

No-regret path planning for temporal logic tasks in partially-known environments

Jianing Zhao¹ , Keyi Zhu², Mingyang Feng¹, Shaoyuan Li¹ and Xiang Yin¹ 

Abstract

In this paper, we investigate the graph-based robot path planning problem for high-level specifications described by co-safe linear temporal logic (*scLTL*) formulae. Our focus is on scenarios where the map geometry of the workspace is only partially-known. Specifically, we assume the existence of unknown regions, where the robot lacks prior knowledge of their successor regions unless it physically reaches these areas. In contrast to the standard non-deterministic synthesis approach that optimizes the worst-case cost, in the paper, we propose using regret as the metric for planning in such partially-known environments. Regret measures the difference between the actual cost incurred and the best-response cost the robot could have achieved if it were aware of the actual environment from the start. We present a formal model for this problem setting and develop an efficient algorithm to find an optimal strategy in the sense that it meets the *scLTL* specification while minimizing the regret of the strategy. Our approach provides a quantitative method for evaluating the trade-off between exploration and non-exploration, rather than relying on the heuristic determinations used in many existing works. Case studies on firefighting and collaborative robots are provided to illustrate the effectiveness of our framework. Furthermore, we conduct numerical experiments on a large number of randomly generated systems and compare the performance of the regret-based strategy with other path planning strategies. The experimental results indicate that regret is a highly meaningful metric for path planning in partially-unknown environments, especially in cases where no probabilistic *a priori* knowledge is available.

Keywords

Path planning; formal methods; linear temporal logic; regret minimization

Received 10 June 2024; Revised 16 October 2024; Accepted 17 December 2024

Senior Editor: Carla Seatzu

Associate Editor: Patricia Pena

1. Introduction

1.1. Backgrounds and motivations

Path planning and decision-making are central problems in autonomous robotics. In this context, one needs to design finite or infinite paths for robots based on system dynamics and the underlying environment, ensuring they meet specific requirements such as reaching target states while avoiding obstacles (LaValle, 2006). With the increasing demand for complex functionalities in autonomous robots, there is a growing need for path planning under complex logic tasks evolving spatial and temporal constraints. For instance, in search and rescue scenarios, robots may need to persistently surveil specific regions of interest according to some predefined rules (Fiaz and Baras, 2020). Similarly, in intelligent warehouse systems, robots must efficiently pick up and deliver items between various locations in a specific logic order (Scher and Kress-Gazit, 2020). Therefore, robot path planning for *high-level specifications* using formal

methods has been drawing increasingly more attentions in the past years; see, for example, Lin (2014), Kress-Gazit et al. (2018), Luckcuck et al. (2019), Belta and Sadraddini (2019), Mahulea et al. (2020), and Yin et al. (2024).

Among many formal languages, Linear Temporal Logic (LTL) stands out as one of the most popular specifications in describing and synthesizing high-level tasks for robotic applications. Specifically, by introducing temporal operators such as “always” or “eventually,” LTL formulae can express complex formal requirements such as “first

¹Department of Automation, Shanghai Jiao Tong University, Shanghai, China

²Department of Mechanical Engineering, Michigan State University, East Lansing, MI, USA

Corresponding author:

Xiang Yin, Department of Automation, Shanghai Jiao Tong University, 800 Dongchuan RD. Minhang District, Shanghai 200240, China.

Email: yinxiang@sjtu.edu.cn

reaching a desired region and then repeatedly visiting a target region.” In the context of robot path planning for LTL tasks, the mobility of the robot is often abstracted as a labeled transition system that considers the connectivity between regions and the dynamics of the robot. Following the automata-theoretical approach for LTL model checking (Vardi and Wolper, 1986; Baier and Katoen, 2008), path planning problems can be then transformed into graph-search problems over the product space of the transition model and the automaton representation of the LTL formula. For instance, for co-safe LTL (scLTL) tasks, a fragment of general LTL tasks achievable within finite horizons, the LTL planning problem simplifies to a short path search problem leading to accepting states (Kloetzer and Mahulea, 2016; Cho et al., 2017; Liu et al., 2024). For general fragments of LTL tasks evaluated over infinite paths, Smith et al. (2011), Kloetzer and Mahulea (2020), Luo et al. (2021), Luo and Zavlanos (2022), and Hustiu et al. (2024) studied the generation of optimal plans using the “prefix–suffix” structure.

When uncertainties exist in the environment, such as uncontrollable components or unknown obstacles, simply finding an open-loop plan is insufficient. Instead, one may seek to synthesize feedback strategies that can react to the dynamic environment. Mathematically, uncertainties in the environment can be abstracted as non-deterministic transitions in the system model. In this context, path planning for LTL tasks can be generally categorized into two categories:

- When one has prior knowledge regarding the probability distributions of non-deterministic environments, the synthesis problem can be formulated within the framework of Markov decision processes (MDPs) so that probabilistic performance in the expected sense can be optimized; see, for example, Fu and Topcu (2014), Ding et al. (2014), Guo and Zavlanos (2018), Lacerda et al. (2019), and Cai et al. (2021b).
- When no probabilistic information regarding uncertainties is available, the behaviors of the environment are usually treated to be purely non-deterministic, and a typical approach is to solve a robust synthesis problem ensuring worst-case performance metrics such as total cost for finite tasks and long-run average cost for infinite tasks; see, for example, Wolff et al. (2012, 2013), Bloem et al. (2014), and Fu and Topcu (2016).

While stochastic models such as MDPs offer a suitable framework for quantifying uncertainty, such probabilistic information is not always available in many applications, especially when exploring an unknown workspace for the first time. Hence, one has to handle the non-stochastic control settings. However, relying solely on worst-case analysis may also lead to a highly conservative strategy.

To illustrate the conservativeness of worst-case synthesis, let us first consider a simple motivating example as follows. Suppose that a robot moves in a *partially-known* workspace shown in Figure 1. Initially, the robot is aware of the

presence of certain walls (denoted by solid lines) as well as a door (represented by the dashed line between regions 2 and 5). However, it is unknown a priori whether the door is locked or not. The robot will only ascertain the status of the door upon reaching region 2. The objective of the robot is to reach target region 5 with shortest distance. If one takes a worst-case strategy, then the robot will follow the red trajectory. This is because the short-cut from regions 2 to 5 may not exist when the door is locked; if it goes to region 2, then in the worst-case, it will spend additional effort to go back. However, by taking the red trajectory, the robot may *heavily regret* by thinking that it should have taken the short-cut at region 2 if it knows *with hindsight* that the door is open. However, a more natural and human-like plan is to first go to region 2 to take a look at whether the door is locked. If not, then it can take the short-cut, which saves 7 units cost. Otherwise, the robot needs to go back to the red trajectory. Compared with the red path, although this approach may have two more units cost than the worst-case, it takes the potential huge advantage of exploring the unknown regions.

The simple example above suggests that even in a non-stochastic setting, worst-case performance may not always be a suitable metric for quantifying a planning strategy. Particularly, for partially-known environments that require explorations, worst-case analysis overlooks the trade-off between the cost incurred from exploration and the potential benefits gained through exploration. On the other hand, exploration is not always necessary if it does not provide more benefit. For example, in Figure 1, if the distance between region 2 and the unknown wall is farther, say 3 units of cost, then the robot may choose not to explore the unknown region. This is because, if it explores and finds no shortcut due to the presence of the wall, then it will regret not choosing the worst-case plan without exploration. Therefore, a systematic approach is needed to effectively and quantitatively balance the trade-off between “to explore” or “not to explore.”

1.2. Our results and contributions

In this paper, we address a new type of optimal path planning problem for robots operating in a *partially-known*

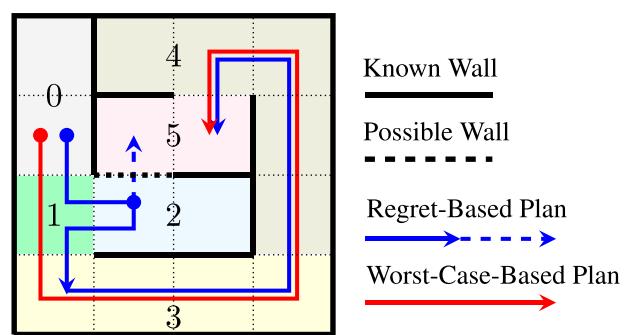


Figure 1. A motivating example, where a robot needs to reach region 5 from region 0 with partially-known environment information.

environment under scLTL specifications. Specifically, we assume that while the location of each region in the workspace is perfectly known, the robot does not have prior knowledge of the connectivities for some of these regions unless it physically explores them. This situation arises in many scenarios where the robot only maintains a basic map of the workspace and needs to explore certain areas online to gather detailed information. It is important to note that the terminologies “partially-known” environments and “non-deterministic” environments are distinct. The latter refers to scenarios where the environment’s outcome is entirely random, meaning it might behave differently even with identical visits. However, a partially-known environments is referred to the case where the robot has *information uncertainty* regarding the true world initially, but the underlying actual environment is still fixed and deterministic.

Mathematically speaking, the path planning problem in such a partially-known environment can be viewed as a specialized type of reactive synthesis problem, where the environment adopts a location-based strategy predetermined at the initial instant. That is, the environment player must fix its decision for each vertex at the initial stage of the game, as the underlying environment is unknown but static. However, our approach goes beyond the standard worst-case analysis framework typically used for qualitative synthesis problems in such scenarios. Specifically, to evaluate the performance of planning strategies in partially-known environments, we propose using the generic notion of *regret* borrowed from the game theory literature (Blackwell, 1956), as the optimality metric. The regret of a plan under a fixed but unknown environment is defined as the difference between its actual cost and the best-response cost it could have achieved with hindsight once the actual environment is known. Our overall objective is to synthesize a planning strategy that minimizes regret for all potential but unknown actual environments while satisfying the scLTL specification.

The main contribution of this paper is the application of the generic regret metric to the planning problem in partially-known environments. This application is non-trivial as it requires formal modeling of the exploration process, the development of an efficient synthesis algorithm, and experimental evaluation to demonstrate the applicability of the metric. More specifically, the main results and contributions of this paper are summarized as follows:

- First, we establish a formal framework for modeling the path planning problem in partially-known environments. Specifically, we propose the structure of *partially-known weighted transition systems* (PK-WTS) as the “possible world” model that encompasses the set of all potential actual environments. Building on the PK-WTS, we formally describe the evolution of knowledge during the exploration process and introduce regret as the metric for evaluating the planning problem.

- We then present an efficient algorithm for synthesizing regret-optimal planning strategy under scLTL specifications. Our algorithm consists of two stages. First, by integrating exploration knowledge into the system model, we introduce a novel game arena termed the *knowledge-based game arena*. We demonstrate that solving the regret-optimal planning problem is equivalent to addressing a quantitative two-player game, where the environment adopts a specific strategy called the strongly positional strategy. Then leveraging the structural properties of this game type, we develop an efficient value iteration algorithm to solve the game. Our overall approach exhibits exponential complexity in the number of regions with unknown transitions, which is in general much small compared with the total of states, but maintains polynomial complexity in the size of the transition system and the specification automaton.
- Finally, we implement our algorithm in two real-world case studies, which demonstrate the effectiveness of our approach. Furthermore, we conduct numerical experiments on a large number of randomly generated systems, comparing the actual cost performance among the proposed regret-based strategy with the standard best-case and worst-case strategies. Our statistical results demonstrate that regret is a highly meaningful metric for path planning in partially-unknown environments, particularly when there is no probabilistic a priori knowledge available.

1.3. Related works

We discuss related works in the literature from the following four perspectives.

1.3.1. LTL Planning in known environments. Optimal path planning for single or multiple robots under LTL constraints has been extensively studied in the past decades. For instance, Smith et al. (2011) and Ulusoy et al. (2013) investigated synthesizing optimal plans in the prefix–suffix structure to minimize per-cycle costs while achieving both LTL formulae and surveillance tasks. However, the major challenge in LTL planning for multi-robot systems is the curse of dimensionality as the number of robots increases. To address this issue, efficient and asymptotically optimal approaches such as sampling-based methods using rapid random trees (RRT) have been developed (Kantaros and Zavlanos, 2018, 2020; Vasile et al., 2020). Petri-nets-based LTL planning techniques have also been developed to leverage the structural properties in concurrent systems (Lacerda and Lima, 2019; Kloetzer and Mahulea, 2020; Lv et al., 2023). Furthermore, distributed approaches have been widely adopted to synthesize local plans for each robot to mitigate the complexity challenge (Kloetzer and Belta, 2009; Luo and Zavlanos, 2022; Yu and Dimarogonas, 2022). Researchers have also investigated path planning problems for variants of LTL tasks, such as counting temporal logics (Sahin et al., 2020), LTLf (Brafman et al.,

2019), and HyperLTL (Wang et al., 2020). In addition to the LTL specification, many recent works have also adopted the signal temporal logic (STL) specification to incorporate real-valued and real-time requirements; see, for example, Gundana and Kress-Gazit (2021), Yu et al. (2023), Leung et al. (2023), Cardona and Vasile (2024), and Yu et al. (2024). Nonetheless, these works all assume that the environment is known, meaning that the map geometry and semantic structure are available during the planning stage.

1.3.2. Planning in unknown environments. There are existing works that address planning problems in the literature when the environment is unknown or partially-known. For example, in Fridovich-Keil et al. (2019), the authors tackled the motion planning problem in an a priori unknown environment, ensuring both safety and liveness. The challenge of high-speed navigation of quadrotors in unknown environments was studied in Zhou et al. (2021), where trajectory replanning techniques were used. In Ho et al. (2024), a sampling-based approach was developed to synthesize strategies for nondeterministic hybrid systems with unknown environmental components. However, these works focus on continuous motion planning settings and do not consider a quantitative performance metric like regret. Recently, there have been related works addressing the challenge of LTL planning in unknown or partially-known environments, where some information such as workspace connectivities, transition probabilities, and region semantics is not precisely known beforehand. For example, in Guo and Dimarogonas (2015), the authors presented a replanning-based algorithm that updates the system model and execution plan when changes in the environment are detected. Similarly, an efficient iterative planning algorithm was proposed in Lahijanian et al. (2016) that not only adjusts the plan on-the-fly in the presence of unforeseen obstacles but also modifies the specification itself when the originally defined task cannot be fully achieved. In Ayala et al. (2013), it was assumed that both the environment map and the planning strategy are incrementally constructed based on information acquired by sensors. In the framework of MDPs, learning-based approaches have also been utilized to tackle the challenge of LTL planning in stochastic environments with unknown transition probabilities; see, for example, Hasanbeig et al. (2019), Bozkurt et al. (2020), and Cai et al. (2021a, 2023). More recently, Kantaros et al. (2022) investigated the scLTL planning problem under environments with known map geometries but with semantic uncertainties. Specifically, it addressed the issue of perceptual uncertainty by developing a novel sampling-based algorithm to iteratively generate plans online. However, all these approaches are either based on stochastic control settings or rely on best/worst-case optimality metrics. Still, the metric of regret is not investigated in these works.

1.3.3. Planning in non-deterministic domains. In the literature on artificial intelligence, there is extensive research

on planning in non-deterministic domains. A fundamental problem in this area is the fully observable non-deterministic (FOND) planning problem, where the state is perfectly observable, but the outcomes of actions are non-deterministic and cannot be predicted during planning; see, for example, Bertoli et al. (2006) and Muise et al. (2014). The FOND problem has also been extended to partially observable non-deterministic (POND) planning, where the state is not directly observable and is inferred through an observation relation (Cimatti et al., 2003; Geffner and Geffner, 2018). Our partially-known planning problem falls under the broader class of FOND problems, as we assume the current state is perfectly observable, while the non-determinism comes from unknown successor patterns. However, compared to the standard FOND problem, the key difference in our work is that we introduce an optimality condition based on the regret metric. Moreover, although a partially-known environment represents a form of non-determinism, it has specific structural properties that allow us to design an efficient algorithm.

1.3.4. Regret-optimal graph games. Mathematically, the regret-optimal planning problem addressed in this paper falls within the category of two-player graph games with quantitative objectives (Chatterjee et al., 2014; Kwiatkowska et al., 2022). Specifically, it can be viewed as an instance of regret minimization games where the environment employs a particular type of positional strategies. Regret minimization is an emerging topic in the context of graph games; see, for example, Filiot et al. (2010), Hunter et al. (2017), and Cadilhac et al. (2019). Particularly, the setting in Filiot et al. (2010) is closely related to our problem setting: it addresses a reachability game with minimal regret using the graph-unfolding technique. Note that scLTL specification is essentially a reachability requirement over the product space between the system model and the specification automaton. However, Filiot et al. (2010) consider a setting where the strategy of the environment-player is unrestricted in the sense that it allows to change its decision freely each time it visits the same state. This unrestricted setting does not align with the partially-known environment scenario, where the underlying environment is fixed, and the player must make consistent decisions when visiting the same state multiple times. In principle, this gap in strategy spaces can be addressed by constructing the knowledge-based game arena, as we proposed in the paper. However, the algorithm in Filiot et al. (2010) relies on a graph-unfolding technique, which incurs pseudo-polynomial complexity to solve the regret minimization problem in the constructed game. Our approach, on the other hand, leverages the key structural property of knowledge accumulation in path planning, reducing the complexity of regret minimization from pseudo-polynomial to polynomial using a value iteration algorithm. In the context of robotic applications, the recent work (Muvvala et al., 2022) used regret to optimize human–robot collaboration strategies. However, the purposes of using regret in

our work and in [Muvvala et al. \(2022\)](#) are distinct. Here, we utilize regret to address the exploration issue in partially-known environments, whereas [Muvvala et al. \(2022\)](#) focus on human–robot collaboration. Additionally, while [Muvvala et al. \(2022\)](#) directly adopted the synthesis algorithm from [Filiot et al. \(2010\)](#), our work presents a more efficient algorithm customized to the structural properties of our problem setting.

1.3.5. Non-stochastic optimal control. Finally, it is worth noting that the metric of regret has found widespread adoption in the fields of online learning and online optimization ([Hazan, 2016](#); [Shalev-Shwartz et al., 2012](#)) as a measure of solution efficiency. Particularly, in the context of non-stochastic optimal control for dynamical systems, regret has proven to be a meaningful metric compared to standard H_2 or H_∞ control settings ([Hazan et al., 2020](#); [Zhou et al., 2023](#); [Yan et al., 2023](#); [Goel and Hassibi, 2023](#)). For example, experimental results in [Goel and Hassibi \(2023\)](#) demonstrate that regret-optimal controllers perform better across various unknown disturbances compared to other robust controllers. However, due to the fundamental differences between controlling continuous dynamical systems and synthesizing reactive systems over discrete state-spaces, these results are not directly applicable to our problem with high-level specifications over symbolic state-spaces. In fact, our results can be seen as an extension of the general findings in continuous-time systems, showing that regret is also a suitable metric for reactive synthesis for symbolic systems in the presence of unknown uncertainties.

1.4. Organizations

The remaining part of this paper is organized as follows. In Section 2, we review some necessary preliminaries and the standard LTL planning in fully-known environments. In Section 3, we present a formal model for partially-known environments and introduce regret as the performance metric. In Section 4, we transfer the regret-minimizing planning problem as a quantitative two-player game on a new structure named *knowledge-based game arena*. In Section 5, we propose an efficient algorithm to solve the regret-minimizing synthesis problem. Case studies on firefighting robots and collaborative robots are provided in Section 6 to show the effectiveness of our algorithm. Numerical experiments are also provided to compare the performance of the regret-based strategy with the other strategies. Finally, we conclude the paper in Section 7.

Some preliminary and partial results in this paper were presented in the conference version ([Zhao et al., 2023](#)). Specifically, the knowledge-based game arena was introduced in [Zhao et al. \(2023\)](#), but the solution to the regret minimization problem is based on a slight modification of the algorithm in [Filiot et al. \(2010\)](#) over the constructed game graph. Therefore, the complexity of the

overall algorithm is pseudo-polynomial in the size of the game graph. However, compared with [Zhao et al. \(2023\)](#), this journal version has several key differences. First, we propose a more efficient algorithm that leverages the structural properties of the problem, in contrast to [Zhao et al. \(2023\)](#), which uses a standard regret-minimization algorithm that is more complex. Furthermore, our work includes detailed hardware experiments as well as thorough numerical analyses, providing strong support for using regret as the metric in our context.

2. Preliminaries

In this section, we briefly review some necessary preliminaries and the standard approach for solving the LTL planning problem in a fully-known environment. Let A be a set of symbols. We denote by A^ω (respectively, A^*) the sets of all infinite (respectively, finite) sequences of symbols over A ; we denote by 2^A the power-set of A . We use notation “ $\{\}$ ” to denote an unordered set, and use notation “ $\langle \rangle$ ” to denote an ordered set. Notation \mathbb{R}^+ denotes the set of all positive real numbers.

2.1. Weighted transition systems

In the context of path planning for high-level specifications, the workspace of the agent is usually abstracted as a discrete transition system with weights ([Smith et al., 2011](#); [Belta et al., 2017](#)). Specifically, when the environment of the workspace is fully-known, the mobility of the agent (or map geometry) is usually modeled as a *weighted transition system* (WTS) defined as follows.

Definition 1. (Weighted Transition Systems) A weighted transition system (WTS) is a 6-tuple

$$T = (X, x_0, \delta_T, w, \mathcal{AP}, L),$$

where X is a set of states representing different regions of the workspace; $x_0 \in X$ is the initial state representing the starting region of the agent; $\delta_T : X \rightarrow 2^X$ is the transition function such that, starting from each state $x \in X$, the agent can move directly to any of its successor state $x' \in \delta_T(x)$. We also refer $\delta_T(x)$ to as the successor states of x . Function $w : X \times X \rightarrow \mathbb{R}^+$ is a cost function such that $w(x, x')$ represents the cost incurred when the agent moves from x to x' ; \mathcal{AP} is the set of atomic propositions representing basic properties of our interest; and $L : X \rightarrow 2^{\mathcal{AP}}$ is a labeling function assigning each state a set of atomic propositions.

Given a WTS T , an infinite *path* of T is an infinite sequence of states $\rho = x_0x_1x_2\cdots \in X^\omega$ such that $x_{i+1} \in \delta_T(x_i)$, $i \geq 0$. A finite path is defined analogously. We denote by $\text{Path}^\omega(T)$ and $\text{Path}^*(T)$ the sets of all infinite paths and finite paths in T , respectively. Given a finite path $\rho = x_0x_1\cdots x_n \in \text{Path}^*(T)$, its cost is defined as the sum of all transition weights in it, that is,

$$\text{cost}(\rho) = \sum_{i=0}^{n-1} w(x_i, x_{i+1}).$$

The *trace* of an infinite or finite path $\rho = x_0x_1x_2(\dots)$ is an infinite or finite sequence over $2^{\mathcal{AP}}$ denoted by $L(\rho) = L(x_0)L(x_1)(\dots)$. Analogously, we denote by $\text{Trace}^\omega(T)$ and $\text{Trace}^*(T)$ the sets of all infinite traces and finite traces in T , respectively.

2.2. Linear temporal logic specifications

The syntax of general LTL formula is given as follows

$$\phi = \top \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \bigcirc\phi \mid \phi_1 U \phi_2,$$

where \top stands for the “true” predicate; $a \in \mathcal{AP}$ is an atomic proposition; \neg and \wedge are Boolean operators “negation” and “conjunction,” respectively; \bigcirc and U denote temporal operators “next” and “until,” respectively. One can also derive other temporal operators such as “eventually” by $\Diamond\phi = \top U \phi$ and “always” by $\Box\phi = \neg\Diamond\neg\phi$. LTL formulae are evaluated over infinite words; the readers are referred to Baier and Katoen (2008) for the detailed semantics of LTL. Specifically, an infinite word $\tau \in (2^{\mathcal{AP}})^\omega$ is an infinite sequence over alphabet $2^{\mathcal{AP}}$. We write $\tau \models \phi$ if τ satisfies LTL formula ϕ .

In this paper, we focus on a widely used fragment of LTL formulae called the *co-safe LTL* (scLTL) formulae. Specifically, an scLTL formula requires that the negation operator \neg can only be applied in front of atomic propositions. Consequently, one cannot use temporal “always” \Box in scLTL. Although the semantics of LTL are defined over infinite words, it is well known that any infinite word satisfying an scLTL formula has a *finite good prefix*. Specifically, a good prefix is a finite word $\tau' = \tau_1 \cdots \tau_n \in (2^{\mathcal{AP}})^*$ such that $\tau'\tau'' \models \phi$ for any $\tau'' \in (2^{\mathcal{AP}})^\omega$. We denote by $\mathcal{L}_{\text{pref}}^\phi \subseteq (2^{\mathcal{AP}})^*$ the set of all finite good prefixes of scLTL formula ϕ .

For any scLTL formula ϕ , its good prefixes $\mathcal{L}_{\text{pref}}^\phi$ can be accepted by a *deterministic finite automaton* (DFA) (Belta et al., 2017) defined as follows.

Definition 2. (Deterministic Finite Automata) A deterministic finite automaton is a 5-tuple $\mathcal{A} = (Q, q_0, \Sigma, f, Q_F)$, where Q is the set of states; $q_0 \in Q$ is the initial state; Σ is the alphabet; $f : Q \times \Sigma \rightarrow Q$ is a transition function; and $Q_F \subseteq Q$ is the set of accepting states.

The transition function is extended to $f : Q \times \Sigma^* \rightarrow Q$ recursively such that $\forall q \in Q, s \in \Sigma^*, \sigma \in \Sigma : f(q, s\sigma) = f(f(q, s), \sigma)$. A finite word $\tau \in \Sigma^*$ is said to be *accepted* by \mathcal{A} iff $(q_0, \tau) \in Q_F$; we denote by $\mathcal{L}(\mathcal{A})$ the set of all accepted words. Then for any scLTL formula ϕ defined over \mathcal{AP} , it is well known that we can always build a DFA over alphabet $\Sigma = 2^{\mathcal{AP}}$, denoted by $\mathcal{A}_\phi = (Q, q_0, 2^{\mathcal{AP}}, f, Q_F)$, such that $\mathcal{L}(\mathcal{A}_\phi) = \mathcal{L}_{\text{pref}}^\phi$.

2.3. Path planning for scLTL specifications

Given a WTS T and an scLTL formula ϕ , the path planning problem is to find a finite path (a.k.a. a plan) $\rho \in \text{Path}^*(T)$ such that $L(\rho) \in \mathcal{L}_{\text{pref}}^\phi$ and, at the same time, its cost $\text{cost}(\rho)$ is minimized. This problem can be reduced as a graph-search problem over the *product system*.

Definition 3. (Product Systems) Given WTS $T = (X, x_0, \delta_T, w, \mathcal{AP}, L)$ and DFA $\mathcal{A}_\phi = (Q, q_0, \Sigma, f, Q_F)$, the product system is a new (unlabeled) WTS

$$P = T \otimes \mathcal{A}_\phi = (S, s_0, \delta_P, w_P, S_F),$$

where $S = X \times Q$ is the set of states; $s_0 = (x_0, q_0)$ is the initial state; $\delta_P : S \rightarrow 2^S$ is the transition function defined by: for any $s = (x, q) \in S$, we have

$$\delta_P(s) = \{(x', q') \in S \mid x' \in \delta_T(x) \wedge q' = f(q, L(x))\};$$

$w_P : S \times S \rightarrow \mathbb{R}$ is the weight function defined by:

$$\forall s = (x, q), s' = (x', q') \in S : w_P(s, s') = w(x, x');$$

and $S_F = X \times Q_F$ is the set of accepting states.

By construction, for any path $\rho = (x_0, q_0) \cdots (x_n, q_n)$ in the product system, $(x_n, q_n) \in S_F$ implies that (i) $\rho = x_0 \cdots x_n \in \text{Path}^*(T)$, and (ii) $L(\rho) \in \mathcal{L}_{\text{pref}}^\phi$. Therefore, to solve the optimal scLTL planning problem, it suffices to find a path with minimum weight from the initial state to accepting states S_F in the product system.

3. Planning in partially-known environments

The above reviewed graph-search-based LTL planning method crucially depends on the information of knowing the mobility of the agent, or the environment map T is perfectly known. This method, however, is not suitable for the case of *partially-known* environments. To be specific, this work considers a partially-known environment characterized by the following assumptions:

- A1. The agent knows the existence of all regions in the environment as well as their semantics (atomic propositions hold at each region);
- A2. The successor regions of each region in the real world are fixed, but the agent may not know, a priori, what are the actual successor regions it can move to;
- A3. Once the agent physically reaches a region, it will know the actual successor regions of this region precisely.

In this section, we will provide a formal model for such a partially-known environment using the new structure of *partially-known weighted transition systems* and use *regret* as a new metric for evaluating the performance of the agent’s plan in such an environment.

3.1. Modeling of partially-known environments

Definition 4. (Partially-Known WTS) A partially-known weighted transition system (*PK-WTS*) is a 6-tuple

$$\mathbb{T} = (X, x_0, \Delta, w, \mathcal{AP}, L),$$

where, similar to the *WTS*, X is the set of states with initial state $x_0 \in X$, $w: X \times X \rightarrow \mathbb{R}$ is the cost function and $L: X \rightarrow 2^{\mathcal{AP}}$ is a labeling function that assigns each state a set of atomic propositions. Different from the *WTS*,

$$\Delta: X \rightarrow 2^{2^X}$$

is called a successor-pattern function that assigns each state $x \in X$ a family of successor states.

The intuition of the *PK-WTS* \mathbb{T} is explained as follows. Essentially, *PK-WTS* is used to describe the *possible world* from the perspective of the agent. Specifically, under assumptions A1 and A3, the agent has some prior information regarding the successor states of each unknown region but does not know which one is true before it actually visits the region. Therefore, in *PK-WTS* \mathbb{T} , for each state $x \in X$, we have

$$\Delta(x) = \{o_1, \dots, o_{|\Delta(x)|}\},$$

where each $o_i \in 2^X$ is called a *successor-pattern* representing a possible set of actual successor states at state x . Hereafter, we will also refer each $o_i \in \Delta(x)$ to as an *observation* at state x since the agent “observes” its successor states when exploring state x . Therefore, for each state $x \in X$, we say x is a

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

We partition the state space as $X = X_{kno} \dot{\cup} X_{un}$, where X_{kno} is the set of known states and X_{un} is the set of unknown states. We assume that the initial state x_0 is known since the agent has already stayed at x_0 so that it has the precise information regarding the successor states of x_0 .

In reality, the agent must move in a specific environment that is compatible with the possible world \mathbb{T} , although itself does not know this a priori.

Definition 5. (Compatible WTS) We say a *WTS* $T = (X, x_0, \delta_T, w, \mathcal{AP}, L)$ is compatible with *PK-WTS* \mathbb{T} , denoted by $T \in \mathbb{T}$, if for any $x \in X$, we have $\delta_T(x) \in \Delta(x)$.

Clearly, if all states in \mathbb{T} are known, then its compatible *WTS* is unique.

Example 1. (Running Example) We use the motivating example in Figure 1 as the simple running example to illustrate all concepts. This partially-known environment in this example can be modeled (by simplification) as the *PK-WTS* shown in Figure 2. Here, the successor-pattern

function Δ is defined by: $\Delta(x_0) = \{\{x_1\}\}$, $\Delta(x_1) = \{\{x_2, x_3\}\}$, $\Delta(x_2) = \{\{x_1\}, \{x_1, x_5\}\}$, and $\Delta(x_3) = \{\{x_4\}\}$. Therefore, x_2 is the unique unknown state, while all other states are known. Specification, $\{x_1\} \in \Delta(x_2)$ and $\{x_1, x_2\} \in \Delta(x_2)$ represent, respectively, the two possible situations that the door is locked and the door is open at state x_2 . Note that the status of the door is assumed to be unchanged during the planning process. We assign the labeling function $L: X \rightarrow 2^{\mathcal{AP}}$ by: $L(x_5) = \{\text{target}\}$ and $L(x_i) = \emptyset$ for each $i = 0, 1, \dots, 4$. The task can be described by scLTL formula $\phi = \Diamond \text{target}$, whose corresponding DFA is given in Figure 3.

Remark 1. In our problem formulation, we assumed that only transition relations could be partially known, while the semantics of each region, that is, the label of each state, are perfectly known. This assumption can actually be relaxed by refining the state space of the transition system. Specifically, if it is known that a state has multiple possible sets of atomic propositions, but the actual set is unknown a priori until the robot explores the state, one can introduce a set of “copy states” between the original state and its successor states. Each copy state corresponds to a different possible set of atomic propositions that may hold at the original state. In this way, the uncertainty about atomic propositions at the original state can be modeled by an unknown transition function from the original state to the copy states. Furthermore, in our problem formulation, we assumed prior knowledge of possible successor patterns for an unknown state. If a state is fully unknown in the sense that no such prior

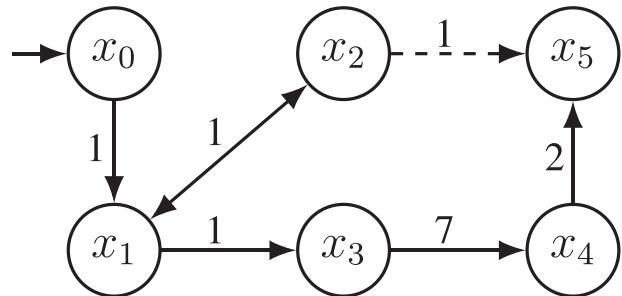


Figure 2. *PK-WTS* \mathbb{T} for the motivating example in Figure 1. For simplicity and without loss of generality, we omit some backward transitions, such as the transition from x_1 back to x_0 . Instead, we retain only the two-way transitions between x_1 and x_2 in the model since the agent needs to backtrack if there is no direct transition from x_2 to x_5 .



Figure 3. DFA \mathcal{A}_ϕ for scLTL formula $\phi = \Diamond \text{target}$.

knowledge is available, then we can account for all possible successor patterns by considering $\Delta(x) = 2^{Ne(x)}$, where $Ne(x)$ represents the set of states physically adjacent to state x .

3.2. Knowledge by explorations

Based on Assumption A3, when the agent visits a known state $x \in X_{kno}$, it will not gain any useful information about the environment since $\Delta(x)$ is already a singleton. However, when the agent visits an unknown state $x \in X_{un}$, it will gain new information and successor-pattern at this state will become known from then on. Therefore, we refer the visit to an unknown state to as an *exploration*. To capture the result of an exploration, we introduce the concept of *knowledge state*.

Definition 6. (Knowledge States) Given PK-WTS \mathbb{T} , a knowledge state is a tuple $\kappa = (x, o) \in X \times 2^X$ such that $o \in \Delta(x)$. For each knowledge state κ , we denote by $x(\kappa)$ and $o(\kappa)$, respectively, its first and second components, that is., $\kappa = (x(\kappa), o(\kappa))$. We denote by

$$Kw = \{\kappa \in X \times 2^X \mid o(\kappa) \in \Delta(x(\kappa))\}, \quad (1)$$

the set of all possible knowledge states.

Intuitively, a knowledge state $(x, o) \in X \times 2^X$ represents the information the agent obtains when exploring known state x , that is, the agent knows that the successor states of x are o . Therefore, as the agent moves in the partially-unknown environment, it collects more information. This accumulated information is then captured by the concept of *knowledge set*.

Definition 7. (Knowledge Sets) Given PK-WTS \mathbb{T} , a knowledge set $\mathcal{K} = \langle \kappa_1, \dots, \kappa_{|\mathcal{K}|} \rangle$, where $\kappa_i \in Kw$, 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.

It is worth noting that a knowledge state is defined as an *ordered* set to capture the information about which state is explored first. With a slight abuse of notation, for each state $x \in X$, we write $x \in \mathcal{K}$ if $(x, o) \in \mathcal{K}$ for some observation $o \in \Delta(x)$. Therefore, we further require that for each state $x \in X$ can only appear at most once in a knowledge set and we denote by $o_{\mathcal{K}}(x) \in \Delta(x)$ the unique observation such that $(x, o_{\mathcal{K}}(x)) \in \mathcal{K}$. This is because the environment is fixed and once we have explored state x , we know its successor-pattern forever according to Assumptions A2 and A3.

Suppose that the agent maintains a knowledge set to record its exploration history. Once a new unknown state is explored, its information can be updated according to the *knowledge update function*.

Definition 8. (Knowledge Updates) The knowledge update function $update : \mathbb{KW} \times Kw \rightarrow \mathbb{KW}$ is defined

by: for any knowledge set $\mathcal{K} = \langle \kappa_1, \dots, \kappa_{|\mathcal{K}|} \rangle \in \mathbb{KW}$ and knowledge state $\kappa \in Kw$, we have

$$update(\mathcal{K}, \kappa) = \begin{cases} \mathcal{K}, & \text{if } x(\kappa) \in \mathcal{K} \text{ or } x(\kappa) \in X_{kno} \\ \langle \kappa_1, \dots, \kappa_{|\mathcal{K}|}, \kappa \rangle, & \text{otherwise} \end{cases}.$$

We define $\mathcal{K}_0 = \emptyset$ as the initial knowledge set.

That is, if state x has already been explored, then we skip this repeated knowledge; otherwise, this new knowledge set is added to the knowledge state in order.

Finally, based on the current knowledge set it maintains, the agent can refine the possible world \mathbb{T} by eliminating uncertainties that have been explored.

Definition 9. (Refined PK-WTS) Let $\mathbb{T} = (X, x_0, \Delta, w, \mathcal{AP}, L)$ be a PK-WTS and $\mathcal{K} \in \mathbb{KW}$ be a knowledge set. The refined PK-WTS is a new PK-WTS

$$\mathbb{T}_{\mathcal{K}} = (X, x_0, \Delta', w, \mathcal{AP}, L) \quad (2)$$

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

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

Example 2. (Running Example Cont.) We still consider the PK-WTS in Figure 2. Initially, the knowledge set of the agent is $\mathcal{K}_0 = \emptyset$. When the agent visits unknown state x_2 , it will gain a knowledge state $\kappa_1 = (x_2, \{x_1\})$ or $\kappa_2 = (x_2, \{x_1, x_3\})$. Then, once the agent gains κ_1 or κ_2 , the knowledge set will be updated to $\mathcal{K}_1 = update(\mathcal{K}_0, \kappa_1) = \langle (x_2, \{x_1\}) \rangle$ or $\mathcal{K}_2 = update(\mathcal{K}_0, \kappa_2) = \langle (x_2, \{x_1, x_3\}) \rangle$, respectively. Both cases, their refined PK-WTSs induce unique compatible WTSs.

3.3. History-based strategies

In the partially-known setting, the agent makes decisions not only based on the finite sequence of states it has visited. In addition, it should also consider what it observed (successor-pattern) at each state visited.

Definition 10. (Histories) Given a PK-WTS \mathbb{T} , a history in \mathbb{T} is a finite sequence of knowledge states

$$h = \kappa_0 \kappa_1 \cdots \kappa_n = (x_0, o_0)(x_1, o_1) \cdots (x_n, o_n) \in Kw^* \quad (3)$$

such that

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

For history h in the form of equation (3), we call $x_0 x_1 \cdots x_n \in X^*$ its path. We denote by $Path^*(\mathbb{T})$ and $Hist^*(\mathbb{T})$ the set of all (finite) paths and histories generated in PK-WTS \mathbb{T} , respectively.

Intuitively, the first condition says that the agent can only move to one of its actual successor states in o_i . The second

condition captures the fact that the actual environment is partially-known but *fixed*; hence, the agent must observe the same successor-pattern for different visits of the same state. Note that, compared with knowledge sets, histories contain all states visited in order including those repeated state. However, knowledge sets only contain those unknown states explored for the first time.

Therefore, under the setting of partially-known environment, the plan is no longer an open-loop sequence. Instead, it is a feedback *strategy* that determines the next state the agent should go to based what has been visited and what has known, which are environment dependent.

Definition 11. (Planning Strategies) A (history-based) planning strategy is a function

$$\xi : \text{Hist}^*(\mathbb{T}) \rightarrow X \cup \{\text{stop}\}$$

such that for any $h = \kappa_1 \cdots \kappa_n$, where $\kappa_i = \langle x_i, o_i \rangle$, we have either (i) $\xi(h) \in o_n$, that is, it decides to move to some successor state; or (ii) $\xi(h) = \text{stop}$, that is, the plan is terminated. We denote by $\text{Stra}(\mathbb{T})$ the set of all strategies for \mathbb{T} .

Note that, in general, a strategy should map a history to an executable *action* for the robot. However, our paper addresses a planning problem for systems with deterministic underlying transitions (though some transitions may be unknown *a priori*). As a result, a strategy can directly assign the target state for the robot, and there are no explicit actions defined in our model.

Also, we note that, although a strategy is designed to handle all possible actual environments in the possible world \mathbb{T} , when it applies to an actual environment $T \in \mathbb{T}$, the outcome can be completely determined.

Definition 12. (Induced Paths) Given a planning strategy $\xi \in \text{Stra}(\mathbb{T})$ and actual environment $T \in \mathbb{T}$, the finite path induced by strategy ξ in environment $T \in \mathbb{T}$ is the unique path

$$\rho_\xi^T = x_0 x_1 \cdots x_n \in X^*$$

such that

- (i) $\forall i < n : \xi((x_0, \delta_T(x_0)) \cdots (x_i, \delta_T(x_i))) = x_{i+1}$; and
- (ii) $\xi((x_0, \delta_T(x_0)) \cdots (x_n, \delta_T(x_n))) = \text{stop}$.

Note that the agent does not know *a priori* which $T \in \mathbb{T}$ is the actual environment. To guarantee the accomplishment of the LTL task, a strategy $\xi \in \text{Stra}(\mathbb{T})$ should satisfy

$$\forall T \in \mathbb{T} : \rho_\xi^T \in \mathcal{L}_{\text{pref}}^\phi. \quad (4)$$

We denote by $\text{Stra}_\phi(\mathbb{T}) \subseteq \text{Stra}(\mathbb{T})$ the set of all strategies satisfying equation (4).

3.4. Problem formulation

To evaluate the performance of strategy ξ , a standard approach is to minimize the *worst-case cost* of the strategy among all possible environments, that is,

$$\text{cost}_{\text{worst}}(\xi) := \max_{T \in \mathbb{T}} \text{cost}(\rho_\xi^T). \quad (5)$$

However, as we have illustrated by the motivating example in Figure 1, this metric cannot capture the potential benefit obtained from exploring unknown states and the agent may regret due to the unexploration. To address this issue, in this work, we propose to use *regret* as the metric to evaluate the performance of a strategy.

Definition 13. (Regret) Given a partially-known environment described by PK-WTS \mathbb{T} and a task described by an scLTL ϕ , the regret of strategy ξ is defined by

$$\text{reg}_{\mathbb{T}}(\xi) = \max_{T \in \mathbb{T}} \left(\text{cost}(\rho_\xi^T) - \min_{\xi' \in \text{Stra}_\phi(\mathbb{T})} \text{cost}(\rho_{\xi'}^T) \right). \quad (6)$$

The intuition of the above notion of regret is as follows. For each strategy $\xi \in \text{Stra}_\phi(\mathbb{T})$ and each actual environment $T \in \mathbb{T}$, $\text{cost}(\rho_\xi^T)$ is the actual cost incurred when applying this strategy to this specific environment, while $\min_{\xi' \in \text{Stra}_\phi(\mathbb{T})} \text{cost}(\rho_{\xi'}^T)$ is cost of the *best-response strategy the agent should have taken* if it knows the actual environment T with hindsight. Therefore, their difference is the regret of the agent when applying strategy ξ in environment T . Note that, the agent does not know the actual environment T precisely *a priori*. Therefore, the regret of the strategy is considered as the worst-case regret among all possible environments $T \in \mathbb{T}$.

Example 3. (Running Example Cont.) Still, we consider the running example in Figure 2. Let ξ_1 be the strategy that does not explore state x_2 , which is referred to as the worst-based strategy and ξ_2 be the strategy that explores state x_2 first, which is referred to as the regret-based strategy. For this PK-WTS \mathbb{T} , there are two compatible WTS $T_1, T_2 \in \mathbb{T}$, where $\delta_{T_1}(x_2) = \{x_1\}$ and $\delta_{T_2}(x_2) = \{x_1, x_5\}$. Then

- for environment T_1 , we have $\text{cost}(\rho_{\xi_1}^{T_1}) = 11$, $\text{cost}(\rho_{\xi_2}^{T_1}) = 13$ and $\min_{\xi \in \text{Stra}_\phi(\mathbb{T})} \text{cost}(\rho_\xi^{T_1}) = 11$;
- for environment T_2 , we have $\text{cost}(\rho_{\xi_1}^{T_2}) = 11$, $\text{cost}(\rho_{\xi_2}^{T_2}) = 3$ and $\min_{\xi \in \text{Stra}_\phi(\mathbb{T})} \text{cost}(\rho_\xi^{T_2}) = 3$.

Therefore, the regrets for the two strategies ξ_1 and ξ_2 are $\text{reg}_{\mathbb{T}}(\xi_1) = \max(11 - 11, 11 - 3) = 8$ and $\text{reg}_{\mathbb{T}}(\xi_2) = \max(13 - 11, 3 - 3) = 2$, respectively.

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

Problem 1. (Regret-Optimal scLTL Planning) Given a possible world represented by PK-WTS \mathbb{T} and an scLTL task ϕ , synthesize a strategy $\xi \in \text{Stra}_\phi(\mathbb{T})$ such that

$$\forall \xi' \in \text{Stra}_\phi(\mathbb{T}) : \text{reg}_{\mathbb{T}}(\xi) \leq \text{reg}_{\mathbb{T}}(\xi'). \quad (7)$$

4. Knowledge-based games

In this section, we show that the regret-optimal scLTL planning problem can be reduced to a quantitative two-player graph-game by incorporating knowledge into the state-space.

4.1. Knowledge-based game arena

Given PK-WTS $\mathbb{T} = (X, x_0, \Delta, w, \mathcal{AP}, L)$, its *skeleton system* is a WTS

$$\mathcal{T} = (X, x_0, \delta_{\mathcal{T}}, w, \mathcal{AP}, L),$$

where for any $x \in X$, we have $\delta_{\mathcal{T}}(x) = \bigcup_{o \in \Delta(x)} o$, that is, the successor states of x is defined as the union of all possible successor-patterns. To incorporate with the task information, let $\mathcal{A}_\phi = (Q, q_0, \Sigma, f, Q_F)$ be the DFA that accepts all good-prefixes of scLTL formula ϕ . We construct the product system between \mathcal{T} and \mathcal{A}_ϕ , denoted by

$$\mathcal{P} = \mathcal{T} \otimes \mathcal{A}_\phi = (S = X \times Q, s_0, \delta_{\mathcal{P}}, w_{\mathcal{P}}, S_F),$$

where “ \otimes ” is the product operator defined in Section 2.3.

However, the state-space of \mathcal{P} is still not sufficient for the purpose of decision-making since the explored knowledges along the trajectory are missing. Therefore, we further incorporate the knowledge set into the product state-space and explicitly split the movement choice of the agent and the non-determinism of the environment. This leads to the *knowledge-based game arena* that captures all interactions between the agent and the partially-known environment.

Definition 14. (Knowledge-Based Game Arenas)

Given PK-WTS \mathbb{T} , the knowledge-based game arena is a bipartite graph

$$G = (V = V_a \dot{\cup} V_e, v_0, E),$$

where

- $V_a \subseteq X \times Q \times \mathbb{K}^W$ is the set of agent vertices;
- $V_e \subseteq X \times Q \times \mathbb{K}^W \times X$ is the set of environment vertices;
- $v_0 = (x_0, q_0, \mathcal{K}_0) \in V_a$ is the initial (agent) vertex, where \mathcal{K}_0 is the initial knowledge set;
- $E \subseteq (V_a \times V_e) \cup (V_e \times V_a)$ is the set of edges defined by: for any $v_a = (x_a, q_a, \mathcal{K}_a) \in V_a$ and $v_e = (x_e, q_e, \mathcal{K}_e, \hat{x}_e) \in V_e$, we have
 - $\langle v_a, v_e \rangle \in E$ whenever
 - (i) $(x_e, q_e, \mathcal{K}_e) = (x_a, q_a, \mathcal{K}_a)$; and
 - (ii) $\hat{x}_e \in o_{\mathcal{K}_a}(x_a)$.
 - $\langle v_e, v_a \rangle \in E$ whenever
 - (i) $x_a = \hat{x}_e$; and
 - (ii) $(x_a, q_a) \in \delta_{\mathcal{P}}(x_e, q_e)$; and
 - (iii) $\mathcal{K}_a = \text{update}(\mathcal{K}_e, (x_a, o))$ for some $o \in \Delta(x_a)$.

The intuition of the knowledge-based game arena is explained as follows. The graph is bipartite with two types of vertices: *agent vertices* from which the agent

chooses a feasible successor state to move to and *environment vertices* from which the environment chooses the actual successor-pattern in the possible world. More specifically, for each agent vertex $v_a = (x_a, q_a, \mathcal{K}_a)$, the first component x_a represents its physical location in the system, the second component q_a represents the current DFA state for task ϕ and the third component \mathcal{K}_a represents the knowledge set of the agent obtained along the trajectory. At each agent vertex, the agent chooses to move to a successor state. Note that since x_a is the current location, it has been explored and we have $x_a \in \mathcal{K}_a$, that is, we know that the actual successor states of x_a are $o_{\mathcal{K}_a}(x_a)$. Therefore, it can move to any environment state $v_e = (x_e, q_e, \mathcal{K}_e, \hat{x}_e)$ by “remembering” the successor state $\hat{x}_e \in o_{\mathcal{K}_a}(x_a)$ it chooses.

Now, at each environment state $v_e = (x_e, q_e, \mathcal{K}_e, \hat{x}_e)$, the meanings of the first three components are the same as those for agent state. The last component \hat{x}_e denotes the state it is moving to. Therefore, v_e can reach agent state $v_a = (x_a, q_a, \mathcal{K}_a)$, where the first two components are just the transition in the product system synchronizing the movements of the WTS and the DFA. Note that we have $x_a = \hat{x}_e$ since the movement has already been decided by the agent. However, for the last component of knowledge set \mathcal{K}_a , we need to consider the following two cases:

- If state x_a has already been explored, then the agent must observe the same successor-pattern as before. Therefore, the knowledge set is not updated;
- If state x_a has not yet been explored, then the new explored knowledge $(x_a, o) \in \mathbb{K}^W$ should be added to the knowledge set \mathcal{K}_a . Note that since this is the first time the agent visits x_a , any possible observations $o \in \Delta(x_a)$ consistent with the prior information are possible. Therefore, it is the environment’s choice which o to be actually observed.

Example 4. (Running Example Cont.) For PK-WTS \mathbb{T} and DFA \mathcal{A}_ϕ shown in in Figures 2 and 3, respectively, the knowledge-based game arena G is provided in Figure 4, where three knowledge sets $\mathcal{K}_0, \mathcal{K}_1, \mathcal{K}_2$ are given in Example 2.

Remark 2. Let $n = |X_{un}|$ be the number of the unknown states in X . Then the knowledge-based game arena G contains at most $n! \cdot 2^n \cdot |X| \cdot |Q| \cdot |\delta_{\mathcal{T}}|$ vertices, where $|\delta_{\mathcal{T}}|$ is the number of transitions in the skeleton system. Therefore, the resulting arena G is polynomial in the sizes of the PK-WTS and DFA, but exponential in the number of unknown states. In practice, however, the number of unknown states is much smaller than the total number of states.

Consider the environment vertices with at least two successors, that is, $v_e \in V_e$ satisfying $\text{Succ}(v_e) \geq 2$, which we call as “environment vertices with decisions.” Then we have the following structural property for G .

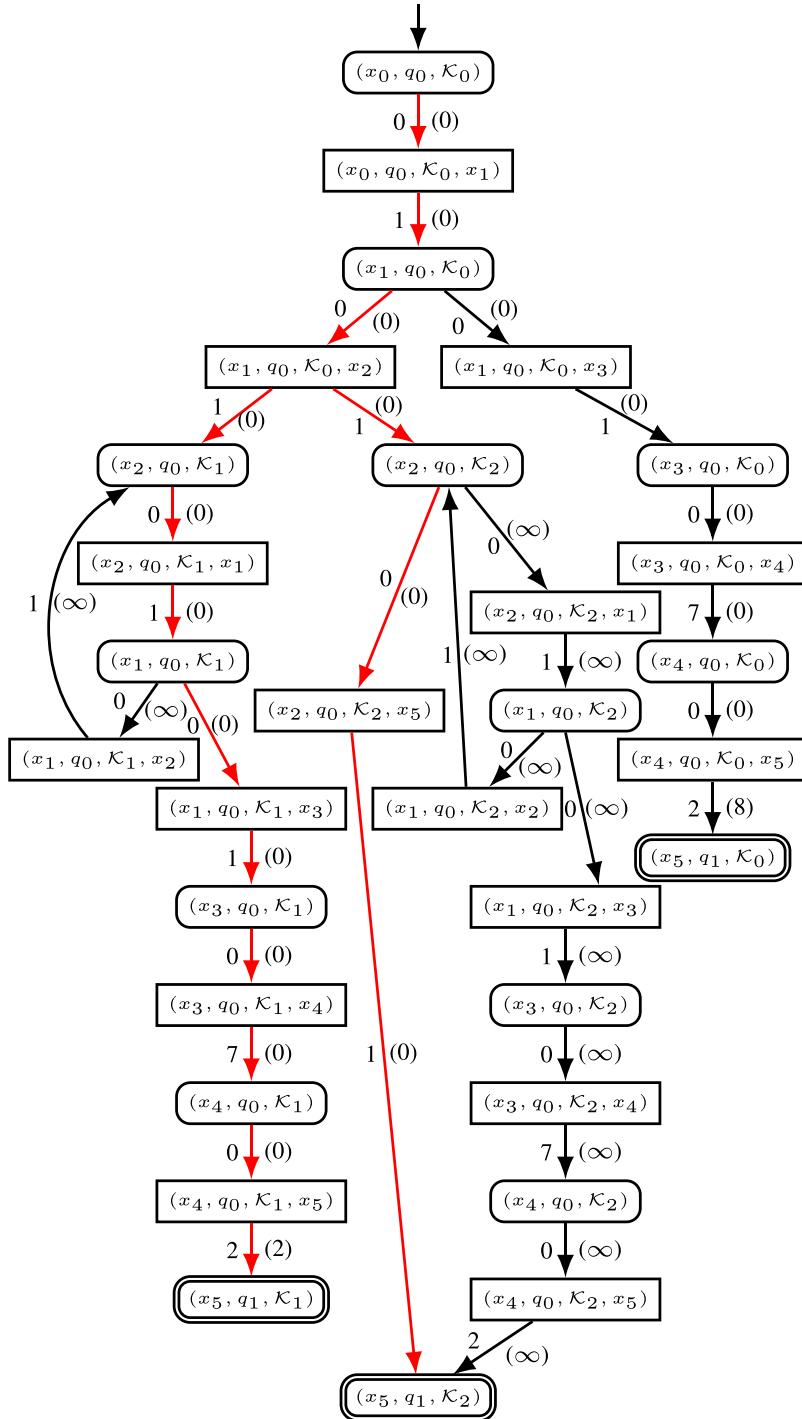


Figure 4. Knowledge-based Game Arena G . For each edge e , the number without parentheses denotes the cost of $\text{cost}_G(e)$, while the number within parentheses denotes the weight of $\mu(e)$.

Proposition 1. For the knowledge-based game arena G , there are no cycles encompassing two distinct environment vertices with decisions.

Proof. We prove this proposition by contradiction. Suppose that there exists a cycle that encompasses two different environment vertices with decisions. That is, there are two different environment vertices $v_e, v'_e \in V_e$ satisfying i) $v_e \neq v'_e$; ii) there is a sequence starting from v_e to v'_e ; and iii) there is also a sequence

starting from v'_e to v_e . Then based on the construction of G , we know that, after each environment vertex with decisions, the knowledge sets will be updated to a different one, that is,

$$\forall v_e \in V_e, \forall v_a \in \text{Succ}(v_e) : \text{Succ}(v_e) \geq 2 \Rightarrow \mathcal{K}(v_a) \neq \mathcal{K}(v_e), \quad (8)$$

where $\mathcal{K}(\cdot)$ denotes the knowledge set (third) component of a vertex in G . From ii), we know that, $\mathcal{K}(v'_e)$ is updated from

$\mathcal{K}(v_e)$; and from iii), we know that, $\mathcal{K}(v_e)$ is updated from $\mathcal{K}(v'_e)$. Since knowledge sets are ordered sets, to satisfy ii) and iii), we have $\mathcal{K}(v_e) = \mathcal{K}(v'_e)$. However, this contradicts with (8). The proof is thus completed. \square

With the above proposition, we draw the following corollary that states the relation between the sequences of G and the environment vertices with decisions.

Corollary 1. *Given the knowledge-based game arena G , if two sequences in G end with the same vertex, then they must visit the same environment vertices with decisions in the same order.*

Proof. By the knowledge updates defined in Definition 8, we know that such two sequences visit the same environment vertices with decisions. Since there is no cycles encompassing two environment vertices with decisions, the two sequences visit them in the same order. \square

4.2. Strategies and plays

Given a game arena G , we call a finite sequence of vertices $\pi = v_0v_1\cdots v_n \in V^*$ a *play* on G if $v_0 \in V_a$ is the initial vertex and $(v_i, v_{i+1}) \in E$, $\forall i = 0, \dots, n-1$; we denote by $\text{Play}(G)$ the set of all finite plays on G . We call π a *complete play* if $\text{last}(\pi) \in V_a$, where $\text{last}(\pi)$ denotes the last vertex in π . Then for a complete play $\pi = v_0v_1\cdots v_{2n} \in (V_aV_e)^*V_a$, where $v_{2i} = (x_i, q_i, \mathcal{K}_i)$, $i = 0, \dots, n$, it induces a path denoted by $\pi_{\text{path}} = x_0x_1\cdots x_n$ as well as a history

$$\pi_{\text{his}} = (x_0, o_{\mathcal{K}_0}(x_0))(x_1, o_{\mathcal{K}_1}(x_1))\cdots(x_n, o_{\mathcal{K}_n}(x_n)).$$

Note that, here, we have $\mathcal{K}_0 \subseteq \mathcal{K}_1 \subseteq \cdots \subseteq \mathcal{K}_n$. On the other hand, for any history $h = \kappa_0\kappa_1\cdots\kappa_n \in \text{Kw}^*$, there exists a unique complete play in G , denoted by π_h , such that its induced history is h .

Since the first two components of G are from the product of \mathcal{T} and \mathcal{A}_ϕ , for any complete play π , we have $L(\pi_{\text{path}}) \in \mathcal{L}_{\text{pref}}^\phi$ if and only if the second component of $\text{last}(\pi)$ is an accepting state in the DFA. Therefore, we define

$$V_F = \{(x_a, q_a, \mathcal{K}_a) \in V_a \mid q_a \in Q_F\}$$

as the set of accepting vertices representing the satisfaction of the scLTL task. Here we note that without loss of generality, we terminate the construction of the product system $\mathcal{P} = \mathcal{T} \otimes \mathcal{A}_\phi$ at the accepting states. That is, in the constructed \mathcal{P} and G , we only focus on the runs that satisfy the scLTL specification ϕ for the first time. Also, since only edges from V_e to V_a represent actual movements, we define a weight function for G as

$$w_G : V \times V \rightarrow \mathbb{R} \quad (9)$$

where for any $v_e = (x_e, q_e, \mathcal{K}_e, \hat{x}_e)$ and $v_a = (x_a, q_a, \mathcal{K}_a)$, we have $w_G(v_e, v_a) = w(x_e, x_a)$ and $w_G(v_a, v_e) = 0$. Then the cost of a play $\pi = v_0v_1\cdots v_n \in V^*$ is defined as $\text{cost}_G(\pi) = \sum_{i=0}^{n-1} w_G(v_i, v_{i+1})$.

A *strategy* for the agent-player is a function $\sigma_a : V^*V_a \rightarrow V_e \cup \{\text{stop}\}$ such that for any $\pi \in V^*V_a$, either $\langle \text{last}(\pi), \sigma_a(\pi) \rangle \in E$ or $\sigma_a(\pi) = \text{stop}$. Analogously, a strategy for the environment-player is a function $\sigma_e : V^*V_e \rightarrow V_a$ such that for any $\pi \in V^*V_e$, we have $\langle \text{last}(\pi), \sigma_e(\pi) \rangle \in E$. We denote by $\Sigma_a(G)$ and $\Sigma_e(G)$ the sets of all strategies for the agent and the environment, respectively. We say a strategy σ is *positional* if $\forall \pi, \pi' : \text{last}(\pi) = \text{last}(\pi') \Rightarrow \sigma(\pi) = \sigma(\pi')$. Given strategies $\sigma_a \in \Sigma_a(G)$ and $\sigma_e \in \Sigma_e(G)$, the *outcome play* π_{σ_a, σ_e} is the unique sequence $v_0v_1\cdots v_n \in V^*V_a$ such that

- $\forall i < n : v_i \in V_a \Rightarrow \sigma_a(v_0v_1\cdots v_i) = v_{i+1}$; and
- $\forall i < n : v_i \in V_e \Rightarrow \sigma_e(v_0v_1\cdots v_i) = v_{i+1}$; and
- $\sigma_a(v_0v_1\cdots v_n) = \text{stop}$.

It is important to note that the environment-player cannot play arbitrarily in the game arena, as the environment is fixed. In other words, the environment must commit to a specific successor-pattern it chooses at each unknown state when the game begins. This leads to the following definition.

Definition 15. (Strongly Positional Strategies) A strategy of the environment-player $\sigma_e \in \Sigma_e(G)$ is said to be *strongly positional* if: for any two plays $\pi, \pi' \in V^*V_e$ where $\sigma_e(\pi) = (x, q, \mathcal{K})$, $\sigma_e(\pi') = (x', q', \mathcal{K}')$, we have

- (i) $\text{last}(\pi) = \text{last}(\pi') \Rightarrow \sigma_e(\pi) = \sigma_e(\pi');$
- (ii) $x = x' \Rightarrow o_{\mathcal{K}}(x) = o_{\mathcal{K}'}(x).$

We denote by $\mathfrak{S}_e \subseteq \Sigma_e(G)$ the set of all strongly positional strategies for the environment player.

For the agent-player, we say $\sigma_a \in \Sigma_a(G)$ is a *winning strategy* if for any $\sigma_e \in \mathfrak{S}_e$, we have $\text{last}(\pi_{\sigma_a, \sigma_e}) \in V_F$. We denote by $\mathfrak{S}_a \subseteq \Sigma_a(G)$ the set of all winning strategies. Similarly to Definition 13, we can also define the *regret* of an agent-player's strategy $\sigma_a \in \mathfrak{S}_a$ in G by

$$\text{reg}_G(\sigma_a) = \max_{\sigma_e \in \mathfrak{S}_e} \left(\text{cost}_G(\pi_{\sigma_a, \sigma_e}) - \min_{\sigma'_a \in \mathfrak{S}_a} \text{cost}_G(\pi_{\sigma'_a, \sigma_e}) \right) \quad (10)$$

Essentially, an agent-player's strategy $\sigma_a \in \mathfrak{S}_a$ uniquely induces a corresponding strategy in \mathbb{T} , denoted by $\xi_{\sigma_a} \in \text{Stra}_\phi(\mathbb{T})$ as follows:

$$\forall h \in \text{Hist}^*(\mathbb{T}) : \xi_{\sigma_a}(h) = \widehat{X}(\sigma_a(\pi_h)), \quad (11)$$

where $\widehat{X}(\cdot)$ denotes the forth component of an environment vertex. Also, a strongly positional strategy of the environment-player $\sigma_e \in \mathfrak{S}_e$ essentially corresponds to a possible actual environment $T \in \mathbb{T}$ since it needs to specify an observation $o \in \Delta(x)$ for each unexplored x , and once x is explored, the observation is fixed based on the construction of G . Finally, since $\text{cost}_G(\cdot)$ is defined only according to its first component, for any play $\pi \in \text{Play}^*(G)$, we have $\text{cost}_G(\pi) = \text{cost}(\pi_{\text{path}})$. Therefore, we obtain the following result directly.

Proposition 2. Given the PK-WTS \mathbb{T} , scLTL task ϕ , and the knowledge-based game arena G , for any strategy $\zeta \in \text{Stra}_\phi(\mathbb{T})$, there exists a unique corresponding agent-player strategy $\sigma_a \in \mathfrak{S}_a$ such that $\text{reg}_{\mathbb{T}}(\zeta) = \text{reg}_G(\sigma_a)$.

Based on the above result, we know that, to solve Problem 1, it suffices to find an agent-player strategy $\sigma_a \in \mathfrak{S}_a$ in arena G that minimizes the regret defined in (10). In what follows, we propose an efficient algorithm to synthesize such a strategy.

To develop the efficient algorithm, we first show that a positional strategy is sufficient to solve the regret-minimizing problem, based on which a value iteration can be applied to synthesize the strategy.

Lemma 1. A positional strategy is sufficient to minimize the regret for the agent-player in game arena G , that is, there is a positional strategy $\sigma_a \in \mathfrak{S}_a$ such that

$$\forall \sigma'_a \in \mathfrak{S}_a : \text{reg}_G(\sigma_a) \leq \text{reg}_G(\sigma'_a)$$

Proof. Consider a winning strategy $\sigma_a^F \in \Sigma_a$ for the agent-player in G , which means that for any $\sigma_e \in \mathfrak{S}_e$, the outcome play $\pi_{\sigma_a^F, \sigma_e}$ satisfies $\text{last}(\pi_{\sigma_a^F, \sigma_e}) \in V_F$. Clearly, strategy σ_a^F needs at most *finite memory* since all plays $\pi_{\sigma_a^F, \sigma_e}, \forall \sigma_e \in \mathfrak{S}_e$ are finite. Now we construct a positional strategy $\sigma_a \in \mathfrak{S}_a$ based on σ_a^F such that

$$\forall \sigma_e \in \mathfrak{S}_e : \text{cost}_G(\pi_{\sigma_a, \sigma_e}) \leq \text{cost}_G(\pi_{\sigma_a^F, \sigma_e}). \quad (12)$$

The construction of strategy σ_a is as follows: for any environment strategy $\sigma_e \in \mathfrak{S}_e$ with $\pi_{\sigma_a^F, \sigma_e} = v_0 v_1 \cdots v_n$ being the outcome play and for all $v_k \in V_a$, we define

$$\forall i \geq k : v_i = v_k \Rightarrow \sigma_a(v_k) = v_{k+1}. \quad (13)$$

Note that strategy σ_a is well defined. By construction, we directly have

$$\text{last}(\pi_{\sigma_a, \sigma_e}) = \text{last}(\pi_{\sigma_a^F, \sigma_e}) \in V_F, \forall \sigma_e \in \mathfrak{S}_e \quad (14)$$

Thus σ_a is a winning strategy. Furthermore, for any $\sigma_e \in \mathfrak{S}_e$, the outcome play π_{σ_a, σ_e} only visits the states in $\pi_{\sigma_a^F, \sigma_e}$, since the environment-player plays only strongly positional strategies. Then (12) holds as we can obtain $\pi_{\sigma_a^F, \sigma_e}$ by removing all cycles in $\pi_{\sigma_a^F, \sigma_e}$ based on the above construction. Then it follows that

$$\forall \sigma_e \in \mathfrak{S}_e : \text{reg}_G(\sigma_a) \leq \text{reg}_G(\sigma_a^F). \quad (15)$$

Therefore, we have

$$\text{reg}_G(\sigma_a) \leq \text{reg}_G(\sigma_a^F). \quad (16)$$

That is, given any winning strategy for the agent-player, we can always find a positional winning strategy making (16) hold. The proof is thus completed. \square

5. Game-based synthesis algorithms

In this section, we present the solution of the regret-minimizing game on the knowledge-based game arena.

5.1. Regret-minimizing strategy synthesis

Recall that, given a knowledge set $\mathcal{K} \in \mathbb{KW}$, $\mathbb{T}_{\mathcal{K}}$ is the refined PK-WTS as defined in Definition 9. We define the optimistic response w.r.t. a knowledge set as follows.

Definition 16. (Optimistic Responses) Let \mathbb{T} be a PK-WTS and $T \in \mathbb{T}$ be an actual environment compatible to the possible world. The best response of the agent w.r.t. WTS T is defined as the minimum cost required to achieve the scLTL task in T , that is,

$$\text{br}(T) = \min\{\text{cost}(\rho) | \rho \in \text{Path}^*(T), \rho \models \phi\}. \quad (17)$$

Then, given a knowledge set $\mathcal{K} \in \mathbb{KW}$, the optimistic response of the agent w.r.t. knowledge set \mathcal{K} is defined as the minimum of the best responses among all WTSs compatible with the refined PK-WTS, that is,

$$\text{opr}(\mathcal{K}) = \min\{\text{br}(T) | T \in \mathbb{T}_{\mathcal{K}}\}. \quad (18)$$

With a slight abuse of notation, for each agent vertex $v_a = (x_a, q_a, \mathcal{K}_a)$ in G , we also define the optimistic response for vertex v_a as the optimistic response w.r.t. its knowledge set \mathcal{K}_a , that is, $\text{opr}(v_a) := \text{opr}(\mathcal{K}_a)$.

Note that, the optimistic response for a knowledge set can be easily computed by (weighted) shortest path search algorithms such as the standard Dijkstra's algorithm. Specifically, given a weighted transition system or a graph T , for any two states (vertices) x and x' , we denote by $\text{SP}_T(x, x')$ the weighted length of the shortest path from x to x' in T ; for a set of states (vertices) X' , we denote by $\text{SP}_T(x, X') = \min_{x' \in X'} \text{SP}_T(x, x')$. Then we have

$$\text{opr}(\mathcal{K}) = \text{SP}_{\mathcal{T}_{\mathcal{K}} \otimes \mathcal{A}_\phi}(q_0, Q_F), \quad (19)$$

where $\mathcal{T}_{\mathcal{K}}$ is the skeleton system of $\mathbb{T}_{\mathcal{K}}$.

The optimistic response can be utilized to compute the regret of an outcome play as follows. Suppose that the agent employs a winning strategy in an underlying environment, resulting in an outcome play $\pi = v_0 v_1 \cdots v_n$, where $v_a = (x_a, q_a, \mathcal{K}_a) \in V_F$. Along this executed path, the accumulated knowledge of the agent is \mathcal{K}_a . Hence, $\text{opr}(v_a)$ serves as a lower-bound estimate of the cost required to reach an accepting vertex in V_F (not necessarily v_a) in environments compatible with the current knowledge. Additionally, $\text{cost}_G(\pi)$ represents the actual cost incurred when following the current strategy. Therefore, the difference $\text{cost}_G(\pi) - \text{opr}(v_a)$ essentially quantifies the regret accrued along the outcome play π .

However, due to the possible presence of cycles, there is generally an infinite number of outcome plays leading to accepting vertices in G . Therefore, enumerating all such accepting outcome plays is not feasible. To tackle this challenge, our approach is to focus on the shortest paths, considering them as *critical paths*. We will demonstrate that incorporating such critical information is adequate for our purposes. Formally, for each accepting

vertex $v_a \in V_F$, let π_{v_0, v_a}^{sp} be a shortest path from v_0 to v_a in G w.r.t. cost function w_G . Note that, in general there may be multiple shortest paths v_0 to v_a ; here, we just choose an arbitrary one and we prove that which one to choose will not affect our final result. For each edge $e \in E$, we use notation $e \in \pi_{v_0, v_a}^{sp}$ to denote that edge e is on path π_{v_0, v_a}^{sp} . Then, we define

$$E_{SP} = \left\{ e \in E \mid \exists v_a \in V_F \text{ s.t. } e \in \pi_{v_0, v_a}^{sp} \right\}$$

as the set of *critical edge* that are involved in at least one shortest path from the initial vertex v_0 to an accepting vertex $v_a \in V_F$.

Based on the above notions, now we define a new weight function

$$\mu : E \rightarrow \mathbb{R} \quad (20)$$

that assigns each edge in the knowledge-based game arena G as follows:

- for any $\langle v_a, v_e \rangle \in V_a \times V_e$, we set $\mu(v_a, v_e) = 0$;
- for any $\langle v_e, v_a \rangle \in V_e \times V_a$, we have
 - if $\langle v_e, v_a \rangle \notin E_{SP}$, then we set $\mu(v_a, v_e) = \infty$;
 - if $\langle v_e, v_a \rangle \in E_{SP}$, then we have
 - (i) if $v_a \notin V_F$, we set $\mu(v_a, v_e) = 0$;
 - (ii) if $v_a \in V_F$, we set

$$\mu(v_e, v_a) = SP_G(v_0, v_a) - opr(v_a). \quad (21)$$

In the above definition of μ , we essentially assess the regret of a play when it terminates at a final state. It is important to note that instead of using the actual cost incurred during a play to evaluate regret, we utilize the shortest path weight from the initial state to the accepting state. Our subsequent analysis will show that to minimize regret, the agent must follow a shortest path from the initial state to an accepting state in the knowledge-based game arena. As a result, we assign infinite weight to those edges that are either not part of the shortest path to the accepting states or do not lead to an accepting state at all. However, relying solely on a shortest path search is still not sufficient, as the environment makes choices at vertices where new unknown states are explored. Therefore, we further incorporate a min-max game to capture the worst-case regret of the agent's strategy across all possible environments. This leads to our main synthesis algorithm outlined in Algorithm 1.

Algorithm 1: Regret-Minimizing LTL Planning

```

Input: PK-WTS  $\mathbb{T}$  and scLTL  $\phi$ 
Output: Optimal plan  $\xi^*$  and minimized  $\text{reg}_{\mathbb{T}}(\xi^*)$ 
1 Construct skeleton-WTS  $\mathcal{T}$  and DFA  $\mathcal{A}_\phi$  from  $\phi$ ;
2 Construct knowledge-based game arena  $G$ ;
3 Define weight function  $\mu : E \rightarrow \mathbb{R}$  according to
Eq. (20);
4  $\sigma_a^*, \text{reg}^* \leftarrow \text{SolveMinMax}(G, \mu)$ ;
5 Obtain strategy  $\xi^*$  from  $\sigma_a^*$ ;
6 if  $\text{reg}^* < \infty$  then
7   return optimal plan  $\xi^*$  with regret value  $\text{reg}^*$ 
8 else
9   return “no solution exists”

10 procedure  $\text{SolveMinMax}(G, \mu)$ 
// Initialization
11 foreach  $v \in V$  do
12   if  $v \in V_F$  then
13      $W^{(0)}(v) \leftarrow 0$  and  $\sigma_a^{(0)}(v) \leftarrow \text{STOP}$ 
14   else
15      $W^{(0)}(v) \leftarrow \infty$ 
// Value Iteration
16 repeat
17   foreach  $v_e \in V_e$  do
18      $W^{(k+1)}(v_e) \leftarrow \max_{v_a \in \text{Succ}(v_e)} (W^{(k)}(v_a) + \mu(v_e, v_a))$ 
19   foreach  $v_a \in V_a$  do
20     if  $v_a \in V_F$  then
21        $W^{(k+1)}(v_a) \leftarrow 0$  and  $\sigma_a^{(k+1)}(v_a) \leftarrow \text{STOP}$ 
22     else
23        $W^{(k+1)}(v_a) \leftarrow \min_{v_e \in \text{Succ}(v_a)} (W^{(k)}(v_e) + \mu(v_a, v_e))$ 
        $\sigma_a^{(k+1)}(v_a) \leftarrow \arg \min_{v_e \in \text{Succ}(v_a)} (W^{(k)}(v_e) + \mu(v_a, v_e))$ 
24    $k \leftarrow k + 1$ 
25 until  $\forall v \in V : W^{(k+1)}(v) = W^{(k)}(v)$ ;
26 return  $\sigma_a^{(k)}, W^{(k)}(v_0)$ 

```

Specifically, Algorithm 1 works as follows. First, we construct the skeleton WTS \mathcal{T} and the DFA \mathcal{A}_ϕ (line 1). Next, we create the knowledge-based game graph along with the weight function μ (lines 2 and 3). Then, we engage in solving a min-max game over G to attain a minimal weight as per μ . This process is detailed by the procedure $\text{SolveMinMax}(G, \mu)$ in lines 7–23. In this min-max game, the player aims to maximize the weight at each environment vertex while striving to minimize it at each agent vertex. It is worth noting that min-max games typically require a termination condition. However, in our

specific knowledge-based game arena, by Proposition 1, we know that, there are no cycles encompassing two distinct environment vertices with decisions (i.e., environment vertices with at least two successors). This structural property ensures that condition line 22 can be fulfilled within $|V|$ iterations. Moreover, this structural property also guarantee that considering the shortest path between distinct environment vertices with decisions is adequate, as the environment has no choice within each cycle in G . Finally, we output the optimal strategy σ_a^* for the min-max game, which uniquely induces a planning strategy ζ^* as the optimal solution.

Example 5. (Running Example Cont.) Finally, we present the solution to the running example, whose knowledge-based game arena G is depicted in Figure 4. In this arena, there are three accepting vertices in V_F . For each of these vertices $v_a \in V_F$, we determine the shortest path from v_0 to v_a and define the weight function μ accordingly, shown as the numbers within parentheses in Figure 4. For instance, consider the accepting vertex $v_a = (x_5, q_1, \mathcal{K}_1)$. The shortest path from v_0 to v_a has a length of 13. Given that $\mathcal{K}_1 = \langle (x_2, x_1) \rangle$, the optimistic response is $opr(\mathcal{K}_1) = 11$. Therefore, the weight assigned to the incoming edge for $v_a = (x_5, q_1, \mathcal{K}_1)$ is calculated as $13 - 11 = 2$.

In the min-max game, the agent essentially faces a decision at state $(x_1, q_0, \mathcal{K}_0)$: whether to explore state x_2 or not. If it decides to explore state x_2 , the worst-case regret it can attain is $\max\{2, 0\} = 2$. On the other hand, if it chooses not to explore, the worst-case regret becomes 8, which is the weight assigned to the incoming edge for accepting vertex $(x_5, q_1, \mathcal{K}_0)$. Consequently, the agent will decide to explore state x_2 , leading to the regret-optimal strategy as depicted by the red parts in Figure 4.

Remark 3. Let us discuss the complexity of the proposed algorithm. Since the iteration of the procedure $SolveMinMax(G, \mu)$ is directly conducted on the original game arena G , the space complexity of Algorithm 1 is exactly $|V|$. Furthermore, since the game graph G is acyclic when removing those edges with infinite weight, the value iteration of $SolveMinMax(G, \mu)$ can be finished in at most $|V|$ steps with time complexity $O(|V|^2)$ (Brihaye et al., 2017). Therefore, the strategy with minimal regret can be computed in the polynomial time in the size of the arena G , which is also is polynomial in the sizes of the PK-WTS and DFA, but exponential in the number of unknown states as we have discussed in Remark 2.

Remark 4. It is worth noting that, in Filiot et al. (2010), an algorithm has been proposed for synthesizing regret-minimizing strategies for reachability objectives. As discussed in Zhao et al. (2023), by some slight modifications, this algorithm can be applied to the knowledge-based graph with cost function $cost_G$ to obtain a regret-optimal strategy. However, their algorithm is designed for general two-player games without considering the

structural properties present in the partially-known environment setting. Specifically, the approach in Filiot et al. (2010) requires “unfolding” the game arena based on the ratio between the maximal and minimal weights of the edges in the arena. This process can lead to extremely high space complexity, especially when the ratio value is very large and the game arena has multiple cycles involved. On the other hand, our algorithm leverages the structural properties inherent in the problem setting based on the knowledge-based game arena. Once ordered knowledge is incorporated in the game arena, the need for unfolding is eliminated, resulting in a more efficient algorithm.

5.2. Structural properties of the game arena

Before building the correctness of our algorithm, we present some important and necessary results showing the properties of the knowledge-based game arena G .

We first show that the optimistic responses in Definition 16 are sufficient to compute the regret for a strategy. For convenience, we define the regret of an agent strategy $\sigma_a \in \mathfrak{S}_a$ against an environment player $\sigma_e \in \mathfrak{S}_e$ as:

$$\text{reg}_G^{\sigma_e}(\sigma_a) = \text{cost}_G(\pi_{\sigma_a, \sigma_e}) - \min_{\sigma'_a \in \mathfrak{S}_a} \text{cost}_G(\pi_{\sigma'_a, \sigma_e}). \quad (22)$$

Then, based on the definition in (10), we have $\text{reg}_G(\sigma_a) = \max_{\sigma_e \in \mathfrak{S}_e} \text{reg}_G^{\sigma_e}(\sigma_a)$.

Lemma 2. Given two strategies $\sigma_a \in \mathfrak{S}_a$ and $\sigma_e \in \mathfrak{S}_e$, the regret of strategy σ_a against strategy σ_e satisfies

$$\text{reg}_G^{\sigma_e}(\sigma_a) \geq \text{cost}_G(\pi_{\sigma_a, \sigma_e}) - opr(\text{last}(\pi_{\sigma_a, \sigma_e})). \quad (23)$$

In particular, there is an environment strategy $\sigma'_e \in \mathfrak{S}_e$ such that $\text{reg}_G^{\sigma'_e}(\sigma_a) = \text{cost}_G(\pi_{\sigma_a, \sigma'_e}) - opr(\text{last}(\pi_{\sigma_a, \sigma'_e}))$.

Proof. For the above two strategies σ_a and σ_e , we write $\text{last}(\pi_{\sigma_a, \sigma_e}) = (x, q, \mathcal{K})$. Let us consider the refined PK-WTS w.r.t. \mathcal{K} , that is, $\mathbb{T}_{\mathcal{K}}$ and let $\mathbb{T}_{\mathcal{K}} = \{T_1, T_2, \dots, T_m\}$. Then for each environment T_i , by the construction of G , there is a corresponding environment strategy $\sigma_e^i \in \mathfrak{S}_e$. Clearly, $\sigma_e \in \{\sigma_e^1, \dots, \sigma_e^m\}$. By the construction of G , we have

$$\pi_{\sigma_a, \sigma_e^i} = \pi_{\sigma_a, \sigma_e}, \forall i = 1, \dots, m. \quad (24)$$

Then it follows that $\text{cost}_G(\pi_{\sigma_a, \sigma_e^i}) = \text{cost}_G(\pi_{\sigma_a, \sigma_e})$. By equation (10), we have

$$\text{reg}_G^{\sigma_e^i}(\sigma_a) = \text{cost}_G(\pi_{\sigma_a, \sigma_e}) - \min_{\sigma'_a \in \mathfrak{S}_a} \text{cost}_G(\pi_{\sigma'_a, \sigma_e^i}) \quad (25)$$

On the other hand, based on Definition 16, we have

$$\begin{aligned} opr(\mathcal{K}) &= \min\{\text{br}(T_i) \mid i = 1, \dots, m\} \\ &= \min_{i \in \{1, \dots, m\}} \min_{\sigma'_a \in \mathfrak{S}_a} \text{cost}_G(\pi_{\sigma'_a, \sigma_e^i}). \end{aligned} \quad (26)$$

With $opr(\mathcal{K}) = opr(\text{last}(\pi_{\sigma_a, \sigma_e}))$ and $\sigma_e \in \{\sigma_e^1, \dots, \sigma_e^m\}$, it follows that

$$\text{reg}_G^{\sigma_e}(\sigma_a) \geq \text{cost}_G(\pi_{\sigma_a, \sigma_e}) - \text{opr}(\text{last}(\pi_{\sigma_a, \sigma_e})) \quad (27)$$

Let $\sigma'_e = \sigma_e^j, j = \arg \min_{i \in \{1, \dots, m\}} \min_{\sigma'_a \in \mathcal{S}_a} \text{cost}_G(\pi_{\sigma'_a, \sigma'_e})$. We have $\text{reg}_G^{\sigma'_e}(\sigma_a) = \text{cost}_G(\pi_{\sigma_a, \sigma'_e}) - \text{opr}(\text{last}(\pi_{\sigma_a, \sigma'_e}))$. The proof is thus completed. \square

With the above result, we can use the optimistic response to compute the regret for any agent strategy $\sigma_a \in \mathcal{S}_a$ by $\text{reg}_G(\sigma_a) = \max_{\sigma_e \in \mathcal{S}_e} (\text{cost}(\pi_{\sigma_a, \sigma_e}) - \text{opr}(\text{last}(\pi_{\sigma_a, \sigma_e})))$.

Recall that, in the definition of weight function μ , it assigns ∞ weight to edges that are not involved in any shortest path. Next, we present the result stating that it is sufficient to only consider strategies that result in shortest outcome plays.

Proposition 3. For any winning strategy $\sigma_a \in \mathcal{S}_a$, there is another positional winning strategy $\sigma'_a \in \mathcal{S}_a$ such that for any environment strategy $\sigma_e \in \mathcal{S}_e$, we have

- (i) $\text{last}(\pi_{\sigma'_a, \sigma_e}) = \text{last}(\pi_{\sigma_a, \sigma_e})$;
- (ii) $\text{cost}_G(\pi_{\sigma'_a, \sigma_e}) = \text{SP}_G(v_0, \text{last}(\pi_{\sigma_a, \sigma_e}))$.

Proof. Let σ_a be a winning strategy. We consider each possible environment strategy $\sigma_e \in \mathcal{S}_e$ and the resulting outcome play π_{σ_a, σ_e} . For each π_{σ_a, σ_e} , we denote by $\mathcal{V}_{\pi_{\sigma_a, \sigma_e}}$ the ordered set of environment vertices with decisions that π_{σ_a, σ_e} visits in sequence. For convenience, we denote $\mathcal{V}_{\pi_{\sigma_a, \sigma_e}} := \langle \tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_m \rangle$, where m is the size of set $\mathcal{V}_{\pi_{\sigma_a, \sigma_e}}$. Based on the construction of G , we know that, along π_{σ_a, σ_e} , the knowledge set will be updated to a new one only at the vertices in $\mathcal{V}_{\pi_{\sigma_a, \sigma_e}}$. That is, denoting

$$\pi_{\sigma_a, \sigma_e} = v_{00} v_{01} \cdots \tilde{v}_1 v_{11} v_{12} \cdots \tilde{v}_2 v_{21} \cdots \tilde{v}_m v_{m1} v_{m2} \cdots v_n, \quad (28)$$

we have

- i) $\mathcal{K}(\tilde{v}_1) \subset \mathcal{K}(\tilde{v}_2) \subset \cdots \subset \mathcal{K}(\tilde{v}_m)$;
- ii) $\mathcal{K}(v_{i1}) = \mathcal{K}(v_{i2}) = \cdots = \mathcal{K}(\tilde{v}_{i+1})$ for any $0 \leq i \leq m$,

where for convenience we denote $v_{00} = v_0$ and $\tilde{v}_{m+1} = v_n$.

Now, we construct a positional strategy $\sigma'_a : V_a \rightarrow V_e \cup \{\text{stop}\}$ from the given strategy σ_a as follows. For each environment strategy σ_e with the corresponding play π_{σ_a, σ_e} in the form of (28), we search the shortest path from v_{i1} to \tilde{v}_{i+1} as $\pi_{v_{i1}, \tilde{v}_{i+1}}^{sp} \sim \cdots \sim \pi_{v_{m1}, v_n}^{sp}$ for each $i = 0, 1, \dots, m-1$. Since the environment vertex in sequence $v_{i1} \cdots \tilde{v}_{i+1}$ has only one successors by the definition of $\mathcal{V}_{\pi_{\sigma_a, \sigma_e}}$, we can define σ'_a by: for each agent vertex in $v_{i1} \cdots \tilde{v}_{i+1}$, its successor is exactly the successor in $v_{i1} \cdots \tilde{v}_{i+1}$; and $\sigma'_a(v_n) = \text{stop}$, that is, strategy σ'_a is defined such that

$$\pi_{\sigma'_a, \sigma_e} = \pi_{v_0, v_1}^{sp} \sim \pi_{v_{11}, v_2}^{sp} \sim \cdots \sim \pi_{v_{m1}, v_n}^{sp}. \quad (29)$$

Note that by the above construction of σ'_a , this strategy may not be defined on every agent vertex $v_a \in V_a$. However, in what follows, we show that the above construction of σ'_a is sufficient to be a well-defined positional strategy. That is,

for any two environment strategies $\sigma_e, \sigma'_e \in \mathcal{S}_e$, if they have a common vertex v_a , then vertex v_a in $\pi_{\sigma'_a, \sigma_e}$ and $\pi_{\sigma'_a, \sigma'_e}$ has the same successor. We prove this by contradiction. Suppose that there are two environment strategies $\sigma_e, \sigma'_e \in \mathcal{S}_e$ such that they have a common vertex v_a but two different successor v_e, v'_e in $\pi_{\sigma'_a, \sigma_e}$ and $\pi_{\sigma'_a, \sigma'_e}$, respectively. By the construction of σ'_a , we know that successor v'_e is defined based on the outcome play $\pi_{\sigma_a, \sigma'_e}$, which can be still write in the following form of

$$\pi_{\sigma_a, \sigma'_e} = v'_{00} v'_{01} \cdots \tilde{v}'_1 v'_{11} v'_{12} \cdots \tilde{v}'_2 v'_{21} \cdots \tilde{v}'_m v'_{m1} v'_{m2} \cdots v'_n. \quad (30)$$

Let v_{ij} and v'_{kl} be the agent vertex v_a in plays (28) and (30), respectively, that is, $v_{ij} = v'_{kl} = v_a$. Correspondingly, we have $v_{i,j+1} = v_e$ and $v'_{k,l+1} = v'_e$. By Corollary 1, we have $i = k$ and $\tilde{v}_\tau = \tilde{v}'_\tau$ for any $\tau = 1, \dots, i$. Specifically, we have $v_{i1} = v'_{k1}$. Since $v_{i,j+1} \neq v_{k,l+1}$, based on the construction of σ'_a , it holds that $\tilde{v}_{i+1} \neq \tilde{v}'_{i+1}$ due to the fact that if $\tilde{v}_{i+1} = \tilde{v}'_{i+1}$, then it would hold that $v_{i,j+1} = v_{k,l+1}$. Then, based on the construction of σ'_a , we know that, given the strategy σ_a , for two different environment strategies $\sigma_e, \sigma'_e \in \mathcal{S}_e$, the outcome plays π_{σ_a, σ_e} and $\pi_{\sigma_a, \sigma'_e}$ visit the same environment vertex with decisions $\tilde{v}_1, \dots, \tilde{v}_i$ and v_{i1} but two different environment vertices \tilde{v}_{i+1} and \tilde{v}'_{i+1} . Since strategy σ_a is a well-defined strategy, each agent vertex in both sequence $v_{i1} \cdots \tilde{v}_{i+1}$ and sequence $v'_{i1} \cdots \tilde{v}'_{i+1}$ has only one successor. Therefore, to visit two different environment vertices \tilde{v}_{i+1} and \tilde{v}'_{i+1} from the same agent vertex v_{i1} , there should be at least one environment vertex in $v_{i1} \cdots \tilde{v}_{i+1}$ and $v'_{i1} \cdots \tilde{v}'_{i+1}$ that has more than one successor in G . However, this contradicts with the fact that there is no environment vertex with decisions between \tilde{v}_i and \tilde{v}_{i+1} , and similarly between \tilde{v}_i and \tilde{v}'_{i+1} . Therefore, strategy σ'_a is a well-defined strategy.

Next, we argue strategy σ'_a satisfies that $\text{last}(\pi_{\sigma'_a, \sigma_e}) = \text{last}(\pi_{\sigma_a, \sigma_e})$ for any $\sigma_e \in \mathcal{S}_e$. This is obvious since we have $\text{last}(\pi_{\sigma'_a, \sigma_e}) = \text{last}(\pi_{\sigma_a, \sigma_e}) = v_n$ in (28) and (29).

Finally, we prove the constructed strategy σ'_a satisfies $\text{cost}_G(\pi_{\sigma'_a, \sigma_e}) = \text{SP}_G(v_0, \text{last}(\pi_{\sigma_a, \sigma_e}))$ for any $\sigma_e \in \mathcal{S}_e$. Based on Corollary 1, for any play π' satisfying $\text{last}(\pi') = \text{last}(\pi_{\sigma_a, \sigma_e})$, it visits the same environment vertices with decisions $\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_m$ in the same order. By the construction (29), we know that play $\pi_{\sigma'_a, \sigma_e}$ captures the shortest paths between \tilde{v}_i and \tilde{v}_{i+1} for each $i = 1, \dots, m$. Therefore, we have $\text{cost}_G(\pi_{\sigma'_a, \sigma_e}) = \text{SP}_G(v_0, \text{last}(\pi_{\sigma_a, \sigma_e}))$. The proof is thus completed. \square

Finally, we show that different actual environments that agree on the same run must result in the same knowledge-set in the constructed game arena.

Lemma 3. Let $\xi \in \text{Stra}_\phi(\mathbb{T})$ be a strategy and $T, T' \in \mathbb{T}$ be two possible actual environments. We denote by σ_ξ be the unique strategy of the agent-player in G corresponding to ξ , and by σ_T and $\sigma_{T'}$ be the unique strategies of the environment-player in G corresponding to T and T' , respectively. Then we have

$$\rho_{\xi}^T = \rho_{\xi}^{T'} \Leftrightarrow T' \in \mathbb{T}_{\mathcal{K}(\text{last}(\pi_{\sigma_{\xi}, \sigma_T}))}. \quad (31)$$

Proof. (\Rightarrow) Since $\rho_{\xi}^T = \rho_{\xi}^{T'}$, by the construction of G , we know that $\pi_{\sigma_{\xi}, \sigma_T} = \pi_{\sigma_{\xi}, \sigma_{T'}}$. Accordingly, we have $\mathcal{K}(\text{last}(\pi_{\sigma_{\xi}, \sigma_T})) = \mathcal{K}(\text{last}(\pi_{\sigma_{\xi}, \sigma_{T'}}))$. By Definition 9, we have $T' \in \mathbb{T}_{\mathcal{K}(\text{last}(\pi_{\sigma_{\xi}, \sigma_{T'}}))} = \mathbb{T}_{\mathcal{K}(\text{last}(\pi_{\sigma_{\xi}, \sigma_T}))}$.

(\Leftarrow) By contraposition. Suppose that $\rho_{\xi}^T \neq \rho_{\xi}^{T'}$ and let x be the last state in their common prefix. Since ξ is the common strategy, we know that $\delta_T(x) \neq \delta_{T'}(x)$. Therefore, $(x, \delta_{T'}(x)) \notin \mathcal{K}(\text{last}(\pi_{\sigma_{\xi}, \sigma_T}))$, which implies that $T' \notin \mathbb{T}_{\mathcal{K}(\text{last}(\pi_{\sigma_{\xi}, \sigma_T}))}$. \square

5.3. Correctness of the proposed algorithm

Now, we formally prove the correctness of our algorithm. Specifically, we show that our algorithm is

- (i) **sound**: in the sense that σ_a^* is a winning strategy for the agent-player if $\text{reg}^* < \infty$;
- (ii) **complete**: in the sense that the agent-player has no winning strategy if $\text{reg}^* = \infty$;
- (iii) **optimal**: in the sense that for any winning strategy $\sigma_a \in \mathfrak{S}_a$, we have $\text{reg}_G(\sigma_a^*) \leq \text{reg}_G(\sigma_a)$.

First, we show the soundness of our algorithm.

Proposition 4. Let $(\sigma_a^*, \text{reg}^*)$ be the solution returned by procedure $\text{SolveMinMax}(G, \mu)$. If $\text{reg}^* < \infty$, then strategy σ_a^* is a winning strategy. Moreover, for any environment strategy $\sigma_e \in \mathfrak{S}_e$, we have

$$\text{cost}_G(\pi_{\sigma_a^*, \sigma_e}) = \text{SP}_G(v_0, \text{last}(\pi_{\sigma_a^*, \sigma_e})). \quad (32)$$

Proof. By the definition of μ , it holds that, for any $\pi \in \text{Play}(G)$ such that $\text{last}(\pi) \in V_F$, we have

$$\text{cost}_G(\pi) < \infty \Rightarrow \text{cost}_G(\pi) = \text{SP}_G(v_0, \text{last}(\pi))$$

Now, we show that the returned σ_a^* is a winning strategy if $\text{reg}^* < \infty$. By Brihaye et al. (2017, Proposition 5), strategy σ_a^* produced by the value iteration procedure $\text{SolveMinMax}(G, \mu)$ satisfies

$$\begin{aligned} \text{reg}^* &= \max_{\sigma_e \in \Sigma_e(G)} \text{cost}_G^{\mu}(\pi_{\sigma_a^*, \sigma_e}) \\ &= \min_{\sigma_a \in \Sigma_a(G)} \max_{\sigma_e \in \Sigma_e(G)} \text{cost}_G^{\mu}(\pi_{\sigma_a, \sigma_e}) < \infty \end{aligned} \quad (33)$$

Since $\mathfrak{S}_e \subseteq \Sigma_e(G)$, we know that, for any $\sigma_e \in \mathfrak{S}_e$, we have $\text{cost}_G^{\mu}(\pi_{\sigma_a^*, \sigma_e}) < \infty$, which means that σ_a^* is a winning strategy. Based on Proposition 3 and the definition of weight function μ , we know that

$$\text{cost}_G(\pi_{\sigma_a^*, \sigma_e}) = \text{SP}_G(v_0, \text{last}(\pi_{\sigma_a^*, \sigma_e}))$$

The proof is thus completed. \square

Note that, in equation (33), the value computed by procedure SolveMinMax is $\text{reg}^* = \max_{\sigma_e \in \Sigma_e(G)} \text{cost}_G^{\mu}(\pi_{\sigma_a^*, \sigma_e})$. According to the definition, reg^* is not necessarily the regret of synthesized strategy σ_a^* as the environment player can only take strategy in $\mathfrak{S}_e \subseteq \Sigma_e(G)$. Next, we show that the computed value reg^* provides a lower-bound for the regret values of all winning strategies.

Lemma 4. For any winning strategy $\sigma_a \in \mathfrak{S}_a$, we have $\text{reg}_G(\sigma_a) \geq \text{reg}^*$.

Proof. Let $\sigma_a^* = \arg \max_{\sigma_e \in \Sigma_e(G)} \text{cost}_G^{\mu}(\pi_{\sigma_a^*, \sigma_e})$ be the optimal strategy of the environment-player against σ_a^* . For any $\sigma_a \in \mathfrak{S}_a$, we define

$$T(\sigma_a) = \arg \min_{T' \in \mathbb{T}_{\mathcal{K}(\text{last}(\pi_{\sigma_a, \sigma_a^*}))}} \text{cost}(\rho_{\xi_{\sigma_a}}^{T'}) \quad (34)$$

as the actual environment that agrees on the knowledge-set explored by σ_a and σ_a^* and with minimal cost. Let $\sigma_{T(\sigma_a)}$ be the strategy of the environment-player corresponding to environment $T(\sigma_a)$. Then, we claim that

$$\text{cost}_G(\pi_{\sigma_a, \sigma_{T(\sigma_a)}}) - \min_{\sigma'_a \in \mathfrak{S}_a} \text{cost}_G(\pi_{\sigma'_a, \sigma_{T(\sigma_a)}}) \geq \text{reg}^*. \quad (35)$$

To see this, suppose that there exists $\sigma_a^3 \in \mathfrak{S}_a$ such that

$$\text{cost}_G(\pi_{\sigma_a^3, \sigma_{T(\sigma_a^3)}}) - \min_{\sigma'_a \in \mathfrak{S}_a} \text{cost}_G(\pi_{\sigma'_a, \sigma_{T(\sigma_a^3)}}) < \text{reg}^*. \quad (36)$$

From (33), we know that

$$\text{cost}_G^{\mu}(\pi_{\sigma_a^3, \sigma_a^*}) \geq \text{cost}_G^{\mu}(\pi_{\sigma_a^*, \sigma_a^*}) = \text{reg}^*. \quad (37)$$

Then it follows that

$$\text{cost}_G(\pi_{\sigma_a^3, \sigma_{T(\sigma_a^3)}}) - \min_{\sigma'_a \in \mathfrak{S}_a} \text{cost}_G(\pi_{\sigma'_a, \sigma_{T(\sigma_a^3)}}) < \text{cost}_G^{\mu}(\pi_{\sigma_a^3, \sigma_a^*}). \quad (38)$$

Based on the definition of weight function μ , we have

$$\text{cost}_G^{\mu}(\pi_{\sigma_a^3, \sigma_a^*}) \leq \text{cost}_G(\pi_{\sigma_a^3, \sigma_a^*}) - \text{opr}(\text{last}(\pi_{\sigma_a^3, \sigma_a^*})). \quad (39)$$

By the construction of G , given play $\pi_{\sigma_a^3, \sigma_a^*}$ with the corresponding strategy being $\xi_{\sigma_a^3}$, there must exist an environment $T' \in \mathbb{T}$ such that $\rho_{\xi_{\sigma_a^3}}^{T'}$ and $\pi_{\sigma_a^3, \sigma_a^*}$ satisfy the one-to-one correspondence. Therefore, we have $T' \in \mathbb{T}_{\mathcal{K}(\text{last}(\pi_{\sigma_a^3, \sigma_a^*}))}$.

Based on Lemma 3, we know that $\rho_{\xi_{\sigma_a^3}}^{T(\sigma_a^3)} = \rho_{\xi_{\sigma_a^3}}^{T'}$. This further implies that $\pi_{\sigma_a^3, \sigma_{T(\sigma_a^3)}} = \pi_{\sigma_a^3, \sigma_a^*}$, and thus

$$\text{cost}_G(\pi_{\sigma_a^3, \sigma_{T(\sigma_a^3)}}) = \text{cost}_G(\pi_{\sigma_a^3, \sigma_a^*}) \quad (40)$$

Combining (38), (39), and (40), we have

$$\min_{\sigma'_a \in \mathfrak{S}_a} \text{cost}_G(\pi_{\sigma'_a, \sigma_{T(\sigma_a^3)}}) > \text{opr}(\text{last}(\pi_{\sigma_a^3, \sigma_{T(\sigma_a^3)}})) \quad (41)$$

However, based on the definition of $T(\sigma_a^3)$ in (34) and the optimistic response in Definition 16, we know that

$$\begin{aligned} \text{opr}(\text{last}(\pi_{\sigma_a^3, \sigma_{T(\sigma_a^3)}})) &= \min_{\zeta' \in \text{Stra}_\phi(\mathbb{T})} \text{cost}(\rho_{\zeta'}^{T(\sigma_a^3)}) \\ &= \min_{\sigma'_a \in \mathcal{S}_a} \text{cost}_G(\sigma'_a, \sigma_{T(\sigma_a^3)}) \end{aligned} \quad (42)$$

which contradicts with (41). Therefore, our claim in equation (35) holds. Then, for any agent strategy $\sigma_a \in \mathcal{S}_a$, there exists an environment strategy $\sigma_e \in \mathcal{S}_e$ such that

$$\text{cost}_G(\pi_{\sigma_a, \sigma_e}) - \min_{\sigma'_a \in \mathcal{S}_a} \text{cost}_G(\pi_{\sigma'_a, \sigma_e}) \geq \text{reg}^\star, \quad (43)$$

which implies that $\text{reg}_G(\sigma_a) \geq \text{reg}^\star$ for any $\sigma_a \in \mathcal{S}_a$. \square

Based on the above lemma, we are now ready to prove the completeness of our algorithm.

Proposition 5. Let $(\sigma_a^\star, \text{reg}^\star)$ be the solution returned by procedure $\text{SolveMinMax}(G, \mu)$. If $\text{reg}^\star = \infty$, then the agent-player has no winning strategy.

Proof. By contraposition. Suppose that the agent-player has a winning strategy $\sigma_a \in \mathcal{S}_a$. Since $\forall \sigma_e \in \mathcal{S}_e : \text{last}(\pi_{\sigma_a, \sigma_e}) \in V_F$, we have $\text{reg}_G(\sigma_a) < \infty$. By Lemma 4, we further have $\text{reg}^\star \leq \text{reg}_G(\sigma_a)$, which means that $\text{reg}^\star < \infty$. \square

Finally, we establish the optimality of our algorithm, which completes the correctness proof.

Theorem 1. Let $(\sigma_a^\star, \text{reg}^\star)$ be the solution returned by procedure $\text{SolveMinMax}(G, \mu)$. Strategy σ_a^\star minimizes the regret of the agent-player in arena G and reg^\star is indeed the regret value of strategy σ_a^\star . Formally, we have

$$\text{reg}^\star = \text{reg}_G(\sigma_a^\star) \leq \text{reg}_G(\sigma_a), \forall \sigma_a \in \mathcal{S}_a. \quad (44)$$

Proof. First, we show that $\text{reg}_G(\sigma_a^\star) \leq \text{reg}^\star$. By Proposition 4, we know that, the strategy σ_a^\star satisfies that for any $\sigma_e \in \mathcal{S}_e$, we have

$$\text{cost}_G(\pi_{\sigma_a^\star, \sigma_e}) = \text{SP}_G(v_0, \text{last}(\pi_{\sigma_a^\star, \sigma_e})) \quad (45)$$

According to Lemma 2, we have

$$\begin{aligned} \text{reg}_G(\sigma_a^\star) &= \max_{\sigma_e \in \mathcal{S}_e} (\text{cost}_G(\pi_{\sigma_a^\star, \sigma_e}) - \text{opr}(\text{last}(\pi_{\sigma_a^\star, \sigma_e}))) \\ &= \max_{\sigma_e \in \mathcal{S}_e} (\text{SP}_G(v_0, \text{last}(\pi_{\sigma_a^\star, \sigma_e})) - \text{opr}(\text{last}(\pi_{\sigma_a^\star, \sigma_e}))) \\ &= \max_{\sigma_e \in \mathcal{S}_e} \text{cost}_G^\mu(\pi_{\sigma_a^\star, \sigma_e}). \end{aligned} \quad (46)$$

From (33), we know that

$$\text{reg}^\star = \max_{\sigma_e \in \Sigma_e(G)} \text{cost}_G^\mu(\pi_{\sigma_a^\star, \sigma_e}). \quad (47)$$

Since $\mathcal{S}_e \subseteq \Sigma_e(G)$, it naturally holds that

$$\text{reg}_G(\sigma_a^\star) \leq \text{reg}^\star. \quad (48)$$

By Lemma 4, we have $\text{reg}_G(\sigma_a) \geq \text{reg}^\star$ for any $\sigma_a \in \mathcal{S}_a$. Therefore, $\text{reg}_G(\sigma_a^\star) = \text{reg}^\star$, which completes the proof. \square

5.4. Comparison with other strategies

In addition to the regret-based strategy synthesized in this section, in the non-stochastic setting, the agent may also choose to adopt the following two commonly used strategies:

- **Worst-Case Strategies:** The agent makes decisions conservatively to minimize the cost function as defined in equation (5). This strategy can be synthesized by solving a min-max game over the knowledge-based game arena G with respect to the original weight function w_G as defined in equation (9).
- **Best-Case Strategies:** The agent makes decisions optimistically by assuming that all possible transitions exist based on its current knowledge. Specifically, at each instant, the agent maintains the knowledge set \mathcal{K} and solves a shortest path search problem in the skeleton system of the refined PK-WTS \mathcal{T}_K . The planned path is updated once a new unknown state is explored in the optimistic path.

Note that when probabilistic information about those unknown transitions is available, one can achieve a quantitative trade-off between the worst-case and best-case strategies in the context of MDPs. Essentially, the objective of the proposed regret-based strategy is to achieve a reasonable trade-off between the worst-case and best-case strategies in the non-stochastic setting assuming no such probabilistic information is available. In the next section, we will present experimental results to justify that regret indeed achieves this trade-off.

6. Case study and numerical experiments

In this section, we present both simulation and experimental results to show the effectiveness of the regret-based strategy when exploring the unknown regions in the partially-known environments.

6.1. Case study 1: A team of firefighting robots

In this subsection, we present a case study to illustrate the proposed framework. We consider a team of firefighting robots consisting of a ground robot and a UAV moving in a district as shown in Figure 5(a). The entire firefighting mission in this district is undertaken by the collaboration of the UAV and the ground robot as follows:

- Initially, the district map is completely unknown to the robotic system. Upon the occurrence of a fire alarm, the UAV will first take off and reconnoiter over the district to obtain some rough information regarding the workspace. Suppose that, after the reconnaissance, the UAV obtains a look down picture of the entire district. According to the district picture, the system will know the map geometry and the semantics. Specifically, it knows the

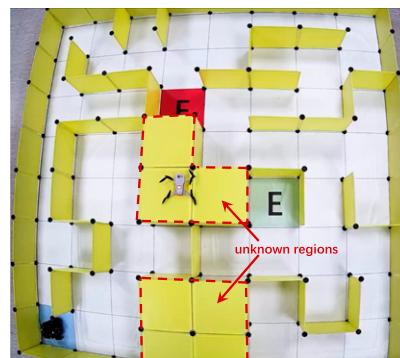
positions of the fire and extinguisher denoted as “F” and “E” in [Figure 5\(a\)](#).

- However, since some regions are covered by roofs, the connectivities still remain partially-known to the system after the reconnaissance. Therefore, the system only maintains a possible world map as shown in [Figure 5\(b\)](#). More detailed connectivities for some unknown regions in the possible world still remain to be explored by the ground robot.
- Then, based on the possible world map obtained by the UAV, the ground robot needs to accomplish the fire-fighting mission. Specifically, its objective is to first go to the region with extinguisher to get fire-extinguishers and then move to the region with fire. Let $\mathcal{AP} = \{\text{fire, extinguisher}\}$. Then the firefighting mission can be described by the following scLTL formula:

$$\phi = (\neg \text{fire} \ U \ \text{extinguisher}) \wedge \Diamond \text{fire}$$

Note that, to figure out the (non-)existence of those potential transitions under roofs, the ground robot is equipped with an onboard camera and it has to move to the adjacent areas to explore.

Given the above partially-known environment and the scLTL task, we first synthesize the regret-optimal strategy based on Algorithm 1. Based on the regret-optimal strategy, the robot chooses to explore the below unknown region but not to explore the above unknown region. This decision is based on the potential cost savings and additional costs associated with exploring these regions. For the below unknown region, if there are no obstacles, the robot can take a shortcut to reach the region “E” saving 14 units of cost compared to strategies that do not explore this region. Even if there are no available transitions, the robot will spend at most 12 more units of cost. On the other hand, for the above unknown region, reaching region “E” without exploring this region can save four units of cost if there are no obstacles. However, if there are no available transitions, an additional six units of cost will be incurred. In [Figure 6\(a\)](#) and [6\(b\)](#), we illustrate two possible trajectories under the regret-optimal strategy. In the former case, there are transitions available to



(a) Firefighting Scenario

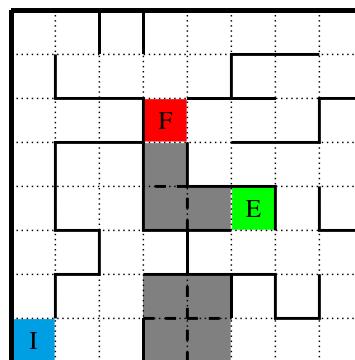
go through the below unknown regions, leading to potential cost savings. In the latter case, the ground robot has to backtrack and take a longer path to reach “E” due to the non-existence of shortcut transitions.

If the ground robot follows the best-case strategy, it will prioritize exploring unknown regions first to find shortcuts. [Figure 6\(e\)](#) illustrates a scenario where the robot successfully finds shortcuts under both roofs, resulting in efficient paths. However, this strategy is highly susceptible to uncertainty and not robust. In the worst-case scenario depicted in [Figure 6\(f\)](#), the robot encounters obstacles under both roofs, leading to a much longer path. Conversely, if the ground robot follows the worst-case strategy, it will always avoid exploration, assuming obstacles are always present under roofs in the worst-case scenario. As a result, regardless of the actual environment (T_1 or T_2), the robot will take the same path, as shown in [Figure 6\(c\)](#) and [6\(d\)](#). The video for the above case study is available at <https://youtu.be/W4pRJ7zrr40>.

6.2. Case study 2: Collaborative mobile robots

In the previous subsection, although two different robots are involved, it is still a single-agent planning problem as the UAV is only used to gather information. In this subsection, we use a case study to show that, in fact, our approach can be further extended in two directions:

- **Multi-Agent Settings:** For the case of multi-agent systems, when each agent always maintains precise information of other agents, for example, by consistent communications, our approach can still be applied. Specifically, one can simply build product of the mobility of each agent. Then the multi-agent planning problem becomes a centralized synthesis problem over the product state-space. Similar approach has also been used in other path planning problems; see, for example, [Guo and Dimarogonas \(2015\)](#).
- **Partially-Known Atomic Propositions:** Throughout the paper, we assume that the transition function in some



(b) Possible World

Figure 5. Experiment setting for Case Study 1.

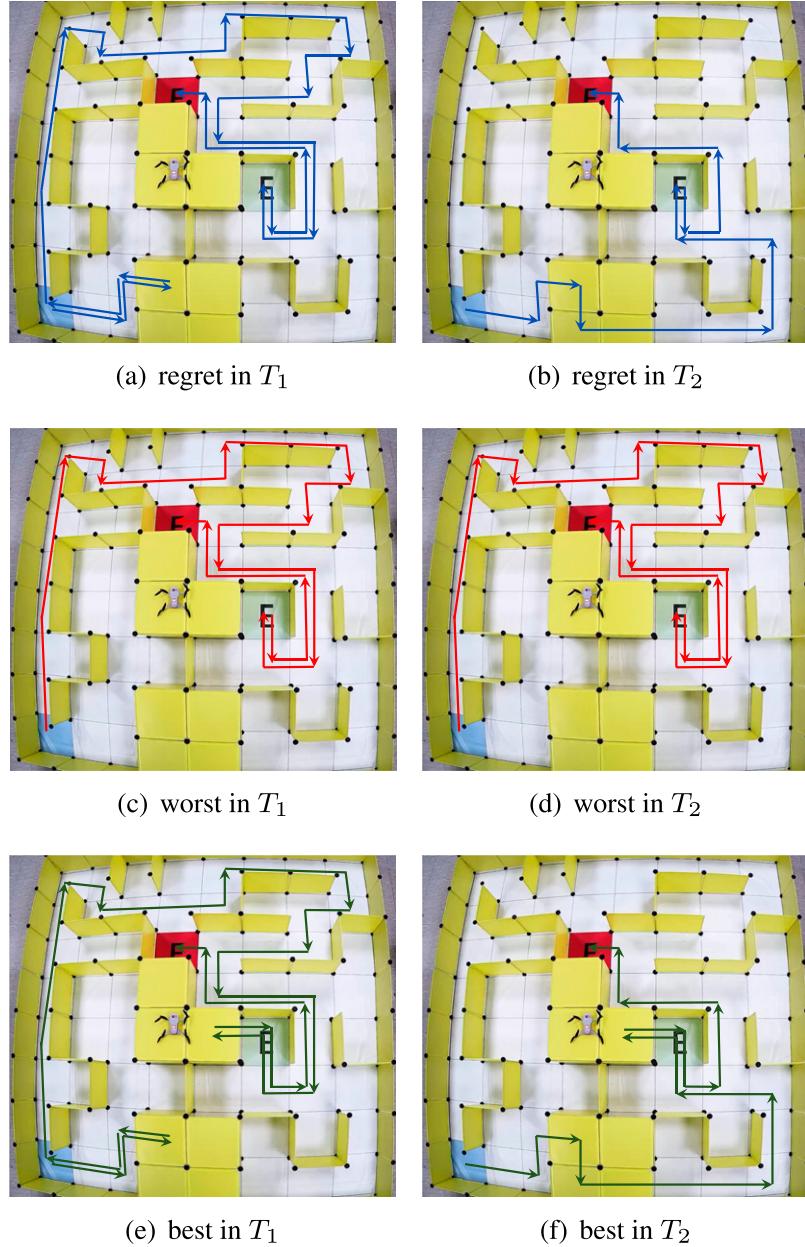


Figure 6. In T_1 , both unknown regions have shortcuts, while in T_2 , no unknown region has a shortcut.

regions is not fully known a priori. In fact, all our results can be applied to the case where the atomic propositions in some regions are not fully known a priori. For instance, we can introduce a labeling-pattern function $\mathbb{L}: X \rightarrow 2^{\mathcal{AP}}$, such that $\mathbb{L}(x) = \{\emptyset, \{a\}\}$ means that it is unknown a priori whether or not state x has property a unless the agent explores x physically. Mathematically speaking, this different setting is only a modeling issue. To be more specific, when building the knowledge-based game arena, a knowledge state should be defined as $(x, o) \in X_{un} \times 2^{\mathcal{AP}}$. Then the regret-optimal synthesis algorithm based on the knowledge-based game arena remains unchanged. We will use the following case study to illustrate this.

Let us consider a team of two mobile robots moving in a workspace shown in Figure 7 to collaboratively achieve a global task. Specifically, the workspace has five warehouses storing two types of materials, A and B. However, the robot only knows for sure a priori that some warehouses contain some materials, while there is uncertainty regarding the contents of other warehouses. The robot needs to physically reach those warehouses to check the availability of the materials. In Figure 7, \boxed{A} and $\boxed{\bar{A}}$ represent, respectively, that the warehouse contains A for sure and the warehouse may contain A; the same for material B. The global task for the team of two robots is that both types of materials can be picked up and be delivered to the target region T by any of the robots. We use atomic proposition A^i to represent that

robot i is at a region with material A^i ; the same for B^i and T^i . Then, the global task is described by the following scLTL formula

$$\phi = \bigvee_{i=1,2} [\Diamond(A^i \wedge \Diamond(B^i \wedge \Diamond T^i)) \vee \Diamond(B^i \wedge \Diamond(A^i \wedge \Diamond T^i))] \vee \\ [\Diamond(A^1 \wedge \Diamond T^1) \wedge \Diamond(B^2 \wedge \Diamond T^2)] \vee [\Diamond(A^2 \wedge \Diamond T^2) \wedge \Diamond(B^1 \wedge \Diamond T^1)]$$

which corresponds to four cases of which material is picked up and delivered by which robot.

In Figure 7(a)–7(c), we show the trajectories of two robots following the regret-based strategy in three different possible worlds. Since warehouse \boxed{A} is far away from the initial locations of both robots, the regret-optimal strategy is that, each robot will first independently explore warehouse \boxed{A} close to it.

- Figure 7(a) first shows an ideal case where robot 1 successfully finds materials in \boxed{A} and \boxed{B} . In this case, once robot 1 finds material B, robot 2 will terminate all of its action since the entire task can already be accomplished by robot 1.
- Figure 7(b) shows the case where robot 1 finds material in \boxed{A} but does not find material in \boxed{B} . In this case, robot 2 will keep moving towards warehouse \boxed{B} after finding nothing in \boxed{A} , and robot 1 will move directly towards the target region to deliver material A.
- Figure 7(c) shows the case where both robots do not find material A initially. Then robot 1 will explore warehouse \boxed{B} and robot 2 will also move towards warehouse \boxed{B} in case robot 1 does not find the material. However, once robot 1 finds material B, it will terminates its current action and changes to move towards warehouse \boxed{A} .

The video for the above case study is available at <https://youtu.be/YS1-0ne1c0Q>.

6.3. Numerical experiments on randomly generated environments

In this section, we present numerical experiments to test the efficiency of our algorithm and compare the performance of

the regret-based strategy against other strategies. The main purposes of these experiments are twofold: (i) to justify that regret is a meaningful metric for decision-making under partially-known environments, and (ii) to demonstrate that our algorithm is significantly more scalable compared to the standard regret-minimizing game approach that does not consider the structural properties of this problem. All the following numerical experiments are conducted on a MacBook (8 G/256G, Apple M1).

6.3.1. Comparison with the graph-unfolding approach. As discussed in Zhao et al. (2023), the algorithm in Filiot et al. (2010) can be applied to the knowledge-based game arena (with some slight modifications) to obtain a regret-optimal strategy. We refer this approach to as the *graph-unfolding approach* because it requires further unfolding of the knowledge-based game arena based on the weight function. In the first set of experiments, we compare our algorithm with the graph-unfolding approach by randomly generate PK-WTS \mathbb{T} as follows. The entire workspace has $|X|$ number of states, and for each state, it has a randomly assigned number of successor states, ranging from 1 to 2. These transitions are assumed to be known. In addition to the known states and transitions, we randomly add one more possible transitions. The minimal and the maximal transition costs are set to be 1 and 2, respectively. The robot is assigned with a simple scLTL task $\phi = \Diamond \text{target}$ and the atomic proposition *target* is randomly assigned to states in the generated PK-WTS. Whenever a \mathbb{T} is generated, we apply our algorithm and the graph-unfolding approach independently and record the sizes of the constructed graphs. It should be noted that, for our algorithm, we only need to construct the knowledge-based game arena G with $|V|$ vertices and $|E|$ edges, while to apply the graph-unfolding approach, one needs further unfolding to obtain a new graph G' with $|V'|$ vertices and $|E'|$ edges. We increase the state number $|X|$ from 15, 20, 30, 50, 80 to 100. For each $|X|$, we repeated to randomly generate \mathbb{T} for 100 times and compute the average statistics for each algorithm. The results are presented in Table 1. Clearly, our algorithm uses significantly less space and time than the graph-unfolding

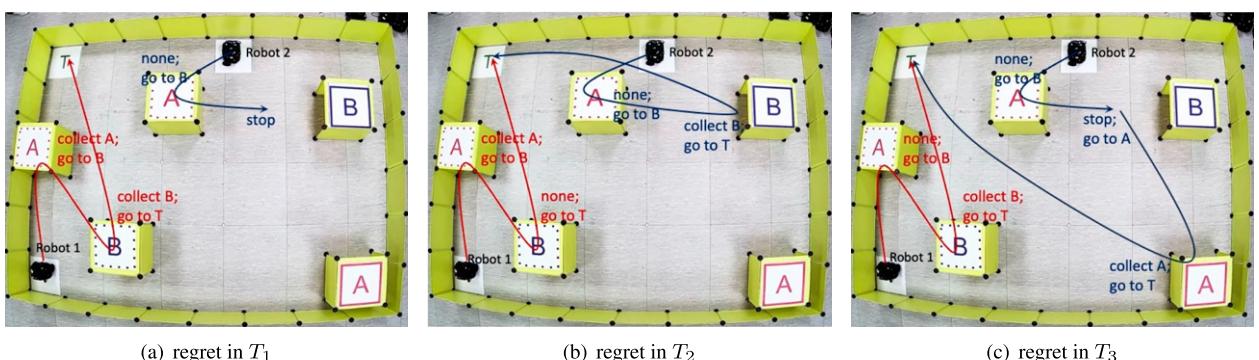


Figure 7. Experiment conducted on two collaborative mobile robots, where the solid rectangle and dashed rectangle in the map denote known and unknown regions, respectively.

approach, which is consistent with our previous theoretical complexity analysis. Note that, in this set of experiments, the structure of each PK-WTS is intentionally kept simple, as our primary goal here is to compare the synthesis algorithms rather than to test the scalability of each algorithm as the system becomes more complex. The scalability issue will be analyzed in the second set of experiments.

6.3.2. Scalability test and performance comparison. In the second set of experiments, we compare the performance of the regret-based strategy synthesized by our algorithm with those of the worst/best-case strategies. Similar to the previous experiments, each time we still randomly generate a PK-WTS with $|X|$ states. Here, we increase the complexity of the generated system by: (i) allowing each state to have at most four known successors; (ii) assigning transition costs between 1 and 100; and (iii) increasing the number of unknown transitions as the system size grows. Specifically, we randomly add $\lceil \log |\delta_{kno}| \rceil$ possible transitions, where $|\delta_{kno}|$ denotes the total number of known transitions. Furthermore, we consider two more complicated scLTL specifications for the robot:

$$\phi_1 = \Diamond(m \wedge \Diamond w)$$

and

$$\phi_2 = \Diamond(m \wedge \Diamond(w \wedge \Diamond(m \wedge \Diamond(w \wedge \Diamond(m \wedge \Diamond(w)))))))$$

where the atomic propositions m and w denote “material” and “warehouse,” respectively, and they are randomly assigned to states in the generated PK-WTS. Intuitively, ϕ_1 and ϕ_2 require the robot to move between material and warehouse once and five times, respectively. For each system generated and each scLTL formula, we only use our algorithm to synthesize the regret-based strategy. We also synthesize the worst-case-based and best-case-based strategies for the purpose of later performance comparison. Still, for each system size, we repeated the test randomly for 100 times. [Table 2](#) shows the average statistics of our synthesis algorithm for formula ϕ_2 . Note that, compared with [Table 1](#), the running time grows faster as the system size increases. This is because we also increase the number of unknown

states as the system size grows. However, the exponential complexity, in terms of the unknown states, seems to be unavoidable due to the inherent lack of information. Intuitively, if one has no prior knowledge but still aims to perform planning, they must account for all possible scenarios, which leads to exponential growth in the decision space due to the combinatorial nature of the problem.

Note that, although the complexity of the synthesis algorithm only depends on the PK-WTS \mathbb{T} , for each time the robot is executed, the actual performance of the algorithm depends on the underlying actual environment $T \in \mathbb{T}$, which is unknown a priori. To compare the performance of our regret-based strategy with other strategies, we need to test the performance of each strategy for a large number of underlying actual environment compatible with the possible world following *an unknown distribution*. To this end, for each possible transition in the generated PK-WTS \mathbb{T} , we further assume that there is a probability $p \in [0, 1]$ that there is an obstacle in the corresponding path, that is, the transition does not exist. Note that this probability information is not used in the planning algorithm, as we consider a non-stochastic setting; it is only used for the valuation purposes. Based on the above settings, for each randomly generated PK-WTS \mathbb{T} with different sizes, we further randomly generate an actual environment $T \in \mathbb{T}$ using parameter $p \in \{0, 0.2, 0.5, 0.8, 1\}$. The regret-based strategy, the worst-case-based strategy and the best-case-based strategy are then independently applied to the same environment $T \in \mathbb{T}$ and we record the corresponding actual cost of each strategy. For each combination of state number $|X|$ with probability p , we generate 100 actual environments in total. The statistic results for the average cost for each strategy among the 100 randomly generated instances for tasks ϕ_1 and ϕ_2 are shown in [Figures 8](#) and [9](#), respectively.

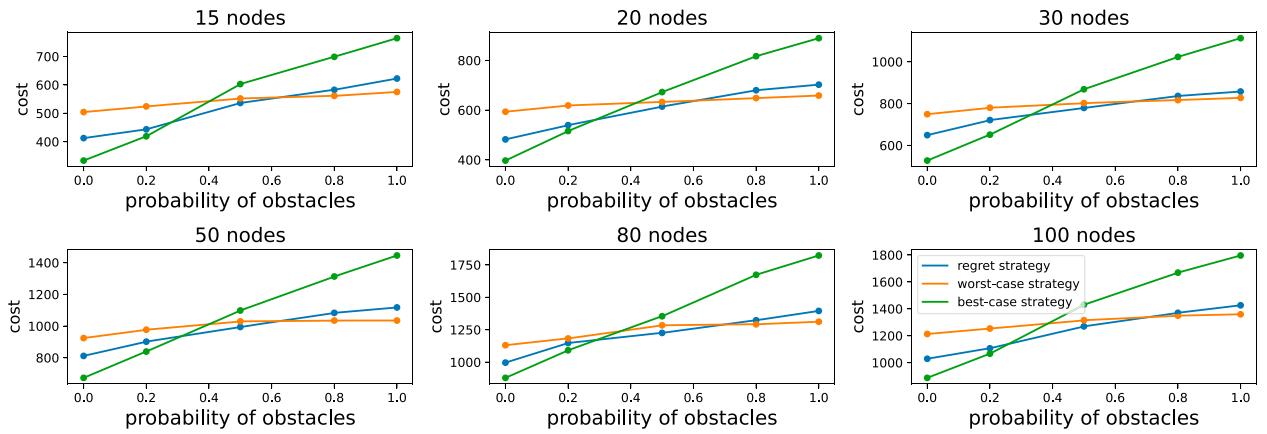
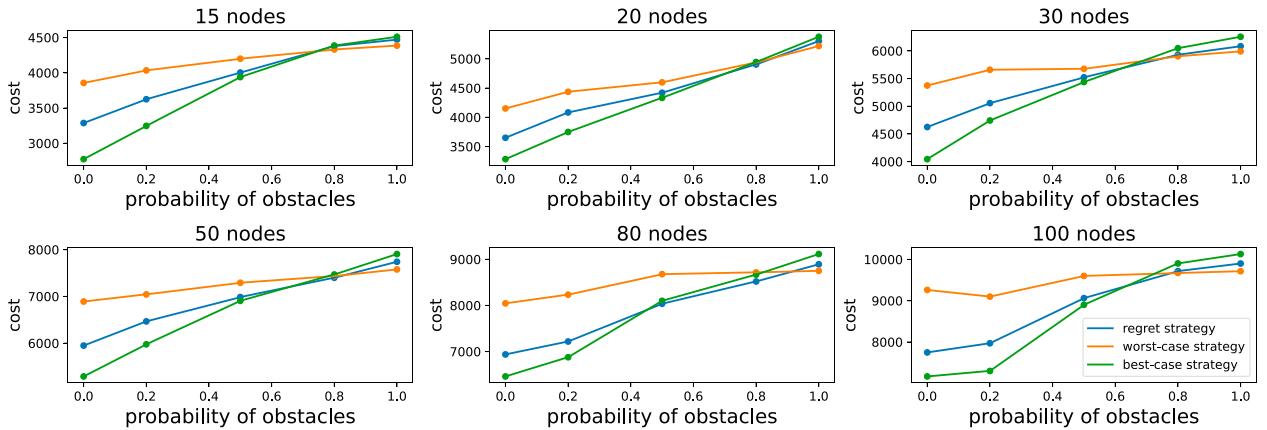
6.3.4. Analysis of the experimental results. Clearly, if the obstacle probability is $p = 0$, that is, the possible transitions always exist, then the best-case strategy should have the minimum cost. On the other hand, if the obstacle probability is $p = 1$, that is, the possible transitions always do not exist, the obviously the worst-case strategy should have the minimum cost. However, our aim here is to address the

Table 1. Numerical experiments on the comparison between our approach and the graph-unfolding approach. In our approach, “model time” refers to the time required to construct the game arena, while in the graph-unfolding approach, “model time” refers to the time required to construct the unfolded game arena.

$ X $	$ \delta_T $	Our approach				Graph-unfolding approach			
		Vertices ($ V $)	Edges ($ E $)	Model Time	Solve Time	Vertices ($ V' $)	Edges ($ E' $)	Model Time	Solve Time
15	23	176	215	0.0052	0.0063	119904	145987	1.436	6.129
20	29	247	306	0.0091	0.0099	229344	284345	3.525	17.387
30	47	611	722	0.0387	0.0293	1394106	1645963	28.13	237.14
50	78	815	999	0.0991	0.0571	2769884	3413719	66.76	1866.3
80	121	1048	1248	0.1115	0.0793	4367065	5199855	200.86	3422.1
100	154	1319	1562	0.2331	0.1694	10543919	12667063	831.81	-

Table 2. Numerical scalability experiments on randomly generated systems for task ϕ_2 .

PK-WTS \mathbb{T}		PS \mathcal{P}		Game arena G			
$ X $	$ \delta_{\mathcal{T}} $	$ \mathcal{S} $	$ \delta_{\mathcal{P}} $	$ \mathcal{V} $	$ \mathcal{E} $	Model time	Solve time
15	37	100	254	2176	3097	0.3697	0.1739
20	51	121	328	3810	5570	1.0731	0.4416
30	79	198	522	6129	8866	1.4202	1.0163
50	132	293	753	41575	60363	32.467	23.253
80	192	485	1173	83508	118266	56.018	40.084
100	261	577	1396	172811	227353	126.39	101.93

**Figure 8.** The average performances of different strategies in randomly generated environments for task ϕ_1 .**Figure 9.** The average performances of different strategies in randomly generated environments for task ϕ_2 .

scenario where such probability information is not available beforehand. According to Figures 8 and 9, we further have the following observations.

- Regret-Based versus Worst-Case-Based: We note that the performance of worst-case strategy is the least sensitive to the probability of obstacles. However, it is generally conservative especially when the probability of obstacles is low. Compared with the worst-case

strategy, our regret-based strategy has better performance in a wide range of probabilities. Particularly, by comparing Figures 8 and 9, this advantage becomes more significant in the complex task ϕ_2 than in the simpler task ϕ_1 .

- Regret-Based versus Best-Case-Based: We note that the performance of the best-case strategy is the most sensitive to the probability of obstacles. The advantage of our regret-based strategy compared to the best-case

strategy is better illustrated in [Figure 8](#) for task ϕ_1 . Specifically, we observe that the regret-based strategy still performs better over a wide range of probabilities. However, in [Figure 9](#) for task ϕ_2 , the performances of the regret-based strategy and the best-case strategy become very close, with the former not outperforming the latter in more than half of the probability range. This is due to the long horizon required to accomplish task ϕ_2 , that is, the robot must repeat visits to two regions for five times across the entire workspace. The best-case strategy, which always opts for exploration first, benefits from the long horizon since the cost is averaged over time. This explains why the advantage of the regret-based strategy over the best-case strategy is less significant for ϕ_2 compared to ϕ_1 .

Therefore, the experimental results further support our earlier claim that regret is a meaningful metric when the environment is partially known and no probabilistic information is available a priori. Essentially, the regret-based strategy achieves a reasonable trade-off between the worst-case and best-case strategies. Note that the advantage of this metric is more pronounced for “mid-horizon tasks” where the robot only needs to traverse the unknown workspace a few times rather than repeatedly working in it. In the latter case, using the best-case strategy to explore unknown states freely can be beneficial, as the exploration cost will eventually average out over time.

7. Conclusions

In this paper, we introduced a novel approach for optimal path planning to satisfy scLTL specifications within partially-known environments. We utilized the concept of regret to achieve the tradeoff between actual costs in an environment and potential benefits from exploring unknown regions. Our framework included a knowledge-based model to formally represent partially-known scenarios, along with an efficient algorithm for synthesizing optimal strategies with minimal regret. Case studies and numerical experiments demonstrated the efficacy of our approach, particularly in scenarios lacking probabilistic prior information.

Our work brings new insights and provides a new framework for path planning in partially-known environments. Regarding the future works, we identify several important directions for extending our regret-based path planning framework.

- First, in our current work, we focus on solving the regret-optimal planning problem within a monolithic system model. While this approach can be, in principle, extended to multi-robot teams with information exchanges, the complexity scales exponentially as the number of agents increases. Therefore, in future research, we plan to explore computationally efficient methods, such as

sampling-based approaches, to tackle the regret-optimal planning problem in multi-agent settings.

- Second, our current work focuses on addressing information uncertainty while assuming the underlying environment is purely deterministic. In future research, we aim to further investigate the regret-optimal reactive control synthesis problem under partially-known but non-deterministic environments. This task presents a greater challenge than planning in such environments, as the environment is not constrained to a strongly positional strategy for all vertices in the game.
- Third, in this work, we aim to synthesize a regret-optimal strategy in an offline manner, taking all future possibilities into account. However, in some applications, the robot can only incrementally build a partial map within its horizon during exploration. In the future, we would like to investigate how to design a limited-lookahead version of the planning strategy that minimizes regret on-the-fly, based on real-time information.
- Finally, this work considers a purely non-stochastic control setting, assuming a complete absence of prior probabilistic information. However, in certain scenarios, it is possible to estimate rough probabilistic information even without precise knowledge. For instance, one might know that the probability of a specific transition falls within an interval, leading to the framework of interval MDPs. In future studies, we aim to explore how the regret-based metric can be integrated with such rough probabilistic information for the purpose of path planning.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the National Natural Science Foundation of China (62173226, 92367203).

ORCID iDs

Jianing Zhao  <https://orcid.org/0000-0002-1579-0522>
Xiang Yin  <https://orcid.org/0000-0003-1944-1570>

Supplemental Material

Supplemental material for this article is available online.

References

- Ayala AM, Andersson SB and Belta C (2013) Temporal logic motion planning in unknown environments. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Tokyo, Japan, 03–07 November 2013. IEEE, 5279–5284.
Baier C and Katoen JP (2008) *Principles of Model Checking*. Cambridge: MIT Press.

- Belta C and Sadraddini S (2019) Formal methods for control synthesis: an optimization perspective. *Annual Review of Control, Robotics, and Autonomous Systems* 2: 115–140.
- Belta C, Yordanov B and Gol EA (2017) *Formal Methods For Discrete-Time Dynamical Systems*. Cham, Switzerland: Springer, 89.
- Bertoli P, Cimatti A, Roveri M, et al. (2006) Strong planning under partial observability. *Artificial Intelligence* 170(4-5): 337–384.
- Blackwell D (1956) An analog of the minimax theorem for vector payoffs. *Pacific Journal of Mathematics* 6(1): 1–8.
- Bloem R, Chatterjee K, Greimel K, et al. (2014) Synthesizing robust systems. *Acta Informatica* 51: 193–220.
- Bozkurt AK, Wang Y, Zavlanos MM, et al. (2020) Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In: IEEE International Conference on Robotics and Automation (ICRA), 31 May - 31 August, 2020, Virtual, 10349–10355.
- Brafman RI and De Giacomo G (2019) Planning for LTLf/LDLf goals in non-markovian fully observable nondeterministic domains. In: International Joint Conference on Artificial Intelligence (IJCAI), August 10-16, 2019, Macao, China, 1602–1608.
- Brihaye T, Geeraerts G, Haddad A, et al. (2017) Pseudopolynomial iterative algorithm to solve total-payoff games and min-cost reachability games. *Acta Informatica* 54(1): 85–125.
- Cadilhac M, Pérez GA and Mvd B (2019) The impatient may use limited optimism to minimize regret. *International Conference on Foundations of Software Science and Computation Structures*. Cham, Switzerland: Springer, 133–149.
- Cai M, Peng H, Li Z, et al. (2021a) Learning-based probabilistic LTL motion planning with environment and motion uncertainties. *IEEE Transactions on Automatic Control* 66(5): 2386–2392.
- Cai M, Xiao S, Li Z, et al. (2021b) Optimal probabilistic motion planning with potential infeasible ltl constraints. *IEEE Transactions on Automatic Control* 68(1): 301–316.
- Cai M, Aasi E, Belta C, et al. (2023) Overcoming exploration: deep reinforcement learning for continuous control in cluttered environments from temporal logic specifications. *IEEE Robotics and Automation Letters* 8(4): 2158–2165.
- Cardona GA and Vasile CI (2024) Planning for heterogeneous teams of robots with temporal logic, capability, and resource constraints. *The International Journal of Robotics Research* 43(13): 2089–2111.
- Chatterjee K, Randour M and Raskin JF (2014) Strategy synthesis for multi-dimensional quantitative objectives. *Acta Informatica* 51(3): 129–163.
- Cho K, Suh J, Tomlin CJ, et al. (2017) Cost-aware path planning under co-safe temporal logic specifications. *IEEE Robotics and Automation Letters* 2(4): 2308–2315.
- Cimatti A, Pistore M, Roveri M, et al. (2003) Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence* 147(1–2): 35–84.
- Ding X, Smith SL, Belta C, et al. (2014) Optimal control of markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control* 59(5): 1244–1257.
- Fiaz UA and Baras JS (2020) *Fast, composable rescue mission planning for uavs using metric temporal logic*. In: 21st IFAC World Congresse, July 11-17, 2020, Virtual, 15404–15411.
- Filiot E, Gall TL and Raskin JF (2010) Iterated regret minimization in game graphs. *International Symposium on Mathematical Foundations of Computer Science*. Berlin, Heidelberg, Germany: Springer, 342–354.
- Fridovich-Keil D, Fisac JF and Tomlin CJ (2019) Safely probabilistically complete real-time planning and exploration in unknown environments. In: International Conference on Robotics and Automation (ICRA), Montreal, Canada, 20–24 May 2019. IEEE, 7470–7476.
- Fu J and Topcu U (2014) Probably approximately correct mdp learning and control with temporal logic constraints. *Robotics: Science and Systems*. USA: MIT Press.
- Fu J and Topcu U (2016) Synthesis of joint control and active sensing strategies under temporal logic constraints. *IEEE Transactions on Automatic Control* 61(11): 3464–3476.
- Geffner T and Geffner H (2018) Compact policies for fully observable non-deterministic planning as sat. *Proceedings of the International Conference on Automated Planning and Scheduling* 28: 88–96.
- Goel G and Hassibi B (2023) Regret-optimal estimation and control. *IEEE Transactions on Automatic Control* 68(5): 3041–3053.
- Gundana D and Kress-Gazit H (2021) Event-based signal temporal logic synthesis for single and multi-robot tasks. *IEEE Robotics and Automation Letters* 6(2): 3687–3694.
- Guo M and Dimarogonas DV (2015) Multi-agent plan re-configuration under local ltl specifications. *The International Journal of Robotics Research* 34(2): 218–235.
- Guo M and Zavlanos MM (2018) Probabilistic motion planning under temporal tasks and soft constraints. *IEEE Transactions on Automatic Control* 63(12): 4051–4066.
- Hasanbeig M, Kantaros Y, Abate A, et al. (2019) Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In: 58th IEEE Conference on Decision and Control (CDC), Nice, France, 11–13 December 2019. IEEE, 5338–5343.
- Hazan E (2016) Introduction to online convex optimization. *Foundations and Trends® in Optimization* 2(3–4): 157–325.
- Hazan E, Kakade S and Singh K (2020) The nonstochastic control problem. *Algorithmic Learning Theory* PMLR, 408–421.
- Ho QH, Sunberg ZN and Lahijanian M (2024) Sampling-based reactive synthesis for nondeterministic hybrid systems. *IEEE Robotics and Automation Letters* 9(2): 931–938.
- Hunter P, Pérez GA and Raskin JF (2017) Reactive synthesis without regret. *Acta Informatica* 54(1): 3–39.
- Hustiu S, Mahulea C, Kloetzer M, et al. (2024) On multi-robot path planning based on Petri net models and LTL specifications. *IEEE Transactions on Automatic Control* 69(9): 6373–6380.
- Kantaros Y and Zavlanos MM (2018) Sampling-based optimal control synthesis for multirobot systems under global temporal tasks. *IEEE Transactions on Automatic Control* 64(5): 1916–1931.

- Kantaros Y and Zavlanos MM (2020) Stylus*: a temporal logic optimal control synthesis algorithm for large-scale multi-robot systems. *The International Journal of Robotics Research* 39(7): 812–836.
- Kantaros Y, Kalluraya S, Jin Q, et al. (2022) Perception-based temporal logic planning in uncertain semantic maps. *IEEE Transactions on Robotics* 38(4): 2536–2556.
- Kloetzer M and Belta C (2009) Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics* 26(1): 48–61.
- Kloetzer M and Mahulea C (2016) Multi-robot path planning for syntactically co-safe LTL specifications. In: 13th International Workshop on Discrete Event Systems (WODES), Xi'an, China, 30 May 2016–01 June. IEEE, 452–458.
- Kloetzer M and Mahulea C (2020) Path planning for robotic teams based on ltl specifications and petri net models. *Discrete Event Dynamic Systems* 30(1): 55–79.
- Kress-Gazit H, Lahijanian M and Raman V (2018) Synthesis for robots: guarantees and feedback for robot behavior. *Annual Review of Control, Robotics, and Autonomous Systems* 1: 211–236.
- Kwiatkowska M, Norman G and Parker D (2022) Probabilistic model checking and autonomy. *Annual Review of Control, Robotics, and Autonomous Systems* 5: 385–410.
- Lacerda B and Lima PU (2019) Petri net based multi-robot task coordination from temporal logic specifications. *Robotics and Autonomous Systems* 122: 103289.
- Lacerda B, Faruq F, Parker D, et al. (2019) Probabilistic planning with formal performance guarantees for mobile service robots. *The International Journal of Robotics Research* 38(9): 1098–1123.
- Lahijanian M, Maly MR, Fried D, et al. (2016) Iterative temporal planning in uncertain environments with partial satisfaction guarantees. *IEEE Transactions on Robotics* 32(3): 583–599.
- LaValle SM (2006) *Planning Algorithms*. United Kingdom: Cambridge University Press.
- Leung K, Aréchiga N and Pavone M (2023) Backpropagation through signal temporal logic specifications: infusing logical structure into gradient-based methods. *The International Journal of Robotics Research* 42(6): 356–370.
- Lin H (2014) Mission accomplished: an introduction to formal methods in mobile robot motion planning and control. *Unmanned Systems* 2(02): 201–216.
- Liu Z, Guo M and Li Z (2024) Time minimization and online synchronization for multi-agent systems under collaborative temporal logic tasks. *Automatica* 159: 111377.
- Luckeuck M, Farrell M, Dennis LA, et al. (2019) Formal specification and verification of autonomous robotic systems: a survey. *ACM Computing Surveys* 52(5): 1–41.
- Luo X and Zavlanos MM (2022) Temporal logic task allocation in heterogeneous multirobot systems. *IEEE Transactions on Robotics* 38(6): 3602–3621.
- Luo X, Kantaros Y and Zavlanos MM (2021) An abstraction-free method for multirobot temporal logic optimal control synthesis. *IEEE Transactions on Robotics* 37(5): 1487–1507.
- Lv P, Luo G, Ma Z, et al. (2023) Optimal multi-robot path planning for cyclic tasks using petri nets. *Control Engineering Practice* 138: 105600.
- Mahulea C, Kloetzer M and González R (2020) *Path Planning of Cooperative Mobile Robots Using Discrete Event Models*. Hoboken: John Wiley & Sons.
- Muisse C, Belle V and McIlraith S (2014) Computing contingent plans via fully observable non-deterministic planning. In: Proceedings of the AAAI Conference on Artificial Intelligence, July 27–31, 2014, Québec City, Québec, Canada, Vol. 28.
- Muvvala K, Amorese P and Lahijanian M (2022) Let's collaborate: regret-based reactive synthesis for robotic manipulation. In: 2022 International Conference on Robotics and Automation (ICRA), Philadelphia, PA, 23–27 May 2022. IEEE, 4340–4346.
- Sahin YE, Nilsson P and Ozay N (2020) Multirobot coordination with counting temporal logics. *IEEE Transactions on Robotics* 36(4): 1189–1206.
- Scher G and Kress-Gazit H (2020) Warehouse automation in a day: from model to implementation with provable guarantees. In: 16th IEEE International Conference on Automation Science and Engineering (CASE), Hong Kong, China, 20–21 August 2020, 280–287.
- Shalev-Shwartz S, Adsdadsa N, Sdfkj M, et al. (2012) Online learning and online convex optimization. *Foundations and Trends® in Machine Learning* 4(2): 107–194.
- Smith SL, Tumová J, Belta C, et al. (2011) Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research* 30(14): 1695–1708.
- Ulusoy A, Smith SL, Ding XC, et al. (2013) Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research* 32(8): 889–911.
- Vardi MY and Wolper P (1986) An automata-theoretic approach to automatic program verification. In: 1st Symposium in Logic in Computer Science (LICS), June 16–18, 1986, Cambridge, MA, USA. IEEE Computer Society.
- Vasile CI, Li X and Belta C (2020) Reactive sampling-based path planning with temporal logic specifications. *The International Journal of Robotics Research* 39(8): 1002–1028.
- Wang Y, Nalluri S and Pajic M (2020) Hyperproperties for robotics: planning via hyperltl. In: 2020 IEEE International Conference on Robotics and Automation (ICRA), 31 May – 31 August, 2020, Virtual. IEEE, 8462–8468.
- Wolff EM, Topcu U and Murray RM (2012) Optimal control with weighted average costs and temporal logic specifications. *Robotics: Science and Systems*. USA: MIT Press.
- Wolff EM, Topcu U and Murray RM (2013) Optimal control of non-deterministic systems for a computationally efficient fragment of temporal logic. In: 52nd IEEE Conference on Decision and Control (CDC). IEEE, 3197–3204.
- Yan YH, Zhao P and Zhou ZH (2023) Online non-stochastic control with partial feedback. *Journal of Machine Learning Research* 24(273): 1–50.
- Yin X, Gao B and Yu X (2024) Formal synthesis of controllers for safety-critical autonomous systems: developments and challenges. *Annual Reviews in Control* 57: 100940.

- Yu P and Dimarogonas DV (2022) Distributed motion coordination for multirobot systems under LTL specifications. *IEEE Transactions on Robotics* 38(2): 1047–1062.
- Yu P, Gao Y, Jiang FJ, et al. (2023) Online control synthesis for uncertain systems under signal temporal logic specifications. *The International Journal of Robotics Research*.
- Yu P, Tan X and Dimarogonas DV (2024) Continuous-time control synthesis under nested signal temporal logic specifications. *IEEE Transactions on Robotics* 43(6): 765–790.
- Zhao J, Zhu K, Li S, et al. (2023) To explore or not to explore: regret-based ltl planning in partially-known environments. In: 22nd IFAC World Congress, 9 July – 14 July 2023, Yokohama, Japan, 12171–12177.
- Zhou B, Pan J, Gao F, et al. (2021) Raptor: robust and perception-aware trajectory replanning for quadrotor fast flight. *IEEE Transactions on Robotics* 37(6): 1992–2009.
- Zhou H, Song Y and Tzoumas V (2023) Safe non-stochastic control of control-affine systems: an online convex optimization approach. *IEEE Robotics and Automation Letters* 8(12): 7873–7880.