To Explore or Not to Explore: Regret-Based LTL Planning in Partially-Known Environments [*]

Jianing Zhao [*] **Keyi Zhu** [*] **Shaoyuan Li** [*] **Xiang Yin** [*]

[*] *Department of Automation, Shanghai Jiao Tong University*
Shanghai 200240, China
(e-mail: {jnzhao,bai12wp,syli,yinxiang} @sjtu.edu.cn)

Abstract: In this paper, we investigate the optimal robot path planning problem for high-level specifications described by co-safe linear temporal logic (LTL) formulae. We consider the scenario where the map geometry of the workspace is *partially-known*. Specifically, we assume that there are some unknown regions, for which the robot does not know their successor regions *a priori* unless it reaches these regions physically. In contrast to the standard game-based approach that optimizes the worst-case cost, in the paper, we propose to use *regret* as a new metric for planning in such a partially-known environment. The regret of a plan under a fixed but unknown environment is the difference between the actual cost incurred and the best-response cost the robot could have achieved if it realizes the actual environment with hindsight. We provide an effective algorithm for finding an optimal plan that satisfies the LTL specification while minimizing its regret. A case study on firefighting robots is provided to illustrate the proposed framework. We argue that the new metric is more suitable for the scenario of partially-known environment since it captures the trade-off between the actual cost spent and the potential benefit one may obtain for exploring an unknown region.

Keywords: Discrete event systems, regret, autonomous robots, LTL planning.

1. INTRODUCTION

Path planning is one of the central problems in autonomous robots. In this context, one needs to design a finite or infinite path for the robot, according to its dynamic and the underlying environment, such that some desired requirements can be fulfilled. In many robotics applications such as search and rescue, persistent surveillance or warehouse delivery, the planning tasks are usually complicated evolving spatial and/or temporal constraints. Therefore, in the past years, robot path planning for *high-level specifications* using formal logics has been drawing increasingly more attentions in the literature; see, e.g., Kress-Gazit et al. (2018); Mahulea et al. (2020); Kloetzer and Mahulea (2020); Yu et al. (2022).

Linear temporal logic (LTL) is one of the most popular languages for describing high-level specifications, which supports temporal operators such as "always", "eventually" or "next". In the context of robotic applications, path planning and decision-making for LTL specifications have been investigated very extensively recently. For example, Smith et al. (2011) studied how to generate an optimal open-loop plan, in the so-called "prefix-suffix" structure, such that a given LTL formula is fulfilled. When the results of control actions are non-deterministic, algorithms for synthesizing reactive strategies have been developed using two-player games Fu and Topcu (2016). The LTL path planning problem has also been studied for stochastic

systems Guo and Zavlanos (2018) to provide probabilistic guarantees and for multi-robot systems Yu and Dimarogonas (2022) under both global and local tasks.

The aforementioned works on LTL path planning all assume that the environment is known in the sense that the map geometry and the semantic structure are both available at the planning stage. In practice, however, the environment may be partially-known such that the robot needs to explore the map geometry as well as the region semantics on-the-fly. To this end, in Guo and Dimarogonas (2015), the authors provided a re-planning algorithm based on the system model updated online. In Lahijanian et al. (2016), the authors proposed an iterative planning algorithm in uncertain environments where unknown obstacles may appear. A learning-based algorithm is proposed in Cai et al. (2021) for LTL planning in stochastic environments with unknown transition probabilities. Recently, Kantaros et al. (2022) investigated the LTL planning problem under environments with known map geometries but with semantic uncertainties.

In this paper, we also investigate the LTL path planning for robots in *partially-known* environments. Specifically, here we assume that the location of each region in the map is perfectly known but, for some regions, the robot does not know their successor regions *a priori* unless it reaches these regions physically. For example, in Figure 1, the dashed line between regions 2 and 5 denotes a possible wall that may prevent the robot from reaching region 5 directly from region 2. Initially, the robot knows the possibility of the wall, but it will actually know the (non-)existence of the wall only when reaching region 2. Here, we distinguish
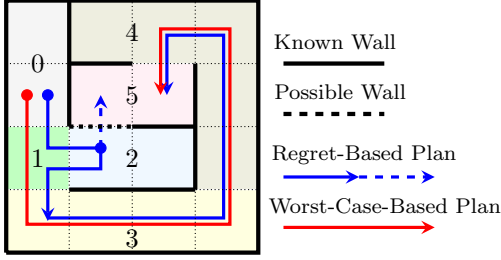
Fig. 1. A motivating example, where a robot needs to reach region 5 from regin 0 with partially-known environment information.

between the terminologies of *non-determinsitic* environments and *partially-known* environments. Specifically, the former is referred to the scenario where the outcome of the environment is purely random in the sense that even for the same visit, the environment may behave differently. However, the partially-known environment 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.

To solve the path planning problem in partially-known environments, a direct approach is to follow the same idea for planning in non-deterministic environments, where game-based approaches are usually used to minimize the *worst-case cost*. Still, let us consider Figure 1, where the robot aims to reach target region 5 with shortest distance. Using a worst-case-based approach, the robot will follow the red trajectory. This is because the short-cut from regions 2 to 5 may not exist; 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 wall does not exist. Therefore, a more natural and human-like plan is to first go to region 2 to take a look at whether there is a wall. 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.

In this paper, we formulate and solve a new type of LTL optimal path planning problem for robots working in an aforementioned partially-known environment. We adopt the notion of *regret* from game theory Yannakakis and Togelius (2018) as the optimality metric. We propose the structure of *partially-known weighted transition systems* (PK-WTS) as the model that contains the set of all possible actual environments. 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 after it knows the actual environment with hindsight. A value iteration algorithm is developed for computing an optimal strategy such that (i) it satisfies the LTL requirement under any possible environment; and (ii) minimizes its regret. We illustrate by case studies that, compared with the worst-case-based synthesis for non-deterministic environments, the proposed regret-based synthesis is more suitable for partially-known environments.

The regret minimization problem is an emerging topic in the context of graph games; see, e.g., Filiot et al. (2010); Hunter et al. (2017); Cadilhac et al. (2019). Particularly, Filiot et al. (2010) is most related to our problem setting, where it solves a reachability game, via a graph-unfolding algorithm, with minimal regret for a player while the other player plays unrestricted strategies or regret-minimizing strategies as well. Furthermore, in the context of robotic applications, the recent work Muvvala et al. (2022) uses regret to optimize human-robot collaboration strategies, based on the algorithm in Filiot et al. (2010). However, when it refers to the exploration problem, since the actual environment is fixed, two critical issues arise in the two-player graph game: i) the environment-player plays a positional strategy rather than the unrestricted manner in Filiot et al. (2010); Muvvala et al. (2022); and ii) to this end, the agent-player needs to "memorize" the choices of the environment-player, which renders the mechanism of *knowledge update*. Therefore, it still remains open in the literature of reactive synthesis/graph game to solve a regret-minimizing reachability game against the environment-player playing positional strategies. In this work, we use regret to capture the issue of exploration in partially-known environments and present an efficient algorithm to solve the regret-minimizing planning problem.

2. LTL PLANNING IN FULLY-KNOWN ENVIRONMENTS

2.1 Weighted Transition Systems

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)

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

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; $\delta_T : X \to 2^X$ 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; $w : X \times X \to \mathbb{N}$ 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; and $L : X \to 2^{\mathcal{AP}}$ 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 $\rho = x_0 x_1 x_2 \cdots \in X^\omega$ such that $x_{i+1} \in \delta_T(x_i), i \geq 0$. A finite path is defined analogously. We denote by $\mathsf{Path}^\omega(T)$ and $\mathsf{Path}^*(T)$ the sets of all infinite paths and finite paths in T, respectively. Given a finite path $\rho = x_0 x_1 \cdots x_n \in \mathsf{Path}^*(T)$, its cost is defined as the sum of all transition weights in it, which is denoted by $\mathsf{cost}(\rho) = \sum_{i=0}^{n-1} w(x_i, x_{i+1})$. The *trace* of an infinite path $\rho = x_0 x_1 x_2 \cdots \in X^\omega$ is an infinite sequence over $2^{\mathcal{AP}}$ denoted by $L(\rho) = L(x_0) L(x_1) \cdots$. Analogously, we denote by $\mathsf{Trace}^\omega(T)$ and $\mathsf{Trace}^*(T)$ the sets of all infinite traces and finite traces in T, respectively.

Remark 1. Throughout this paper, we assume that T is fixed and connected, that is, for any $x, x' \in X$, there exists a path fragment $x_1 x_2 \cdots x_n \in X^*$ such that $x_1 = x$, $x_{i+1} \in \delta_T(x_i), i = 1, \ldots, n-1$ and $x_n = x'$. Such a setting

is common and without loss of generality, (c.f. MacDonald and Smith (2019)) when the agent is able to "turn around" between any two adjacent regions.

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 $\Diamond\phi = \top U \phi$. LTL formulae are evaluated over infinite words; the readers are referred to Baier and Katoen (2008) for the 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 "always" $\square$ in scLTL. Although the semantics of LTL are defined over infinite words, it is well-known that any infinite word satisfying a co-safe LTL 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 $\mathcal{L}_{pref}^\phi$ the set of all finite good prefixes of scLTL formula ϕ.

For any scLTL formula ϕ, its good prefixes $\mathcal{L}_{pref}^\phi$ can be accepted by a *deterministic finite automaton* (DFA). Formally, a DFA 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; $f : Q \times \Sigma \to Q$ is a transition function; and $Q_F \subseteq Q$ is the set of accepting states. The transition function can also be extended to $f : Q \times \Sigma^* \to Q$ recursively. A finite word $\tau \in \Sigma^*$ is said to be *accepted* by $\mathcal{A}$ if $f(q_0, \tau) \in Q_F$; we denote by $\mathcal{L}(\mathcal{A})$ the set of all accepted words. Then for any scLTL formula ϕ defined over $\mathcal{AP}$, 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 $\mathcal{L}(\mathcal{A}_\phi) = \mathcal{L}_{pref}^\phi$.

2.3 Path Planning for scLTL Specifications

Given a WTS T and an scLTL formula ϕ, the path planning problem is to find an finite path (a.k.a. a plan) $\rho \in \mathsf{Path}^*(T)$ such that $L(\rho) \in \mathcal{L}_{pref}^\phi$ and, at the same time, its cost $\mathsf{cost}(\rho)$ is minimized.

To solve the scLTL planning problem, the standard approach is to build the *product system* between WTS $T = (X, x_0, \delta_T, w, \mathcal{AP}, L)$ and DFA $\mathcal{A}_\phi = (Q, q_0, \Sigma, f, Q_F)$, which 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; $\delta_P : S \to 2^S$ is the transition function defined by: for any $s = (x, q) \in S$, we have $\delta_P(s) = \{(x', q') \in S \mid x' \in \delta_T(x) \wedge q' = f(q, L(x))\}$; $w_P : S \times S \to \mathbb{N}$ is the weight function defined by: for any $s = (x, q), s' = (x', q') \in S$, we have $w_P(s, s') = w(x, x')$; 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 $\rho = x_0 \cdots x_n \in \mathsf{Path}^*(T)$ and $L(\rho) \in \mathcal{L}_{pref}^\phi$. Therefore, to solve the 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 shortest-path-search-based LTL planning method crucially depends on that the mobility of the robot, or the environment map T is perfectly known. This method, however, is not suitable for the case of *partially-known* environments. To be specific, we consider a partially-known environment in the following setting:

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 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 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 a partially-known environment.

3.1 Partially-Known Weighted Transition Systems

Definition 1. (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 a WTS, X is the set of states with initial state $x_0 \in X$, $w : X \times X \to \mathbb{N}$ is the cost function and $L : X \to 2^{\mathcal{AP}}$ is a labeling function that assigns each state a set of atomic propositions. Different from the WTS, $\Delta : X \to 2^{2^X}$ 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-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 $\Delta(x) = \{o_1, \ldots, o_{|\Delta(x)|}\}$, 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 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. Therefore, we can 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.

In reality, the agent is moving in a specific environment that is compatible with the possible world $\mathbb{T}$, although itself does not know this a priori. Formally, 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 $\forall x \in X : \delta_T(x) \in \Delta(x)$. Clearly, if all states in $\mathbb{T}$ are known, then its compatible WTS is unique. Similar to Remark 1, we still assume that each compatible environment $T \in \mathbb{T}$ is fixed and connected.

3.2 History and Knowledge Updates

In the partially-known setting, the agent cannot make decision 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. Note that, 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. Only when the agent visits an unknown state, 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 call a tuple $\kappa = \langle x, o \rangle \in X \times 2^X$, where $o \in \Delta(x)$, a *knowledge* obtained when exploring state x, which means the agent knows that the successor states of x are o. For each knowledge κ, we denote by $\kappa(x)$ and $\kappa(o)$, respectively, its first and second components, i.e., $\kappa = \langle \kappa(x), \kappa(o) \rangle$. We denote by

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

the set of all possible knowledges.

A *history* in $\mathbb{T}$ is a finite sequence of knowledges

$$\hbar = \kappa_0 \kappa_1 \cdots \kappa_n = \langle x_0, o_0 \rangle \langle x_1, o_1 \rangle \cdots \langle x_n, o_n \rangle \in \mathsf{Kw}^* \quad (2)$$

such that

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

Intuitively, the first condition says that the agent can only go 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 will observe the same successor-pattern for different visits of the same state. For history $\hbar = \kappa_0 \kappa_1 \cdots \kappa_n \in \mathsf{Kw}^*$, we call $\kappa_0(x) \kappa_1(x) \cdots \kappa_n(x)$ its path. We denote by $\mathsf{Path}^*(\mathbb{T})$ and $\mathsf{Hist}^*(\mathbb{T})$ the set of all paths and histories of $\mathbb{T}$, respectively.

Along each history $\hbar = \kappa_0 \kappa_1 \cdots \kappa_n \in \mathsf{Kw}^*$, the agent obtain a set of knowledges (or simply, a knowledge-set)

$$\mathcal{K} = \{ \kappa_i \mid i = 0, \ldots, n \} \subseteq \mathsf{Kw}, \qquad (3)$$

which is an *unordered* set of knowledges. We define

$$\mathbb{KW} = \{ \mathcal{K} \in 2^{\mathsf{Kw}} \mid \forall \kappa, \kappa' \in \mathcal{K} : \kappa(x) = \kappa'(x) \Rightarrow \kappa(o) = \kappa'(o) \} \quad (4)$$

as the set of knowledge-sets. Therefore, given a knowledge-set $\mathcal{K} \in \mathbb{KW}$, we say state $x \in X$ has been explored in $\mathcal{K}$, if $\langle x, o \rangle \in \mathcal{K}$ for some o; we denote by $X(\mathcal{K})$ the set of explored states in $\mathcal{K}$. If x has been explored in $\mathcal{K}$, we denote by $o_{\mathcal{K}}(x) \in 2^X$ the unique observation such that $\langle x, o_{\mathcal{K}}(x) \rangle \in \mathcal{K}$.

Then the agent can maintain a finer possible world by incorporating with the knowledges it obtained. Specifically, by having knowledge-set $\mathcal{K} \in \mathbb{KW}$, the agent can update the PK-WTS $\mathbb{T} = (X, x_0, \Delta, w, \mathcal{AP}, L)$ to a finer PK-WTS

$$\mathbb{T}' = \mathsf{update}(\mathbb{T}, \mathcal{K}) = (X, x_0, \Delta', w, \mathcal{AP}, L),$$

where for any $x \in X$, we have

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

Note that, the above update function is well-defined since conflict knowledges $\langle x, o \rangle, \langle x, o' \rangle \in \mathsf{Kw}$ such that $o \neq o'$ cannot belong to the same knowledge-set by the definition of history and Equation (4).

Remark 2. Note that in the domain of knowledges as defined in Equation (1), we can omit those tuples $\langle x, o \rangle$ when $x \in X_{kno}$ is a known state since o here does not provide any additional information. Here, we still leave this reduction information when constructing a knowledge-set for the sake of unified description. One can easily omit this part when implementing our algorithm.

3.3 Strategy and Regret

Under the setting of partially-known environment, the plan is no longer an open-loop sequence. Instead, it is a *strategy* that determines the next state the agent should go to based what has been visited and what has known, which are environment dependent. Formally, a strategy is a function $\xi : \mathsf{Hist}^*(\mathbb{T}) \to X \cup \{stop\}$ such that for any $\hbar = \kappa_1 \cdots \kappa_n$, where $\kappa_i = \langle x_i, o_i \rangle$, either (i) $\xi(\hbar) \in o_n$, i.e., it decides to move to some successor state; or (ii) $\xi(\hbar) = stop$, i.e., the plan is terminated. We denote by $\mathsf{Stra}(\mathbb{T})$ the set of all strategies for $\mathbb{T}$.

Although a strategy is designed to handle all possible actual environments in the possible world $\mathbb{T}$, when it is applied to an actual environment $T \in \mathbb{T}$, the outcome of the strategy can be completely determined. We denote by $\rho_\xi^T = x_0 x_1 \cdots x_n \in X^*$ the finite path induced by strategy ξ in environment $T \in \mathbb{T}$, which is the unique path such that

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

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 ξ should satisfy

$$\forall T \in \mathbb{T} : \rho_\xi^T \in \mathcal{L}_{pref}^\phi. \qquad (5)$$

We denote by $\mathsf{Stra}_\phi(\mathbb{T})$ all strategies satisfying (5).

To evaluate the performance of strategy ξ, one approach is to consider the *worst-case cost* of the strategy among all possible environment, i.e., $\mathsf{cost}_{worst}(\xi) := \max_{T \in \mathbb{T}} \mathsf{cost}(\rho_\xi^T)$. However, as we have illustrated by the 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 capture this issue, in this work, we adopt the notion of *regret* as the metric to evaluate the performance of a strategy.

Definition 2. (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

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

4

The intuition of the above notion of regret is explained as follows. For each strategy ξ and each actual environment T, $\mathsf{cost}(\rho_\xi^T)$ is the actual cost incurred when applying this strategy to this specific environment, while $\min_{\xi' \in \mathsf{Stra}_\phi(\mathbb{T})} \mathsf{cost}(\rho_{\xi'}^T)$ 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 $T \in \mathbb{T}$.

3.4 Problem Formulation

After presenting the PK-WTS modeling framework as well as the regret-based performance metric, we are now ready to formulate the problem that we solve in this work.

Problem 1. (Regret-Based LTL Planning). Given a possible world represented by PK-WTS $\mathbb{T}$ and an scLTL task ϕ, synthesize a strategy ξ such that i) $\rho_\xi^T \in \mathcal{L}_{pref}^\phi$ for any $T \in \mathbb{T}$; and ii) $\mathrm{reg}_\mathbb{T}(\xi)$ is minimized.

4. GAME-BASED SYNTHESIS ALGORITHM

In this section, we solve the regret-based LTL planning problem. Our approach is to first build a knowledge-based game arena that is consistent with all possible histories. Then, based on the game arena, an effective value iteration algorithm is developed, which provides the optimal strategy with minimum regret.

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 $\delta_\mathcal{T}(x) = \bigcup \{o \in \Delta(x)\}$, i.e., 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 $\mathcal{A}_\phi$, denoted by

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

where the product "$\otimes$" has been defined in Section 2.3 and recall that its state-space is $S = X \times Q$.

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 following *knowledge-based game arena*.

Definition 3. (Knowledge-Based Game Arena). 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

- $V_a = X \times Q \times \mathbb{KW}$ is the set of *agent vertices*;

- $V_e = X \times Q \times \mathbb{KW} \times X$ is the set of *environment vertices*;
- $v_0 = (x_0, q_0, \kappa_0) \in V_a$ is the initial (agent) vertex, where $\kappa_0 \in \mathsf{Kw}$ is the initial knowledge of the agent, i.e., $\kappa_0(x) = x_0$ and $\Delta(x_0) = \{\kappa_0(o)\}$;
- $E \subseteq V \times V$ 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-1) if $x_a \in X(\mathcal{K}_e)$, then $\mathcal{K}_a = \mathcal{K}_e$;
    - (iii-2) if $x_a \notin X(\mathcal{K}_e)$, then we have
      $$\mathcal{K}_a \in \{\mathcal{K}_e \cup \{\langle x_a, o \rangle\} \mid o \in \Delta(x_a)\}.$$

The intuition of the knowledge-based game arena G 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 state 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 state, it has been explored and we have $x_a \in X(\mathcal{K}_a)$, i.e., 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 $v_e = (x_a, q_a, \mathcal{K}_a, \hat{x})$ by "remembering" the successor state $\hat{x} \in o_{\mathcal{K}_a}(x_a)$ 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 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 $x_a = \hat{x}_e$ has already been explored, then the agent must observe the same successor-pattern as before. Therefore, the knowledge-set is not updated;
- If state $x_a = \hat{x}_e$ has not yet been explored, then the new explored knowledge $\langle x_a, o \rangle \in \mathsf{Kw}$ should be added to the knowledge-set $\mathcal{K}_e$. However, 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, the resulting knowledge-set $\mathcal{K}_e$ is non-deterministic.

4.2 Strategies and Plays in the Game Arena

We call a finite sequence of vertices $\pi = v_0 v_1 \cdots v_n \in V^*$ a *play* on G if $\langle v_i, v_{i+1} \rangle \in E$ and we denote by $\mathsf{Play}^*(G)$ the set of all finite plays on G. We call π a *complete play* if $\mathsf{last}(\pi) \in V_a$, where $\mathsf{last}(\pi)$ denotes the last vertex in π. Then for a complete play $\pi = v_0 v_1 \cdots v_{2n} \in V^* V_a$, where

$v_{2i} = (x_i, q_i, \mathcal{K}_i), i = 0, \ldots, n$, it induces a path denoted by $\pi_{path} = x_0 x_1 \cdots x_n$ as well as a history

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

Note that, in the above, we have $\mathcal{K}_0 \subseteq \mathcal{K}_1 \subseteq \cdots \subseteq \mathcal{K}_n$ and the knowledge-set constructed along history π_{his} is exactly $\mathcal{K}_n$. On the other hand, for any history $\hbar = \kappa_0 \kappa_1 \cdots \kappa_n \in$ Kw*, there exists a unique complete play in G, denoted by $\pi_\hbar$, such that its induced history is $\hbar$.

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

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

the set of accepting vertices representing the satisfaction of the scLTL task. Also, since only edges from V_e to V_a represent actual movements, we define a weight function for G as $w_G : V \times V \to \mathbb{N}$, where for any $v_a = (x_a, q_a, \mathcal{K}_a)$ and $v_e = (x_e, q_e, \mathcal{K}_e, \hat{x}_e)$, we have $w_G(v_a, v_e) = 0$ and $w_G(v_e, v_a) = w(x_e, x_a)$. The the cost of a play $\pi = v_0 v_1 \cdots v_n \in V^*$ is defined as $\mathsf{cost}_G(\pi) = \sum_{i=0}^{n-1} w_G(v_i, v_{i+1})$.

A strategy for the agent-player is a function $\sigma_a : V^* V_a \to V_e \cup \{stop\}$ such that for any $\pi \in V^* V_a$, either $\langle \mathsf{last}(\pi), \sigma_a(\pi) \rangle \in E$ or $\sigma_a(\pi) = stop$. Analogously, a strategy for the environment-player is a function $\sigma_e : V^* V_e \to V_a$ such that for any $\pi \in V^* V_e$, we have $\langle \mathsf{last}(\pi), \sigma_e(\pi) \rangle \in E$. We denote by $\Sigma_a(G)$ and $\Sigma_e(G)$ the sets of all strategies for the agent and the environment respectively. In particular, a strategy $\sigma \in \Sigma_a(G) \cup \Sigma_e(G)$ is said to be positional if $\forall \pi, \pi' : \mathsf{last}(\pi) = \mathsf{last}(\pi') \Rightarrow \sigma(\pi) = \sigma(\pi')$ and we denote by $\Sigma_a^1(G)$ and $\Sigma_e^1(G)$ the corresponding sets of all positional strategies respectively. 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 $v_0 v_1 \cdots v_n \in V^* V_a$ s.t.

- $\forall i < n : v_i \in V_a \Rightarrow \sigma_a(v_0 v_1 \cdots v_i) = v_{i+1}$; and
- $\forall i < n : v_i \in V_e \Rightarrow \sigma_e(v_0 v_1 \cdots v_i) = v_{i+1}$; and
- $\sigma_a(v_0 v_1 \cdots v_n) = stop$.

For the environment-player, under assumption A3, we can further characterize its set of strategies as

$$\mathfrak{S}_e = \left\{ \sigma_e \in \Sigma_e^1(G) : \begin{array}{l} \forall v_e, v_e' \in V_e : X(v_e) = X(v_e') \\ \Rightarrow X(\sigma_e(v_e)) = X(\sigma_e(v_e')) \end{array} \right\} \quad (7)$$

where we denote by $X(\cdot)$ the first component of an agent vertex $v_a = (x_a, q_a, \mathcal{K}_a) \in V_a$ as well as an environment vertex $v_e = (x_e, q_e, \mathcal{K}_e, \hat{x}_e) \in V_e$, i.e., for such v_a, v_e, we have $X(v_a) = x_a$ and $X(v_e) = x_e$. For the agent-player, we say σ_a is winning if for any $\sigma_e \in \mathfrak{S}_e$, we have $\mathsf{last}(\pi_{\sigma_a, \sigma_e}) \in V_F$. We denote by $\mathfrak{S}_a \subseteq \Sigma_a(G)$ the set of all winning strategies. By Remark 1, we know $|\mathfrak{S}_a^F| \geq 1$. Similarly to Definition 2, we can also define the *regret* of strategy $\sigma_a \in \mathfrak{S}_a$ in G by

$$\mathrm{reg}_G(\sigma_a) = \max_{\sigma_e \in \mathfrak{S}_e} \left(\mathsf{cost}_G(\pi_{\sigma_a, \sigma_e}) - \min_{\sigma_a'} \mathsf{cost}_G(\pi_{\sigma_a, \sigma_e}) \right) \quad (8)$$

In particular, we denote by $\mathrm{reg}_G^{\sigma_e}(\sigma_a) := \mathsf{cost}_G(\pi_{\sigma_a, \sigma_e}) - \min_{\sigma_a' \in \mathfrak{S}_a} \mathsf{cost}_G(\pi_{\sigma_a, \sigma_e})$ the regret of agent strategy $\sigma_a \in \mathfrak{S}_a$ w.r.t. an environment strategy $\sigma_e \in \mathfrak{S}_e$.

Essentially, an agent-player's strategy $\sigma_a \in \mathfrak{S}_a$ uniquely defines a corresponding strategy in $\mathbb{T}$, denoted by $\xi_{\sigma_a} \in$ Stra$_\phi(\mathbb{T})$ as follows: for any $\hbar \in$ Hist$^*(\mathbb{T})$, we have $\xi_{\sigma_a}(\hbar) = \sigma_a(\pi_\hbar)$. The environment-player's strategy $\sigma_e \in \mathfrak{S}_e$ 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. Since $\mathsf{cost}_G(\cdot)$ is defined only according to its first component, for any play $\pi \in V^* V_a$, we have $\mathsf{cost}_G(\pi) = \mathsf{cost}(\pi_{path})$.

In summary, in order to solve Problem 1, it suffices to find a winning strategy $\sigma_a \in \mathfrak{S}_a$ of the agent-player in the knowledge-based game arena G, such that $\mathrm{reg}_G(\sigma_a)$ as defined in (8) is minimized. Then, based on the optimal strategy $\sigma_a \in \mathfrak{S}_a$, we can induce a strategy $\xi_{\sigma_a} \in$ Stra$(\mathbb{T})$ for PK-WTS $\mathbb{T}$. Therefore, hereafter, we will only focus on the game graph G without considering $\mathbb{T}$. Furthermore, it suffices to find a *positional winning strategy* $\sigma_a \in \mathfrak{S}_a \cap \Sigma_a^1(G) := \mathfrak{S}_a^1$ for the agent-player since it is unnecessary to record the choices of the environment-player based on the construction of the knowledge-based game arena G.

4.3 Synthesis via Min-Max Games

Since the optimal strategy on the arena G, we denote by σ_a^*, satisfies $\mathrm{reg}(\sigma_a^*) = \min_{\sigma_a \in \mathfrak{S}_a} \mathrm{reg}_G(\sigma_a)$, it is natural for us to reduce the strategy synthesis problem to the solution of a *min-max reachability game* Brihaye et al. (2017) in which the optimal strategy σ^* satisfies

$$\max_{\sigma_e \in \mathfrak{S}_e} W(\pi_{\sigma^*, \sigma_e}) = \min_{\sigma_a \in \mathfrak{S}_a} \max_{\sigma_e \in \mathfrak{S}_e} W(\pi_{\sigma_a, \sigma_e}) \quad (9)$$

where $W : \mathsf{Play}^*(G) \to \mathbb{N}$ is some cost function. To this end, given knowledge-based game arena G, we first define the *best response* w.r.t. a knowledge $\mathcal{K}_a \in \mathbb{KW}$ as

$$br(\mathcal{K}_a) = \textsc{ShortestPath}(V_{x_0}(\mathcal{K}_a), V_F) \quad (10)$$

where $V_{x_0}(\mathcal{K}_a) := \{ v_a \in V_a \mid X(v_a) = x_0 \wedge \mathcal{K}(v_a) = \mathcal{K}_a \}$ is the set of agent vertices whose first component are x_0 and third component is $\mathcal{K}_a$ and we denote by $\mathcal{K}(\cdot)$ the third component of an agent vertex. Given two subsets $V_1, V_2 \subseteq V$ on G, the shortest path between them $\textsc{ShortestPath}(V_1, V_2)$ can be directly computed via a Dijkstra's algorithm.

For the agent-player, the cost of all its positional winning strategies is bounded by a constant $B = M^G |V| \in \mathbb{N}$, where $M^G := \max_{(v, v') \in E} w_G(v, v')$ denotes the largest cost of the transitions in G. To this end, we define $\hat{G} = (\hat{V}, \hat{v}_0, \hat{E})$ as the *information game arena*, where $\hat{V} \subseteq V \times [0, B]$, $\hat{v}_0 = (v_0, 0)$, $\hat{E} \subseteq \hat{V} \times \hat{V}$ is defined by: for any $\hat{v} = (v, u), \hat{v}' = (v', u') \in \hat{V}$, we have $(\hat{v}, \hat{v}') \in \hat{E}$ if i) $(v, v') \in E$ and ii) $u' = u + w_G(v, v')$. We also have partition $\hat{V} = \hat{V}_a \dot{\cup} \hat{V}_e$, based on whether the first component of each vertex belongs to V_a or V_e. Given $\hat{G}$, we define cost function $\mu : \hat{E} \to \mathbb{N}$ by

- for any $(\hat{v}, \hat{v}') \in \hat{E} : \hat{v}' \notin \hat{V}_F$, we have $\mu(\hat{v}, \hat{v}') = 0$;
- for any $(\hat{v}, \hat{v}') \in \hat{E} : \hat{v}' = ((x, q, \mathcal{K}), u) \in \hat{V}_F$, we have

$$\mu(\hat{v}, \hat{v}') = u - br(\mathcal{K}). \quad (11)$$

Based on the above construction, one can easily build the "one-to-one correspondence" between a play ρ in G and the play $\hat{\rho}$ in $\hat{G}$, as well as strategies σ and $\hat{\sigma}$, which is omitted here. Given a play $\hat{\rho} = \hat{v}_0 \hat{v}_1 \cdots \hat{v}_n \in \mathsf{Play}^*(\hat{G})$, we denote its cost w.r.t μ by $\mathsf{cost}_{\hat{G}}^\mu(\hat{\rho}) := \sum_{i=0}^n \mu(\hat{v}_i, \hat{v}_{i+1})$. Then we obtain the following result and the detailed proof is given in Appendix A.

Proposition 1. Given game arena G with the information game arena $\hat{G}$ and cost function $\mu : E \to \mathbb{N}$, for each $\sigma_a \in \mathfrak{S}_a^1$, we have

$$\text{reg}_G(\sigma_a) = \max_{\hat{\sigma}_e \in \hat{\mathfrak{S}}_e} \text{cost}_{\hat{G}}^\mu(\pi_{\hat{\sigma}_a, \hat{\sigma}_e}) \tag{12}$$

With the result of Proposition 1, to obtain the optimal regret-minimizing strategy, it suffices for us to solve a min-max reachability game on $\hat{G}$ w.r.t. cost function μ. Now, we summarize our solution as Algorithm 1 and obtain the following result and the proof is given in Appendix B.

Theorem 1. The strategy ξ^* obtained from Algorithm 1 correctly solves the Regret-Based LTL Planning Problem defined in Problem 1.

Algorithm 1: Optimal Regret-Minimizing LTL Plan

Input: PK-WTS $\mathbb{T}$ and scLTL ϕ;
**Output:** Optimal Regret-Minimizing Plan ξ^*
1 Construct skeleton-WTS $\mathcal{T}$ and DFA $\mathcal{A}_\phi$ from ϕ;
2 Construct product system $\mathcal{P} = \mathcal{T} \otimes \mathcal{A}_\phi$;
3 Construct knowledge-based game arena G;
4 Construct information game arena $\hat{G}$;
5 **for** $(\hat{v}, \hat{v}') \in \hat{E}$ **do**
6 $\quad$ define its cost $\mu(\hat{v}, \hat{v}')$;
7 Obtain strategy $\hat{\sigma}_a^* = \texttt{SolveMinMax}(\hat{G}, \mu)$;
8 Obtain strategy σ_a^* from $\hat{\sigma}_a^*$;
9 Obtain strategy $\xi^* = \xi_{\sigma_a^*}$;
10 **procedure** $\texttt{SolveMinMax}(\hat{G}, \mu)$
11 **for** $\hat{v} \in \hat{V}$ **do**
12 $\quad$ **if** $\hat{v} \in \hat{V}_F$ **then**
13 $\quad\quad$ $W^{(0)}(\hat{v}) = 0$ and $\hat{\sigma}_a^{(0)}(\hat{v}) = stop$;
14 $\quad$ **else**
15 $\quad\quad$ $W^{(0)}(\hat{v}) = \infty$;
16 **while** $\exists \hat{v} \in \hat{V} : W^{(k+1)}(\hat{v}) \neq W^{(k)}(\hat{v})$ **do**
17 $\quad$ **for** $\hat{v}_e \in \hat{V}_e$ **do**
18 $\quad\quad$ $W^{(k+1)}(\hat{v}_e) = \max_{\hat{v}_a \in \text{Succ}(\hat{v}_e)} \left(W^{(k)}(\hat{v}_a) + \mu(\hat{v}_e, \hat{v}_a) \right)$;
19 $\quad$ **for** $\hat{v}_a \in \hat{V}_a$ **do**
20 $\quad\quad$ **if** $\hat{v}_a \in \hat{V}_a$ **then**
21 $\quad\quad\quad$ $W^{(k+1)}(\hat{v}_a) = 0$ and $\hat{\sigma}_a^{(k+1)}(\hat{v}_a) = stop$;
22 $\quad\quad$ **else**
23 $\quad\quad\quad$ $W^{(k+1)}(\hat{v}_a) = \min_{\hat{v}_e \in \text{Succ}(\hat{v}_a)} \left(W^{(k)}(\hat{v}_e) + \mu(\hat{v}_e, \hat{v}_a) \right)$;
24 $\quad\quad\quad$ $\hat{\sigma}_a^{(k+1)}(\hat{v}_a) = \arg\min_{\hat{v}_e \in \text{Succ}(v_a)} \left(W^{(k)}(\hat{v}_e) + \mu(\hat{v}_e, \hat{v}_a) \right)$;
25 $\quad$ $k \leftarrow k + 1$;
26 **return** $\sigma_a^{(k)}$;

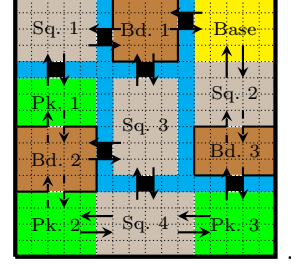
5. CASE STUDY: A TEAM OF FIREFIGHTING ROBOTS

In this section, 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 working in an urban district shown in Figure 2(a). Specifically, the blue regions are *rivers*, the black regions are *bridges*, the grey regions are *squares*, the green regions are *parks*,
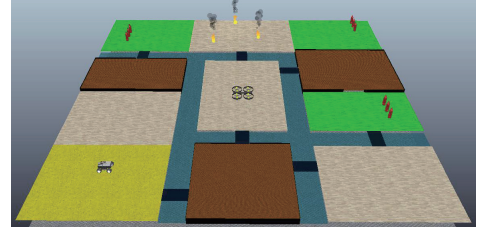
(a) Firefighting Scenario.
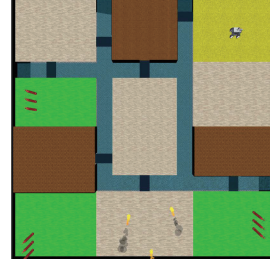
(b) Perspective from UAV. (c) Possible World $\mathbb{T}$.

Fig. 2. Simulation Setting.

the brown regions are *buildings* and the yellow region is the *base* of the team of robots. The firefighting mission in this district is undertaken by the collaboration of the UAV and the ground robot. Specifically, we assume that the district map is completely unknown to the robotic system initially. When a fire alarm is reported, the UAV takes off first and reconnoiters over the district, which allows the system to obtain some rough information of the distinct and leads to a possible world map. More detailed connectivities for some unknown regions in the possible world still remain to be explored by the ground robot. In order to accomplish the firefighting mission, the ground robot needs to first go to some regions with extinguisher to get fire-extinguishers and then move to the region with fire. Let $\mathcal{AP} = \{\text{fire}, \text{extinguisher}\}$. The mission can be described by the following scLTL formula:

$$\phi = (\neg\text{fire } U \text{ extinguisher}) \wedge \Diamond\text{fire} \tag{13}$$

Suppose that, after the reconnaissance, the UAV will get a look down picture of the entire district as shown in Figure 2(b). Based on the distinct picture, the system will know the map geometry and the semantics. Specifically, it knows that there is a fire in Square 4 and there are extinguishers in Parks 1-3. The connectivities of all open regions including rivers, bridges, squares, parks and the base are known. However, the UAV cannot tell if each building has corresponding doors connecting with its adjacent open regions. In order to figure out the (non-)existence of those potential doors, the ground robot has to move to the adjacent areas to explore. Therefore, the possible world model $\mathbb{T}$ of the robots is shown in Figure 2(c), where the dashed arrows denote some potential doors, whose existences are unknown.

Now, the environment is partially-known in the sense that the accessibility of the buildings are unknown to the robotic system until the ground robot reaches their adjacent regions. Then based on the possible world model $\mathbb{T}$ and the scLTL ϕ, we can synthesize an strategy ξ^* that minimizes the regret while achieving ϕ. We have implemented our algorithm in robot simulator V-REP (Videos are available at https://youtu.be/lLRT2pLfABA). As shown
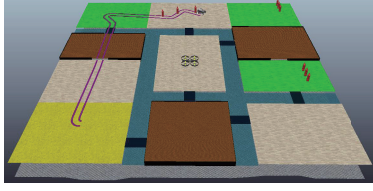
(a) Trajectory in Environment T_1
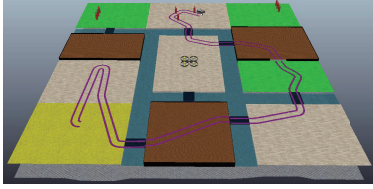
(b) Trajectory in Environment T_2

Fig. 3. Simulation Results

in Figure 3, we consider two different actual environments $T_1, T_2 \in \mathbb{T}$ compatible to the possible world $\mathbb{T}$ when applying the same strategy ξ^*. To minimize its regret, the ground robot needs to first go to Square 2 to explore if there exists a door connecting Building 3. Environment T_1 in Figure 3(a) actually corresponds to the case of the existence of such a door. Then by passing through this door, the ground robot can easily find a extinguisher in Park 3 and then quench the fire in Square 4. This actual path is shorter than the path planned based on the worst-case by assuming no such a door exists. However, environment T_2 in Figure 3(b) corresponds to the case, where there is no door between Square 2 and Building 3. Note that, the ground robot will still first go to Square 2. However, when it realizes that there is no such a door, it will turn back and go to Park 1, where it finds a extinguisher and luckily, there is a door to Building 2. Then the ground robot will cross through Building 2 and finally reaches Square 4 to quench the fire. Although this actual path is longer than the path planned based on the worst-case, the robot will not regret that much since it does not know, *a priori*, the non-existence of such a door.

6. CONCLUSIONS

In this paper, we proposed a new approach for optimal path planning for scLTL specifications under partially-known environments. We adopted the notion of regret to evaluate the trade-off between cost incurred in an actual environment and the potential benefit of exploring unknown regions. A knowledge-based model was developed to formally describe the partially-known scenario and an effective algorithm was proposed to synthesize an optimal strategy with minimum regret. In the future, we would like to extend our results to multi-agent systems with general LTL specifications.

REFERENCES

Baier, C. and Katoen, J.P. (2008). *Principles of Model Checking*. MIT press.

Brihaye, T., Geeraerts, G., Haddad, A., and Monmege, B. (2017). Pseudopolynomial iterative algorithm to solve total-payoff games and min-cost reachability games. *Acta Informatica*, 54(1), 85–125.

Cadilhac, M., Pérez, G.A., and Bogaard, M.v.d. (2019). The impatient may use limited optimism to minimize regret. In *International Conference on Foundations of Software Science and Computation Structures*, 133–149. Springer.

Cai, M., Peng, H., Li, Z., and Kan, Z. (2021). Learning-based probabilistic LTL motion planning with environment and motion uncertainties. *IEEE Trans. Automatic Control*, 66(5), 2386–2392.

Filiot, E., Gall, T.L., and Raskin, J.F. (2010). Iterated regret minimization in game graphs. In *International Symposium on Mathematical Foundations of Computer Science*, 342–354.

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.

Guo, M. and Dimarogonas, D.V. (2015). Multi-agent plan reconfiguration under local ltl specifications. *The International Journal of Robotics Research*, 34(2), 218–235.

Guo, M. and Zavlanos, M.M. (2018). Probabilistic motion planning under temporal tasks and soft constraints. *IEEE Transactions on Automatic Control*, 63(12), 4051–4066.

Hunter, P., Pérez, G.A., and Raskin, J.F. (2017). Reactive synthesis without regret. *Acta Informatica*, 54(1), 3–39.

Kantaros, Y., Kalluraya, S., Jin, Q., and Pappas, G.J. (2022). Perception-based temporal logic planning in uncertain semantic maps. *IEEE Transactions on Robotics*.

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.

Lahijanian, M., Maly, M.R., Fried, D., Kavraki, L.E., Kress-Gazit, H., and Vardi, M.Y. (2016). Iterative temporal planning in uncertain environments with partial satisfaction guarantees. *IEEE Transactions on Robotics*, 32(3), 583–599.

MacDonald, R.A. and Smith, S.L. (2019). Active sensing for motion planning in uncertain environments via mutual information policies. *The International Journal of Robotics Research*, 38(2-3), 146–161.

Mahulea, C., Kloetzer, M., and González, R. (2020). *Path Planning of Cooperative Mobile Robots Using Discrete Event Models*. John Wiley & Sons.

Muvvala, K., Amorese, P., and Lahijanian, M. (2022). Let's collaborate: Regret-based reactive synthesis for robotic manipulation. *arXiv preprint arXiv:2203.06861*.

Smith, S.L., Tumová, J., Belta, C., and Rus, D. (2011). Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research*, 30(14), 1695–1708.

Yannakakis, G.N. and Togelius, J. (2018). *Artificial Intelligence and Games*, volume 2. Springer.

Yu, P. and Dimarogonas, D.V. (2022). Distributed motion coordination for multirobot systems under LTL specifications. *IEEE Transactions on Robotics*.

Yu, X., Yin, X., Li, S., and Li, Z. (2022). Security-preserving multi-agent coordination for complex temporal logic tasks. *Control Engineering Practice*, 123, 105130.

Appendix A. PROOF OF PROPOSITION 1

In what follows, we first present three lemmas based on which we finish the proof of Proposition 1.

Lemma 1. A *positional* winning strategy $\sigma_a \in \mathfrak{S}_a^1$ can minimize the regret of the agent-player in G.

Proof. Consider a winning strategy for the agent-player $\sigma_a^F \in \mathfrak{S}_a$, which means that for any $\sigma_e \in \mathfrak{S}_e(G)$, the *outcome play* $\pi_{\sigma_a^F, \sigma_e}$ satisfies $\mathsf{last}(\pi_{\sigma_a^F, \sigma_e}) \in V_F$. The existence of such winning strategy σ_a^F can be guaranteed by Remark 1. It is obvious that strategy σ_a^F needs at most finite memory since all the outcome plays $\pi_{\sigma_a^F, \sigma_e}, \forall \sigma_e \in \mathfrak{S}_e$ are finite. Then we show the existence of a memoryless strategy $\sigma_a : V_a \to V_e$ corresponding to σ_a^F such that

$$\forall \sigma_e \in \mathfrak{S}_e : \mathsf{cost}_G(\pi_{\sigma_a, \sigma_e}) \leq \mathsf{cost}_G(\pi_{\sigma_a^F, \sigma_e}). \quad (A.1)$$

We define σ_a by: for any $\sigma_e \in \mathfrak{S}_e$ with $\pi_{\sigma_a^F, \sigma_e} = v_0 v_1 \cdots v_n$ being the output play and for all $v_k \in V_a$, we have

$$\nexists i \geq k : v_i = v_k \Rightarrow \sigma_a(v_k) = v_{k+1}. \quad (A.2)$$

It is obvious that strategy σ_a is well defined. Furthermore, for any $\sigma_e \in \mathfrak{S}_e$, the output play π_{σ_a, σ_e} only visits the states in $\pi_{\sigma_a^F, \sigma_e}$, since the environment-player plays positional strategies. Then (A.1) holds since we obtain π_{σ_a, σ_e} by removing all cycles in $\pi_{\sigma_a^F, \sigma_e}$ based on the above construction. It directly follows that

$$\forall \sigma_e \in \mathfrak{S}_e : \mathrm{reg}_G^{\sigma_e}(\sigma_a) \leq \mathrm{reg}_G^{\sigma_e}(\sigma_a^F). \quad (A.3)$$

Then we have

$$\mathrm{reg}_G(\sigma_a) \leq \mathrm{reg}_G(\sigma_a^F). \quad (A.4)$$

That is, given any winning strategy for the agent-player, we can always find a positional strategy making (A.4) hold. The proof is thus completed.

Therefore, to synthesize a strategy that minimizes the regret, it suffices to consider positional winning strategies on G. For each positional winning strategy, we aim to compute its regret on game arena G. However, given an agent-strategy $\sigma_a \in \mathfrak{S}_a$, it is typically to compute its the cost against all environment-player strategies $\sigma_e \in \mathfrak{S}_e$. In fact, given an agent-player strategy, there may be more than one environment-player strategies resulting in the same output play. For those environment-player strategies, it suffices to consider an σ_e' that minimizes the regret $\mathrm{reg}_G^{\sigma_e}(\sigma_a)$. Therefore, we have the following result.

Lemma 2. Given $\sigma_a \in \mathfrak{S}_a^1$, for any $\sigma_e \in \mathfrak{S}_e$ resulting in the same output play π_{σ_a, σ_e}, we have

$$\mathrm{reg}_G^{\sigma_e}(\sigma_a) \geq \mathsf{cost}_G(\pi_{\sigma_a, \sigma_e}) - br(\mathcal{K}(\mathsf{last}(\pi_{\sigma_a, \sigma_e'}))) \quad (A.5)$$

Specifically, there exists $\sigma_e' \in \mathfrak{S}_e$ such that

$$\min_{\sigma_a \in \mathfrak{S}_a} \mathsf{cost}_G(\pi_{\sigma_a, \sigma_e'}) = br(\mathcal{K}(\mathsf{last}(\pi_{\sigma_a, \sigma_e'}))) \quad (A.6)$$

and thus $\mathrm{reg}_G^{\sigma_e'}(\sigma_a) = \mathsf{cost}_G(\pi_{\sigma_a, \sigma_e'}) - br(\mathcal{K}(\mathsf{last}(\pi_{\sigma_a, \sigma_e'})))$.

Proof. Based on the construction of the knowledge-based game arena, we know that, there is a "one-to-one" correspondence between an environment-player strategy $\sigma_e \in \mathfrak{S}_e$ and an actual environment $T \in \mathbb{T}$. For each actual environment T, we denote by $\mathcal{K}^T$ the knowledge obtained by the agent after exploring all the unknown states. Given a play $\pi = v_0 v_1 \cdots v_n \in \mathsf{Play}^*(G)$, each $v_i \in V_e$ such that $\mathsf{Succ}(v_e) \geq 2$ represents an update of the knowledge, i.e., for such $v_i = (x_i, q_i, \mathcal{K}_i, \hat{x}_i)$ and $v_{i+1} = (x_{i+1}, q_{i+1}, \mathcal{K}_{i+1})$,

we have $\mathcal{K}_i \subset \mathcal{K}_{i+1}$. Accordingly, we know that, for each knowledge $\mathcal{K} \in \mathbb{KW}$, there are a sequence of knowledge updates $\mathcal{K}_1, \mathcal{K}_2, \ldots$ such that

   i) $\mathcal{K} \subset \mathcal{K}_1 \subset \cdots \subset \mathcal{K}_k$;
   ii) $\mathcal{K}_k = \mathcal{K}^T$ for some $T \in \mathbb{T}$.

In particular, we have $k = 0$ if $\mathcal{K} = \mathcal{K}^T$ for some $T \in \mathbb{T}$ initially. Therefore, given the two strategies σ_a and σ_e with $\mathsf{last}(\pi_{\sigma_a, \sigma_e}) := (x, q, \mathcal{K})$, we denote by $T_1, T_2, \ldots, T_m \in \mathbb{T}$ the actual environments such that ii) could hold. Furthermore, we denote by $\sigma_e^1, \sigma_e^2, \ldots, \sigma_e^m \in \mathfrak{S}_e$ the corresponding environment-player strategies. Then it follows that

$$\pi_{\sigma_a, \sigma_e^i} = \pi_{\sigma_a, \sigma_e}, \forall i = 1, \ldots, m \quad (A.7)$$

Obviously, we have $\mathsf{cost}_G(\pi_{\sigma_a, \sigma_e^i}) = \mathsf{cost}_G(\pi_{\sigma_a, \sigma_e})$ and thus

$$\mathrm{reg}^{\sigma_e^i}(\sigma_a) = \mathsf{cost}_G(\pi_{\sigma_a, \sigma_e}) - \min_{\sigma_a' \in \mathfrak{S}_a} \mathsf{cost}_G(\pi_{\sigma_a', \sigma_e^i})$$

Based on the construction of $br(\mathcal{K})$, we have

$$br(\mathcal{K}) = \min_{i \in \{1, \ldots, m\}} \min_{\sigma_a' \in \mathfrak{S}_a} \mathsf{cost}_G(\pi_{\sigma_a', \sigma_e^i}) \quad (A.8)$$

It directly follows that

$$\mathrm{reg}^{\sigma_e^i}(\sigma_a) \geq \mathsf{cost}_G(\pi_{\sigma_a, \sigma_e}) - br(\mathcal{K}) \quad (A.9)$$

Therefore, (A.5) holds. Furthermore, (A.6) also holds by denoting $\sigma_e' = \arg \min_{i \in \{1, \ldots, m\}} \mathrm{reg}^{\sigma_e^i}(\sigma_a)$. The proof is thus completed.

To this end, given an agent-player strategy $\sigma_a \in \mathfrak{S}_a$, since $\mathrm{reg}(\sigma_a) = \max_{\sigma_e \in \mathfrak{S}_e} \mathrm{reg}_G^{\sigma_e}(\sigma_a)$, it suffices to consider the environment-player strategies σ_e' such that (A.6) holds.

On the other hand, based on the construction of $\hat{G}$, we directly have the following result.

Lemma 3. Given knowledge-based game arena G with its information game arena being $\hat{G}$, for any $\sigma_a \in \mathfrak{S}_a^1$ and any $\sigma_e \in \mathfrak{S}_e$, we have

$$\mathsf{cost}_G(\pi_{\sigma_a, \sigma_e}) = \mathsf{cost}_{\hat{G}}(\hat{\pi}_{\hat{\sigma}_a, \hat{\sigma}_e}) \quad (A.10)$$

where $\hat{\sigma}_a$ and $\hat{\sigma}_e$ denote the corresponding strategies of σ_a and σ_e in $\hat{G}$, respectively.

By Lemma 2 and Lemma 3, we know that, given a strategy $\sigma_a \in \mathfrak{S}_a^1$, for any $\sigma_e' \in \mathfrak{S}_e$ satisfying (A.5), we have

$$\mathrm{reg}_G^{\sigma_e'}(\sigma_a) = \mathsf{cost}_{\hat{G}}(\hat{\pi}_{\hat{\sigma}_a, \hat{\sigma}_e'}) - br(\mathcal{K}(\mathsf{last}(\pi_{\sigma_a, \sigma_e'}))) \quad (A.11)$$

Based on the construction of $\mu : \hat{E} \to \mathbb{N}$, we have

$$\mathrm{reg}_G^{\sigma_e'}(\sigma_a) = \mathsf{cost}_{\hat{G}}^{\mu}(\hat{\pi}_{\hat{\sigma}_a, \hat{\sigma}_e'}) \quad (A.12)$$

Accordingly, we have

$$\mathrm{reg}_G(\sigma_a) = \max_{\hat{\sigma}_e \in \hat{\mathfrak{S}}_e} \mathsf{cost}_{\hat{G}}^{\mu}(\hat{\pi}_{\hat{\sigma}_a, \hat{\sigma}_e}) \quad (A.13)$$

Above all, the proof of Proposition 1 is thus completed.

Appendix B. PROOF OF THEOREM 1

Based on the result of Proposition 1, we know that, to synthesize a regret-minimizing strategies on G, it suffices to synthesize a strategy that solves the *min-max game* on the information game arena $\hat{G}$ with respect to cost function $\mu : \hat{E} \to \mathbb{N}$. Furthermore, the correctness of the procedure $\texttt{SolveMinMax}(\hat{G}, \mu)$ could be easily validated according to Brihaye et al. (2017). The proof is thus completed.