2024 IEEE 63rd Conference on Decision and Control (CDC)
December 16-19, 2024. MiCo, Milan, Italy

Reactive Planning for Teams of Heterogeneous Robots with Dynamic Collaborative Temporal Logic Missions

Yuqing Zhang, Samarth Kalluraya, George J. Pappas, Yiannis Kantaros

Abstract— **Several task and motion planning algorithms have been proposed recently for teams of robots assigned to collaborative high-level tasks specified using Linear Temporal Logic (LTL). However, the majority of prior works cannot effectively adapt to new missions that may arise in the field due to unexpected service requests. To address this novel challenge, we propose a reactive planning algorithm for teams of heterogeneous robots with collaborative LTL-encoded missions that dynamically change. The robots are heterogeneous in terms of their skills while the mission requires them to apply these skills in specific regions/objects in a temporal/logical order. Our method designs paths that can adapt to unexpected changes in the mission and effectively address potential mission violations arising due to conflicting logical task requirements or a limited number of robots. We achieve this by locally allocating new sub-tasks to the robots based on their capabilities, minimizing disruptions to the existing team plan, and strategically prioritizing the most crucial sub-tasks according to user-specified priorities. We provide theoretical guarantees and numerical experiments to demonstrate the efficiency of our method.**

I. Introduction

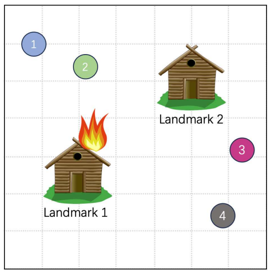
Linear Temporal Logic (LTL) has emerged as one of the main approaches to define robot tasks with temporal and logical requirements that go beyond the classical reach-avoid ones [1]. Temporal logic planning methods initially focused on robots operating in known environments [2]–[8]. These works have been extended to handle unknown static environments [9]–[11], unknown dynamic environments [12]–[14], unknown system dynamics [15]–[17], and unexpected robot failures [18]–[21]. A recent survey can be found in [22]. A key assumption in these works is that the task remains fixed over time. However, this may not hold in practice in case of unexpected mission disruptions or service requests.

To tackle this novel challenge, we present a multi-robot planning algorithm designed to adapt to unforeseen mission changes. Specifically, we consider robots with heterogeneous capabilities (e.g., mobility, sensing, or manipulation) tasked with a collaborative nominal mission encoded as an LTL formula. The mission requires them to apply their skills at specific areas or objects in a known environment. New tasks, not necessarily pre-assigned to robots, may emerge unexpectedly, modeled as LTL formulas. This gives rise to a new mission defined as the conjunction of the nominal and the new tasks. Our goal is to design online paths that can adapt to these dynamic mission requirements. A key challenge is determining which sub-task each robot should undertake, as soon as new tasks are announced, to minimize potential mission violations. Such violations can occur either due to conflicting logical requirements or due to limited number of available robots. To address this, we propose a joint task re-allocation and re-planning framework. First, the task re-allocation algorithm assigns robots to the new sub-tasks. To minimize potential mission violations, previous sub-tasks may be re-assigned to different robots, prioritizing critical sub-tasks based on user-defined priorities. We show that this algorithm minimizes disruptions to the team's behavior by minimizing the number of task re-assignments. Second, after the sub-tasks are re-assigned, we revise the existing robot plans to accommodate the new mission requirements. We validate the efficiency of the proposed algorithm both theoretically and via numerical experiments.

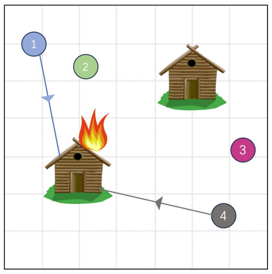
[1]Authors are with the Department of Electrical and Systems Engineering, Washington University in St. Louis, St. Louis, MO, USA. zyuqing,k.samarth,ioannisk@wustl.edu. [2]Author is with the Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA, USA. pappasg@seas.upenn.edu.
This work was supported by the ARL grant DCIST CRA W911NF-17-2-0181 and the NSF grant CCF #2403758.

Related Works: Several task allocation methods have been proposed that assign either individual LTL tasks [23], [24] or sub-tasks of a collaborative LTL mission [25], [26] to robots. These approaches typically perform task assignment offline and do not consider online changes in the mission. While these methods can be used to globally re-allocate sub-tasks when new missions are announced, the computational cost of global task re-assignment at runtime may render it impractical. To the contrary, our method attempts to minimize the number of task re-assignments. Related are also the works on designing least-violating plans [27]–[31]. These works design *single-robot* plans that minimally violate temporal logic specifications in the presence of timing and environmental constraints, or exogenous disturbances that may render certain parts of a *fixed* mission hard or impossible to satisfy. In contrast, this paper focuses on designing minimum-violation *multi-robot* paths in the presence of *dynamic* mission requirements. Closer to our approach are the reactive planning algorithms proposed in [32], [33]. These algorithms design plans ensuring that a nominal/global temporal logic specification is always satisfied while also adapting the plans to accomplish local service requests. However, unlike our method, they consider single-robot systems and assume that mission conflicts do not occur.

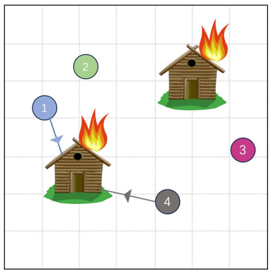
Contribution: *First*, we address a new planning problem for teams of heterogeneous robots with dynamically changing temporal logic missions. *Second*, we propose a task re-allocation and re-planning algorithm that can (asymptotically) compute minimum-violation plans in cases of mission conflicts or limited number of available robots. *Third*, we demonstrate the efficiency of our algorithm through numerical experiments.



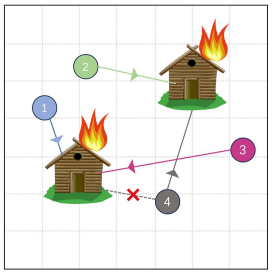
(a) Starting positions



(b) Offline-designed paths



(c) New tasks released



(d) Online revised paths

Fig. 1. Consider 4 robots, illustrated as colored disks. The set of skills is $\mathcal{C} = \{c_0, c_1, c_2, c_3\}$, where c_0, c_1, c_2, and c_3 refer to mobility, fire extinguishing, taking photos, and object recognition skills. The robots can be divided into 4 groups based on their skills: $\mathcal{T}_{c_0} = \{1, 2, 3, 4\}$, $\mathcal{T}_{c_1} = \{1, 2, 3\}$, $\mathcal{T}_{c_2} = \{3, 4\}$, $\mathcal{T}_{c_3} = \{4\}$. The initial task is $\phi_{\text{cur}}(0) = \Diamond[\pi_{c_1}(1, \ell_1) \wedge \Diamond(\pi_{c_2}(4, \ell_1))]$, which requires eventually robot 1 to extinguish the fire in landmark 1 and then robot 4 to take photos. The offline-designed robot paths for this task are shown in Fig. 1(b). At time t, while the robots are still on the way to ℓ_1, a fire also breaks out at landmark ℓ_2, prompting an additional mission $\phi_{\text{new}}(t) = \Diamond\pi_{c_1}(\ell_2) \wedge [\neg\pi_{c_1}(\ell_2)\mathcal{U}\pi_{c_3}(\ell_2)]$; see Fig. 1(c). This task requires the robots to recognize the object under fire at ℓ_2 and then extinguish the fire. Addressing this problem requires assigning new tasks to the robots and then revising the original paths. For instance, here, robot 2 can take over the fire extinguishing task $\pi_{c_1}(\ell_2)$ since it is currently free. However, the task $\pi_{c_3}(\ell_2)$ of identifying the object can be undertaken only by robot 4, which, however, is currently engaged with a take-photo task $\pi_{c_2}(4, \ell_1)$. This requires re-assigning the previous take-photo task to the robot 3 which is currently free. Here, mission violations did not occur; see Sec. II-C.

II. Problem Formulation

A. Teams of Heterogeneous Robots

Consider a team of N robots with the following dynamics: $\mathbf{p}_j(t+1) = \mathbf{f}_j(\mathbf{p}_j(t), \mathbf{u}_j(t)), j \in \mathcal{R} = \{1, 2, ..., N\}$, where $\mathbf{p}_j(t) \in \mathbb{R}^m$ and $\mathbf{u}_j(t) \in \mathbb{R}^n$ stand for state of robot j and its control input at time t. Hereafter, we succinctly denote dynamics of the robot team by $\mathbf{p}(t+1) = \mathbf{f}(\mathbf{p}(t), \mathbf{u}(t)), \mathbf{p}(t) \in \mathbb{R}^{mN}, \mathbf{u}(t) \in \mathbb{R}^{nN}$. We assume that $\mathbf{p}(t)$ is known for all time steps t. The robots are heterogeneous in terms of their capabilities/skills. The robots have collectively $C > 0$ skills collected in a set $\mathcal{C}$. We also define the set $\mathcal{C}_j \subseteq \mathcal{C}$ collecting all skills that robot j can apply. We assume that a robot can apply one skill at a time and that each skill can be executed perfectly. Based on the individual robot abilities, we can divide the robots into C sub-teams $\mathcal{T}_c = \{j \in \mathcal{R} \mid c \in \mathcal{C}_j\}$ that collect all robots that possess skill $c \in \mathcal{C}$; see Fig. 1.

B. Specifying Dynamic Missions

The robots operate in an environment $\Omega \subseteq \mathbb{R}^d$, $d \in \{2, 3\}$ that contains $M > 0$ regions/objects of interest $\ell_e, e \in \{1, 2, ..., M\}$ at locations $\mathbf{x}_e$. We assume that the obstacle-free space $\Omega_{\text{free}} \subseteq \Omega$ of the environment and the locations $\mathbf{x}_e$ are known. The robots are tasked with accomplishing collaborative high-level tasks requiring them to apply their skills at the regions/objects of interest in a temporal logical order. We formally describe the mission as a Linear Temporal Logic (LTL) specification ϕ. LTL is a type of formal logic whose basic ingredients are a set of atomic propositions collected in a set $\mathcal{AP}$, the Boolean operators, (i.e., conjunction $\wedge$ and negation $\neg$), and two temporal operators, next $\bigcirc$ and until $\mathcal{U}$. For brevity, we abstain from presenting the derivations of other Boolean and temporal operators, e.g., *always* $\square$, *eventually* $\Diamond$, *implication* $\Rightarrow$. We consider LTL tasks constructed using the following atomic propositions:

$$\pi_c(\ell_e) = \begin{cases} \texttt{True}, \text{if any robot } j \in \mathcal{T}_c \text{ applies skill } c \text{ at } \ell_e, \\ \texttt{False}, \text{otherwise.} \end{cases} \tag{1}$$

Notice that $\pi_c(\ell_e)$ is true if any robot $j \in \mathcal{T}_c$ applies the skill c at ℓ_e. Building upon (1), we define the following predicate:

$$\pi_c(j, \ell_e) = \begin{cases} \texttt{True}, \text{if robot } j \in \mathcal{T}_c \text{ applies skill } c \text{ at } \ell_e \\ \texttt{False}, \text{otherwise.} \end{cases} \tag{2}$$

Essentially, the key difference of (2) from (1) is that the former requires a specific robot $j \in \mathcal{T}_c$ to apply skill c at ℓ_e. We also define atomic propositions of the form

$$\bar{\pi}_c(\ell_e) = \neg\pi_c(\ell_e), \tag{3}$$

that is true if none of the robots in $\mathcal{T}_c$ applies skill c at ℓ_e. The set $\mathcal{AP}$ contains predicates π of the form (1)-(3).[1]

We denote the mission at time t by $\phi_{\text{cur}}(t)$ initialized as $\phi_{\text{cur}}(0) = \phi_{\text{nom}}$ using a nominal LTL task ϕ_{nom}; see Fig. 1. We assume that this nominal mission is defined over predicates of the form (2)-(3). At unknown time steps t, a new task is announced modeled as an LTL formula $\phi_{\text{new}}(t)$ with sub-tasks that are not necessarily pre-assigned to robots. Specifically, $\phi_{\text{new}}(t)$ may be defined over predicates of the form (1)-(3). Then, the mission gets updated as follows:

$$\phi_{\text{cur}}(t) \leftarrow \phi_{\text{cur}}(t) \wedge \phi_{\text{new}}(t). \tag{4}$$

C. Reactive Robot Plans

Given a feasible task $\phi_{\text{cur}}(0)$, we can design a plan τ_0, i.e., an infinite sequence of multi-robot states and actions satisfying $\phi_{\text{cur}}(0)$ using existing planners [3], [6]. This plan is defined as $\tau_0 = \tau_0(0), \tau_0(1), \dots, \tau_0(t) \dots$, where $\tau_0(t) = [\mathbf{p}(t), \mathbf{s}(t)]$ and $\mathbf{s}(t) = [s_1(t), \dots, s_N(t)]$, $s_j(t) \in \mathcal{C}_j$ at time t. In other words, $s_j(t)$ determines the skill that robot j should apply at time t; if robot j does not need to apply any skill at time t, then we denote this by $s_j(t) = \varnothing$.

Let $\mathbf{t} = t_1, \dots, t_m, \dots$ be a possibly infinite sequence of unknown time steps t_m at which a new mission $\phi_{\text{new}}(t_m)$ is announced. As soon as a new mission $\phi_{\text{cur}}(t_m)$ is constructed as per (4), our goal is to design a multi-robot plan, denoted by $\tau_{t_m} = \tau_{t_m}(t_m), \tau_{t_m}(t_m+1), \dots, \tau_{t_m}(t_m+k) \dots$, so that the new task $\phi_{\text{cur}}(t_m)$ is satisfied. We denote the overall multi-robot plan by $\tau = \tau_0(0 : t_1), \tau_{t_1}(t_1 : t_2), \dots, \tau_{t_m-1}(t_{m-1} : t_m), \tau_{t_m}$. With slight abuse of notation, $\tau_{t_m}(\alpha : \beta)$ captures

[1]The atomic propositions in (3) are introduced only because they facilitate the definition of a penalty function.

the part of a plan τ_{t_m} from $t = \alpha$ until $t = \beta$. Hereafter, for simplicity, we denote by t (instead of t_m) the time step when the most recent task was announced.

There are two key challenges in designing τ_t. First, new sub-tasks (modeled as predicates of the form (2) in $\phi_{\text{new}}(t)$) should be assigned to robots based on their capabilities so that the resulting mission $\phi_{\text{cur}}(t)$ is feasible. To ensure that there are no unassigned sub-tasks/predicates, past sub-tasks may need to be re-allocated to other robots [21]. Second, such a feasible assignment may not exist either due to conflicting logical mission requirements or because of a limited number of available robots. In this case, an assignment should be generated yielding a robot plan τ_t that minimizes mission violations. In what follows, we construct an objective function measuring violation of $\phi_{\text{cur}}(t)$ by a plan τ_t, given a fixed assignment of all predicates to robots.

D. Mission Violation Cost Function for Robot Planning

To define a cost function measuring mission violation by a given plan, we need to introduce the following definitions. First, we define a penalty function for each predicate in $\mathcal{AP}$:

Definition 2.1 (Penalty Function): The penalty function $F : \mathcal{AP} \rightarrow \mathbb{R}_+$ returns the penalty for treating a false predicate $\pi \in \mathcal{AP}$ as true. The larger the penalty of a predicate, the more important the corresponding task is.

Second, given an LTL mission, we translate it, offline, into a Nondeterministic Büchi Automaton (NBA) [1], [34].

Definition 2.2 (NBA): A Nondeterministic Büchi Automaton (NBA) B over $\Sigma = 2^{\mathcal{AP}}$ is defined as a tuple $B = \left(\mathcal{Q}_B, \mathcal{Q}_B^0, \Sigma, \delta_B, \mathcal{Q}_B^F\right)$, where $\mathcal{Q}_B$ is the set of states, $\mathcal{Q}_B^0 \subseteq \mathcal{Q}_B$ is a set of initial states, Σ is an alphabet, $\delta_B : \mathcal{Q}_B \times \Sigma \rightarrow 2^{\mathcal{Q}_B}$ is a non-deterministic transition relation, and $\mathcal{Q}_B^F \subseteq \mathcal{Q}_B$ is a set of accepting/final states.

Third, consider a task $\phi_{\text{cur}}(t)$ with all predicates assigned to robots. Using existing temporal logic planners we can define plans of the form $\bar{\tau}_t$, where $\bar{\tau}_t = \bar{\tau}_t(t), \bar{\tau}_t(t+1), \dots, \bar{\tau}_t(t+k), \dots$ and $\bar{\tau}_t(t+k) = [\mathbf{p}(t+k), \mathbf{s}(t+k), q_B(t+k)]$, for some $k \geq 0$; hereafter, for simplicity, we will replace $t + k$ with t'. In the state $\bar{\tau}_t(t')$, $q_B(t')$ denotes the NBA state that has been reached once the robots have executed the plan $\bar{\tau}_t$ up to the time step t'. Informally, $q_B(t')$ captures how much mission progress has been made. Eliminating the NBA state from $\bar{\tau}_t$ yields the plan τ_t discussed earlier. A plan $\bar{\tau}_t$ satisfies $\phi_{\text{cur}}(t)$ if it goes through states containing an accepting NBA state an infinite number of times; a more formal definition can be found in [1]. Feasible plans $\bar{\tau}_t$ can be computed using existing temporal logic planning methods and they are typically represented in a prefix-suffix form, i.e., $\bar{\tau}_t = \bar{\tau}_t^{\text{pre}}[\bar{\tau}_t^{\text{suf}}]^\omega$. The prefix $\bar{\tau}_t^{\text{pre}}$ is executed first followed by the indefinite execution of the suffix $\bar{\tau}_t^{\text{suf}}$; in $\bar{\tau}_t$, ω stands for indefinite repetition. The prefix part is defined as $\bar{\tau}_t^{\text{pre}} = \bar{\tau}_t(t), \bar{\tau}_t(t+1), \dots, \bar{\tau}_t(t+T)$, for some horizon $T \geq 0$, where $q_B(t+T) \in \mathcal{Q}_B^F$, and the suffix part is defined as $\bar{\tau}_t^{\text{suf}} = \bar{\tau}_t(t+T+1), \bar{\tau}_t(t+T+2), \dots, \bar{\tau}_t(t+T+K)$, for some $K \geq 0$ where $\bar{\tau}_t(t+T+1) = \bar{\tau}_t(t+T+K) = \bar{\tau}_t(t+T)$.

Fourth, consider the NBA states $q'_B = q_B(t'), q''_B = q_B(t'+1)$ in $\bar{\tau}_t$. We denote by $b_{q'_B,q''_B}$ the Boolean formula, defined over $\mathcal{AP}$, for which it holds that if $\sigma \models b_{q'_B,q''_B}$ then $q''_B \in \delta_B(q'_B, \sigma)$ where $\sigma \in \Sigma$. Such Boolean formulas can be constructed automatically using existing tools such as [34]. If $\bar{\tau}_t$ satisfies $\phi_{\text{cur}}(t)$, then for all time steps t', we have that $\sigma(t') \models b_{q'_B,q''_B}$ where $\sigma(t') = L([\mathbf{p}(t'), \mathbf{s}(t')])$ and L is labeling function $L : \mathbb{R}^N \times \mathcal{C}^N \rightarrow \Sigma$ determining which atomic propositions are true given $\mathbf{p}(t')$ and $\mathbf{s}(t')$.

Using the above definitions, we can now define our cost function. Given a fixed plan $\bar{\tau}_t$ and mission $\phi_{\text{cur}}(t)$, let $q'_B = q_B(t'), q''_B = q_B(t'+1)$. Consider the case where we have that $\sigma(t') \not\models b_{q'_B,q''_B}$, i.e., the transition from q'_B to q''_B cannot be enabled based on the current multi-robot state $\mathbf{p}(t')$ and action $\mathbf{s}(t')$. There exists at least one $\sigma^* \in \Sigma$, such that the concatenation of the symbols $\sigma(t')$ and σ^* satisfies $b_{q'_B,q''_B}$, i.e., $\sigma(t')\sigma^* \models b_{q'_B,q''_B}$. Thus, the predicates in σ^*, if assumed true at time t, allow the transition from q'_B to q''_B. We allow this assumption by taking into account the total penalty for treating σ^* as true. The *violation* score of the symbol $\sigma(t')$ over an NBA transition from q'_B to q''_B is defined as $\mathbb{C}_{\sigma(t')} = \min_{\forall \sigma^* \in \Sigma^*}(\sum_{\pi \in \sigma^*} F(\pi))$ where $\Sigma^* = \{\sigma \in \Sigma \mid \sigma(t')\sigma \models b_{q'_B,q''_B}\}$ and $\sigma(t') = L([\mathbf{p}(t'), \mathbf{s}(t')])$. Thus the violation score is the lowest possible penalty that we can take to enable this transition. The violation score $\mathbb{C}_{\bar{\tau}_t}$ associated with a prefix-suffix plan $\bar{\tau}_t$ is the sum of all violation scores for each transition in the plan, i.e.,

$$\mathbb{C}_{\bar{\tau}_t} = \sum_{\bar{t}=t}^{t+T+K} \mathbb{C}_{\sigma(\bar{t})}, \tag{5}$$

where $\sigma(\bar{t}) = L([\mathbf{p}(\bar{t}), \mathbf{s}(\bar{t})])$ is the symbol to enable the transition from $q_B(\bar{t})$ to $q_B(\bar{t}+1)$.

Example 2.3 (Violation Cost Function): Consider the formula $\phi = \Diamond\pi_1 \wedge \Diamond\pi_2$, where $\pi_1 = \pi_{c_1}(1, \ell_1)$, $\pi_2 = \pi_{c_2}(1, \ell_2)$, $F(\pi_1) = 5$ and $F(\pi_2) = 10$. A transition in the corresponding NBA is enabled if this Boolean formula $b_{q'_B,q''_B} = \pi_1 \wedge \pi_2$ is true. Notice that this transition is infeasible to activate as it requires robot 1 to be present in two locations simultaneously. Given a symbol $\sigma(t') = \pi_1$, then $\Sigma^* = \{\pi_2\}$. Thus, $\mathbb{C}_{\sigma(t')} = F(\pi_2) = 10$.

E. Problem Statement

This paper addresses the following problem; see Fig. 1.

Problem 1: Consider an initial task $\phi_{\text{cur}}(0)$ and a plan τ_0 satisfying it. Given a new task $\phi_{\text{new}}(t)$ announced at t (a) design an online task (re)allocation method (re)assigning sub-tasks/predicates to the robots and (b) revise the current plan (i.e. design τ_t) to satisfy the updated mission. The requirements (a)-(b) should be met so that (5) is minimized.

Remark 2.4 (Independence of Sub-tasks): We assume that the ability of a robot to fulfill a predicate does not depend on any other predicates assigned to it or other robots. This also means that any predicate/sub-task initially assigned to a robot i can be re-assigned to any other robot j as long as it has the skill required to complete it.

III. Reactive Temporal Logic Planning

In this section, we present an algorithm to address Problem 1. In Section III-A, we provide a brief overview of an existing

Algorithm 1: Minimum Violation Task Allocation

```
Input: (i) NBA $B_t$ for $\phi_{\text{cur}}(t)$, (ii) Current NBA state $q_B^{\text{cur}}$; (iii)
       Set of predicates $\mathcal{AP}_n$; (iv) Set of NBA transitions $\mathcal{E}$
Output: Revised NBA $B_t$
1  for every $\pi \in \mathcal{AP}_n$ do
2      if Find robot $i$ replacement for new task then
3          Define the ordered set of edges $\mathcal{E}_\pi$;
4          for every $e = (q'_B, q''_B) \in \mathcal{E}_\pi$ do
5              Rewrite: $b_{q'_B,q''_B} = \bigvee_{d=1}^{D} b^d_{q'_B,q''_B}$;
6              for $d = 1, \dots, D$ do
7                  Define $\mathcal{G}$ and functions $V^d_{q'_B,q''_B}, g^d_{q'_B,q''_B}$;
8                  Apply Alg. 2 to compute a sequence of
                     re-assignments $p = p(0), \dots, p(P)$;
9                  Re-assign atomic predicates as per $p$;
10                 Revise $b^d_{q'_B,q''_B}$ in $B_t$ as per $p$;
```

planner that we employ to generate τ_0. This occurs offline. In Section III-B-III-C, we present a task re-allocation algorithm that is executed as soon as new tasks are announced. Our task allocation algorithm builds upon our earlier work [21]. A key difference is that the proposed algorithm accounts for potential mission conflicts that may arise due to dynamically changing task requirements. In Section III-D, we propose an online re-planning algorithm that revises the team plan to accommodate the new tasks and mitigate potential mission conflicts. This algorithm is executed as soon as the new, and possibly previous, sub-tasks are (re)assigned to the robots.

A. Offline Planning

Consider a nominal task $\phi_{\text{cur}}(0)$ defined over predicates (2)-(3). We design a feasible plan τ_0 using the sampling-based planner developed in [3] due to its abstraction-free and scalability benefits; any other motion planner can be employed. This planner incrementally builds a tree $\mathcal{T}$ that explores both the robot motion space and the automaton state-space. The tree $\mathcal{T}$ is defined as $\mathcal{T} = \{\mathcal{V}_\mathcal{T}, \mathcal{E}_\mathcal{T}, \texttt{Cost}\}$. The set $\mathcal{V}_\mathcal{T}$ consists of nodes defined as $\mathbf{q}(t) = [\mathbf{p}(t), \mathbf{s}(t), q_B(t)]$. The root $\mathbf{q}(0)$ of the tree is defined using the initial state $\mathbf{p}(0)$, a null vector $\mathbf{s}(0)$, and an initial NBA state $q_B(0) \in \mathcal{Q}_B^0$. The set of edges $\mathcal{E}_\mathcal{T}$ captures transitions among the nodes in $\mathcal{V}_\mathcal{T}$. Moreover, the cost function $\texttt{Cost} : \mathcal{V}_\mathcal{T} \to \mathbb{R}^+$ computes the cost of reaching node $\mathbf{q}(t)$ from the root by following a path, i.e., a sequence of tree nodes, $\mathbf{d} = \mathbf{q}(0), \mathbf{q}(1), \dots, \mathbf{q}(t)$. We define this function as in (5), i.e., $\texttt{Cost}(\mathbf{q}(t)) = \mathbb{C}_\mathbf{d}$. This sampling-based planner is asymptotically optimal i.e., as the number of tree nodes goes to infinity, the probability of computing the optimal (prefix-suffix) plan τ_0 satisfying an LTL task $\phi_{\text{cur}}(0)$ goes to 1. This also implies that if $\phi_{\text{cur}}(0)$ is infeasible, then the planner will asymptotically compute the least violating plan.

B. Setting Up the Task Allocation Process

As the robots execute the initial plan τ_0, a new task $\phi_{\text{new}}(t)$ may be announced at an unknown time step t giving rise to a new mission $\phi_{\text{cur}}(t)$ as defined in (4). Let $\mathcal{AP}_n \subseteq \mathcal{AP}$ be a set collecting all 'unassigned' atomic predicates (1) in $\phi_{\text{new}}(t)$. To revise the current plan, first, we need to assign the new sub-tasks in $\mathcal{AP}_n$ to new robots. As discussed in Section II-C, it is possible that the sub-tasks in $\mathcal{AP}_n$ can be accomplished only by robots that are currently busy with other sub-tasks. In such cases, it is necessary to reassign the previous tasks to accommodate the new requirements while mitigating any mission conflicts that may arise. Next, we set up this task allocation process summarized in Alg. 1.

Given the updated LTL formula $\phi_{\text{cur}}(t)$, we construct its corresponding NBA denoted by B_t. Then, we determine how much progress the robots have made towards accomplishing this new task $\phi_{\text{cur}}(t)$. Formally, this is represented by the NBA state that the robots can reach in B_t, starting from an initial state q_B^0 given the sequence of actions they have applied up to time t [1].[2] We denote this state by q_B^{cur}. Due to the non-deterministic nature of the NBA there may exist multiple candidate states q_B^{cur}. We select the closest one to the accepting NBA states using existing 'distance' metrics defined over automata [3]. Next, we compute all NBA states that can be reached from the current state q_B^{cur} through a multi-hop path. This step can be implemented by treating the NBA as a directed graph and checking which states $q'_B \in \mathcal{Q}_B$ can be reached from q_B^{cur}. We collect these states (including q_B^{cur}) in a set $\hat{\mathcal{Q}}_B^{\text{cur}} \subseteq \mathcal{Q}_B$. Let $e = (q'_B, q''_B)$ denote an NBA transition from $q'_B \in \hat{\mathcal{Q}}_B^{\text{cur}}$ to $q''_B \in \hat{\mathcal{Q}}_B^{\text{cur}}$ for which there exists an unassigned predicate in the corresponding Boolean formula $b_{q'_B,q''_B}$. We collect all these edges e in a set $\mathcal{E}$. The NBA B_t, the state q_B^{cur}, the set of unassigned sub-tasks $\mathcal{AP}_n$, and the set of NBA transitions $\mathcal{E}$ serve as inputs to Alg. 1.

Let $\pi \in \mathcal{AP}_n$ be an unassigned predicate and $\mathcal{E}_\pi \subseteq \mathcal{E}$ be a set of NBA edges where π appears in the corresponding Boolean formulas $b_{q'_B,q''_B}$. The key idea in Alg. 1 is to inspect all edges $e \in \mathcal{E}_\pi$ in parallel and allocate π to a robot. It is important to note that we do not require the robot assigned to undertake π in each edge to be the same since we assume independent sub-tasks; see Remark 2.4. This process is repeated sequentially for all unassigned predicates so that the resulting LTL formula remains feasible.

A necessary condition to preserve the feasibility of the LTL formula after task allocation is that $b_{q'_B,q''_B}$ should be feasible (i.e., it can become 'true') for all $e = (q'_B, q''_B) \in \mathcal{E}_\pi$, and $\pi \in \mathcal{AP}_n$. In other words, there should exist a symbol $\sigma = L([\mathbf{p}, \mathbf{s}])$ generated by the robots, that satisfies the revised formulas $b_{q'_B,q''_B}$ arising after task assignment. A challenge arising here is that there may not exist enough available robots capable of taking over all unassigned predicates or $b_{q'_B,q''_B}$ is infeasible regardless of the task assignments due to logical conflicts. This implies the necessity of sacrificing the completion of a sub-task/predicate, incurring a penalty as per F (see Definition 2.1). In this case, our goal is to allocate tasks to minimize a violation task allocation objective. To

[2]This computation is done without sacrificing satisfaction of any predicate required to activate the NBA transitions, i.e., q_B^{cur} is reached from q_B^0 using the currently implemented plan while the cost of this plan, as per (5), over the NBA B_t is 0. We note that depending on the previous and the new task, as well as the actions that the robots have applied so far, such an NBA state may not exist. For instance, consider the nominal task $\phi_{\text{cur}}(0) = \Diamond A$, for some predicate A. Assume that at some time t, when $\phi_{\text{cur}}(0)$ has already been satisfied, a new task $\phi_{\text{new}}(t) = \Diamond B \wedge \Box\neg A$ is announced. In this case, there is no q_B^{cur} in B_t that can be reached using the current robot actions; this can be confirmed by inspection of B_t [34]. In this case, we select q_B^{cur} to be the initial state in B_t.

formally define this objective, we need to introduce the following definitions [21]; see Ex. 3.3. For every edge in $\mathcal{E}_\pi$, we re-write the Boolean $b_{q'_B,q''_B}$ in a disjunctive normal form (DNF), i.e., $b_{q'_B,q''_B} = \bigvee_{d=1}^{D} b^d_{q'_B,q''_B}$, for some $D > 0$. For each Boolean formula $b^d_{q'_B,q''_B}$, we collect all robots that appear in $b^d_{q'_B,q''_B}$ as the set $\mathcal{R}^d_{q'_B,q''_B} \subseteq \mathcal{R}$. Also, we define a set $\mathcal{AP}_i \subseteq \mathcal{AP}$ that collects all atomic predicates that appear in $b^d_{q'_B,q''_B}$ associated with skills $c \in \mathcal{C}_i$, $i \in \mathcal{R}$, assuming that all these predicates are assigned to robot i. Using $\mathcal{AP}_i$, we can define $\Sigma_i = 2^{\mathcal{AP}_i}$. Using these definitions, we can define the following functions that capture (i) the tasks/predicates that if a robot i undertakes, then $\phi_{\text{cur}}(t)$ will become infeasible and (ii) which robots are currently busy with other sub-tasks.

Definition 3.1 (Function $V^d_{q'_B,q''_B}$): The set-valued function $V^d_{q'_B,q''_B} : \mathcal{R} \rightarrow \Sigma_i$, given as input a robot index $i \in \mathcal{R}$, returns a set collecting all symbols $\sigma_i \in \Sigma_i$ that if robot $i \in \mathcal{R}$ generates, then $b^d_{q'_B,q''_B}$ will be 'false' regardless of the values of the other predicates. We define $V^d_{q'_B,q''_B}(i) = \emptyset$ for all robots $i \in \mathcal{R} \setminus \mathcal{R}^d_{q'_B,q''_B}$.

Definition 3.2 (Function $g^d_{q'_B,q''_B}$): The function $g^d_{q'_B,q''_B} : \mathcal{R} \rightarrow \mathcal{AP}$, given as an input a robot index $i \in \mathcal{R}$, returns a set collecting the atomic predicates that are assigned to robot i in $b^d_{q'_B,q''_B}$ excluding the negated ones. We define $g^d_{q'_B,q''_B}(i) = \varnothing$, for all robots $i \notin \mathcal{R}^d_{q'_B,q''_B}$ and for all robots $i \in \mathcal{R}^d_{q'_B,q''_B}$ appearing only in negated predicates.

Example 3.3 (Functions $V^d_{q'_B,q''_B}$ and $g^d_{q'_B,q''_B}$): Consider 5 robots divided into teams $\mathcal{T}_{c_0} = \{1, 2, 3, 4, 5\}$, $\mathcal{T}_{c_1} = \{1, 3\}$, $\mathcal{T}_{c_2} = \{3, 4, 5\}$, $\mathcal{T}_{c_3} = \{4\}$, and $\mathcal{T}_{c_4} = \{5\}$. The skills c_0, c_1, c_2, c_3, and c_4 refer to mobility, fire extinguishing, object recognition, taking photos, and cleaning skills. Consider the Boolean formula $b^d_{q'_B,q''_B} = \pi_1 \wedge \pi_2 \wedge \bar{\pi}_3 \wedge \pi_4 \wedge \pi_5$, where $\pi_1 = \pi_{c_1}(1, \ell_1), \pi_2 = \pi_{c_2}(4, \ell_3), \bar{\pi}_3 = \bar{\pi}_{c_1}(\ell_3), \pi_4 = \pi_{c_4}(5, \ell_3)$, and $\pi_5 = \pi_{c_3}(\ell_2)$; $\bar{\pi}_3$ is defined as in (3). Observe that π_5 is unassigned. The robots that are crucial for satisfaction of $b^d_{q'_B,q''_B}$ are $\mathcal{R}^d_{q'_B,q''_B} = \{1, 3, 4, 5\}$.We have $g^d_{q'_B,q''_B}(1) = \pi_1$, $g^d_{q'_B,q''_B}(3) = \varnothing$, $g^d_{q'_B,q''_B}(4) = \pi_2$, $g^d_{q'_B,q''_B}(5) = \pi_4$, and $g^d_{q'_B,q''_B}(i) = \varnothing$ for robots $i \in \mathcal{R} \setminus \mathcal{R}^d_{q'_B,q''_B}$. As for $V^d_{q'_B,q''_B}$, it holds that $V^d_{q'_B,q''_B}(i) = \emptyset$ for all $i \notin \mathcal{T}_{c_1}$. As for the robots in $\mathcal{T}_{c_1}$, we have that $V^d_{q'_B,q''_B}(1) = \{\pi_3\}$ and that $V^d_{q'_B,q''_B}(3) = \{\pi_2, \pi_3, \pi_2\pi_3\}$ since if robot 3 satisfies π_2 it will satisfy π_3.

C. Minimum-Violation Task Allocation Algorithm

In this section, we present our task allocation algorithm that utilizes the definitions introduced earlier; see Alg. 1. Consider an unassigned atomic predicate $\pi = \pi_m(\ell_e) \in \mathcal{AP}_n$ and an edge $e \in \mathcal{E}_\pi$ associated with a Boolean formula $b_{q'_B,q''_B} = \bigvee_{d=1}^{D} b^d_{q'_B,q''_B}$ [lines 1-5, Alg. 1]. Then for each sub-formula $b^d_{q'_B,q''_B}$ (in parallel), we search for an assignment using graph-search methods [lines 6-10, Alg. 1]. Particularly, for each sub-formula $b^d_{q'_B,q''_B}$, we can define a directed graph $\mathcal{G} = \{\mathcal{V}_\mathcal{G}, \mathcal{E}_\mathcal{G}\}$ capturing all possible assignments. In this graph, we have that $\mathcal{V}_\mathcal{G}$ and $\mathcal{E}_\mathcal{G}$ denote the set of nodes and edges, respectively [line 7, Alg. 1]. The set of nodes is defined as $\mathcal{V}_\mathcal{G} = \mathcal{R} \cup a_0$, where a_0 is an artificial node; the purpose of a_0 will be explained later. An edge from a node a to a' ($\neq a$) exists if $a' \in \mathcal{T}_c$, where c is the skill required to satisfy the predicate $g^d_{q'_B,q''_B}(a)$. The directed edge indicates that robot a' can take over predicate of robot a in $b^d_{q'_B,q''_B}$. Also, a direct edge from a_0 to a exists if $a \in \mathcal{T}_m$. We emphasize that we do not explicitly construct this graph; instead, we only require knowledge of all teams $\mathcal{T}_c$.

As discussed in Section III-B, there may not exist any available robots that can undertake π. As a result, already assigned predicates may need to be re-allocated. Thus, we compute a path in $\mathcal{G}$ dictating this sequence of re-assignments [line 8, Alg. 1]. Let $p = p(0), p(1), ..., p(P)$ denote such a path over $\mathcal{G}$, where $p(k) \in \mathcal{V}_\mathcal{G}$, for all $k \in \{1, 2, ..., P-1\}$, and $p(0) = a_0$ [line 1, Alg. 2]. The transition from $p(k)$ to $p(k+1)$ means that $p(k+1)$ will relinquish its current sub-task (which is $g^d_{q'_B,q''_B}(p(k+1)))$ to take over the sub-task of robot $p(k)$ (which is $g^d_{q'_B,q''_B}(p(k))$). This also means that the robot $p(1)$ will take over the unassigned task.

Observe that if the robot $p(P)$ was associated with a predicate in $b^d_{q'_B,q''_B}$, i.e., $g^d_{q'_B,q''_B}(p(P)) \neq \emptyset$, then this means that the task of $p(P)$ will be sacrificed causing a penalty of $F(g^d_{q'_B,q''_B}(p(P))) > 0$. Similarly, if the robot $p(k+1)$ undertakes the sub-task $g_{q'_B,q''_B}(p(k))$, this will yield a penalty of $F(g^d_{q'_B,q''_B}(p(k))) > 0$ if $g^d_{q'_B,q''_B}(p(k)) \in V^d_{q'_B,q''_B}(p(k+1))$ (i.e., the task that the robot $p(k+1)$ will take over will result in violating the boolean formula $b^d_{q'_B,q''_B}$). We collect in a set $\mathcal{K}$ all indices $k \in \{1, \ldots, P\}$ where $g^d_{q'_B,q''_B}(p(P)) \neq \emptyset$ or $g^d_{q'_B,q''_B}(p(k)) \in V^d_{q'_B,q''_B}(p(k+1))$. This way, we can define the violation cost of a path p as follows:

$$\mathbb{C}_\mathcal{G}(p) = \begin{cases} \sum_{k \in \mathcal{K}} F(g^d_{q'_B,q''_B}(p(k))), \text{if } \mathcal{K} \neq \emptyset, \\ 0, \text{otherwise}, \end{cases} \tag{6}$$

where the function $g^d_{q'_B,q''_B}$ is applied to the Boolean formula $b^d_{q'_B,q''_B}$ before re-allocating tasks to robots as per p.[3]

Our goal is to compute the path p that minimizes (6). To compute it, we adopt a Breadth First Search (BFS) search approach; see Alg. 2 and Ex. 3.4. We use a queue data structure $\mathcal{Q}$, initialized as $\mathcal{Q} = [a_0]$, similar to the traditional BFS algorithms [line 1, Alg. 2]. When a node a is removed from $\mathcal{Q}$ [line 2, Alg. 2], then each adjacent node a' is added to $\mathcal{Q}$ if it has not been explored yet (as in standard BFS) [lines 7-8, Alg. 2]. This also prevents cases where a single robot will be assigned to complete two tasks at the same time. Then, we compute the paths p connecting these unexplored nodes a' to the root a_0 and then corresponding cost $\mathbb{C}_\mathcal{G}(p)$ [line 12, Alg. 2]. Then, we update a^* to point to the node with the lowest cost $C^* = \mathbb{C}_\mathcal{G}(p)$ [lines 13-14, Alg. 2]. Nodes a' for which it holds that $g^d_{q'_B,q''_B}(a') = \varnothing$ are not included in $\mathcal{Q}$ [line 11, Alg. 2]. The reason is that these robots are not assigned to any task and, therefore, reassignment for them is not meaningful. As a result, such nodes are never expanded. The search process ends as soon as a path p with $\mathbb{C}_\mathcal{G}(p) = 0$ is found [lines 5-6, Alg. 2]. If

[3]In fact, $\mathbb{C}_\mathcal{G}(p)$ is equal to the minimum penalty $\mathbb{C}_{\sigma(\bar{t})}$ that any plan τ_t will incur in order to make $b^d_{q'_B,q''_B}$ true at time $\bar{t}$; see also (5).

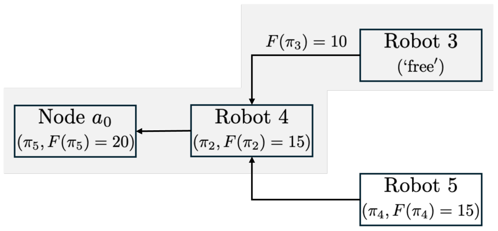


Fig. 2. Graphical depiction of Alg. 2 for Ex. 3.4. The shaded path stands for the minimum violation path p computed by Alg. 2.

this happens, then this means no sub-tasks were sacrificed. If such a path does not exist, Alg. 2 will exhaustively search all possible paths to compute the one with the minimum violation penalty cost [lines 15-16, Alg. 2]. Once Alg. 1 terminates, all formulas $b_{q'_B,q''_B}$ are revised by assigning predicates as per the corresponding paths p [lines 9-10, Alg. 1]. This revised NBA is an input to the online re-planner.

Example 3.4 (Task Re-allocation): Consider the Boolean formula of Ex. 3.3: $b^d_{q'_B,q''_B} = \pi_1 \wedge \pi_2 \wedge \bar{\pi}_3 \wedge \pi_4 \wedge \pi_5$, where $\pi_1 = \pi_{c_1}(1, \ell_1), \pi_2 = \pi_{c_2}(4, \ell_3), \bar{\pi}_3 = \bar{\pi}_{c_1}(\ell_3)$, $\pi_4 = \pi_{c_4}(5, \ell_3)$ and $\pi_5 = \pi_{c_3}(\ell_2)$. The penalty function F is defined as $F(\pi_1) = 10, F(\pi_2) = 15, F(\bar{\pi}_3) = 10, F(\pi_4) = 15$, and $F(\pi_5) = 20$. The tree constructed by Alg. 2 to assign π_5 to a robot is shown in Fig. 2. Observe that the root a_0 has only one child node, as only robot 4 can undertake π_5 (since $\mathcal{T}_{c_3} = \{4\}$). The node associated with robot 4 is currently busy with task π_2 requiring skill c_2. Thus, the children of that node are associated with robots 3 and 5. Observe that robot 3 is currently free (i.e., $g^d_{q'_B,q''_B}(3) = \varnothing$) while robot 5 is busy with π_4 (i.e., $g^d_{q'_B,q''_B}(5) = \pi_4$). The BFS algorithm terminates at that point since the nodes associated with robots 3 and 5 cannot be expanded further since the former is 'free' and the latter cannot be replaced by any other robot (robot 5 is the only one that has the c_4 skill). The output of Alg. 2 is $C^* = 10$ and $p = a_0, 4, 3$. Observe in Fig. 2 that there are two paths in the constructed tree. The first path is $p_1 = a_0, 4, 3$: robot 3 takes over the task of robot 4 (i.e., π_2). However, $\pi_2 \in V^d_{q'_B,q''_B}(3)$ (see Ex. 3.3) incurring a cost $F(\bar{\pi}_3) = 10$. And robot 4 can take over π_5 with no penalty since $V^d_{q'_B,q''_B}(4) = \varnothing$. Thus, we have that $\mathbb{C}_{\mathcal{G}}(p_1) = 10$. The second path is $p_2 = a_0, 4, 5$ which has a cost of $15 > 10$. Notice that if $F(\pi_5) < 10$, then Alg. 2 would return $p = a_0$ with $C^* = F(\pi_5)$ i.e., the team would give up on π_5.

D. Online Re-planning

Assume that at time t a new task is announced and the current joint robot-NBA state is $[\mathbf{p}(t), \mathbf{s}(t), q_B(t)]$ (before task allocation). Let B_t be the revised automaton, generated by Alg. 1, i.e., after task re-allocation. A straightforward solution to design a new plan τ_t (see Section II-C) accommodating the updated mission is to apply the sampling-based planner, discussed in Section III-A, to build a new tree. The root of the new tree will be $[\mathbf{p}(t), \mathbf{s}(t), q^{\text{cur}}_B]$ where q^{cur}_B is belongs to the revised automaton B_t and denotes the current NBA state (see Section III-B). Nevertheless, re-planning from scratch for *all* robots may be unnecessary given the 'local' task re-allocations. It may also be impractical for large robot teams. To address this, we leverage the tree, denoted

Algorithm 2: Breadth First Search

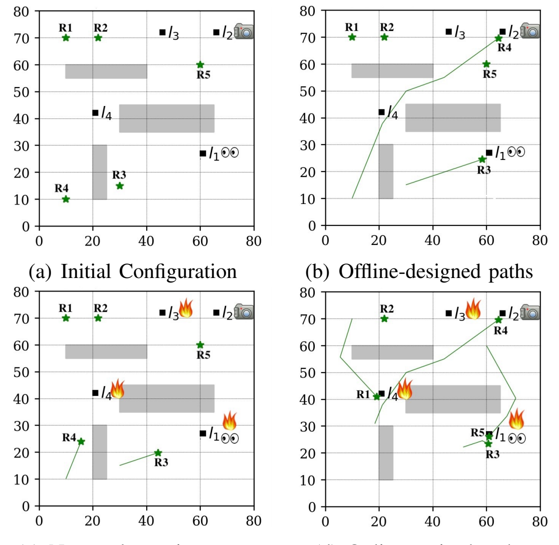
```
Input: (i) Unassigned predicate $\pi = \pi_c(\ell_e)$, (ii) $V^d_{q'_B,q''_B}$, (iii)
       $g^d_{q'_B,q''_B}$, (iv) Teams $\mathcal{T}_c, \forall c \in \mathcal{C}$
Output: Path $p$
1  Initialize: $\mathcal{Q} = [a_0]$; $a^* = a_0; p = a_0$; $C^* = F(\pi)$;
2  while ($\sim$empty($\mathcal{Q}$)) $\vee$ ($C^* > 0$) do
3      $a \leftarrow$ POP($\mathcal{Q}$);
4      Compute path $p$ from $a$ to $a_0$;
5      if $\mathbb{C}_{\mathcal{G}}(p) = 0$ then
6          Return path $p$ from $a$ to $a_0$;
7      for $a'$ adjacent to $a$ in $\mathcal{G}$ do
8          if $a'$ not explored then
9              Label $a'$ as explored;
10             Parent($a'$) = $a$;
11             Append $a'$ to $\mathcal{Q}$ if $g^d_{q'_B,q''_B}(a') \neq \varnothing$ ;
12             Compute path $p$ from $a'$ to $a_0$;
13             if $\mathbb{C}_{\mathcal{G}}(p) < C^*$ then
14                 $a' = a^*$, $C^* = \mathbb{C}_{\mathcal{G}}(p)$
15     if empty($\mathcal{Q}$) then
16         Return path $p$ from $a^*$ to $a_0$ ;
```

by $\mathcal{T}$, used to construct the previous plan. We provide a brief overview of our approach. First, we extract from $\mathcal{T}$ the sub-tree that is rooted at $[\mathbf{p}(t), \mathbf{s}(t), q_B(t)]$. Second, since the NBA has been updated due to the new tasks, we revise the NBA states along the branches of the sub-tree accordingly. Specifically, the NBA state $q_B(t)$ (that belongs to the previous NBA) is replaced by q^{cur}_B that belongs to B_t. Then, based on the predicates that are satisfied along each branch of the tree, we accordingly compute the corresponding NBA states. This gives rise to a revised sub-tree. Then we apply the sampling-based planner to compute a new plan where the new tree is initialized using the revised sub-tree. We note that the planner may find a path τ_t with zero violation cost according to (5), even if Alg. 2 made some re-assignments with non-zero violation cost as per (6). The reason is that the planner may compute paths that enable NBA transitions where re-assignment occurred with zero cost; see Sec. V-B.

IV. Algorithm Analysis

Proposition 4.1 (Optimality of Alg. 2): Consider a new predicate $\pi \in \mathcal{AP}_n$ and Boolean formula $b^d_{q'_B,q''_B}$ that contains π. Alg. 2 will compute the optimal sequence p of re-assignments as per $\mathbb{C}_{\mathcal{G}}(p)$ defined in (6). If there exist more than one optimal path/re-allocation, it will select the one with the minimum number of re-assignments.

Proof: This result holds by the construction of Alg. 2. Specifically, if there exists a path p with $\mathbb{C}_{\mathcal{G}}(p) = 0$, then Alg. 2 will find it by the completeness of the BFS algorithm. Also, since, by construction, search over $\mathcal{G}$ occurs in a breadth-first manner, Alg. 2 will compute the path with the minimum number of hops from the root a_0. This equivalently results in the minimum number of re-assignments. If there does not exist a path p satisfying $\mathbb{C}_{\mathcal{G}}(p) = 0$ then Alg. 2 will exhaustively search over the entire graph and it will return the path from node a^* to a_0 achieving the smallest cost $\mathbb{C}_{\mathcal{G}}(p)$. If there exists more than one path achieving the same optimal cost, Alg. 2 returns the one that will be computed first. Due to the breadth-first nature of the search process, this path has



(a) Initial Configuration (b) Offline-designed paths

(c) New task requirements (d) Online revised paths

Fig. 3. Graphical illustration of the paths designed for Case Study I.

the smallest number of hops among all other paths achieving the same violation cost. ■

Proposition 4.2 (Soundness/Optimality of Re-planning): Consider an allocation of predicates by Alg. 1 as soon as new tasks are released at time t. (i) If the re-planning algorithm (Sec. III-D) returns a new prefix-suffix plan $\bar{\tau}_t$, then this plan satisfies ϕ. (ii) If there exists a feasible prefix-suffix plan satisfying the $\phi_{\text{cur}}(t)$, then the probability that the re-planning algorithm will compute a prefix-suffix plan $\bar{\tau}_t$ with the minimum violation cost as per (5) goes to 1 as the size of the constructed tree goes to infinity.

Proof: This result holds due to the soundness and asymptotic optimality of the employed planner [3]. ■

V. Experiments

We conducted our experiments using Python 3 on a computer with Intel Core i5 2.4GHz and 8Gb RAM.

A. Case Study I: Limited Number of Available Robots

Consider the team of robots with skills as described in Ex. 3.3 residing in the environment shown in Fig. 3(a). The initial mission is $\phi_{\text{cur}}(0) = \Diamond\pi_1 \wedge \Diamond\pi_2$, where $\pi_1 = \pi_{c_2}(3, \ell_1)$ and $\pi_2 = \pi_{c_3}(4, \ell_2)$, corresponding to an NBA B with 4 states. This task requires eventually robot 3 to recognize the object at ℓ_1 and eventually robot 4 to take photos at ℓ_2 with penalty function $F(\pi_1) = 5, F(\pi_2) = 10$. The plan τ_0 was computed in 0.16 seconds; see Fig. 3(b). At $t = 3$, while the robots are still on the way to their destinations (i.e., $q_B(t) \in \mathcal{Q}_B^0$ in B and π_1 and π_2 have not been satisfied yet), a fire breaks out at landmarks ℓ_3, ℓ_1, and ℓ_4; see Fig.3(c). Thus, a user specifies an additional mission $\phi_{\text{new}}(3) = (\Diamond\xi) \wedge (\bar{\pi}_2 \mathcal{U} \xi)$, where $\xi = \pi_3 \wedge \pi_4 \wedge \pi_5$, $\pi_3 = \pi_{c_1}(\ell_3)$, $\pi_4 = \pi_{c_1}(\ell_1)$, and $\pi_5 = \pi_{c_1}(\ell_4)$ with penalties $F(\pi_3) = 10$, $F(\pi_4) = 20$, $F(\pi_5) = 30$, and $F(\bar{\pi}_2) = 15$. This mission requires the robots to extinguish fires at ℓ_3, ℓ_1, and ℓ_4, simultaneously, before taking photos at ℓ_1. The updated task $\phi_{\text{cur}}(t)$ results in a new automaton B_t with 6 states. The state q_B^{cur} in B_t is an initial one, since no progress has been towards completing $\phi_{\text{cur}}(t)$. Also, we have that $\mathcal{AP}_n = \{\pi_3, \pi_4, \pi_5\}$ and $|\mathcal{E}| =$ 6. In all formulas $b^d_{q'_B, q''_B}$ across the edges in $\mathcal{E}$, Alg. 2 re-assigns π_1 to robot 5, assigns π_4 to robot 3, assigns π_5 to robot 1, while π_3 remains unassigned. The latter occurs as π_3 has the lowest priority among others in $\mathcal{AP}_n$ and there are no available robots to satisfy π_3, π_4 and, π_5 simultaneously; see Fig. 3(d). The cost of this assignment for every $b^d_{q'_B, q''_B}$, as per (6), is $F(\pi_3) = 10$. Task allocation and re-planning required 0.0017 and 0.096 seconds, respectively; the average depth of the trees constructed by Alg. 2 across predicates was 2. The violation cost of the new path, as per (5), is 10, which occurred due to sacrificing π_3. Consider also the case where $\phi_{\text{new}}(t)$ is announced at $t = 9$, when π_1 (but not π_2) has already been completed. In this case, the current NBA state q_B^{cur} in B_t does not belong to the set of initial states. The reason is that progress towards completing $\phi_{\text{cur}}(9)$ has already been made.[4] Then, we have $|\mathcal{E}| = 2$ and $\mathcal{AP}_n = \{\pi_3, \pi_4, \pi_5\}$. The tasks in $\mathcal{AP}_n$ are assigned as before. Task allocation and re-planning required 0.01 secs and 0.15 secs, respectively. The average depth of the trees constructed by Alg. 1 was 1.67.

B. Case Study II: Logical Mission Conflicts

Consider the same team of robots as in Case Study I. The initial mission is $\phi_{\text{cur}}(0) = \Diamond\pi_1 \wedge \Box\bar{\pi}_2$, where $\pi_1 = \pi_{c_1}(1, \ell_1)$, $\bar{\pi}_2 = \bar{\pi}_{c_3}(\ell_1)$, $F(\pi_1) = 10$, and $F(\bar{\pi}_2) = 20$, requiring robot 1 to eventually extinguish a fire at ℓ_1 while prohibiting any robot in $\mathcal{T}_{c_3} = \{4\}$ taking photos at ℓ_1. This formula corresponds to an NBA with 2 states. At time $t = 2$, while robot 1 is on its way to ℓ_1 (as per τ_0), the new mission is released: $\phi_{\text{new}}(t) = \Diamond\pi_3 \vee \Diamond\pi_4$, where $\pi_3 = \pi_{c_3}(\ell_1)$ and $\pi_4 = \pi_{c_3}(\ell_3)$, with $F(\pi_3) = 5, F(\pi_4) = 5$. This task requires any robot to eventually take photos at either ℓ_1 or ℓ_3. Observe that $\Diamond\pi_3$ is in conflict with the original requirement $\Box\bar{\pi}_2$. The updated mission $\phi_{\text{cur}}(t)$ corresponds to an NBA B_t with 4 states. Given that the robots have not made any progress towards accomplishing $\phi_{\text{cur}}(t)$, the current NBA state q_B^{cur} in B_t is an initial one. We have $|\mathcal{E}| = 3$ and $\mathcal{AP}_n = \{\pi_3, \pi_4\}$. Alg. 1 keeps π_3 unassigned and assigns π_4 to robot 4 across all sub-formulas $b^d_{q'_B, q''_B}$; none of the remaining predicates are re-assigned to other robots. We note that although the assignment cost in some of the Boolean formulas $b^d_{q'_B, q''_B}$ is non-zero (as per (6)), the (re)planner computes a plan with zero violation cost (as per (5)). For instance, consider the formula $b^d_{q'_B, q''_B} = \pi_1 \wedge \bar{\pi}_2 \wedge \pi_3$. In this case, π_3 remains un-assigned (as Alg. 2 returns the path $p = a_0$) incurring a penalty of $F(\pi_3) = 5$. Consider also the formula $b^d_{q'_B, q''_B} = \pi_1 \wedge \bar{\pi}_2 \wedge \pi_4$, where π_4 is assigned to robot 4 by Alg. 2 incurring zero penalty. The re-assignment process requires 0.008 seconds. Given this task allocation, our re-planning algorithm designed a path with zero violation

[4]If $\phi_{\text{new}}(t)$ is announced when $\phi_{\text{cur}}(0)$ is already satisfied, then there is no q_B^{cur} in B_t as $\phi_{\text{cur}}(t)$ has already been violated by past actions. Then, we set q_B^{cur} to be an initial state; see Sec. III-B. This will result in plans that make the robots perform tasks that have already been accomplished in the past (i.e., π_1 and π_2). We assume that the robots are equipped with sensing mechanisms allowing them to determine, as they execute the new plans, whether a sub-task (e.g., π_1), depending on its nature, can be deemed as 'completed' and, therefore, the corresponding action in the plan can be neglected, or it needs to be re-done.

cost in 0.15 secs. Note that if '$\vee\Diamond\pi_4$' did not exist in ϕ_{new}, the resulting plan would have a non-zero violation cost.

C. Case Study III: Large Scale Robot Teams

Consider a team of $N \in \{6, 16, 26\}$ robots with $\mathcal{C} = \{c_0, c_1, c_2, c_3, c_4, c_5, c_6\}$. The construction of the sub-teams depends on N. For instance, when $N = 6$, we have: $\mathcal{T}_{c_0} = \{1, ..., 6\}$, $\mathcal{T}_{c_1} = \{1\}$, $\mathcal{T}_{c_2} = \{2\}$, $\mathcal{T}_{c_3} = \{3\}$, $\mathcal{T}_{c_4} = \{3, 4\}$, $\mathcal{T}_{c_5} = \{5\}$ and $\mathcal{T}_{c_6} = \{6\}$. For all $N \in \{6, 16, 26\}$, the initial mission is $\phi_{\text{cur}}(0) = \Diamond\pi_{c_3}(1, \ell_1) \wedge [\bar{\pi}_{c_3}(\ell_1)\mathcal{U}\pi_{c_2}(2, \ell_1)]$ and that at $t = 5$, the new task is announced: $\phi_{\text{new}}(5) = \Diamond\pi_{c_5}(\ell_4) \wedge [\bar{\pi}_{c_5}(\ell_4)\mathcal{U}\pi_{c_1}(\ell_4)] \wedge \Diamond\pi_{c_3}(\ell_3) \wedge \Diamond[\pi_{c_1}(\ell_1) \wedge \pi_{c_4}(\ell_1)] \wedge \Diamond[\pi_{c_2}(\ell_5) \wedge \Diamond\pi_{c_3}(\ell_5)]$. In all cases, we have that $|\mathcal{E}| = 1800$. The task allocation process required 0.59, 0.64, and 0.542 secs for $N = 6$, 16, and 26, respectively. The violation score is zero for all cases due to the sufficiently large number of robots. These runtimes are comparable, as the average depth of the BFS trees was similar, with values of 1.33, 1.17, and 1 for $N = 6$, 16, and 26 respectively. The replanning times for $N = 6$, 16, and 26 were 0.64, 0.84, and 0.66 secs, respectively.

VI. Conclusions

We proposed a new reactive multi-robot planning algorithm that can adapt to unexpected mission changes while effectively handling potential mission violations. We validated the proposed method both theoretically and experimentally. Our future work will focus on extensions to unknown environments or missions expressed in natural language.

References

[1] C. Belta, B. Yordanov, and E. A. Gol, *Formal methods for discrete-time dynamical systems*. Springer, 2017, vol. 89.

[2] C. I. Vasile and C. Belta, "Sampling-based temporal logic path planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, Japan, November 2013, pp. 4817–4822.

[3] X. Luo, Y. Kantaros, and M. M. Zavlanos, "An abstraction-free method for multirobot temporal logic optimal control synthesis," *IEEE Transactions on Robotics*, 2021.

[4] J. Tumova and D. V. Dimarogonas, "Multi-agent planning under local ltl specifications and event-based synchronization," *Automatica*, vol. 70, pp. 239–248, 2016.

[5] Y. Kantaros and M. M. Zavlanos, "Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 812–836, 2020.

[6] D. Gujarathi and I. Saha, "Mt*: Multi-robot path planning for temporal logic specifications," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022, pp. 13 692–13 699.

[7] Z. Liu, M. Guo, and Z. Li, "Time minimization and online synchronization for multi-agent systems under collaborative temporal logic tasks," *Automatica*, vol. 159, p. 111377, 2024.

[8] A. Fang, T. Yin, J. Lin, and H. Kress-Gazit, "Continuous execution of high-level collaborative tasks for heterogeneous robot teams," *arXiv preprint arXiv:2406.18019*, 2024.

[9] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local ltl specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.

[10] S. C. Livingston, P. Prabhakar, A. B. Jose, and R. M. Murray, "Patching task-level robot controllers based on a local μ-calculus formula," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 4588–4595.

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

[12] S. Kalluraya, G. J. Pappas, and Y. Kantaros, "Multi-robot mission planning in dynamic semantic environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[13] Z. Li, M. Cai, S. Xiao, and Z. Kan, "Online motion planning with soft metric interval temporal logic in unknown dynamic environment," *IEEE Control Systems Letters*, vol. 6, pp. 2293–2298, 2022.

[14] Z. Zhou, Z. Chen, M. Cai, Z. Li, Z. Kan, and C.-Y. Su, "Vision-based reactive temporal logic motion planning for quadruped robots in unstructured dynamic environments," *IEEE Transactions on Industrial Electronics*, 2023.

[15] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee, "Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees," in *IEEE Conference on Decision and Control (CDC)*, Nice, France, December 2019.

[16] A. Balakrishnan, S. Jakšić, E. A. Aguilar, D. Ničković, and J. V. Deshmukh, "Model-free reinforcement learning for spatiotemporal tasks using symbolic automata," in *IEEE Conference on Decision and Control (CDC)*, 2023.

[17] H. Wang, H. Zhang, L. Li, Z. Kan, and Y. Song, "Task-driven reinforcement learning with action primitives for long-horizon manipulation skills," *IEEE Transactions on Cybernetics*, 2023.

[18] F. Huang, X. Yin, and S. Li, "Failure-robust multi-robot tasks planning under linear temporal logic specifications," in *13th Asian Control Conference (ASCC 2022)*. IEEE, 2022.

[19] F. Faruq, D. Parker, B. Laccrda, and N. Hawes, "Simultaneous task allocation and planning under uncertainty," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2018, pp. 3559–3564.

[20] Z. Zhou, D. J. Lee, Y. Yoshinaga, S. Balakirsky, D. Guo, and Y. Zhao, "Reactive task allocation and planning for quadrupedal and wheeled robot teaming," in *IEEE 18th International Conference on Automation Science and Engineering (CASE)*, 2022, pp. 2110–2117.

[21] S. Kalluraya, G. J. Pappas, and Y. Kantaros, "Resilient temporal logic planning in the presence of robot failures," in *62nd IEEE Conference on Decision and Control (CDC)*, 2023, pp. 7520–7526.

[22] X. Yin, B. Gao, and X. Yu, "Formal synthesis of controllers for safety-critical autonomous systems: Developments and challenges," *arXiv preprint arXiv:2402.13075*, 2024.

[23] C. Banks, S. Wilson, S. Coogan, and M. Egerstedt, "Multi-agent task allocation using cross-entropy temporal logic optimization," in *IEEE International Conference on Robotics and Automation*, 2020.

[24] Z. Li, Z. Liu, M. Guo, and W. Bao, "Fast and adaptive multi-agent planning under collaborative temporal logic tasks via poset product," *Research*.

[25] X. Luo and M. M. Zavlanos, "Temporal logic task allocation in heterogeneous multirobot systems," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3602–3621, 2022.

[26] L. Li, Z. Chen, H. Wang, and Z. Kan, "Fast task allocation of heterogeneous robots with temporal logic and inter-task constraints," *IEEE Robotics and Automation Letters*, 2023.

[27] J. Tumova, S. Karaman, C. Belta, and D. Rus, "Least-violating planning in road networks from temporal logic specifications," in *ACM/IEEE International Conference on Cyber-Physical Systems*, 2016.

[28] M. Lahijanian, M. R. Maly, D. Fried, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative temporal planning in uncertain environments with partial satisfaction guarantees," *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 583–599, 2016.

[29] M. Lahijanian and M. Kwiatkowska, "Specification revision for markov decision processes with optimal trade-off," in *IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 7411–7418.

[30] C.-I. Vasile, J. Tumova, S. Karaman, C. Belta, and D. Rus, "Minimum-violation scltl motion planning for mobility-on-demand," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1481–1488.

[31] M. Cai, M. Mann, Z. Serlin, K. Leahy, and C.-I. Vasile, "Learning minimally-violating continuous control for infeasible linear temporal logic specifications," in *American Control Conference*, 2023.

[32] C. I. Vasile and C. Belta, "Reactive sampling-based temporal logic path planning," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 4310–4315.

[33] C. I. Vasile, X. Li, and C. Belta, "Reactive sampling-based path planning with temporal logic specifications," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 1002–1028, 2020.

[34] P. Gastin and D. Oddoux, "Fast ltl to büchi automata translation," in *International Conference on Computer Aided Verification*. Springer, 2001, pp. 53–65.