ is the set of X-states;
- $V_Y = X \times \mathbb{KW} \times \Gamma$ is the set of Y-states;
- $V_Z = X \times \mathbb{KW} \times X \cup \{d\}$ is the set of Z-states;
- $v_0 = (x_0, \mathcal{K}_0) \in Q_X$ is the initial state with $\mathcal{K}_0$ being the initial knowledge set;
- $f_{XY} : V_X \times \Gamma \to V_Y$ is the transition function from X-states to Y-states defined by: for any $v_x = (x, \mathcal{K}) \in V_X$, $\gamma \in \Gamma$, and $v_y = (x, \mathcal{K}, \gamma) \in V_F$, we have
$$v_y = f_{XY}(v_x, \gamma) \tag{16}$$
- $f_{YZ} : V_Y \times (\Sigma \cup \{\epsilon\}) \to V_Z$ is the transition function from Y-states to Z-states defined by: for any $v_y = (x, \mathcal{K}, \gamma) \in V_Y$, we have
  - if $\gamma \cap o_{\mathcal{K}}(x) \neq \emptyset$, we define
$$f_{YZ}(v_y, \sigma) = (x, \mathcal{K}, \delta(x, \sigma)),\ \forall \sigma \in \gamma \cap o_{\mathcal{K}}(x) \tag{17}$$
  - if $\gamma \cap o_{\mathcal{K}}(x) = \emptyset$, we define
$$f_{YZ}(v_x, \epsilon) = d \tag{18}$$
- $f_{ZX} : V_Z \times 2^{\Sigma} \to V_X$ is the transition function from Z-states to X-states defined by: for any $v_z = (x, \mathcal{K}, x') \in V_Z \backslash \{d\}$ and any $o \in \Delta(x')$, we define
$$v_x = f_{ZX}(v_z, o) = (x', \mathcal{K}') \tag{19}$$
with $\mathcal{K}' = \mathtt{update}(\mathcal{K}, (x', o))$;
- Σ is the set of events in G;
- Γ is the set of admissible control decisions in G.

The intuition of the TTS is explained as follows. The graph is tripartite with three types of states: the X-states from which the system chooses a feasible control decision; the Y-states from which the system is executed according the control decision chosen before; and the Z-states from which the actual event-pattern is decided in the possible world.

Consider the Z-states with at least two successors, i.e., $v_z \in V_Z$ satisfying $\mathrm{Succ}(v_z) \geq 2$, which we call as "$Z$-states with decisions". Then we have the following property.

Proposition 1: In the TTS $\mathcal{G}$, there are no cycles encompassing two different Z-states with decisions.

Proof: Due the page limit, all the proofs are omitted in this paper and can be found at `https://jnzhaooo.github.io/files/regret.pdf`. ■

B. Plays and Strategies

We denote by $V = V_X \cup V_Y \cup V_Z$ the state space of $\mathcal{G}$. Furthermore, we define the set of marked states in $\mathcal{G}$ as

$$V_m = \{(x, \mathcal{K}) \in V_X : x \in X_m\} \tag{20}$$

Given a TTS $\mathcal{G}$, we call a finite sequence of states

$$\rho = v_x^0 v_y^0 v_z^0 v_x^1 v_y^1 v_z^1 \cdots v_x^n \in (V_X \cdot V_Y \cdot V_Z)^* V_X$$

a *play* if $v_x^0 = v_0$ and there are $\gamma_i \in \Gamma$, $\sigma_i \in \Sigma \cup \{\epsilon\}$ and $o_i \in 2^{\Sigma}$ such that $v_y^i = f_{XY}(v_x^i, \gamma_i)$, $v_z^i = f_{YZ}(v_y^i, \sigma_i)$, and $v_x^{i+1} = f_{ZX}(v_z^i, o_i)$ hold for each $i = 0, 1, \ldots, n-1$. It is obvious that each play in $\mathcal{G}$ induces a history in $\mathbb{G}$, i.e.,

$$\hbar_\rho = (x_0, o_0)\sigma_0(x_1, o_1)\sigma_1 \cdots (x_n, o_n) \in \mathcal{H}(\mathbb{G}).$$

For convenience, we still use ρ to denote the partial play that does not necessarily end with a X-state.

Since only edges from V_z to V_x represent actual movements, we define a weight function for $\mathcal{G}$ as

$$w_{\mathcal{G}} : V \times V \to \mathbb{R}_{\geq 0}$$

where for any $v_z = (x, \mathcal{K}, x')$ and $v_x = (x', \mathcal{K}')$, we have $w_{\mathcal{G}}(v_z, v_x) = w(x, x')$, $w_{\mathcal{G}}(v_x, v_y) = w_{\mathcal{G}}(v_y, v_z) = 0$. The cost of a play $\rho = v_0 v_1 \cdots v_n \in V^*$ is defined as

$$\mathsf{cost}_{\mathcal{G}}(\rho) = \sum_{i=0}^{n-1} w_{\mathcal{G}}(v_i, v_{i+1}). \tag{21}$$

Given the above TTS $\mathcal{G}$, the strategies are functions

$$\pi_x : V^* V_X \to V_Y \cup \{\text{Null}\},\ \pi_y : V^* V_Y \to V_Z,\ \pi_z : V^* V_Z \to V_X$$

for X-player, Y-player and Z-player, respectively. We denote by $\mathfrak{S}_X(\mathcal{G})$, $\mathfrak{S}_Y(\mathcal{G})$, $\mathfrak{S}_Z(\mathcal{G})$ the sets of all X-strategies, Y-strategies and Z-strategies, respectively. In particular, we say a strategy π is *positional* if $\forall \rho, \rho' : \mathsf{last}(\rho) = \mathsf{last}(\rho') \Rightarrow \pi(\rho) = \pi(\rho')$, where $\mathsf{last}(\cdot)$ denotes the last state of a sequence. We denote by $\mathfrak{S}_j^1(\mathcal{G})$ the set of all positional strategies for j-player, where $j = X, Y, Z$. Given strategies $\pi_x \in \mathfrak{S}_X, \pi_y \in \mathfrak{S}_Y, \pi_z \in \mathfrak{S}_Z$, the outcome play $\rho_{\pi_x, \pi_y, \pi_z}$ is the unique sequence $v_0 v_1 \cdots v_n \in V^* V_X$ such that

- $\forall i < n : v_i \in V_X \Rightarrow \pi_x(v_0 v_1 \cdots v_i) = v_{i+1}$;
- $\forall i < n : v_i \in V_Y \Rightarrow \pi_y(v_0 v_1 \cdots v_i) = v_{i+1}$;
- $\forall i < n : v_i \in V_Z \Rightarrow \pi_z(v_0 v_1 \cdots v_i) = v_{i+1}$;
- $\pi_x(v_0 v_1 \cdots v_n) = \text{Null}$.

$$\mathsf{Reg}_{\mathcal{G}}(\pi_x) = \max_{\pi_z \in \Pi_Z} \left(\max_{\pi_y \in \Pi_Y} \mathsf{cost}_{\mathcal{G}}(\rho_{\pi_x,\pi_y,\pi_z}) - \min_{\pi'_x \in \Pi_X} \max_{\pi_y \in \Pi_Y} \mathsf{cost}_{\mathcal{G}}(\rho_{\pi'_x,\pi_y,\pi_z}) \right) \tag{23}$$

Here we remark that, since the objective of the supervisor is to satisfy the reachability w.r.t. the set V_m, once the system trajectory visits V_m, it is unncessary to restrict the behaviors of the system. To this end, in the TTS $\mathcal{G}$, for a play ρ satisfying $\mathsf{last}(\rho) \in \text{Null}$, we set $\pi_x(\rho) = \text{Null}$.

Next, we aim to characterize the sets of strategies for the X, Y, Z-players that are sufficient to solve the regret-optimal supervisory control problem.

For the Y-player, since there is no restriction for the uncontrollable events and no causal relation between two different events, it is sufficient to consider $\Pi_Y = \mathfrak{S}^1_Y(\mathcal{G})$ for all the possible behaviors generated by Y-player.

For the Z-player, it cannot play arbitrarily since the actual DES is fixed. Therefore, the Z-player must commit to a specific event-pattern it chooses at each unknown state when the game begins. To this end, we define the following notion of *strongly positional strategy*. We say a Z-strategy $\pi_z \in \mathfrak{S}_Z(\mathcal{G})$ is strongly positional if for any two plays $\rho, \rho' \in V^* V_Z$ where $\pi_z(\rho) = (x, \mathcal{K})$ and $\pi_z(\rho') = (x', \mathcal{K}')$, we have

i) π_z is positional; and
ii) $x = x' \Rightarrow o_{\mathcal{K}}(x) = o_{\mathcal{K}'}(x')$.

We denote by $\Pi_Z \subseteq \mathfrak{S}_Z(\mathcal{G})$ the set of all stronly positional strategies for the Z-player.

Finally, we aim to find a winning strategy for X-player that ensures the outcome play visits the state in V_m. Therefore, we define

$$\Pi_X = \left\{ \pi_x \in \mathfrak{S}_X(\mathcal{G}) : \begin{matrix} \mathsf{last}(\rho_{\pi_x,\pi_y,\pi_z}) \in V_m, \\ \forall \pi_y \in \Pi_Y, \forall \pi_z \in \Pi_Z \end{matrix} \right\} \tag{22}$$

Now, similarly to Definition 5, we define the regret of an X-strategy $\pi_x \in \Pi_X$ in $\mathcal{G}$ as (23). It suffices to synthesize an X-strategy $\pi_X \in \Pi_X$ that minimizes $\mathsf{Reg}_{\mathcal{G}}(\pi_x)$. In what follows, we present the solution by introducing two algorithms for micro and macro-strategy synthesis.

C. Strategy Synthesis

First of all, we claim that a positional X-strategy is sufficient to solve the regret-minimizing problem, based on which a value iteration can be applied. This is due to the fact that the knowledge update mechanism has been captured in the state space of TTS $\mathcal{G}$ and it is unnecessary for the X-player to keep additional memory for the knowledge.

Given a knowledge set $\mathcal{K} \in \mathbb{KW}$, we define the optimistic response of $\mathcal{K}$ as

$$\mathsf{opr}(\mathcal{K}) = \min\{\mathsf{minmax}(G) : G \in \mathbb{G}_{\mathcal{K}}\} \tag{24}$$

which can be computed by $\texttt{SolveMinMax}(\mathcal{B}_G, X_m, w_{\mathcal{B}_G})$, where $\mathcal{B}_G$ is the BTS corresponding to each $G \in \mathbb{G}$.

With a slight abuse of notation, for each X-state $v_x = (x, \mathcal{K})$, we define its optimistic response as the optimistic response of its knowledge set $\mathcal{K}$, i.e., $\mathsf{opr}(v_x) = \mathsf{opr}(\mathcal{K})$.

The optimistic response can be utilized to compute the regret of an outcome play as follows. Suppose that the system trajectory ρ reaches a marked state $v_x = (x, \mathcal{K})$. Along this trajectory, the accumulated knowledge is $\mathcal{K}$. Therefore, $\mathsf{opr}(v_x)$ serves as a lower-bound estimate of the cost required to each some marked state in V_m (not necessarily v_x) in the DESs that are compatible with the current knowledge $\mathcal{K}$. Then, the difference $\mathsf{cost}_{\mathcal{G}}(\rho) - \mathsf{opr}(v_x)$ essentially quantifies the regret incurred along the trajectory ρ.

However, due to the possible existence of cycles in $\mathcal{G}$, there is generally an infinite number of winning strategies for X-player, which makes enumerating all of them infeasible. To tackle this issue, we propose to compute *micro-strategies* to capture the strategies that are sufficient to obtain the regret-optimal X-strategy. Based on Proposition 1, we know that, the evolution of Z-states is monotone and for any plays in $\mathcal{G}$ that end with the same state, they must visit the same Z-states with decisions in the same order. Therefore, it suffices to compute the X-strategies between different *layers of Z-states*. To formally capture this, we define the set of initial states for all knowledge sets as

$$V_I = \{v_0\} \cup \{\text{Succ}(v_z) \in V_X : \text{Succ}(v_z) \geq 2, \forall v_z \in V_Z\} \tag{25}$$

For each $v_I \in V_I$, we define the set of all end states as

$$\begin{aligned} V_F(v_I) = &\{v_x \in V_m : \mathcal{K}(v_x) = \mathcal{K}(v_I)\} \\ &\cap \left\{ v_x \in V_X : \begin{matrix} \exists v_z \in V_Z \text{ s.t. } \mathcal{K}(v_z) = \mathcal{K}(v_I) \wedge \\ v_x \in \text{Succ}(v_z) \wedge \mathcal{K}(v_x) \neq \mathcal{K}(v_z) \end{matrix} \right\} \end{aligned} \tag{26}$$

For convenience, in what follows, we use a *tree* in the TTS $\mathcal{G}$ to represent a strategy and we denote by $\mathcal{T}$ the set of all trees in $\mathcal{G}$. For each $\mathcal{T} \in \mathbb{T}$, we denote by $V_{\mathcal{T}}$ and $E_{\mathcal{T}}$ its sets of states and edges. We define the cost of $\mathcal{T}$, denoted by $\mathsf{cost}_{\mathbb{T}}(\mathcal{T})$, as the maximum cost of its plays from the root to the leaves. We denote by $\rho_{\mathcal{T}}$ the play with the maximum cost, i.e., $\mathsf{cost}_{\mathcal{G}}(\rho_{\mathcal{T}}) = \mathsf{cost}_{\mathbb{T}}(\mathcal{T})$. Furthermore, for each $\mathcal{T} \in \mathbb{T}$, it is a map $\mathcal{T} : V_{\mathcal{T}} \to \mathbb{T}$ that assigns each state in $\mathcal{T}$ a subtree. Specifically, we have $\mathcal{T}(v_I) = \mathcal{T}$, where v_I is the root of $\mathcal{T}$.

Definition 7 (Micro-Strategy): Given the TTS $\mathcal{G}$ and for each $v_I \in V_I$, a micro-strategy for X-player is a winning strategy (strategy-tree) $\mathcal{T}_{v_I} \in \mathbb{T}$ such that

i) the root of $\mathcal{T}_{v_I}$ is v_I;
ii) for each $v_x \in V_{\mathcal{T}_{v_I}} \cap V_X$, there is only one successor $v_y \in \text{Succ}(v_x)$ in $\mathcal{G}$ such that $(v_x, v_y) \in E_{\mathcal{T}_{v_I}}$;
iii) for each $v_y \in V_{\mathcal{T}_{v_I}} \cap V_Y$, for any successor $v_z \in \text{Succ}(v_y)$ in $\mathcal{G}$, we have $(v_y, v_z) \in E_{\mathcal{T}_{v_I}}$;
iv) for each $v_z \in V_{\mathcal{T}_{v_I}} \cap V_Z$, all of the successors $v_x \in \text{Succ}(v_z)$ in $\mathcal{G}$ are the leaves;
v) All $v_x \in V_{\mathcal{T}_{v_I}} \cap V_m$ are leaves.

Intuitively, each micro-strategy captures the partial strategy for X-player in $\mathcal{G}$ that ensures for each $v_I \in V_I$ either the

Algorithm 2: Find All Critical Micro-Strategies (FACMiS)

Input: The TTS $\mathcal{G}$ and an initial state v_I
Output: All micro-strategies from v_I to $V_F(v_I)$

1 Compute $V_F(v_I)$ as in (26);
2 Define $\Pi_{v_I} : 2^{V_F(v_I)} \to \mathbb{T} \times \mathbb{R}_{\geq 0}$;
3 Call DFS$(v_I, \{v_I\}, 0)$;
4 **return** Π_{v_I}
5 **procedure** DFS(v, V_{vis}, C)
6 Define subtree $\mathcal{T}_v : Succ(v) \to \mathbb{T}$;
7 $\mathcal{T}_v \leftarrow$ Null;
8 **if** $v \in V_F(v_I)$ **then**
9 $\quad Acc \leftarrow V_{vis} \cap V_F(v_I)$;
10 $\quad$ **if** $Acc \notin \Pi_{v_I}$ **then**
11 $\quad\quad \Pi_{v_I}(Acc) = (\mathcal{T}_v, \mathsf{C})$;
12 $\quad$ **else**
13 $\quad\quad (\mathcal{T}, c) \leftarrow \Pi_{v_I}(Acc)$;
14 $\quad\quad$ **if** $\mathsf{C} < c$ **then**
15 $\quad\quad\quad \Pi_{v_I}(Acc) = (\mathcal{T}_v, \mathsf{C})$;
16 $\quad$ **return** Null
17 **if** $v \in V_X$ **then**
18 $\quad$ Define $\tilde{\mathsf{C}} \leftarrow \infty$, $\tilde{v}_y \leftarrow$ Null, $\tilde{T} \leftarrow$ Null;
19 $\quad$ **foreach** $v_y \in Succ(v)$ **do**
20 $\quad\quad$ **if** $v_y \notin V_{vis}$ **then**
21 $\quad\quad\quad V_{vis} \leftarrow V_{vis} \cup \{v_y\}$;
22 $\quad\quad\quad T \leftarrow$ DFS$(v_y, V_{vis}, \mathsf{C} + w_{\mathcal{G}}(v, v_y))$;
23 $\quad\quad\quad V_{vis} \leftarrow V_{vis} \backslash \{v_y\}$;
24 $\quad\quad\quad$ **if** $\mathsf{C} + w_{\mathcal{G}}(v, v_y) + \mathsf{cost}_{\mathbb{T}}(T) < \tilde{\mathsf{C}}$ **then**
25 $\quad\quad\quad\quad \tilde{\mathsf{C}} \leftarrow \mathsf{C} + w_{\mathcal{G}}(v, v_y) + \mathsf{cost}_{\mathbb{T}}(T)$;
26 $\quad\quad\quad\quad \tilde{v}_y \leftarrow v_y$;
27 $\quad\quad\quad\quad \tilde{T} \leftarrow T$;
28 $\quad$ **if** $\tilde{v}_y \neq Null$ **then**
29 $\quad\quad \mathcal{T}_v(\tilde{v}_y) \leftarrow \tilde{T}$;
30 **if** $v \in V_Y \cup V_Z$ **then**
31 $\quad$ **foreach** $v' \in Succ(v)$ **do**
32 $\quad\quad$ **if** $v' \notin V_{vis}$ **then**
33 $\quad\quad\quad V_{vis} \leftarrow V_{vis} \cup \{v'\}$;
34 $\quad\quad\quad T \leftarrow$ DFS$(v', V_{vis}, \mathsf{C} + w_{\mathcal{G}}(v, v'))$;
35 $\quad\quad\quad V_{vis} \leftarrow V_{vis} \backslash \{v'\}$;
36 $\quad\quad\quad \mathcal{T}_v(v') = T$;
37 **return** $\mathcal{T}_v$

reachability of V_m or a new exploration. For any two micro-strategies $\mathcal{T}_{v_I}$ and $\mathcal{T}'_{v_I}$, we say they are similar if they have the same leaves. For each $v_I \in V_I$, we cal the micro-strategy with the minimum cost as the *critical micro-strategy*.

Now, we present the algorithm for finding all critical micro-strategies as Algorithm 2, which is explained in detail as follows. In Line 2, we define the set of all micro-strategies as Π_{v_I} which assigns each feasible combination Acc of end nodes with a micro-strategy and the cost of the strategy. With a slight abuse of notation, we denote $Acc \in \Pi_{v_I}$ if there is a micro-strategy for Acc in Π_{v_I}. We aim to use a depth-first search procedure to find all the strategies. For DFS, we use v, V_{visit}, and C to denote the current state, the states that have been searched and the minimum cost of the path from v_I to v. For each v, we aim to build a subtree $\mathcal{T}_v$ recursively (Line 7). If the current state v is an end state, then we compute its visited end nodes as Acc (Line 9) and update the optimal micro-strategy for Acc in Lines 10–15. If the current state v is an X-state, then we use $\tilde{\mathsf{C}}$, $\tilde{v}_y$ and $\tilde{T}$ to record the total cost for its optimal subtree, the root of its optimal subtree, and the optimal subtree, respectively (Lines 18–20). We find the unique optimal subtree for v by Lines 21–31. Specifically, we use Lines 23–25 to ensure that there is no cycles in the searched subtree. If the current state v is a Y-state or Z-state, we keep the subtrees starting from all the successors of v. The correctness of the above algorithm is summarized as follows.

Proposition 2: For each $v_I \in V_I$, all its the critical micro-strategies are returned by Algorithm 2.

With the critical micro-strategies for *one layer of Z-states with decisions*, we next introduce the notion of *macro-strategy* that connects the *different layers of Z-states with decisions* by the micro-strategies.

Definition 8 (Macro-Strategy): Given $\mathcal{G}$, a macro-strategy is a map $\xi : V_I \to \{\Pi_{v_I} : v_I \in V_I\}$ such that for each $v_I \in V_I$, we have $\xi(v_I) = \mathcal{T}_{v_I}$ for some micro-strategy $\mathcal{T}_{v_I} \in \Pi_{v_I}$.

Due to the fact that $V_F(v_I) \subseteq V_I$ for each $v_I \in V_I$, it is sufficient to form a *complete X-strategy* in the TTS $\mathcal{G}$ with one macro-strategy with the corresponding micro-strategies, each of which starts from either the initial state v_0 or one leave of the last micro-strategy.

Now, we present the algorithm for synthesizing the regret-optimal macrio-strategy as Algorithm 3. The intuition is explained as follows. For each micro-strategy $\mathcal{T}$ in each Π_{v_I}, we only keep the maximum cost play $\rho_{\mathcal{T}}$ in the new constructed $\tilde{\mathcal{G}}$ (Lines 7 and 9). In particular, if $\rho_{\mathcal{T}}$ does not end with a marked state, it means that its second last state is a Z-state. Accordingly, we keep all the successors of its second last state in $\tilde{\mathcal{G}}$ (Line 15). With the above construction, we reduce the TTS $\mathcal{G}$ to a BTS $\tilde{G}$ in which all the Y-state has only one successor. Furthermore, for the constructed BTS $\tilde{\mathcal{G}}$, if the edge ends with a marked state in V_m, then we set its new weight as the difference between the actual cost c_2 and its optimistic response (Line 12); otherwise, we set the new weight of the final edge as the actual cost c_2 and set all the other edges as zero (Line 18). Then, the final regret-optimal macro-strategy can be obtained by solving a min-max game w.r.t. the BTS $\tilde{G}$, the set of marked states V_m and the new weight function μ. The correctness of such algorithm is summarized as following theorem.

Theorem 1: Given the TTS $\mathcal{G}$, the X-strategy formed by the macro-strategy returned by Algorithm 3 and the micro-strategies returned by Algorithm 2 minimizes the regret for X-player, i.e., Problem 1 is solved with minimized $\mathsf{Reg}^\star$.

Remark 1: The complexity of Algorithm 2 is $O(|\delta| \cdot |X|^{|\delta|})$ where $|\delta|$ is the number of all possible transitions

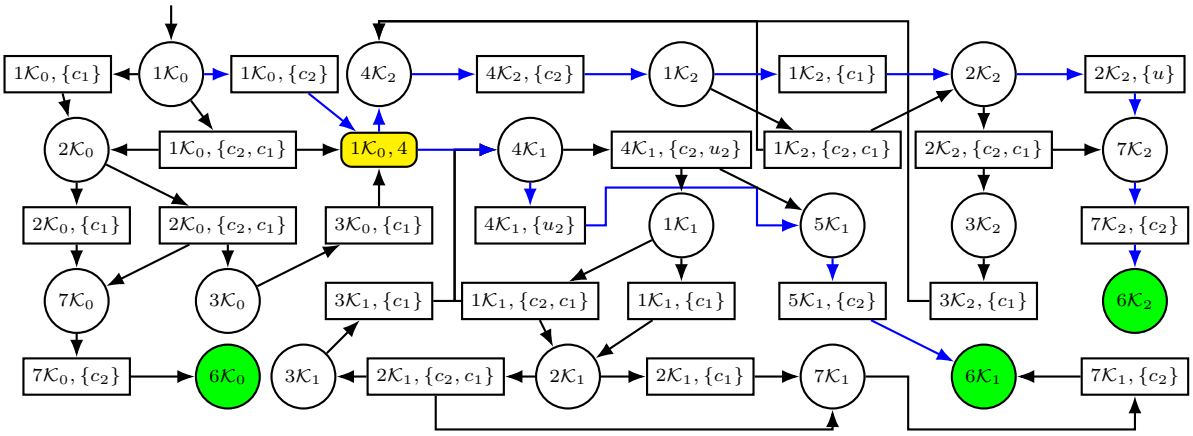


Fig. 2. Tripartite transition system of PK-DES in Fig 1(b).

Algorithm 3: Find Regret-Optimal Macro-Strategy

Input: The TTS $\mathcal{G}$, an initial state v_0, and a set of target states V_m

Output: The regret-optimal macro-strategy $\xi^\star$ and the minimized regret $\mathsf{Reg}^\star$

Construct a new graph $\tilde{\mathcal{G}} = (V, E, \mu)$ with $E \leftarrow \emptyset$;
foreach $v_I \in V_I$ **do**
  Compute $V_F(v_I)$ as in (26);
  Obtain $\Pi_{v_I} \leftarrow \texttt{FACMiS}(v_I, V_F(v_I))$;
  foreach $Acc \in \Pi_{v_I}$ **do**
    $(\mathcal{T}, c) \leftarrow \Pi_{v_I}(Acc)$;
    Find the play with maximum cost $\rho_{\mathcal{T}}$;
    foreach $(v, v') \in \rho_{\mathcal{T}}$ **do**
      $E \leftarrow E \cup \{(v, v')\}$;
      if $v' = \mathit{last}(\rho_{\mathcal{T}})$ **then**
        if $v' \in V_m$ **then**
          $\mu(v, v') \leftarrow c - \mathsf{opr}(v')$;
        else
          foreach $v'' \in \mathit{Succ}(v)$ **do**
            $E \leftarrow E \cup \{(v, v'')\}$;
            $\mu(v, v'') \leftarrow c$;
      else
        $\mu(v, v') \leftarrow 0$;
$\xi^\star, \mathsf{Reg}^\star \leftarrow \texttt{SolveMinMax}(\tilde{\mathcal{G}}, V_m, \mu)$;
return $\xi^\star, \mathsf{Reg}^\star$

in the PK-DES. The complexity of Algorithm 3 is $O(|V|^2 \cdot |\delta| \cdot |X|^{|\delta|})$ where $|V|$ is the size of state space of $\mathcal{G}$.

V. Case Study

Let us still consider the PK-DES as shown in Fig 1(b) where $\Sigma_c = \{c_1, c_2\}$, $w(c_2) = 2$, $w(c_1) = w(u_1) = 10$ and $w(u_2) = 1$. The corresponding TTS $\mathcal{G}$ is as shown in Fig 2. For simplicity, we omit Z-states when the corresponding state is already known as well as the controllable transitions that can not satisfy the reachability specification. Since there is only one unknown state in Fig 1(b), the corresponding Z-state in $\mathcal{G}$ is $(1, \mathcal{K}_0, 4)$ from which there are two different X-states with the different updated knowledge set, i.e., $(4, \mathcal{K}_1)$ and $(4, \mathcal{K}_2)$. We apply the solution synthesis approach provided by Algorithm 2 and Algorithm 3 to obtain the regret-optimal strategic supervisor as the blue transitions in Fig 2, whose regret is returned as $\mathsf{Reg}^\star = 4$.

VI. Conclusion

In this paper, we formulate the regret-optimal supervisory control problem for partially-known DES. We introduce a tripartite transition system to capture all possible actual environments and develop an effective algorithm by decomposing the problem into a two-stage reachability game. For the future direction, we aim to design a more efficient algorithm and extend our approach to broader specifications, including cyclic tasks and the mean-payoff cost metric.

References

[1] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Springer, 2008.

[2] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[3] W. Wonham and K. Cai, "Supervisory control of discrete-event systems," 2019.

[4] N. Ran, T. Li, S. Wang, and Z. He, "Supervisor synthesis for petri nets with uncontrollable and unobservable transitions," *IEEE Transactions on Automation Science and Engineering*, vol. 21, no. 2, pp. 1517–1525, 2023.

[5] Z. Xiang, Y. Chen, N. Wu, and Z. Li, "On the existence of non-blocking bounded supervisors for discrete event systems," *IEEE Transactions on Automatic Control*, 2024.

[6] J. Li and S. Takai, "Synthesis of maximally permissive supervisors for similarity control of partially observed nondeterministic discrete event systems," *Automatica*, vol. 135, p. 109978, 2022.

[7] R. Meira-Góes, J. Weitze, and S. Lafortune, "A compact and uniform approach for synthesizing state-based property-enforcing supervisors for discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 67, no. 7, pp. 3567–3573, 2021.

[8] M. Reniers and K. Cai, "Supervisory control theory with event forcing," *IEEE Transactions on Automatic Control*, 2024.

[9] A. M. Mainhardt and A.-K. Schmuck, "Assume-guarantee synthesis of decentralised supervisory control," *IFAC-PapersOnLine*, vol. 55, no. 28, pp. 165–172, 2022.

[10] X. Yin and S. Lafortune, "Synthesis of maximally-permissive supervisors for the range control problem," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3914–3929, 2016.

[11] W. Ren and D. V. Dimarogonas, "Symbolic supervisory control of periodic event-triggered control systems," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 1968–1973, 2020.
[12] R. Majumdar and A.-K. Schmuck, "Supervisory controller synthesis for nonterminating processes is an obliging game," *IEEE Transactions on Automatic Control*, vol. 68, no. 1, pp. 385–392, 2022.
[13] Y. Ji, X. Yin, and S. Lafortune, "Optimal supervisory control with mean payoff objectives and under partial observation," *Automatica*, vol. 123, p. 109359, 2021.
[14] L. V. Alves, P. N. Pena, and R. H. Takahashi, "Planning on discrete event systems using parallelism maximization," *Control Engineering Practice*, vol. 112, p. 104813, 2021.
[15] J. Fu, A. Ray, and C. M. Lagoa, "Unconstrained optimal control of regular languages," *Automatica*, vol. 40, no. 4, pp. 639–646, 2004.
[16] R. Sengupta and S. Lafortune, "An optimal control theory for discrete event systems," *SIAM Journal on control and Optimization*, vol. 36, no. 2, pp. 488–541, 1998.
[17] R. Su, J. H. Van Schuppen, and J. E. Rooda, "The synthesis of time optimal supervisors by using heaps-of-pieces," *IEEE Transactions on Automatic Control*, vol. 57, no. 1, pp. 105–118, 2011.
[18] Z. Ma and K. Cai, "Optimal secret protections in discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 67, no. 6, pp. 2816–2828, 2021.
[19] L. Grigorov and K. Rudie, "Near-optimal online control of dynamic discrete-event systems," *Discrete Event Dynamic Systems*, vol. 16, pp. 419–449, 2006.
[20] P. Lv, Z. Xu, Y. Ji, S. Li, and X. Yin, "Optimal supervisory control of discrete event systems for cyclic tasks," *Automatica*, vol. 164, p. 111634, 2024.
[21] K. Chatterjee, M. Randour, and J.-F. Raskin, "Strategy synthesis for multi-dimensional quantitative objectives," *Acta informatica*, vol. 51, no. 3, pp. 129–163, 2014.
[22] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic model checking and autonomy," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, pp. 385–410, 2022.
[23] E. Filiot, T. L. Gall, and J.-F. Raskin, "Iterated regret minimization in game graphs," in *International Symposium on Mathematical Foundations of Computer Science*, pp. 342–354, 2010.
[24] P. Hunter, G. A. Pérez, and J.-F. Raskin, "Reactive synthesis without regret," *Acta Informatica*, vol. 54, no. 1, pp. 3–39, 2017.
[25] M. Cadilhac, G. A. Pérez, and M. v. d. Bogaard, "The impatient may use limited optimism to minimize regret," in *International Conference on Foundations of Software Science and Computation Structures*, pp. 133–149, Springer, 2019.
[26] X. Yin and S. Lafortune, "Synthesis of maximally permissive supervisors for partially-observed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 5, pp. 1239–1254, 2015.