Sampling-Based Optimal Control Synthesis for Multi-Robot Systems under Global Temporal Tasks

Yiannis Kantaros, *Student Member, IEEE,* and Michael M. Zavlanos, *Member, IEEE*

***Abstract*—This paper proposes a new optimal control synthesis algorithm for multi-robot systems under global temporal logic tasks. Existing planning approaches under global temporal goals rely on graph search techniques applied to a product automaton constructed among the robots. In this paper, we propose a new sampling-based algorithm that builds incrementally trees that approximate the state-space and transitions of the synchronous product automaton. By approximating the product automaton by a tree rather than representing it explicitly, we require much fewer memory resources to store it and motion plans can be found by tracing sequences of parent nodes without the need for sophisticated graph search methods. This significantly increases the scalability of our algorithm compared to existing optimal control synthesis methods. We also show that the proposed algorithm is probabilistically complete and asymptotically optimal. Finally, we present numerical experiments showing that our approach can synthesize optimal plans from product automata with billions of states, which is not possible using standard optimal control synthesis algorithms or model checkers.**

***Index Terms*—Temporal logic planning, optimal control synthesis, sampling-based motion planning, multi-robot systems.**

I. Introduction

ROBOT motion planning is a fundamental problem that has received considerable research attraction [1]. The basic motion planning problem consists of generating robot trajectories that reach a desired goal region starting from an initial configuration while avoiding obstacles. More recently, a new class of planning approaches have been developed that can handle a richer class of tasks than the classical point-to-point navigation, and can capture temporal and boolean specifications. Such tasks can be, e.g., sequencing or coverage [2], data gathering [3], intermittent communication [4], or persistent surveillance [5], and can be modeled using formal languages, such as Linear Temporal Logic (LTL) [6], [7], that are developed in concurrency theory. Given a task described by a formal language, model checking algorithms can be employed to synthesize correct-by-construction controllers that satisfy the assigned tasks.

Control synthesis for mobile robots under complex tasks, captured by Linear Temporal Logic (LTL) formulas, build upon either bottom-up approaches when independent LTL expressions are assigned to robots [8]–[10] or top-down approaches when a global LTL formula describing a collaborative task is assigned to a team of robots [11], [12], as in this work. Common in the above works is that they rely on model checking theory [6], [7] to find paths that satisfy LTL-specified tasks, without optimizing task performance. Optimal control synthesis under local and global LTL specifications has been addressed in [13], [14] and [15]–[17], respectively. In top-down approaches [15]–[17], optimal discrete plans are derived for every robot using the individual transition systems that capture robot mobility and a Non-deterministic Büchi Automaton (NBA) that represents the global LTL specification. Specifically, by taking the synchronous product among the transition systems and the NBA, a synchronous product automaton can be constructed. Then, representing the latter automaton as a graph and using graph-search techniques, optimal motion plans can be derived that satisfy the global LTL specification and optimize a cost function. As the number of robots or the size of the NBA increases, the state-space of the product automaton grows exponentially and, as a result, graph-search techniques become intractable. Consequently, these motion planning algorithms scale poorly with the number of robots and the complexity of the assigned task.

To mitigate these issues, we propose an optimal control synthesis algorithm for multi-robot systems under global temporal specifications that avoids the explicit construction of the product among the transition systems and the NBA. Motivated by the RRT* algorithm [18], we build incrementally through a Büchi-guided sampling-based algorithm directed trees that approximately represent the state-space and transitions among states of the synchronous product automaton. Specifically, first a tree is built until a path from an initial to an *accepting* state is constructed. This path corresponds to the prefix part of the motion plan and is executed once. Then, a new tree rooted at an accepting state is constructed in a similar way until a cycle-detection method discovers a loop around the root. This cyclic path corresponds to the suffix part of the motion plan and is executed indefinitely. The advantage of the proposed method is that approximating the product automaton by a tree rather than representing it explicitly by an arbitrary graph, as existing works do, results in significant savings in resources both in terms of memory to save the associated data structures and in terms of computational cost in applying graph search techniques. In this way, our proposed model-checking algorithm scales much better compared to existing top-down approaches. Also, we show that the proposed LTL-based planning algorithm is probabilistically complete and asymptotically optimal. We present numerical simulations that show that the proposed approach can synthesize optimal motion plans from product automata with billions of states, which is impossible using existing optimal control synthesis algorithms or the off-the-shelf symbolic model checkers PRISM [19] and NuSMV [20].

Yiannis Kantaros and Michael M. Zavlanos are with the Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC 27708, USA. {yiannis.kantaros,michael.zavlanos}@duke.edu. This work is supported in part by NSF under grant IIS #1302283 and by ONR under grant #N000141812374.

To the best of our knowledge, the most relevant works are presented in [21]–[24]. In [21], a sampling-based algorithm is proposed which builds incrementally a Kripke structure until it is expressive enough to generate a motion plan that satisfies a task specification expressed in deterministic μ-calculus. Specifically, in [21], since control synthesis from ω-regular languages requires cyclic patterns, an RRG-like algorithm is employed to construct a Kripke structure. However, building arbitrary graph structures to represent transition systems, compromises scalability of temporal logic planning methods since, as the number of samples increases, so does the density of the constructed graph increasing in this way the required resources to save the associated structure and search for optimal plans using graph search methods. Therefore, the algorithm proposed in [21] can be typically used only for single-robot motion planning problems in simple environments and for LTL tasks associated with small NBA. Motivated by this limitation, in [22], a sampling-based temporal logic path planning algorithm is proposed, that also builds upon the RRG algorithm, but constructs incrementally sparse graphs representing transition systems that are then used to construct a product automaton. Then, correct-by-construction discrete plans are synthesized applying graph search methods on the product automaton. However, similar to [21], as the number of samples increases, the sparsity of the constructed graph is lost and as a result this method does not scale well to large planning problems either. Common in the works [21], [22] is that a discrete abstraction of the environment is built until it becomes expressive enough to generate a motion plan that satisfies the LTL specification. To the contrary, our proposed sampling-based approach, given a discrete abstraction of the environment [25]–[29], builds trees, instead of arbitrary graphs, to approximate the product automaton. Therefore, it is more economical in terms of memory requirements and does not require the application of expensive graph search techniques to find the optimal motion plan, but instead it tracks sequences of parent nodes starting from desired accepting states. In this way, we can handle more complex planning problems with more robots and LTL tasks that correspond to larger NBA, compared to the ones that can be solved using the approach in [22]. Moreover, we show that our proposed planning algorithm is asymptotically optimal which is not the case in [22].

In our previous work [23], [24], we have proposed sampling-based planning algorithms for multi-robot systems under global temporal logic tasks. Specifically, [23] transforms given transition systems that abstract robot mobility into trace-included transition systems with smaller state-spaces that are still rich enough to construct motion plans that satisfy the global LTL specification. However, this algorithm does not scale well with the number of robots, since it relies on the construction of a product automaton among all agents. A more tractable and memory-efficient approach is proposed in [24] that builds trees incrementally, similar to the approach proposed here, that approximate the product automaton. Here, we extend the work in [24] by improving the sampling process of the algorithm so that samples are drawn among the sets of nodes that are reachable from the current tree rather than drawn arbitrarily from the state-space of the product automaton, as in [24]. Since the state-space of the product automaton can be arbitrarily large drawing samples randomly can result in very slow growth of the tree, since these samples are not necessarily reachable from the current tree, as we show through numerical experiments. As in [24], here too we show that the proposed algorithm is probabilistically complete and asymptotically optimal. Nevertheless, the completeness and optimality proofs are entirely different due to the different sampling process. To the best of our knowledge, this is the first optimal control synthesis algorithm for global temporal task specifications that is probabilistically complete, asymptotically optimal, and scalable, as it is resource efficient both in terms of memory requirements and computational time.

The rest of the paper is organized as follows. In Section II, we provide a brief overview of LTL and in Section III we present the problem formulation. In Section IV we describe our proposed sampling-based planning algorithm and we examine its correctness and optimality in Section V. Numerical experiments are presented in Section VI.

II. Preliminaries

In this section we formally describe Linear Temporal Logic (LTL) by presenting its syntax and semantics. Also, we briefly review preliminaries of automata-based LTL model checking. A detailed overview of this theory can be found in [6].

Linear temporal logic is a type of formal logic whose basic ingredients are a set of atomic propositions $\mathcal{AP}$, the boolean operators, i.e., conjunction $\wedge$, and negation $\neg$, and two temporal operators, next $\bigcirc$ and until $\mathcal{U}$. LTL formulas over a set $\mathcal{AP}$ can be constructed based on the following grammar: $\phi ::= \text{true} \mid \pi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc \phi \mid \phi_1 \ \mathcal{U} \ \phi_2$, where $\pi \in \mathcal{AP}$. For the sake of brevity we abstain from presenting the derivations of other Boolean and temporal operators, e.g., *always* $\Box$, *eventually* $\Diamond$, *implication* $\Rightarrow$, which can be found in [6].

An infinite *word* σ over the alphabet $2^{\mathcal{AP}}$ is defined as an infinite sequence $\sigma = \pi_0\pi_1\pi_2\cdots \in (2^{\mathcal{AP}})^\omega$, where ω denotes an infinite repetition and $\pi_k \in 2^{\mathcal{AP}}$, $\forall k \in \mathbb{N}$. The language $\texttt{Words}(\phi) = \left\{\sigma \in (2^{\mathcal{AP}})^\omega | \sigma \models \phi\right\}$ is defined as the set of words that satisfy the LTL formula ϕ, where $\models \subseteq (2^{\mathcal{AP}})^\omega \times \phi$ is the satisfaction relation.

Any LTL formula ϕ can be translated into a Nondeterministic Büchi Automaton (NBA) over $(2^{\mathcal{AP}})^\omega$ denoted by B [30], which is defined as follows:

Definition 2.1: A *Nondeterministic Büchi Automaton* (NBA) B over $2^{\mathcal{AP}}$ is defined as a tuple $B = \left(\mathcal{Q}_B, \mathcal{Q}_B^0, \Sigma, \rightarrow_B, \mathcal{Q}_B^F\right)$, where $\mathcal{Q}_B$ is the set of states, $\mathcal{Q}_B^0 \subseteq \mathcal{Q}_B$ is a set of initial states, $\Sigma = 2^{\mathcal{AP}}$ is an alphabet, $\rightarrow_B \subseteq \mathcal{Q}_B \times \Sigma \times \mathcal{Q}_B$ is the transition relation, and $\mathcal{Q}_B^F \subseteq \mathcal{Q}_B$ is a set of accepting/final states.

An *infinite run* ρ_B of B over an infinite word $\sigma = \pi_0\pi_1\pi_2\ldots$, where $\pi_k \in \Sigma$, for all $k \in \mathbb{N}$, is a sequence $\rho_B = q_B^0 q_B^1 q_B^2 \ldots$ such that $q_B^0 \in \mathcal{Q}_B^0$ and $(q_B^k, \pi_k, q_B^{k+1}) \in \rightarrow_B$, $\forall k \in \mathbb{N}$. An infinite run ρ_B is called *accepting* if $\texttt{Inf}(\rho_B) \cap \mathcal{Q}_B^F \neq \varnothing$, where $\texttt{Inf}(\rho_B)$ represents the set of states that appear in ρ_B infinitely often. The words σ that result in an

accepting run of B constitute the accepted language of B, denoted by $\mathcal{L}_B$. It is proven [6] that the accepted language of B is equivalent to the words of ϕ, i.e., $\mathcal{L}_B = \texttt{Words}(\phi)$.

III. Problem Formulation

Consider N mobile robots that evolve in a complex workspace $\mathcal{W} \subset \mathbb{R}^d$ according to the following dynamics: $\dot{\mathbf{x}}_i(t) = f_i(\mathbf{x}_i(t), \mathbf{u}_i(t))$, where $\mathbf{x}_i(t)$ and $\mathbf{u}_i(t)$ are the position and the control input associated with robot i, $i \in \{1, \ldots, N\}$. We assume that there are W disjoint regions of interest in $\mathcal{W}$ that are worth investigation or surveillance. The j-th region is denoted by ℓ_j and it can be of any arbitrary shape. Given the robot dynamics, robot mobility in the workspace $\mathcal{W}$ can be represented by a weighted transition system (wTS) obtained through an abstraction process [25]–[29].The wTS for robot i is defined as follows

Definition 3.1 (wTS): A *weighted Transition System* (wTS) for robot i, denoted by wTS_i is a tuple $\text{wTS}_i = \left(\mathcal{Q}_i, q_i^0, \rightarrow_i, w_i, \mathcal{AP}_i, L_i\right)$ where: (a) $\mathcal{Q}_i = \bigcup_{j=1}^{W} \{q_i^{\ell_j}\}$ is the set of states, where a state $q_i^{\ell_j}$ indicates that robot i is at location ℓ_j; (b) $q_i^0 \in \mathcal{Q}_i$ is the initial state of robot i; $\rightarrow_i \subseteq \mathcal{Q}_i \times \mathcal{Q}_i$ is the transition relation for robot i. Given the robot dynamics, if there is a control input $\mathbf{u}_i$ that can drive robot i from location ℓ_j to ℓ_e, then there is a transition from state $q_i^{\ell_j}$ to $q_i^{\ell_e}$ denoted by $(q_i^{\ell_j}, q_i^{\ell_e}) \in \rightarrow_i$; (c) $w_i : \mathcal{Q}_i \times \mathcal{Q}_i \rightarrow \mathbb{R}_+$[1] is a cost function that assigns weights/cost to each possible transition in wTS. For example, such costs can be associated with the distance that needs to be traveled by robot i in order to move from state $q_i^{\ell_j}$ to state $q_i^{\ell_k}$; (d) $\mathcal{AP}_i = \bigcup_{j=1}^{W} \{\pi_i^{\ell_j}\}$ is the set of atomic propositions, where $\pi_i^{\ell_j}$ is true if robot i is inside region ℓ_j and false otherwise; and (e) $L_i : \mathcal{Q}_i \rightarrow 2^{\mathcal{AP}_i}$ is an observation/output relation giving the set of atomic propositions that are satisfied in a state, i.e., $L_i(q_i^{\ell_j}) = \pi_i^{\ell_j}$.

Given the definition of the wTS, we can define the synchronous *Product Transition System* (PTS), which captures all the possible combinations of robots' states in their respective wTS_i, as follows [6]:

Definition 3.2 (PTS): Given N transition systems $\text{wTS}_i = (\mathcal{Q}_i, q_i^0, \rightarrow_i, w_i, \mathcal{AP}_i, L_i)$, the *product transition system* $\text{PTS} = \text{wTS}_1 \otimes \text{wTS}_2 \otimes \cdots \otimes \text{wTS}_N$ is a tuple $\text{PTS} = (\mathcal{Q}_{\text{PTS}}, q_{\text{PTS}}^0, \longrightarrow_{\text{PTS}}, w_{\text{PTS}}, \mathcal{AP}, L_{\text{PTS}})$ where (a) $\mathcal{Q}_{\text{PTS}} = \mathcal{Q}_1 \times \mathcal{Q}_2 \times \cdots \times \mathcal{Q}_N$ is the set of states; (b) $q_{\text{PTS}}^0 = (q_1^0, q_2^0, \ldots, q_N^0) \in \mathcal{Q}_{\text{PTS}}$ is the initial state, (c) $\longrightarrow_{\text{PTS}} \subseteq \mathcal{Q}_{\text{PTS}} \times \mathcal{Q}_{\text{PTS}}$ is the transition relation defined by the rule[2] $\frac{\bigwedge_{\forall i}(q_i \rightarrow_i q_i')}{q_{\text{PTS}} \rightarrow_{\text{PTS}} q_{\text{PTS}}'}$, where with slight abuse of notation $q_{\text{PTS}} = (q_1, \ldots, q_N) \in \mathcal{Q}_{\text{PTS}}$, $q_i \in \mathcal{Q}_i$. The state q_{PTS}' is defined accordingly. In words, this transition rule says that there exists a transition from q_{PTS} to q_{PTS}' if there exists a transition from q_i to q_i' for all $i \in \{1, \ldots, N\}$; (d) $w_{\text{PTS}} : \mathcal{Q}_{\text{PTS}} \times \mathcal{Q}_{\text{PTS}} \rightarrow \mathbb{R}_+$ is a cost function that assigns weights/cost to each possible transition in PTS, defined as $w_{\text{PTS}}(q_{\text{PTS}}, q_{\text{PTS}}') = \sum_{i=1}^{N} w_i(\Pi|_{\text{wTS}_i} q_{\text{PTS}}, \Pi|_{\text{wTS}_i} q_{\text{PTS}}') \geq 0$, where $q_{\text{PTS}}', q_{\text{PTS}} \in \mathcal{Q}_{\text{PTS}}$, and $\Pi|_{\text{wTS}_i} q_{\text{PTS}}$ stands for the projection of state q_{PTS} onto the state space of wTS_i. The state $\Pi|_{\text{wTS}_i} q_{\text{PTS}} \in \mathcal{Q}_i$ is obtained by removing all states in q_{PTS} that do not belong to $\mathcal{Q}_i$; (e) $\mathcal{AP} = \bigcup_{i=1}^{N} \mathcal{AP}_i$ is the set of atomic propositions; and, (f) $L_{\text{PTS}} = \bigcup_{\forall i} L_i : \mathcal{Q}_{\text{PTS}} \rightarrow 2^{\mathcal{AP}}$ is an observation/output relation giving the set of atomic propositions that are satisfied at a state $q_{\text{PTS}} \in \mathcal{Q}_{\text{PTS}}$.

In what follows, we give definitions related to the PTS, that we will use throughout the rest of the paper. An *infinite path* τ of a PTS is an infinite sequence of states, $\tau = \tau(1)\tau(2)\tau(3)\ldots$ such that $\tau(1) = q_{\text{PTS}}^0$, $\tau(k) \in \mathcal{Q}_{\text{PTS}}$, and $(\tau(k), \tau(k+1)) \in \rightarrow_{\text{PTS}}$, $\forall k \in \mathbb{N}_+$, where k is an index that points to the k-th entry of τ denoted by $\tau(k)$. A *finite path* of a PTS can be defined accordingly. The only difference with the infinite path is that a finite path is defined as a finite sequence of states of a PTS. Given the definition of the weights w_{PTS} in Definition 3.2, the *cost* of a finite path τ, denoted by $\hat{J}(\tau)$, can be defined as

$$\hat{J}(\tau) = \sum_{k=1}^{|\tau|-1} w_{\text{PTS}}(\tau(k), \tau(k+1)). \tag{1}$$

In (1), $|\tau|$ stands for the number of states in τ. In words, the cost (1) captures the total cost incurred by all robots during the execution of the finite path τ. Notice that the cost function $\hat{J}(\cdot)$ is additive, i.e., $\hat{J}(\tau_1|\tau_2) = \hat{J}(\tau_1) + \hat{J}(\tau_2)$, where τ_1 and τ_2 are two finite paths of the PTS so that there is a feasible transition from the last state in τ_1 to the first state in τ_2, according to $\rightarrow_{\text{PTS}}$, and $|$ stands for concatenation. Also, since $\hat{J}(\cdot)$ is additive and $w_{\text{PTS}}(q_{\text{PTS}}, q_{\text{PTS}}') \geq 0$ by Definition 3.2, we get that $\hat{J}(\cdot)$ is monotone i.e., $\hat{J}(\tau_1) \leq \hat{J}(\tau_1|\tau_2)$.

The *trace* of an infinite path $\tau = \tau(1)\tau(2)\tau(3)\ldots$ of a PTS, denoted by $\texttt{trace}(\tau) \in \left(2^{\mathcal{AP}}\right)^{\omega}$, where ω denotes infinite repetition, is an infinite word that is determined by the sequence of atomic propositions that are true in the states along τ, i.e., $\texttt{trace}(\tau) = L(\tau(1))L(\tau(2))\ldots$.

Given the PTS and the NBA B that corresponds to the LTL ϕ, we can now define the *Product Büchi Automaton* (PBA) $P = \text{PTS} \otimes B$ [6], as follows:

Definition 3.3 (PBA): Given the product transition system $\text{PTS} = (\mathcal{Q}_{\text{PTS}}, q_{\text{PTS}}^0, \longrightarrow_{\text{PTS}}, w_{\text{PTS}}, \mathcal{AP}, L_{\text{PTS}})$ and the NBA $B = (\mathcal{Q}_B, \mathcal{Q}_B^0, \Sigma, \rightarrow_B, \mathcal{Q}_B^F)$, we can define the *Product Büchi Automaton* $P = \text{PTS} \otimes B$ as a tuple $P = (\mathcal{Q}_P, \mathcal{Q}_P^0, \longrightarrow_P, w_P, \mathcal{Q}_P^F)$ where (a) $\mathcal{Q}_P = \mathcal{Q}_{\text{PTS}} \times \mathcal{Q}_B$ is the set of states; (b) $\mathcal{Q}_P^0 = q_{\text{PTS}}^0 \times \mathcal{Q}_B^0$ is a set of initial states; (c) $\longrightarrow_P \subseteq \mathcal{Q}_P \times 2^{\mathcal{AP}} \times \mathcal{Q}_P$ is the transition relation defined by the rule: $\frac{\left(q_{\text{PTS}} \rightarrow_{\text{PTS}} q_{\text{PTS}}'\right) \wedge \left(q_B \xrightarrow{L_{\text{PTS}}(q_{\text{PTS}})} q_B'\right)}{q_P = (q_{\text{PTS}}, q_B) \longrightarrow_P q_P' = \left(q_{\text{PTS}}', q_B'\right)}$. Transition from state $q_P \in \mathcal{Q}_P$ to $q_P' \in \mathcal{Q}_P$, is denoted by $(q_P, q_P') \in \longrightarrow_P$, or $q_P \longrightarrow_P q_P'$; (d) $w_P(q_{\text{P}}, q_{\text{P}}') = w_{\text{PTS}}(q_{\text{PTS}}, q_{\text{PTS}}') \geq 0$, where $q_{\text{P}} = (q_{\text{PTS}}, q_B)$ and $q_{\text{P}}' = (q_{\text{PTS}}', q_B')$; and (e) $\mathcal{Q}_P^F = \mathcal{Q}_{\text{PTS}} \times \mathcal{Q}_B^F$ is a set of accepting/final states.

In what follows, we assume that the robots have to accomplish a complex collaborative task captured by a global LTL statement ϕ defined over the set of atomic propositions $\mathcal{AP} = \bigcup_{i=1}^{N} \mathcal{AP}_i$. Given such an LTL formula ϕ, we define the *language* $\texttt{Words}(\phi) = \left\{\sigma \in (2^{\mathcal{AP}})^{\omega} | \sigma \models \phi\right\}$, where $\models \subseteq (2^{\mathcal{AP}})^{\omega} \times \phi_i$ is the satisfaction relation, as the set of infinite words $\sigma \in (2^{\mathcal{AP}})^{\omega}$ that satisfy the LTL formula ϕ. Given such

[1] $\mathbb{R}_+$ and $\mathbb{N}_+$ stand for the positive real and natural numbers, respectively.

[2] The notation of this rule is along the lines of the notation used in [6]. In particular, it means that if the proposition above the solid line is true, then so does the proposition below the solid line.

a global LTL formula ϕ and the PBA an infinite path τ of a PTS satisfies ϕ if and only if $\texttt{trace}(\tau) \in \texttt{Words}(\phi)$, which is equivalently denoted by $\tau \models \phi$.

Given an LTL formula ϕ, if there is a path satisfying ϕ, then there exists a path $\tau \models \phi$ that can be written in a finite representation, called prefix-suffix structure, i.e., $\tau = \tau^{\text{pre}}[\tau^{\text{suf}}]^{\omega}$, where the prefix part τ^{pre} is executed only once followed by the indefinite execution of the suffix part τ^{suf} [14], [31]. The prefix part τ^{pre} is the projection of a finite path of the PBA, i.e., a finite sequence of states of the PBA, denoted by p^{pre}, onto $\mathcal{Q}_{\text{PTS}}$, which has the following structure $p^{\text{pre}} = (q^0_{\text{PTS}}, q^0_B)(q^1_{\text{PTS}}, q^1_B)\dots(q^K_{\text{PTS}}, q^K_B)$ with $(q^K_{\text{PTS}}, q^K_B) \in \mathcal{Q}^F_B$. The suffix part τ^{suf} is the projection of a finite path of the PBA, denoted by p^{suf}, onto $\mathcal{Q}_{\text{PTS}}$, which has the following structure $p^{\text{suf}} = (q^K_{\text{PTS}}, q^K_B)(q^{K+1}_{\text{PTS}}, q^{K+1}_B)\dots(q^{K+S}_{\text{PTS}}, q^{K+S}_B)(q^{K+S+1}_{\text{PTS}}, q^{K+S+1}_B)$, where $(q^{K+S+1}_{\text{PTS}}, q^{K+S+1}_B) = (q^K_{\text{PTS}}, q^K_B)$. Then our goal is to compute a plan $\tau = \tau^{\text{pre}}[\tau^{\text{suf}}]^{\omega} = \Pi|_{\text{PTS}} p^{\text{pre}}[\Pi|_{\text{PTS}} p^{\text{suf}}]^{\omega}$, where $\Pi|_{\text{PTS}}$ stands for the projection on the state-space $\mathcal{Q}_{\text{PTS}}$, so that the following objective function is minimized

$$J(\tau) = \hat{J}(\tau^{\text{pre}}) + \hat{J}(\tau^{\text{suf}}), \tag{2}$$

which captures the total cost incurred by all robots during the execution of the prefix and a single execution of the suffix part. In (2), $\hat{J}(\tau^{\text{pre}})$ and $\hat{J}(\tau^{\text{suf}})$ stands for the cost of the prefix and suffix part, where the cost function $\hat{J}(\cdot)$ is defined in (1), i.e., $\hat{J}(\tau^{\text{pre}}) = \sum_{k=1}^{K-1} w_{\text{PTS}}(\Pi|_{\text{PTS}} p^{\text{pre}}(k), \Pi|_{\text{PTS}} p^{\text{pre}}(k+1))$, $\hat{J}(\tau^{\text{suf}}) = \sum_{k=K}^{K+S} w_{\text{PTS}}(\Pi|_{\text{PTS}} p^{\text{suf}}(k) \Pi|_{\text{PTS}} p^{\text{suf}}(k+1))$. Specifically, in this paper we address the following problem.

Problem 1: Given a global LTL specification ϕ, and transition systems wTS_i, for all robots i, determine a discrete team plan τ that satisfies ϕ, i.e., $\tau \models \phi$, and minimizes the cost function (2).

A. A Solution to Problem 1

Problem 1 is typically solved by applying graph-search methods to the PBA, see e.g., [14], [31]. Specifically, to generate a motion plan τ that satisfies ϕ, the PBA is viewed as a weighted directed graph $\mathcal{G}_P = \{\mathcal{V}_P, \mathcal{E}_P, w_P\}$, where the set of nodes $\mathcal{V}_P$ is indexed by the set of states $\mathcal{Q}_P$, the set of edges $\mathcal{E}_P$ is determined by the transition relation $\longrightarrow_P$, and the weights assigned to each edge are determined by the function w_P. Then we find the shortest paths from the initial states to all reachable final/accepting states $q_P \in \mathcal{Q}^F_P$ and projecting these paths onto the PTS results in the prefix parts $\tau^{\text{pre},a}$, where $a \in \{1, \dots, |\mathcal{Q}^F_P|\}$. The respective suffix parts $\tau^{\text{suf},a}$ are constructed similarly by computing the shortest cycle around the a-th accepting state. All the resulting motion plans $\tau^a = \tau^{\text{pre},a}[\tau^{\text{suf},a}]^{\omega}$ satisfy the LTL specification ϕ. Among all these plans, we can easily compute the optimal plan that minimizes the cost function defined in (2) by computing the cost $J(\tau^a)$ for all plans and selecting the one with the smallest cost; see e.g. [14], [16], [17].

IV. Sampling-based Optimal Control Synthesis

Since the size of the PBA can grow arbitrarily large with the number of robots and the complexity of the task, constructing this PBA and applying graph-search techniques to find optimal plans, as discussed in Section III-A, is resource demanding and computationally expensive. In this section, we propose a sampling-based planning algorithm that is scalable and constructs a discrete motion plan τ in prefix-suffix structure, i.e., $\tau = \tau^{\text{pre}}[\tau^{\text{suf}}]^{\omega}$, that satisfies a given global LTL specification ϕ. The procedure is based on the incremental construction of a directed tree that approximately represents the state-space $\mathcal{Q}_P$ and the transition relation $\rightarrow_P$ of the PBA defined in Definition 3.3. The construction of the prefix and the suffix part is described in Algorithm 1. In Algorithm 1, first the LTL formula is translated to a NBA $B = \{\mathcal{Q}_B, \mathcal{Q}^0_B, \rightarrow_B, \mathcal{Q}^F_B\}$ [line 1, Alg. 1]. Then, in lines 2-8, the prefix parts $\tau^{\text{pre},a}$ are constructed, followed by the construction of their respective suffix parts $\tau^{\text{suf},a}$ in lines 9-21. Finally, using the constructed prefix and suffix parts, the optimal plan $\tau = \tau^{\text{pre},a^*}[\tau^{\text{suf},a^*}]^{\omega} \models \phi$ is synthesized in lines 22-23.

Algorithm 1: Construction of Optimal plans $\tau \models \phi$

Input: Logic formula ϕ, Transition systems $\text{wTS}_1, \dots, \text{wTS}_N$, Initial location $q^0_{\text{PTS}} \in \mathcal{Q}_{PTS}$, maximum numbers of iterations $n^{\text{pre}}_{\max}$, $n^{\text{suf}}_{\max}$

Output: Optimal plans $\tau \models \phi$

1 Convert ϕ to a NBA $B = (\mathcal{Q}_B, \mathcal{Q}^0_B, \rightarrow_B, \mathcal{Q}^F_B)$;
2 Define goal set: $\mathcal{X}^{\text{pre}}_{\text{goal}}$;
3 **for** $b_0 = 1 : |\mathcal{Q}^0_B|$ **do**
4 Initial NBA state: $q^0_B = \mathcal{Q}^0_B(b_0)$;
5 Root of the tree: $q^r_P = (q^0_{\text{PTS}}, q^0_B)$;
6 $[\mathcal{G}_T, \mathcal{P}] = \texttt{ConstructTree}(\mathcal{X}^{\text{pre}}_{\text{goal}}, \text{wTS}_1, \dots, \text{wTS}_N, B, q^r_P, n^{\text{pre}}_{\max})$;
7 **for** $a = 1 : |\mathcal{P}|$ **do**
8 $\tau^{\text{pre},a} = \texttt{FindPath}(\mathcal{G}_T, q^r_P, \mathcal{P}(a))$;
9 **for** $a = 1 : |\mathcal{P}|$ **do**
10 Root of the tree: $q^r_P = \mathcal{P}(a)$;
11 Define goal set: $\mathcal{X}^{\text{suf}}_{\text{goal}}(q^r_P)$;
12 **if** $(q^r_P \in \mathcal{X}^{suf}_{goal}) \ \wedge \ (w_P(q^r_P, q^r_P) = 0)$ **then**
13 $\mathcal{G}_T = (\{q^r_P\}, \{q^r_P, q^r_P\}, 0)$;
14 $\mathcal{S}_a = \{q^r_P\}$;
15 **else**
16 $[\mathcal{G}_T, \mathcal{S}_a] = \texttt{ConstructTree}(\mathcal{X}^{\text{suf}}_{\text{goal}}, \text{wTS}_1, \dots, \text{wTS}_N, B, q^r_P, n^{\text{suf}}_{\max})$;
17 **for** $e = 1 : |\mathcal{S}_a|$ **do**
18 $\tilde{\tau}^{\text{suf},e} = \texttt{FindPath}(\mathcal{G}_T, q^r_P, \mathcal{S}_f(e))$;
19 $e^* = \text{argmin}_e(\hat{J}(\tilde{\tau}^{\text{suf},e}))$;
20 $\tau^{\text{suf},a} = \tilde{\tau}^{\text{suf},e^*}$
21 $a_{q^0_B} = \text{argmin}_a(\hat{J}(\tau^{\text{pre},a}) + \hat{J}(\tau^{\text{suf},a}))$;
22 $a^* = \text{argmin}_{a_{q^0_B}}(\hat{J}(\tau^{\text{pre}}_{q^0_B}) + \hat{J}(\tau^{\text{suf}}_{q^0_B}))$;
23 Optimal Plan: $\tau = \tau^{\text{pre},a^*}[\tau^{\text{suf},a^*}]^{\omega}$;

In what follows, we denote by $\mathcal{G}_T = \{\mathcal{V}_T, \mathcal{E}_T, \texttt{Cost}\}$ the tree that approximately represents the PBA P. Also, we denote by q^r_P the root of $\mathcal{G}_T$. The set of nodes $\mathcal{V}_T$ contains the states of $\mathcal{Q}_P$ that have already been sampled and added to the tree structure. The set of edges $\mathcal{E}_T$ captures transitions between nodes in $\mathcal{V}_T$, i.e., $(q_P, q'_P) \in \mathcal{E}_T$, if there is a transition from state $q_P \in \mathcal{V}_T$ to state $q'_P \in \mathcal{V}_T$. The function $\texttt{Cost} : \mathcal{V}_T :\rightarrow$

Algorithm 2: Function $[\mathcal{G}_T, \mathcal{Z}] = \texttt{ConstructTree}(\mathcal{X}_{\text{goal}}, \text{wTS}_1, \ldots, \text{wTS}_N, B, q_P^r, n_{\max})$

$\mathcal{V}_T = \{q_P^r\}$;
$\mathcal{E}_T = \emptyset$;
$\texttt{Cost}(q_P^r) = 0$;
for $n = 1 : n_{max}$ **do**
  $q_{\text{PTS}}^{\text{new}} = \texttt{Sample}(\mathcal{V}_T, \text{wTS}_1, \ldots, \text{wTS}_N)$;
  for $b = 1 : |\mathcal{Q}_B|$ **do**
    $q_B^{\text{new}} = \mathcal{Q}_B(b)$;
    $q_P^{\text{new}} = (q_{\text{PTS}}^{\text{new}}, q_B^{\text{new}})$;
    if $q_P^{new} \notin \mathcal{V}_T$ **then**
      $[\mathcal{V}_T,\ \mathcal{E}_T, \texttt{Cost}] = \texttt{Extend}(q_P^{\text{new}}, \rightarrow_P)$;
    if $q_P^{new} \in \mathcal{V}_T$ **then**
      $[\mathcal{E}_T, \texttt{Cost}] = \texttt{Rewire}(q_P^{\text{new}}, \mathcal{V}_T, \mathcal{E}_T, \texttt{Cost})$;
$\mathcal{Z} = \mathcal{V}_T \cap \mathcal{X}_{\text{goal}}$;

$\mathbb{R}_+$ assigns the cost of reaching node $q_P \in \mathcal{V}_T$ from the root q_P^r of the tree. In other words,

$$\texttt{Cost}(q_P) = \hat{J}(\tau_T), \tag{3}$$

where $q_P \in \mathcal{V}_T$ and τ_T is the path in the tree $\mathcal{G}_T$ that connects the root to q_P.

A. Construction of Prefix Parts

In this Section, we describe how to construct the tree $\mathcal{G}_T = \{\mathcal{V}_T, \mathcal{E}_T, \texttt{Cost}\}$ that will be used for the synthesis of the prefix part [lines 2-8, Alg. 1]. Since the prefix part connects an initial state $q_P^0 = (q_{\text{PTS}}^0, q_B^0) \in \mathcal{Q}_P^0$ to an *accepting* state $q_P = (q_{\text{PTS}},\ q_B) \in \mathcal{Q}_P^F$, with $q_B \in \mathcal{Q}_B^F$, we can define the goal region for the tree $\mathcal{G}_T$, as [line 2, Alg. 1]

$$\mathcal{X}_{\text{goal}}^{\text{pre}} = \{q_P = (q_{\text{PTS}},\ q_B) \in \mathcal{Q}_P \mid q_B \in \mathcal{Q}_B^F\}. \tag{4}$$

The root q_P^r of the tree is an initial state $q_P^0 = (q_{\text{PTS}}^0, q_B^0)$ of the PBA and the following process is repeated for each initial state $q_B^0 \in \mathcal{Q}_B^0$ [line 3-5, Alg. 1]. The construction of the tree is described in Algorithm 2 [line 6, Alg. 1]. In line 4 of Algorithm 1, $\mathcal{Q}_B^0(b_0)$ stands for the b_0-th initial state assuming an arbitrary enumeration of the elements of the set $\mathcal{Q}_B^0$. The set $\mathcal{V}_T$ initially contains only the root q_P^r, i.e., an initial state of the PBA [line 1 , Alg. 2] and, therefore, the set of edges is initialized as $\mathcal{E}_T = \emptyset$ [line 2, Alg. 2]. By convention, we assume that the cost of q_P^r is zero [line 3, Alg. 2].

1) Sampling a state $q_P^{new} \in \mathcal{Q}_P$: The first step in the construction of the graph $\mathcal{G}_T$ is to sample a state from the state-space of the PBA. This is achieved by a sampling function `Sample`; see Algorithm 3. Specifically, we first create a state $q_{\text{PTS}}^{\text{rand}} = \Pi|_{\text{PTS}} q_P^{\text{rand}}$, where q_P^{rand} is sampled from a given discrete distribution $f_{\text{rand}}(q_P|\mathcal{V}_T) : \mathcal{V}_T \rightarrow [0, 1]$ [lines 1-2, Alg. 3]. The probability density function $f_{\text{rand}}(q_P|\mathcal{V}_T)$ defines the probability of selecting the state $q_P \in \mathcal{V}_T$ as the state q_P^{rand} at iteration n of Algorithm 2 given the set $\mathcal{V}_T$. We make the following assumption for $f_{\text{rand}}(q_P|\mathcal{V}_T)$.

Assumption 4.1 (Probability density function f_{rand}): (i) The probability density function $f_{\text{rand}}(q_P|\mathcal{V}_T) : \mathcal{V}_T \rightarrow [0, 1]$ is bounded away from zero on $\mathcal{V}_T$. (ii) The probability density function $f_{\text{rand}}(q_P|\mathcal{V}_T) : \mathcal{V}_T \rightarrow [0, 1]$ remains the same for all iterations n and for a given state $q_P \in \mathcal{V}_T$ is monotonically decreasing with respect to the size of $|\mathcal{V}_T|$. This also implies that for a given $q_P \in \mathcal{V}_T$, the probability $f_{\text{rand}}(q_P|\mathcal{V}_T)$ remains the same for all iterations n if the set $\mathcal{V}_T$ does not change. (iii) Independent samples q_P^{rand} can be drawn from f_{rand}.

By definition of $f_{\text{rand}}(q_P|\mathcal{V}_T)$ we have that q_P^{rand} always belongs to $\mathcal{V}_T$. Also, by assumption 4.1(i), we have that any node $q_P \in \mathcal{V}_T$ has a non-zero probability to be selected as the node q_P^{rand}. Also, notice that assumption 4.1(ii) is reasonable, as it requires that the probability of selecting a given state $q_P \in \mathcal{V}_T$ decreases as the cardinality of $\mathcal{V}_T$ increases. An example of a distribution that satisfies Assumption 4.1 is the discrete uniform distribution

$$f_{\text{rand}}(q_P|\mathcal{V}_T) = \begin{cases} \frac{1}{|\mathcal{V}_T|}, & \text{if } q_P \in \mathcal{V}_T, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Notice that other sampling methods for q_P^{rand} can be employed that do not require the more strict conditions of Assumption 4.1(ii); see Remark A.1 in Appendix A.

Given a state $q_{\text{PTS}}^{\text{rand}}$, we define its *reachable set* in the PTS

$$\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}}) = \{q_{\text{PTS}} \in \mathcal{Q}_{\text{PTS}} \mid q_{\text{PTS}}^{\text{rand}} \rightarrow_{\text{PTS}} q_{\text{PTS}}\} \tag{6}$$

i.e., $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}}) \subseteq \mathcal{Q}_{\text{PTS}}$ collects all the states $q_{\text{PTS}} \in \mathcal{Q}_{\text{PTS}}$ that can be reached from $q_{\text{PTS}}^{\text{rand}}$ in one hop. Then, we sample a state $q_{\text{PTS}}^{\text{new}}$ from a discrete distribution $f_{\text{new}}(q_{\text{PTS}}|q_{\text{PTS}}^{\text{rand}}) : \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}}) \rightarrow [0, 1]$ [line 3, Alg. 3] that satisfies the following assumption.

Assumption 4.2 (Probability density function f_{new}): (i) The probability density function $f_{\text{new}}(q_{\text{PTS}}|q_{\text{PTS}}^{\text{rand}}) : \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}}) \rightarrow [0, 1]$ is bounded away from zero on $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}})$. (ii) For a given $q_{\text{PTS}}^{\text{rand}}$, the distribution $f_{\text{new}}(q_{\text{PTS}}|q_{\text{PTS}}^{\text{rand}})$ remains the same for all iterations n. (iii) Given a state $q_{\text{PTS}}^{\text{rand}}$, independent samples $q_{\text{PTS}}^{\text{new}}$ can be drawn from the probability density function f_{new}.

By definition of $f_{\text{new}}(q_{\text{PTS}}|q_{\text{PTS}}^{\text{rand}})$, we have that $q_{\text{PTS}}^{\text{rand}}$ always belongs to $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}})$. Also, observe that by Assumption 4.2(i), we have that any node $q_P \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}})$ has a non-zero probability to be $q_{\text{PTS}}^{\text{new}}$. Moreover, notice that the state q_P^{rand} can change at every iteration n and, therefore, clearly, so does the reachable set $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}})$. Nevertheless, observe that the set $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}})$ remains the same for all iterations n for a given $q_{\text{PTS}}^{\text{rand}}$. Finally, note that other sampling methods for $q_{\text{PTS}}^{\text{new}}$ can be employed that do not require the more strict conditions of Assumption 4.2(ii); see Remark A.2 in Appendix A.

Remark 4.3 (Reachable set $\mathcal{R}_{PTS}(q_{PTS}^{rand})$): In practice in order to obtain the state $q_{\text{PTS}}^{\text{new}}$ we do not need to construct the reachable set $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}})$, which is a computationally expensive step. In fact, we only need the *reachable* sets $\mathcal{R}_{\text{TS}_i}(q_i^{\text{rand}})$, for all robots i, that collect all states that are reachable from the state $q_i^{\text{rand}} = \Pi|_{\text{TS}_i} q_{\text{PTS}}^{\text{rand}} \in \mathcal{Q}_i$ in one hop, defined as $\mathcal{R}_{\text{TS}_i}(q_i^{\text{rand}}) = \{q_i \in \mathcal{Q}_i | q_i^{\text{rand}} \rightarrow_i q_i\}$. Then, we can define the probability density functions $f_{\text{new},i}(q_i|q_i^{\text{rand}}) : \mathcal{R}_{\text{TS}_i}(q_i^{\text{rand}}) \rightarrow [0, 1]$ that are bounded away from zero on $\mathcal{R}_{\text{TS}_i}(q_i^{\text{rand}})$ and use them to draw a sample q_i^{new} that is reachable from q_i^{rand}, for all $i \in \{1, \ldots, N\}$. Stacking these samples in a vector

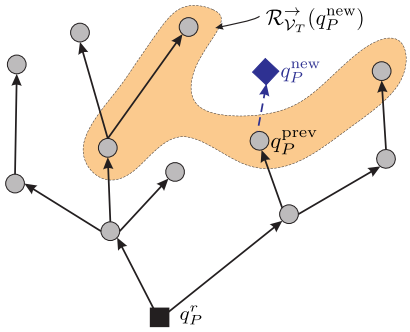


Fig. 1. Graphical depiction of Algorithm 4. The black square stands for the root of the tree and the gray disks represent nodes in the set $\mathcal{V}_T$. Black arrows represent transitions captured by $\mathcal{E}_T$. The blue diamond stands for the state q_P^{new} and the dashed blue arrow represents the new edge that will be added to the set $\mathcal{E}_T$ after the execution of Algorithm 4 (line 5, Alg. 4).

Algorithm 3: Function $\texttt{Sample}(\mathcal{V}_T, \text{wTS}_1, \ldots, \text{wTS}_N)$

1 Pick a state $q_P^{\text{rand}} \in \mathcal{V}_T$ from a given distribution $f_{\text{rand}}(q_P|\mathcal{V}_T) : \mathcal{V}_T \rightarrow [0,1]$;
2 $q_{\text{PTS}}^{\text{rand}} = \Pi|_{\text{PTS}} q_P^{\text{rand}}$;
3 Sample a state $q_{\text{PTS}}^{\text{new}}$ from probability distribution $f_{\text{new}} : \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}}) \rightarrow [0,1]$;
4 **return** q_{PTS}^{new};

we can define $q_{\text{PTS}}^{\text{new}} = [q_1^{\text{new}}, q_2^{\text{new}}, \ldots, q_N^{\text{new}}]$ and the probability density function f_{new} becomes $f_{\text{new}} = f_{\text{new},1} \cdot f_{\text{new},2} \cdot \cdots \cdot f_{\text{new},N}$ that has to satisfy Assumption 4.2. Clearly, the resulting state $q_{\text{PTS}}^{\text{new}}$ belongs to the reachable set $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}})$. Throughout the paper, for simplicity, our analysis is based on f_{new} and not on the probability density functions $f_{\text{new},i}$.

In order to build incrementally a graph whose set of nodes approximates the state-space $\mathcal{Q}_P$ we need to append to $q_{\text{PTS}}^{\text{new}}$ a state from the state-space $\mathcal{Q}_B$ of the NBA B. Let $q_B^{\text{new}} = \mathcal{Q}_B(b)$ [line 7, Alg. 2] be the candidate Büchi state that will be attached to $q_{\text{PTS}}^{\text{new}}$, where $\mathcal{Q}_B(b)$ stands for the b-th state in the set $\mathcal{Q}_B$ assuming an arbitrary enumeration of the elements of the set $\mathcal{Q}_B$. The following procedure is repeated for all $q_B^{\text{new}} = \mathcal{Q}_B(b)$ with $b \in \{1, \ldots, |\mathcal{Q}_B|\}$. First, we construct the state $q_P^{\text{new}} = (q_{\text{PTS}}^{\text{new}}, q_B^{\text{new}}) \in \mathcal{Q}_P$ [line 8, Alg. 2] and then we check if this state can be added to the tree $\mathcal{G}_T$ [lines 9-10, Alg. 2]. If the state q_P^{new} does not already belong to the tree from a previous iteration of Algorithm 2, i.e, if $q_P^{\text{new}} \notin \mathcal{V}_T$ [line 11, Alg. 2], we check which node in $\mathcal{V}_T$ (if there is any) can be the parent of q_P^{new} in the tree $\mathcal{G}_T$. This is achieved by the function `Extend` described in Algorithm 4 [line 10, Alg. 2] and in Section IV-A2. If $q_P^{\text{new}} \in \mathcal{V}_T$, then the *rewiring* step follows described in Algorithm 5 [lines 11-12, Alg. 2] and in Section IV-A3 that aims to reduce the cost of nodes $q_P \in \mathcal{V}_T$.

2) Adding a new edge to $\mathcal{E}_T$: Assume that $q_P^{\text{new}} \notin \mathcal{V}_T$ [lines 9, Alg. 2]. Then the function `Extend` described in Algorithm 4 [line 10, Alg. 2] is used to check if the tree can be extended towards q_P^{new}. The first step in Algorithm 4 is to construct the set $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T}(q_P^{\text{new}}) \subseteq \mathcal{V}_T$ defined as

$$\mathcal{R}^{\rightarrow}_{\mathcal{V}_T}(q_P^{\text{new}}) = \{q_P \in \mathcal{V}_T | q_P \rightarrow_P q_P^{\text{new}}\}, \tag{7}$$

Algorithm 4: Function $\texttt{Extend}(q_P^{\text{new}})$

1 Collect in set $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T}(q_P^{\text{new}})$ all states $q_P \in \mathcal{V}_T$ that satisfy the following transition rule: $(q_P, q_P^{\text{new}}) \in \rightarrow_P$;
2 **if** $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T}(q_P^{new}) \neq \emptyset$ **then**
3 $\quad q_P^{\text{prev}} = \text{argmin}_{q_P \in \mathcal{R}^{\rightarrow}_{\mathcal{V}_T}(q_P^{\text{new}})} [\texttt{Cost}(q_P) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P, \Pi|_{\text{PTS}} q_P^{\text{new}})]$;
4 $\quad \mathcal{V}_T = \mathcal{V}_T \cup \{q_P^{\text{new}}\}$;
5 $\quad \mathcal{E}_T = \mathcal{E}_T \cup \{(q_P^{\text{prev}}, q_P^{\text{new}})\}$;
6 $\quad \texttt{Cost}(q_P^{\text{new}}) = \texttt{Cost}(q_P^{\text{prev}}) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P^{\text{prev}}, \Pi|_{\text{PTS}} q_P^{\text{new}})$;
7 **return** $\mathcal{V}_T$, $\mathcal{E}_T$, $\texttt{Cost}$;

that collects all states $q_P \in \mathcal{V}_T$ that satisfy the transition rule $(q_P, q_P^{\text{new}}) \in \rightarrow_P$, i.e., all states that can directly reach q_P^{new} [line 1, Alg. 4]. If the resulting set $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T}(q_P^{\text{new}})$ is empty then the sample q_P^{new} is not added to the tree. On the other hand, if $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T}(q_P^{\text{new}}) \neq \emptyset$, then the state q_P^{new} is added to the tree [lines 3-6, Alg. 4]. The parent of q_P^{new} is selected as

$$q_P^{\text{prev}} = \text{argmin}_{q_P \in \mathcal{R}^{\rightarrow}_{\mathcal{V}_T}(q_P^{\text{new}})} [\texttt{Cost}(q_P) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P, \Pi|_{\text{PTS}} q_P^{\text{new}})],$$

where $\texttt{Cost}(q_P) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P, \Pi|_{\text{PTS}} q_P^{\text{new}})$ captures the cost of the node q_P^{new} if it gets connected to the root through the node q_P. In other words, the parent q_P^{prev} of node q_P^{new} is selected among all states in $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T}(q_P^{\text{new}})$ so that the cost of q_P^{new} is minimized [line 3, Alg. 4]. The set of nodes and edges is updated as $\mathcal{V}_T = \mathcal{V}_T \cup \{q_P^{\text{new}}\}$ and $\mathcal{E}_T = \mathcal{E}_T \cup \{(q_P^{\text{prev}}, q_P^{\text{new}})\}$ [lines 4 and 5, Alg. 4]. Given the parent q_P^{prev} of the node q_P^{new}, the cost of q_P^{new} is [line 6, Alg. 4]:

$$\texttt{Cost}(q_P^{\text{new}}) = \underbrace{\texttt{Cost}(q_P^{\text{prev}})}_{\text{Cost of reaching } q_P^{\text{prev}} \text{ from the root of the tree } \mathcal{G}_T} + \underbrace{w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P^{\text{prev}}, \Pi|_{\text{PTS}} q_P^{\text{new}})}_{\text{cost of reaching } q_P^{\text{new}} \text{ from } q_P^{\text{prev}}}, \tag{8}$$

due to (1) and (3). Algorithm 4 is illustrated in Figure 1, as well.

3) Rewiring: Once a new state $q_P^{\text{new}} = (q_{\text{PTS}}^{\text{new}}, q_B^{\text{new}})$ has been added to the tree or if the new sample q_P^{new} already belongs to the tree [line 11, Alg. 2], the rewiring step follows [line 12, Alg. 2]. Specifically, we *rewire* the nodes in $q_P \in \mathcal{V}_T$ that can get connected to the root q_P^r through the node q_P^{new} if this rewiring can decrease their cost $\texttt{Cost}(q_P)$. The rewiring process is described in Algorithm 5 and is illustrated in Figure 2.

In Algorithm 5 we first construct the reachable set $\mathcal{R}^{\leftarrow}_{\mathcal{V}_T}(q_P^{\text{new}}) \subseteq \mathcal{V}_T$ defined as

$$\mathcal{R}^{\leftarrow}_{\mathcal{V}_T}(q_P^{\text{new}}) = \{q_P \in \mathcal{V}_T | q_P^{\text{new}} \rightarrow_P q_P\}, \tag{9}$$

that collects all states of $q_P \in \mathcal{V}_T$ that satisfy the transition rule $(q_P^{\text{new}}, q_P) \in \rightarrow_P$, i.e., all states that can be directly reached by q_P^{new} [line 1, Alg. 5]. Then, for all states $q_P \in \mathcal{R}^{\leftarrow}_{\mathcal{V}_T}(q_P^{\text{new}})$ we check if their current cost $\texttt{Cost}(q_P)$ is greater than their cost if they were connected to the root through q_P^{new} [line 3, Alg. 5]. If this is the case for a node $q_P \in \mathcal{R}^{\leftarrow}_{\mathcal{V}_T}(q_P^{\text{new}})$, then

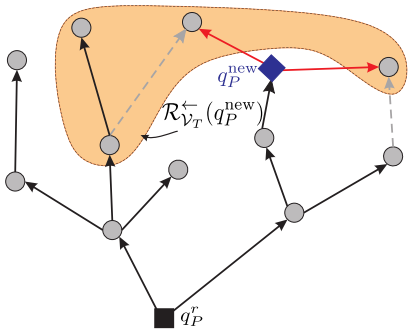


Fig. 2. Graphical depiction of Algorithm 5. The black square stands for the root of the tree and the gray disks and the blue diamond represent nodes in the set $\mathcal{V}_T$. Black arrows represent transitions captured by $\mathcal{E}_T$. The blue diamond stands for the state q_P^{new}. Dashed gray arrows stand for the edges that will be deleted from the set $\mathcal{E}_T$ during the execution of Algorithm 5 (line 4, Alg. 5). Red arrows stand for the new edges that will be added to $\mathcal{E}_T$ during the execution of Algorithm 5 (line 5, Alg. 5).

the new parent of q_P becomes q_P^{new}, i.e, a directed edge is drawn from q_P^{new} to q_P, and the edge that was connecting q_P to its previous parent is deleted [lines 4-5, Alg. 5]. The cost of node q_P is updated as $\texttt{Cost}(q_P) = \texttt{Cost}(q_P^{\text{new}}) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P^{\text{new}}, \Pi|_{\text{PTS}} q_P)$ to take into account the new path through which it gets connected to the root [line 6, Alg. 5]. Once a state q_P gets rewired, the cost of all its *successor* nodes in $\mathcal{G}_T$, collected in the set

$$\mathcal{S}(q_P) = \{q'_P \in \mathcal{V}_T | q'_P \text{ is connected to } q_P \text{ through a multi hop path in } \mathcal{G}_T\}, \quad (10)$$

is updated to account for the change in the cost of q_P [line 6, Alg. 5].

4) Construction of Paths: The construction of the tree $\mathcal{G}_T$ ends after $n_{\max}^{\text{pre}}$ iterations, where $n_{\max}^{\text{pre}}$ is user specified [line 4, Alg. 2]. Then, we construct the set $\mathcal{P} = \mathcal{V}_T \cap \mathcal{X}_{\text{goal}}^{\text{pre}}$ [line 13, Alg. 2] that collects all the states $q_P \in \mathcal{V}_T$ that belong to the goal region $\mathcal{X}_{\text{goal}}^{\text{pre}}$. Given the tree $\mathcal{G}_T$ and the set $\mathcal{P}$ [line 6, Alg. 1] that collects all states $q_P \in \mathcal{X}_{\text{goal}}^{\text{pre}} \cap \mathcal{V}_T$, we can compute the prefix plans [lines 7-8, Alg. 1]. In particular, the path that connects the a-th state in the set $\mathcal{P}$, denoted by $\mathcal{P}(a)$, to the root q_P^r constitutes the α-th prefix plan and is denoted by $\tau^{\text{pre},a}$ [line 8, Algorithm 1]. Its computation is described in Algorithm 6. Specifically, the prefix part $\tau^{\text{pre},a}$ is constructed by tracing the sequence of parents of nodes starting from the node that represents the accepting state $\mathcal{P}(a)$ and ending at the root of the tree [lines 1-7, Alg. 6]. The parent of each node is computed by the function $\texttt{parent} : \mathcal{V}_T \to \mathcal{V}_T$ that maps a node $q_P \in \mathcal{V}_T$ to a unique vertex $q'_P \in \mathcal{V}_T$ if $(q'_P, q_P) \in \mathcal{E}_T$, i.e., $\texttt{parent}(q_P) = q'_P$ if $(q'_P, q_P) \in \mathcal{E}_T$. By convention, we assume that $\texttt{parent}(q_P^r) = q_P^r$. In line 7, $\Pi|_{\text{PTS}} p_T$ stands for the projection of the path p_T onto the state-space of the PTS. In line 4 of Algorithm 6, $|$ stands for the concatenation of paths. Thus, for the resulting prefix plan $\tau^{\text{pre},a}$, it holds that $\tau^{\text{pre},a}(1) = \Pi|_{\text{PTS}} q_P^r$ and $\tau^{\text{pre},a}(|\tau^{\text{pre},a}|) = \Pi|_{\text{PTS}} \mathcal{P}(a)$.

B. Construction of Suffix Parts

Once the prefix plans $\tau^{\text{pre},a}$ for all $a \in \{1, \ldots, |\mathcal{P}|\}$ are constructed, the corresponding suffix plans $\tau^{\text{suf},a}$ are constructed

Algorithm 5: Function $\texttt{Rewire}(q_P^{\text{new}}, \mathcal{V}_T, \mathcal{E}_T, \texttt{Cost})$

1 Collect in set $\mathcal{R}^{\leftarrow}_{\mathcal{V}_T}(q_P^{\text{new}})$ all states of $q_P \in \mathcal{V}_T$ that abide by the following transition rule: $(q_P^{\text{new}}, q_P) \in \to_P$;
2 **for** $q_P \in \mathcal{R}^{\leftarrow}_{\mathcal{V}_T}(q_P^{new})$ **do**
3 **if** $\texttt{Cost}(q_P) > \texttt{Cost}(q_P^{new}) + w_{PTS}(\Pi|_{PTS} q_P^{new}, \Pi|_{PTS} q_P)$ **then**
4 $\mathcal{E}_T = \mathcal{E}_T \setminus \{(\texttt{Parent}(q_P), q_P)\}$;
5 $\mathcal{E}_T = \mathcal{E}_T \cup \{(q_P^{\text{new}}, q_P)\}$;
6 $\texttt{Cost}(q_P) = \texttt{Cost}(q_P^{\text{new}}) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P^{\text{new}}, \Pi|_{\text{PTS}} q_P)$;
7 Update the cost of all successor nodes of $q_P \in \mathcal{V}_T$;
8 **return** $\mathcal{E}_T$, $\texttt{Cost}$;

Algorithm 6: Function $\texttt{FindPath}(\mathcal{G}_T, q_P^{\text{initial}}, q_P^{\text{goal}})$

1 $p_T = \{q_P^{\text{goal}}\}$;
2 $q_P^{\text{prev}} = \texttt{Parent}(q_P^{\text{goal}})$;
3 **while** $q_P^{prev} \neq q_P^{initial}$ **do**
4 $p_T = p_T | \{q_P^{\text{prev}}\}$;
5 $q_P^{\text{prev}} = \texttt{Parent}(q_P^{\text{prev}})$;
6 $p_T = p_T | \{q_P^{\text{initial}}\}$;
7 $p_T = \Pi|_{\text{PTS}} p_T$;
8 **return** p_T;

[lines 9-20, Alg. 1]. Specifically, every suffix part $\tau^{\text{suf},a}$ is a sequence of states in $\mathcal{Q}_P$ that starts from the state $\mathcal{P}(a)$ and ends at the same state $\mathcal{P}(a)$, i.e., a cycle around state $\mathcal{P}(a)$ where any two consecutive states in $\tau_i^{\text{suf},a}$ respect the transition rule $\to_P$. To construct the suffix plan $\tau_i^{\text{suf},a}$ we build a tree $\mathcal{G}_T = \{\mathcal{V}_T, \mathcal{E}_T, \texttt{Cost}\}$ that approximates the PBA P, in a similar way as in Section IV-A, and implement a cycle-detection mechanism to identify cycles around the state $P(a)$. The only differences are that: (i) the root of the tree is now $q_P^r = \mathcal{P}(a)$, i.e., it is an *accepting/final* state [line 10, Alg. 1] detected during the construction of the prefix plans, (ii) the goal region corresponding to the root $q_P^r = \mathcal{P}(a)$, is defined as

$$\mathcal{X}_{\text{goal}}^{\text{suf}}(q_P^r) = \{q_P = (q_{\text{PTS}},\ q_B) \in \mathcal{Q}_P \mid (q_P, L(q_{\text{PTS}}), q_P^r) \in \to_P\}, \quad (11)$$

and, (iii) we first check if $q_P^r \in \mathcal{X}_{\text{goal}}^{\text{suf}}$, i.e., if $(\Pi|_B q_P^r, L(\Pi|_{\text{PTS}} q_P^r), \Pi|_B q_P^r)$ and if the cost of such a self loop has zero cost, i.e., if $w_P(q_P^r, q_P^r) = 0$ [line 12, Alg. 1]. If so, the construction of the tree is trivial, as it consists of only the root, and a loop around it with zero cost [line 13, Alg. 1].[3] If $q_P^r \notin \mathcal{X}_{\text{goal}}^{\text{suf}}$, then the tree $\mathcal{G}_T$ is constructed by Algorithm 2 [line 16, Alg. 1]. Once a tree rooted at $q_P^r = \mathcal{P}(a)$ is constructed, a set $\mathcal{S}_a \subseteq \mathcal{V}_T$ is formed that collects all states $q_P \in \mathcal{V}_T \cap \mathcal{X}_{\text{goal}}^{\text{suf}}(q_P^r)$ [lines 14, 16, Alg. 1]. Then for each state $q_P \in \mathcal{S}_a$, we compute the cost $\hat{J}(\tilde{\tau}^{\text{suf},e})$ of each

[3]Clearly, any other suffix part will have non-zero cost and, therefore, it will not be optimal and it will be discarded by Algorithm 1 [lines 19-20, Alg. 1]. For this reason, the construction of the tree $\mathcal{G}_T$ is terminated if a self-loop around q_P^r is detected.

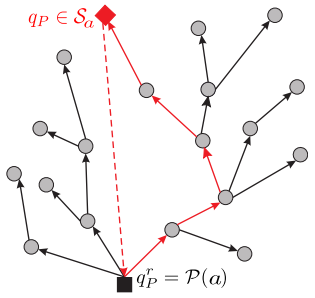


Fig. 3. Graphical depiction of detecting cycles around a final/accepting state $\mathcal{P}(a)$ (black square) which acts as the root of the tree. The red diamond stands for a state $q_P \in \mathcal{S}_a$. Solid red arrows stand for the path that connects the state $q_P \in \mathcal{S}_a$ to the root $\mathcal{P}(a)$. The dashed red arrow implies that a transition from q_P to $\mathcal{P}(a)$ is feasible according to the transition rule $\longrightarrow_P$; however, such a transition is not included in the set $\mathcal{E}_T$. The cycle around the accepting state $\mathcal{P}(a)$ is illustrated by solid and dashed red arrows.

possible suffix plan $\tilde{\tau}^{\text{suf},e}$, for all $e \in \{1,\dots,|\mathcal{S}_a|\}$, associated with the root q_P^r. By construction of the cost functions $\texttt{Cost}$ and $\hat{J}(\cdot)$, it holds that $\hat{J}(\tilde{\tau}^{\text{suf},e}) = \texttt{Cost}(\mathcal{S}_a(e)) + w_{\text{PTS}}(\Pi|_{\text{PTS}}\mathcal{S}_a(e), \Pi|_{\text{PTS}} q_P^r)$, where $\mathcal{S}_a(e)$ stands for the e-th state in the set $\mathcal{S}_a$. Among all detected suffix plans $\tilde{\tau}^{\text{suf},e}$ associated with the *accepting* state $\mathcal{P}(a)$, we select the suffix plan with the minimum cost, which constitutes the suffix plan $\tau^{\text{suf},a}$ [lines 19-20, Alg. 1]. This process is repeated for all $a \in \{1,\dots,|\mathcal{P}|\}$ [line 9, Alg. 1]. In this way, for each prefix plan $\tau^{\text{pre},a}$ we construct its corresponding suffix plan $\tau^{\text{suf},a}$, if it exists.

C. Construction of Optimal Discrete Plans

By construction, any motion plan $\tau^a = \tau^{\text{pre},a}[\tau^{\text{suf},a}]^\omega$, with $\mathcal{S}_a \neq \emptyset$, and $a \in \{1,\dots,|\mathcal{P}|\}$ satisfies the global LTL specification ϕ. The cost $J(\tau^a)$ of each plan τ^a is defined in (2). Given an initial state $q_B^0 \in \mathcal{Q}_B^0$, among all the motion plans $\tau^a \models \phi$, we select the one with the smallest cost $J(\tau^a)$ [line 21, Alg. 1]. The plan with the smallest cost given an initial state q_B^0 is denoted by $\tau_{q_B^0}$. Then, among all plans $\tau_{q_B^0}$, we select again the one with smallest cost $J(\tau_{q_B^0})$, i.e., $\tau = \tau^{a_*}$, where $a_* = \text{argmin}_{a_{q_B^0}} J(\tau_{q_B^0})$ [lines 22-23, Alg. 1].

Remark 4.4 (Execution of plan τ): The motion plan τ generated by Algorithm 1 satisfies the global LTL formula, if all robots pick their next states either synchronously or asynchronously as in [16]. For asynchronous execution of the plan τ, we only need to add 'traveling states' to the transition systems that capture cases where robots i are traveling from states $q_i^{\ell_j} \in \mathcal{Q}_i$ to $q_i^{\ell_e} \in \mathcal{Q}_i$ that satisfy $(q_i^{\ell_j}, q_i^{\ell_e}) \in \rightarrow_i$. More details about the traveling states can be found in [16]. Note that adding traveling states to the trees increases the computational cost of synthesizing plans that can be executed asynchronously and satisfy the assigned LTL formula.

D. Complexity Analysis

The memory resources needed to store the PBA as a graph structure $\mathcal{G}_P = \{\mathcal{V}_P, \mathcal{E}_P, w_P\}$, defined in Section III-A, using its adjacency list is $O(|\mathcal{V}_P| + |\mathcal{E}_P|)$ [32]. On the other hand, the memory needed to store a tree, constructed by Algorithm 2, that approximates the PBA is $O(|\mathcal{V}_T|)$, since $|\mathcal{E}_T| = |\mathcal{V}_T| - 1$. Due to the incremental construction of the tree we get that $|\mathcal{V}_T| \leq |\mathcal{V}_P| < |\mathcal{V}_P| + |\mathcal{E}_P|$ which shows that our proposed algorithm requires fewer memory resources compared to existing optimal control synthesis algorithms that rely on the construction of the PBA [16], [17].

Next, observe that the time complexity of sampling the state $q_{\text{PTS}}^{\text{new}}$ in Algorithm 3 is $O(\sum_i |\mathcal{Q}_i|)$; see also Remark 4.3. Moreover, the time complexity of extending the graph towards q_P^{new} is $O(|\mathcal{V}_T|(N+1))$; see Algorithm 4. The reason is that Algorithm 4 can be equivalently written as a for-loop over the set $\mathcal{V}_T$ where we first examine if $q_P \in \mathcal{V}_T$ can reach q_P^{new}, based on the transition rule $\rightarrow_P$, and then we compute the cost of such a transition while keeping track of the node $q_P \in \mathcal{V}_T$ that incurs the minimum cost. These calculations have cost $O(N+1)$ and the time complexity of the for-loop over $\mathcal{V}_T$ is $O(|\mathcal{V}_T|)$. With this implementation of Algorithm 4, we do not need to construct the set $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T}(q_P^{\text{new}})$.[4] For the same reason, the time complexity of the rewiring step is $O(|\mathcal{V}_T|(N+1))$; see Algorithm 5. Finally the time complexity of Algorithm 6 that finds a path in the tree $\mathcal{G}_T$ is $O(|\mathcal{V}_T|)$. On the other hand, using the Dijkstra algorithm to find the shortest path from an initial to a final state or a cycle around a final state of a PBA is $O(|\mathcal{E}_P| + |\mathcal{V}_P|\log(|\mathcal{V}_p|))$; clearly, it holds that $|\mathcal{E}_P| + |\mathcal{V}_P|\log(|\mathcal{V}_P|) > |\mathcal{V}_T|$. If the PBA is represented as an implicit graph using its transition rule $\rightarrow_P$, then we can apply the uniform cost search algorithm [33], [34] to find the optimal prefix and suffix paths with time and space complexity $O(b^{1+C^*/\epsilon})$, where b is the branching factor of $\mathcal{G}_P$, C^* is the optimal cost of either the prefix or suffix path, and $\epsilon > 0$ is the minimum increase in the cost of the path as we move from one node of $\mathcal{G}_P$ to another. Note that this approach is also memory efficient since it does not require the explicit construction of the PBA but it can become computationally intractable (i) for dense graphs, i.e., as b increases or (ii) for long paths, i.e., as C^*/ϵ increases. In comparison, the computational cost per iteration of our algorithm depends linearly only on $|\mathcal{V}_T|$ and the size of the network and not on the structure of $\mathcal{G}_P$.

V. Correctness and Optimality

In this section, we first characterize the rate at which the constructed trees grow and then we provide the main results pertaining to the probabilistic completeness and optimality of the proposed Algorithm 1. In what follows, we denote by $\mathcal{G}_T^n = \{\mathcal{V}_T^n, \mathcal{E}_T^n, \texttt{Cost}\}$ the tree that has been built by Algorithm 2 at the n-th iteration for the construction of either a prefix or suffix part. Also, we denote the nodes q_P^{rand} and q_P^{new} at iteration n by $q_P^{\text{rand},n}$ and $q_P^{\text{new},n}$, respectively. Moreover, in the following results, we denote the reachable set of $q_P \in \mathcal{Q}_P$ in the state-space of the PBA by:

$$\mathcal{R}_P(q_P) = \{q'_P \in \mathcal{Q}_P \mid q_P \rightarrow_P q'_P\}, \tag{12}$$

that collects all states $q'_P \in \mathcal{Q}_P$ that can be reached from $q_P \in \mathcal{Q}_P$ in one hop.

Proposition 5.1 (Growth Rate of $\mathcal{G}_T^n$): Assume that the reachable set of $q_P^{\text{rand},n} = (q_{\text{PTS}}^{\text{rand},n}, q_B^{\text{rand},n})$ in the PBA is

[4] Definition of $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T}(q_P^{\text{new}})$ is only used in Section V to simplify the proof of completeness and optimality of the proposed algorithm.

non-empty, i.e., that $\mathcal{R}_P(q_P^{\text{rand},n}) = \{q'_P \in \mathcal{Q}_P \mid q_P^{\text{rand},n} \rightarrow q'_P\} \neq \emptyset$. Then, there is at least one $b \in \{1, \ldots, |\mathcal{Q}_B|\}$ so that either the state $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new}}, \mathcal{Q}_B(b))$ will be added to $\mathcal{V}_T^n$ at iteration n if $q_P^{\text{new},n} \notin \mathcal{V}_T^n$, or rewiring to $q_P^{\text{new},n}$ will occur if $q_P^{\text{new},n} \in \mathcal{V}_T^n$ and $\mathcal{R}^{\leftarrow}_{\mathcal{V}_T^n}(q_P^{\text{new},n}) \neq \emptyset$. If $\mathcal{R}_P(q_P^{\text{rand},n}) = \emptyset$, then the tree may remain unaltered at iteration n.

Proof: To show this result, recall that a state $q'_P = (q'_{\text{PTS}}, q'_B)$ belongs to $\mathcal{R}_P(q_P^{\text{rand},n})$ if $q_P^{\text{rand},n} \rightarrow_P q'_P$, i.e., if (i) $q_{\text{PTS}}^{\text{rand},n} \rightarrow_{\text{PTS}} q'_{\text{PTS}}$ and (ii) $q_B^{\text{rand},n} \xrightarrow{L(q_{\text{PTS}}^{\text{rand},n})}_B q'_B$. In what follows, we first examine the case $\mathcal{R}_P(q_P^{\text{rand},n}) = \emptyset$ and then the case $\mathcal{R}_P(q_P^{\text{rand},n}) \neq \emptyset$.

Assume that $\mathcal{R}_P(q_P^{\text{rand},n}) = \emptyset$. This means that for the state $q_P^{\text{rand},n} = (q_{\text{PTS}}^{\text{rand},n}, q_B^{\text{rand},n})$, there are either no states q'_{PTS} that satisfy condition (i), or no states q'_B that satisfy condition (ii), or possibly both. If there are no states q'_{PTS} that satisfy condition (i), then $q_{\text{PTS}}^{\text{rand},n}$ is a terminal state of the PTS, i.e., $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n}) = \emptyset$, where $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$ is defined in (6). As a result, this implies that we cannot create any state $q_{\text{PTS}}^{\text{new},n} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$ and, therefore, it is trivial to see that no states will be added to $\mathcal{V}_T^n$ at iteration n and the tree will not change at iteration n. On the other hand, if there are no states q'_B that satisfy condition (ii), i.e., if there is no b such that $q_B^{\text{rand},n} \xrightarrow{L(q_{\text{PTS}}^{\text{rand},n})}_B \mathcal{Q}_B(b)$, then $q_{\text{PTS}}^{\text{rand},n}$ is a state at which the LTL formula ϕ is violated, by construction of the NBA. However, this does not necessarily mean that the tree will remain the same at iteration n. The reason is that if $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n}) \neq \emptyset$, then there exist states $q_{\text{PTS}}^{\text{new},n} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$ and, consequently, states $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b))$ can be constructed. Such states can possibly be added to the tree if there exists a state $q_P \in \mathcal{V}_T^n$ such that $q_P \in \mathcal{R}^{\rightarrow}_{\mathcal{V}_T^n}(q_P^{\text{new},n})$; see line 2 in Algorithm 4. Also, if $q_P^{\text{new},n} \in \mathcal{V}_T^n$ and if $\mathcal{R}^{\leftarrow}_{\mathcal{V}_T^n}(q_P^{\text{new},n}) \neq \emptyset$ then rewiring to these states may occur; see line 2 in Algorithm 5. Clearly if $q_{\text{PTS}}^{\text{rand},n}$ is both a terminal state of PTS and a state at which ϕ is violated, then the tree will remain unchanged at iteration n.[5]

Next, assume that $\mathcal{R}_P(q_P^{\text{rand},n}) \neq \emptyset$. Following the same logic as in the previous case, this means that $q_{\text{PTS}}^{\text{rand},n}$ is not a terminal state of the PTS, i.e., $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n}) \neq \emptyset$, and $q_{\text{PTS}}^{\text{rand},n}$ is not a state at which the LTL formula ϕ is violated. Since $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n}) \neq \emptyset$ and since $q_{\text{PTS}}^{\text{new},n} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$ by construction, we get that $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b))$ satisfies condition (i) for all b. Next, since $q_{\text{PTS}}^{\text{rand},n}$ is not a state at which the LTL formula ϕ is violated, this means that there is at least one value for b, denoted hereafter by $\bar{b}$, such that $q_B^{\text{rand},n} \xrightarrow{L(q_{\text{PTS}}^{\text{rand},n})}_B \mathcal{Q}_B(\bar{b})$, by construction of the NBA. Thus, we get that $(q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(\bar{b})) \in \mathcal{R}_P(q_P^{\text{rand},n})$. This result along with the fact that $q_P^{\text{rand},n} \in \mathcal{V}_T^n$, by definition of f_{rand}, entail that $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(\bar{b}))$ satisfies $q_P^{\text{rand},n} \in \mathcal{R}^{\rightarrow}_{\mathcal{V}_T^n}(q_P^{\text{new},n})$, i.e., $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T^n}(q_P^{\text{new},n}) \neq \emptyset$. This equivalently means that if $q_P^{\text{new},n} \notin \mathcal{V}_T^n$ then the state $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(\bar{b}))$ will be added to $\mathcal{V}_T^n$ at iteration n; see line 2 in Algorithm 4. Otherwise, if $q_P^{\text{new},n} \in \mathcal{V}_T^n$ and $\mathcal{R}^{\leftarrow}_{\mathcal{V}_T^n}(q_P^{\text{new},n}) \neq \emptyset$ rewiring to $q_P^{\text{new},n}$ will follow (see line 2 in Algorithm 5), completing the proof. ■

[5]Observe that if $\mathcal{R}_P(q_P^{\text{rand},n}) = \emptyset$, then the state $q_P^{\text{rand},n} \in \mathcal{V}_T^n$ will remain forever a leaf node in the tree $\mathcal{G}_T^n$.

Remark 5.2 (Emptiness of $\mathcal{R}_P(q_P^{rand,n})$): Observe that $\mathcal{R}_P(q_P^{\text{rand},n}) = \emptyset$ for a state $q_P^{\text{rand},n} = (q_{\text{PTS}}^{\text{rand},n}, q_B^{\text{rand},n})$, if $q_{\text{PTS}}^{\text{rand},n}$ is either a terminal state of the PTS, i.e., $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n}) = \emptyset$, where $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$ is defined in (6), or a state at which the LTL formula ϕ is violated, or both. For example, if the PTS has no terminal states and the LTL formula does not include the negation operator $\neg$, i.e., there are no states in $\mathcal{Q}_{\text{PTS}}$ that can violate ϕ, then $\mathcal{R}_P(q_P) \neq \emptyset$, $\forall q_P \in \mathcal{Q}_P$.

To show that Algorithm 1 is probabilistically complete and asymptotically optimal we need first to show the following results that rely on the second Borel-Cantelli lemma [35] presented below. The proofs of the following lemmas are provided in Appendix A.

Lemma 5.3 (Borel-Cantelli [35]): Consider a sequence of independent events $A = \{A^n\}_{n=1}^{\infty}$. If $\sum_{n=1}^{\infty} \mathbb{P}(A^n) = \infty$ then $\mathbb{P}(\limsup_{n\rightarrow\infty} A^n) = 1$, i.e., the probability that infinitely many events A^n occur is 1.

Lemma 5.4 (Sampling $q_P^{rand,n}$): Consider any state $q_P \in \mathcal{V}_T^n$ and any fixed iteration index n. Then, there exists an infinite number of subsequent iterations $n+k$, where $k \in \mathcal{K}$ and $\mathcal{K} \subseteq \mathbb{N}$ is a subsequence of $\mathbb{N}$, at which the state $q_P \in \mathcal{V}_T^n$ is selected by Algorithm 3 to be the node $q_P^{\text{rand},n+k}$.

Using Lemma 5.4 we can show the following result for the node $q_{\text{PTS}}^{\text{new},n}$.

Lemma 5.5 (Sampling $q_{PTS}^{new,n}$): Consider any state $q_P^{\text{rand},n} = (q_{\text{PTS}}^{\text{rand},n}, q_B^{\text{rand},n}) \in \mathcal{V}_T^n$ selected by Algorithm 3 and any fixed iteration index n. Then, for any state $q_{\text{PTS}} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$, where $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$ is defined in (6), there exists an infinite number of subsequent iterations $n+k$, where $k \in \mathcal{K}'$ and $\mathcal{K}' \subseteq \mathcal{K}$ is a subsequence of the sequence of $\mathcal{K}$ defined in Lemma 5.4, at which the state $q_{\text{PTS}} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$ is selected by Algorithm 3 to be the node $q_{\text{PTS}}^{\text{new},n+k}$.

By Lemma 5.5, we have the following corollary for the state $q_P^{\text{new},n}$.

Corollary 5.6 (Sampling $q_P^{new,n}$): Consider any state $q_P^{\text{rand},n} = (q_{\text{PTS}}^{\text{rand},n}, q_B^{\text{rand},n}) \in \mathcal{V}_T^n$ selected by Algorithm 3 and any fixed iteration index n. Then, for any state $q_P \in \mathcal{R}_P(q_P^{\text{rand},n})$, where $\mathcal{R}_P(q_P^{\text{rand},n})$ is defined in (12), there exists an infinite number of iterations $n+k$, where $k \in \mathcal{K}'$ and $\mathcal{K}'$ is the subsequence defined in Lemma 5.5, at which the state $q_P \in \mathcal{R}_P(q_P^{\text{rand},n})$, is selected by Algorithm 3 to be the node $q_P^{\text{new},n+k}$.

Using Corollary 5.6, we can show the following result for the reachable set $\mathcal{R}_P(q_P^{\text{rand},n})$.

Lemma 5.7 (Reachable set $\mathcal{R}_P(q_P^{rand,n})$): Consider any state $q_P^{\text{rand},n} = (q_{\text{PTS}}^{\text{rand},n}, q_B^{\text{rand},n}) \in \mathcal{V}_T^n$ selected by Algorithm 3 and any fixed iteration index n. Then, Algorithm 2 will add to $\mathcal{V}_T^{n+k}$ all states that belong to the reachable set $\mathcal{R}_P(q_P^{\text{rand},n})$, where $\mathcal{R}_P(q_P^{\text{rand},n})$ is defined in (12), as $k \rightarrow \infty$, with probability 1, i.e.,

$$\lim_{k\rightarrow\infty} \mathbb{P}\left(\{\mathcal{R}_P(q_P^{\text{rand},n}) \subseteq \mathcal{V}_T^{n+k}\}\right) = 1, \tag{13}$$

Using Lemma 5.4, we can show that Lemma 5.7 holds for all nodes $q_P \in \mathcal{V}_T^n$. This result is stated in the following corollary.

Corollary 5.8 (Reachable set $\mathcal{R}_P(q_P)$): Given any state $q_P = (q_{\text{PTS}}, q_B) \in \mathcal{V}_T^n$, Algorithm 2 will add to $\mathcal{V}_T^n$ all states

that belong to the reachable set $\mathcal{R}_P(q_P)$, as $n \to \infty$, with probability 1, i.e.,

$$\lim_{n\to\infty} \mathbb{P}\left(\{\mathcal{R}_P(q_P) \subseteq \mathcal{V}_T^n\}\right) = 1, \tag{14}$$

Using Corollary 5.8, in the next theorem, we show that Algorithm 1 is probabilistically complete.

Theorem 5.9 (Probabilistic Completeness): If there exists a solution to Problem 1, then Algorithm 1 is probabilistically complete, i.e., it will find with probability 1 a motion plan τ that satisfies the LTL specification ϕ, as $n_{\text{max}}^{\text{pre}} \to \infty$ and $n_{\text{max}}^{\text{suf}} \to \infty$.

Proof: to show this result, we need to show that Algorithm 2 satisfies

$$\lim_{n\to\infty} \mathbb{P}\left(\{\mathcal{V}_T^n \cap \mathcal{X}_{\text{goal}} \neq \emptyset\}\right) = 1, \tag{15}$$

for both goal regions $\mathcal{X}_{\text{goal}}$ defined in (4) and in (11).

To show (15), it suffices to show that the set of nodes $\mathcal{V}_T^n$ will eventually contain all states in the state-space $\mathcal{Q}_P$ that are reachable from the root q_P^r through a multi-hop path that respects the transition rule $\to_P$.[6] We collect these states in the set $\mathcal{R}_P^\infty(q_P^r)$, defined as

$$\mathcal{R}_P^\infty(q_P^r) = \cup_{m=1}^\infty \mathcal{R}_P^m(q_P^r), \tag{16}$$

where the reachable set $\mathcal{R}_P^m(q_P^r)$ collects all states $q_P \in \mathcal{Q}_P$ that are reachable from the root q_P^r in m-hops.[7]

In mathematical terms, we need to show that

$$\lim_{n\to\infty} \mathbb{P}\left(\{\mathcal{R}_P^\infty(q_P^r) = \mathcal{V}_T^n\}\right) = 1. \tag{17}$$

Since we assume that there exists a solution to Problem 1, if (17) holds, then $\mathcal{R}_P^\infty(q_P^r) \cap \mathcal{X}_{\text{goal}} \neq \emptyset$, which equivalently means that (15) holds, as well.

To show that (17) holds it suffices to show that the event $\{\mathcal{R}_P(q_P) \subseteq \mathcal{V}_T^n,\ \forall q_P \in \mathcal{V}_T^n\}$, is equivalent to the event $\{\mathcal{R}_P^\infty(q_P^r) = \mathcal{V}_T^n\}$. Then the result follows due to (14) in Corollary 5.8. To show this, observe that if $q \in \mathcal{R}_P(q_P)$, with $q_P \in \mathcal{V}_T^n$ then, clearly q is reachable from the root q_P^r through a multi-hop path, since by construction of the tree there is a multi-hop path that connects q_P to the root q_P^r, i.e., $q \in \mathcal{R}_P^\infty(q_P^r)$. Next, if $q \in \mathcal{R}_P^\infty(q_P^r) = \mathcal{V}_T^n$ then there exists a node $q_P \in \mathcal{R}_P^\infty(q_P^r) = \mathcal{V}_T^n$, so that q is reachable from q_P due to (16), i.e., $q \in \mathcal{R}_P(q_P)$, completing the proof. ∎

To show that Algorithm 1 is asymptotically optimal, we need the following corollary that is proved in Appendix A.

Corollary 5.10 (Sampling $q_P^{new,n}$): Consider any state $q_P \in \mathcal{V}_T^n$ and any fixed iteration n. Then, there exists and infinite number of subsequent iterations $n+k$, where $k \in \mathcal{K}'$ is the subsequence defined in Lemma 5.5, at which the state $q_P \in \mathcal{V}_T^n$ is selected by Algorithm 3 to be the node $q_P^{\text{new},n+k}$.

Theorem 5.11 (Asymptotic Optimality): Assume that there exists an optimal solution to Problem 1. Then, Algorithm 1 is asymptotically optimal, i.e., the optimal motion plan for a given LTL formula ϕ will be found with probability 1, as $n_{\text{max}}^{\text{pre}} \to \infty$ and $n_{\text{max}}^{\text{suf}} \to \infty$. In other words, the discrete motion plan τ that is generated by this algorithm for a given global LTL specification ϕ satisfies

$$\mathbb{P}\left(\left\{\lim_{n_{\text{max}}^{\text{pre}}\to\infty, n_{\text{max}}^{\text{suf}}\to\infty} J(\tau) = J^*\right\}\right) = 1, \tag{18}$$

where J is the cost function (2), J^* is the optimal cost, and $n_{\text{max}}^{\text{pre}}$ and $n_{\text{max}}^{\text{suf}}$ are the maximum number of iterations of Algorithm 2 for the prefix and suffix part synthesis, respectively.

Proof: To show that Algorithm 2 is asymptotically optimal, we will show that as $n_{\text{max}}^{\text{pre}} \to \infty$ and $n_{\text{max}}^{\text{suf}} \to \infty$, all states $q_P \in \mathcal{V}_T^n$ are connected to the root q_P^r through the path in the PBA that has the minimum cost. A necessary and sufficient condition for this is to show that as $n_{\text{max}}^{\text{pre}} \to \infty$ and $n_{\text{max}}^{\text{suf}} \to \infty$, the set of edges $\mathcal{E}_T^n$ of the tree $\mathcal{G}_T^n$ constructed by Algorithm 2 contains the transitions between states in the plans $\tau^{\text{pre},*}$ and $\tau^{\text{suf},*}$, where $\tau^* = \tau^{\text{pre},*}[\tau^{\text{suf},*}]^\omega$ is the optimal motion plan. Specifically, to prove that, we first show that every node $q_P \in \mathcal{V}_T^n$ will get rewired only a *finite* number of times as $n \to \infty$, which means that the cost of each node will converge to a finite number, as $n \to \infty$ (necessary condition that guarantees convergence). Then we show that this means that the cost of every node q_P has converged to its optimal cost, which equivalently means that the set of edges $\mathcal{E}_T^n$ of the tree $\mathcal{G}_T^n$ constructed by Algorithm 2 contains the transitions between states in the optimal prefix $\tau^{\text{pre},*}$ and suffix $\tau^{\text{suf},*}$ part (sufficient condition).

To show that every node $q_P \in \mathcal{V}_T^n$ will get rewired only a finite number of times as $n \to \infty$ we use contradiction. Assume that as $n \to \infty$, there exists a node q_P for which rewiring will take place infinitely often. Equivalently, this means that the path that connects q_P to the root q_P^r of the tree will change infinitely often. This can happen if: (i) The sets $\mathcal{V}_T^n$ and $\mathcal{E}_T^n$ are infinite as $n \to \infty$. This is not possible by construction of the PBA, since $|\mathcal{V}_T^n| \leq |\mathcal{Q}_P| < \infty$ for all $n \in \mathbb{N}_+$; (ii) The sets $\mathcal{V}_T^n$ and $\mathcal{E}_T^n$ are finite but the cost of each node $q_P \in \mathcal{V}_T^n$ is not bounded below. This is not possible, by definition of the PBA, since the cost of all states $q_P \in \mathcal{Q}_P$ is bounded below by 0; and (iii) The sets $\mathcal{V}_T^n$ and $\mathcal{E}_T^n$ are finite but the path that connects q_P to q_P^r in the tree $\mathcal{G}_T^n$, denoted by $\pi^n(q_P)$, reoccurs periodically. This means that there exist constants $\bar{n}$, $K > 0$ so that $\pi^{m\bar{n}}(q_P) = \pi^{m\bar{n}+K}(q_P)$ for all $n > m\bar{n}$ and $m \in \mathbb{N}_+$

In what follows, we show by contradiction that case (iii) is not possible either. With slight abuse of notation, we denote by $\texttt{Cost}^n(q_P)$ the cost of q_P at iteration n. Since, by assumption, q_P gets rewired indefinitely, we have that

$$\texttt{Cost}^{m\bar{n}}(q_P) > \texttt{Cost}^{m\bar{n}+1}(q_P) > \cdots > \texttt{Cost}^{m\bar{n}+K}(q_P),$$

for any $\bar{n}, K > 0$ and for all $m \in \mathbb{N}_+$. Clearly, this contradicts the fact that $\pi^{m\bar{n}}(q_P) = \pi^{m\bar{n}+K}(q_P)$ which implies that $\texttt{Cost}^{m\bar{n}}(q_P) = \texttt{Cost}^{m\bar{n}+K}(q_P)$. Therefore, as $n \to \infty$, every node $q_P \in \mathcal{V}_T^n$ will get rewired only a finite number of times.

Next, we show by contradiction that when rewiring has ended for all nodes in $\mathcal{V}_T^n$, every node in the constructed tree has achieved its optimal cost. Specifically, assume that

[6]Recall that the root for the construction of the prefix parts is $q_P^r = (q_{\text{PTS}}^0, q_B^0)$, where q_B^0 is each possible state in $\mathcal{Q}_B^0$, and for the construction of the suffix parts the root q_P^r is each possible final state detected during the construction of the prefix parts.

[7]The superscript ∞ in (16) means that the reachable set collects all states that are reachable from q_P^r in the state-space $\mathcal{Q}_P$ in any number of hops.

rewiring has ended for all nodes and that there exists at least one node $q_P \in \mathcal{V}_T^n$ that has not reached its optimal cost. This means that there exists at least one pair of nodes $q_P \in \mathcal{V}_T^n$ and $q'_P \in \mathcal{V}_T^n$ such that (i) $q_P \rightarrow_P q'_P$ and (ii) if q_P gets rewired to q'_P the cost of q_P will decrease. However, by Corollary 5.10, q'_P will be selected by Algorithm 3 to be $q_P^{\text{new},n+k}$ infinitely often, meaning that q'_P will eventually get rewired to q_P by Algorithm 2. This contradicts the fact that rewiring has ended for all nodes in $\mathcal{V}_T^n$. Therefore, when all nodes have been rewired finitely many times and the rewiring process has terminated, every node q_P has achieved its optimal cost.

Finally note that by Theorem 5.9, as $n \rightarrow \infty$ we have that $\mathcal{V}_T^n = \mathcal{R}_P^\infty(q_P^r)$ with probability 1. Since $\mathcal{R}_P^\infty$ is fixed (because $\mathcal{Q}_P$ is finite), so is $\mathcal{V}_T^n$ as $n \rightarrow \infty$. Moreover, since Problem 1 has a solution, $\mathcal{V}_T^n$ also includes the final state q_P^F that appears in $\tau^{\text{pre},*}$ as $n \rightarrow \infty$. Therefore, by the above argument, as $n \rightarrow \infty$, every state in $\mathcal{V}_T^n$, including the final state q_P^F, will reach its optimal cost with probability 1. This means that the cost of the path that corresponds to the prefix part constructed by Algorithm 2 that connects the final state q_P^F to the root q_P^r will be $\hat{J}(\tau^{\text{pre},*})$. Following the same logic, the cost of the respective suffix part, i.e., the cycle around the final state q_P^F will be $\hat{J}(\tau^{\text{suf},*})$ completing the proof. ■

VI. Numerical Experiments

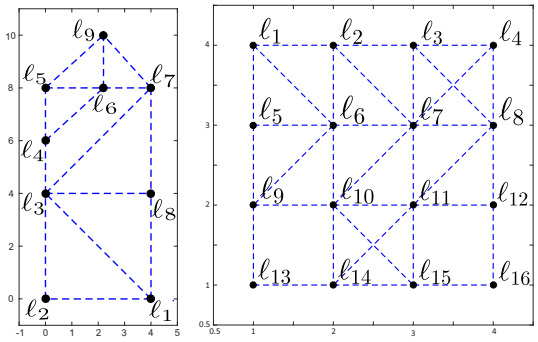
In this section, we present two case studies, implemented using MATLAB R2016a on a computer with Intel Xeon CPU at 2.93 GHz and 4 GB RAM, that illustrate our proposed algorithm and compare it to existing methods. The first case study pertains to a motion planning problem with a PBA that has $3,099,363,912$ states. Recall that the state-space of the PBA defined in Definition 3.3 has $\Pi_{i=1}^N |\mathcal{Q}_i||\mathcal{Q}_B|$ states. This problem cannot be solved by standard optimal control synthesis algorithms, discussed in Section I, that rely on the explicit construction of the PBA defined in Section III, due to memory requirements. Representing the PBA as an *implicit* graph and using the uniform-cost search (UCS) algorithm [33], [34] to find the optimal plan also failed to detect a final state within 24 hours. In fact, our implementation for both approaches of the algorithm presented in Section III-A cannot provide a plan for PBA with more than few millions of states and transitions either due to memory requirement or excessively high runtime. This problem cannot be solved by the off-the-shelf model checker PRISM either, due to excessive memory requirements. Our implementation of [23] failed also to provide a motion plan for the considered case study due to the large state-space of the resulting PBA. On the other hand, NuSMV can generate a feasible, but not the optimal, plan that satisfies the considered LTL-based task. A direct comparison with [22] cannot be made, since in [22] samples of the robot positions are drawn from the continuous space, which is not the case here. Note, however, that as the size of the regions that observe the atomic propositions in [22] becomes smaller, more samples are needed to construct expressive enough transition systems that are needed to generate a motion plan. In this case, the state space of the PBA may become too large to store, let alone apply graph search methods. This issue becomes more pronounced, as the size of the NBA increases. Also, scalability in [22] relies on the construction of a sparse graph rather than a tree as in our proposed method. However, sparsity of the graph is lost as the number of samples increases. Moreover, we also compare the proposed control synthesis algorithm to our previous work [24] and we show a significant improvement in terms of scalability, due to the fast exploration of the state-space of the PBA as predicted by Proposition 5.1. In the second case study, we consider a motion planning problem with a PBA that has 6,144 states. This state-space is small enough to manipulate and construct an optimal plan using the standard method described in Section III-A. In this simulation study, we examine the performance of the proposed algorithm in terms of runtime and optimality. In what follows, we consider discrete uniform distributions for both f_{rand} and f_{new} for all iterations n and also we assume that the weights w_i defined in Definition 3.1 represent distance between locations.

A. Case Study I

In the first simulation study, we consider a team of $N = 9$ robots residing in a workspace with $W = 9$ regions of interest. The transition system describing the motion of each robot has $|\mathcal{Q}_i| = 9$ states and 39 transitions, including self-loops around each state, as shown in Figure 4(a). The collaborative task that is assigned to the robots describes an intermittent connectivity problem, that was defined in our previous work [4]. Specifically, the robots move along the edges of a mobility graph and communicate only when they meet at the vertices of this graph. The communication network is intermittently connected if communication occurs at the vertices of the mobility graph infinitely often. This intermittent connectivity requirement can be captured by a global LTL formula, which for the case study at hand takes the form $\phi = [\square\lozenge(\pi_1^{\ell_5} \wedge \pi_2^{\ell_5})] \wedge [\square\lozenge(\pi_2^{\ell_1} \wedge \pi_3^{\ell_1} \wedge \pi_4^{\ell_1})] \wedge [\square\lozenge(\pi_4^{\ell_7} \wedge \pi_5^{\ell_7} \wedge \pi_6^{\ell_7})] \wedge [\square\lozenge(\pi_6^{\ell_8} \wedge \pi_7^{\ell_8})] \wedge [\square\lozenge(\pi_7^{\ell_4} \wedge \pi_8^{\ell_4})] \wedge [\square\lozenge(\pi_8^{\ell_3} \wedge \pi_9^{\ell_3})] \wedge [\neg(\pi_1^{\ell_5} \wedge \pi_2^{\ell_5}) \mathcal{U} \pi_1^{\ell_7}]$. In words, (a) robots 1 and 2 need to meet at location ℓ_5 infinitely often, (b) robots 2, 3 and 4 need to meet at location ℓ_1, infinitely often, (c) robots 4, 5, and 6 need to meet at location ℓ_7, infinitely often, (d) robots 6 and 7 need to meet at location ℓ_8 infinitely often, (e) robots 7 and 8 need to meet at location ℓ_4, infinitely often, (f) robots 8 and 9 need to meet at location ℓ_3, infinitely often, and (g) robots 1 and 2 should never meet at location ℓ_5 until robot 1 visits location ℓ_7 to collect some available information. This LTL formula corresponds to a NBA with $|\mathcal{Q}_B| = 8$ states, $|\mathcal{Q}_B^0| = 1$, $|\mathcal{Q}_B^F| = 1$, and 36 transitions.[8]

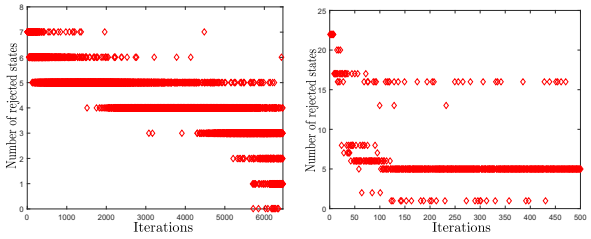
In Algorithm 1, we select $n_{\text{max}}^{\text{pre}} = n_{\text{max}}^{\text{suf}} = 6500$. The first final state was detected in 13 minutes. After 6500 iterations that took approximately 30 minutes, $|\mathcal{P}| = 11$ final states were detected and a tree $\mathcal{G}_T$ with $|\mathcal{V}_T| = 23893$ nodes was constructed. Figure 5(a) depicts the number of rejected states per iteration n of Algorithm 2, i.e., samples $q_P^{\text{new},n} \notin \mathcal{V}_T^n$ that cannot be added to $\mathcal{V}_T^n$ at each iteration n, during the construction of the prefix part. Observe in Figure 5(a), that at every iteration n, there is at least one sample q_P^{new} that either

[8]The translation of the LTL formula to a NBA was made by the tool developed in [36].



(a) wTS_i for Simulation Study I (b) wTS_i for Simulation Study II

Fig. 4. Graphical depiction of the transition systems wTS_i, for all robots i used in simulation study I (Figure 4(a)) and II (Figure 4(b)). Black disks represent the states of wTS_i and red edges stand for feasible transitions among the states.



(a) Simulation Study I (b) Simulation Study II

Fig. 5. Graphical depiction of the number of rejected states per iteration n after running Algorithm 2 for 6500 and 500 iterations for simulation studies I and II, for the synthesis of the prefix part. The resulting trees have 23893 and 3621 nodes, respectively. Red diamonds represent the number of rejected states at each iteration. At iteration n of Algorithm 2, a state sampled from $\mathcal{Q}_{\text{PTS}}$ is taken. Given this state, $|\mathcal{Q}_B|$ states that belong to $\mathcal{Q}_P$ are created. Consequently, at iteration n at most $|\mathcal{Q}_B|$ states can be rejected or accepted. Recall that $|\mathcal{Q}_B| = 8$ and $|\mathcal{Q}_B| = 24$, for simulation studies I and II, respectively.

is added to the tree if $q_P^{\text{new},n} \notin \mathcal{V}_T^n$ or enables the rewiring operation if $q_P^{\text{new},n} \in \mathcal{V}_T^n$. Given the detected final states, the construction of the suffix part follows, where the average time to compute each suffix part $\tau_{\text{pre},a}$, $a = \{1, \ldots, 11\}$ was 17 minutes. Given the prefix and suffix parts, the resulting optimal motion plan that satisfies the considered LTL task was synthesized in less than 1 second and its cost is $J(\tau) = \hat{J}(\tau^{\text{pre}}) + \hat{J}(\tau^{\text{suf}}) = 387.2293 + 387.2293 = 774.4586$ meters. Notice also that storage of each of the constructed trees required approximately only 3MBs while the computation of paths over the trees associated with either the prefix or the suffix part required 0.02 seconds on average rendering our approach resource and computationally efficient.

1) Comparison with [24]: Next, we compare the proposed sampling approach with our previous work [24] in terms of their ability to minimize cost and grow the tree as a function of runtime. Before presenting the comparative results, recall first that [24] consists of two parts. In the *first part*, samples are drawn *randomly* from $\mathcal{Q}_P$. If these samples do not already belong to the tree but they are reachable from the tree, then the tree is extended towards them and the rewiring step follows, as in our proposed algorithm. If they are not reachable from the tree, then they are rejected, as in our proposed algorithm too.

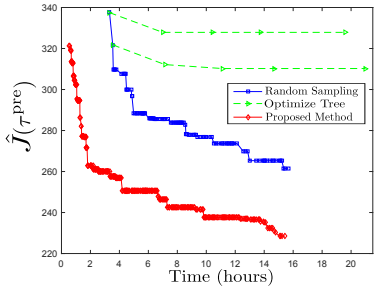
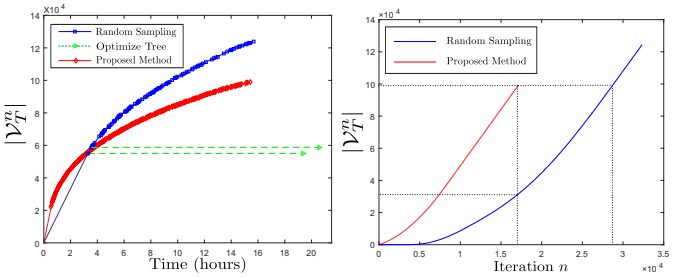


Fig. 6. Simulation Study I: Comparison of the average cost of the best prefix part constructed by Algorithm 1 (red line) and the first part of [24] (blue line) with respect to time. The average cost of the best prefix part is reported every time a new final state is detected. Red diamonds and blue squares denote a new final state detected by Algorithm 1 and [24], respectively. The green dashed lines represent the evolution of the average cost of the best prefix part, when the second part of [24] is executed that stops extending the tree and instead it optimizes its structure.



(a) (b)

Fig. 7. Simulation Study I: Figure 7(a) compares the average size of the tree built by Algorithm 1 (red line) and [24] (blue line) with respect to time, during the synthesis of the prefix part. The size of the tree is reported every time a new final state is detected. Red diamonds and blue squares denote a new final state detected by Algorithm 1 and [24], respectively. The green dashed lines represent the average time that it takes for the second part of [24] to optimize the tree structure when the first two final states are detected. Figure 7(b) illustrates the effect of sampling in the growth rate of the trees during the synthesis of the prefix part. The red and the blue lines show the evolution of the size of the tree with respect to iterations n when the proposed method and the first part of [24] are executed, respectively.

On the other hand, if these samples already belong to the tree, then they are rejected in [24] while in our proposed algorithm, rewiring to these samples follows. A more detailed description of this first part can be found in Algorithm 1 and 7 in [24]. The *second part* of [24], that does not exist in our sampling-based algorithm, pertains to the optimization of the tree structure. Specifically, once the first part of [24] has been executed for a user-specified number of iterations, then a rewiring-based algorithm follows that minimizes the cost of each node; see Algorithm 5 in [24]. This algorithm is terminated once the set of edges of the tree stops changing and it is necessary to obtain asymptotic optimality of the method in [24].

To compare Algorithm 1 and [24], we executed Algorithm 1 and the *first part* of [24] three times for a duration of about 15 hours per experiment. Notice that Algorithm 1 detected 338 final states in 15.42 hours in average while [24] found 108 final states in 15.65 hours in average. In every experiment, every time a new final state was detected, we computed the cost of the best prefix part, i.e., the minimum cost among all detected final states, and the size of the constructed tree. The

evolution of the average cost with time is illustrated in Figure 6 for both algorithms. Observe in Figure 6 that the proposed algorithm can find the first final state in 30 minutes in average, while [24] can do so in 3.5 hours, approximately. Also, observe in Figure 6 that the prefix part generated by [24] has always a higher cost than the one synthesized by Algorithm 1. The reason is that the first part of [24] only *partially* optimizes the tree, as rewiring occurs only for samples that do not already exist in the tree but they are reachable from it. As discussed before, optimization of the tree structure is accomplished by the *second part* of [24] which, however, requires additional computational time. Figure 6 also shows how the cost of the best prefix part changes with time during the execution of the second part of [24]. Observe that when [24] has optimized the tree that was built until the detection of the first final state (the last triangle in the top green dashed line in Figure 6), our proposed method has already detected 338 final states and has also constructed a much better prefix part (the last red rhombus in Figure 6), in terms of the cost function $\hat{J}(\tau^{\text{pre}})$ defined in (1).

Figure 7(a), shows how the average size of the tree changes with time during the execution of our proposed algorithm and the first part of [24]. Observe that [24] explores the state-space of the PBA faster than our proposed method, since at every iteration n the first part of [24] executes fewer operations due to the way the rewiring step is triggered, as discussed before. The time required to execute the second part of [24] that optimizes the tree structure is also depicted in Figure 7(a), that shows that [24] is much slower than our proposed method in synthesizing optimal plans.

Finally, in Figure 7(b), we present the average size of the constructed trees per iteration n during the execution of the proposed algorithm and the first part of [24]. Observe that at any given iteration n the proposed method has built a much larger tree than [24]. The reason is that in [24] samples are taken arbitrarily from the state space $\mathcal{Q}_{\text{PTS}}$ and, therefore, they are not necessarily reachable from the constructed tree and, as a result, they are rejected. On the other hand, here, samples are drawn from reachable sets accelerating the construction of the tree with respect to iterations n, as shown in Proposition 5.1.

2) Comparison with off-the-shelf model checkers: Notice that the off-the-shelf model checker PRISM could not verify the considered LTL specification due to memory requirements. Specifically, PRISM could verify only a smaller part of the considered LTL formula that involved 6 robots, which was $\bar{\phi} = [\square\Diamond(\pi_1^{\ell_5} \wedge \pi_2^{\ell_5})] \wedge [\square\Diamond(\pi_2^{\ell_1} \wedge \pi_3^{\ell_1} \wedge \pi_4^{\ell_1})] \wedge [\square\Diamond(\pi_4^{\ell_7} \wedge \pi_5^{\ell_7} \wedge \pi_6^{\ell_7})]$. In this case, the size of the state-space of the PBA was $|\mathcal{Q}_P| = 9,765,625$. PRISM finished the model-checking process in 1.5 minutes while our method found a plan within 17 minutes. We also applied NuSMV to this problem that was able to generate a feasible plan within few seconds with cost equal to $J(\tau) = J(\tau^{\text{pre}}) + J(\tau^{\text{suf}}) = 336.1216 + 336.1216 = 672.2431$ meters while our method found a plan with cost $J(\tau) = J(\tau^{\text{pre}}) + J(\tau^{\text{suf}}) = 298.5286 + 269.6571 = 568.1857$ meters. Notice that NuSMV can only generate a feasible plan and not the optimal plan, as our proposed algorithm does. The optimal control synthesis method described in Section III-A failed to design a plan that satisfies the considered LTL formula and so did the algorithm presented in [23] due to excessive memory requirements.

B. Case Study II

In the second simulation study, we consider a team of $N = 2$ robots. The transition system describing the motion of each robot is shown in Figure 4(b), and has $|\mathcal{Q}_i| = 16$ states and 70 transitions, including self-loops around each state. The assigned task is expressed in the following temporal logic formula: $\phi = \square\Diamond(\pi_1^{\ell_6} \wedge \Diamond(\pi_2^{\ell_{14}})) \wedge \square(\neg\pi_1^{\ell_9}) \wedge \square(\pi_2^{\ell_{14}} \rightarrow \bigcirc(\neg\pi_2^{\ell_{14}} \mathcal{U} \pi_1^{\ell_4})) \wedge (\Diamond\pi_2^{\ell_{12}}) \wedge (\square\Diamond\pi_2^{\ell_{10}})$ where the respective NBA has $|\mathcal{Q}_B| = 24$ states with $|\mathcal{Q}_B^0| = 1$, $|\mathcal{Q}_B^F| = 4$, and 163 transitions. In words, this LTL-based task requires (a) robot 1 to visit location ℓ_6, (b) once (a) is true robot 2 to visit location ℓ_{14}, (c) conditions (a) and (b) to occur infinitely often, (d) robot 1 to always avoid location ℓ_9, (e) once robot 2 visits location ℓ_{14}, it should avoid this area until robot 1 visits location ℓ_4, (f) robot 2 to visit location ℓ_{12} eventually, and (g) robot 2 to visit location ℓ_{10} infinitely often. In this simulation study, the state space of the PBA consists of $\Pi_{i=1}^{N}|\mathcal{Q}_i||\mathcal{Q}_B| = 6,144$ states, which is small enough so that the method discussed in Section III-A can be used to find the optimal plan. The cost of the optimal plan is $J^* = 14.6569$ meters.

Algorithm 1 was run for various values of the parameters $n_{\text{max}}^{\text{pre}}$ and $n_{\text{max}}^{\text{suf}}$. Observe in Figure 8 that as we increase $n_{\text{max}}^{\text{pre}}$ and $n_{\text{max}}^{\text{suf}}$, the cost of the resulting plans decreases and eventually the optimal plan is found, as expected due to Theorem 5.11. The number of detected final states and runtime for each case are also depicted in the same figure. PRISM verified that there exists a motion plan that satisfies the considered LTL formula in few seconds and NuSMV in less than 1 second. However, neither of them can synthesize the optimal motion plan that satisfies the considered LTL task. For instance, the cost of the plan generated by NuSMV is 30.8995 meters while our algorithm can find the optimal plan with cost $J^* = 14.6569$, as shown in Figure 8. Figure 5(b) depicts the number of rejected states with respect to the iterations n of Algorithm 2. Notice that, as in the previous simulation study, at every iteration n there is at least one state that is added to the tree.

VII. Conclusion

In this paper we proposed a sampling-based control synthesis algorithm for multi-robot systems under global linear temporal logic (LTL) formulas. Existing planning approaches under global temporal goals rely on graph search techniques applied to a synchronous product automaton constructed among the robots. In this paper, we proposed a new sampling-based algorithm to build incrementally trees that approximated the state-space and transitions of the synchronous product automaton increasing in this way significantly scalability of our method compared to existing model-checking approaches. Moreover, we showed that the proposed algorithm is probabilistically complete and asymptotically optimal. Finally, we presented numerical experiments that show that it can be

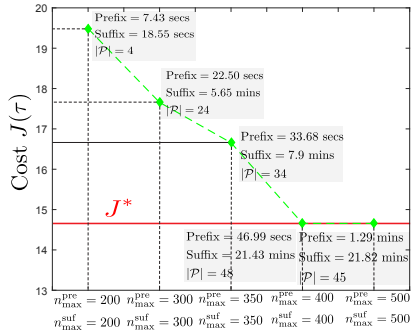


Fig. 8. Evolution of the cost $J(\tau)$ of the optimal motion plan τ for various maximum numbers of iterations, $n^{\text{pre}}_{\max}$ and $n^{\text{suf}}_{\max}$, for Algorithm 2. The time required for the construction of the optimal prefix and suffix part along with the number of detected final states for each case are included in the gray colored box. The red line denotes the optimal cost $J^* = 14.6569$.

used to synthesize optimal plans from product automata with billions of states, which was not possible using standard optimal control synthesis algorithms or off-the-shelf model checkers.

APPENDIX A
PROOFS OF LEMMAS

A. Proof of Lemma 5.4

The proof of this results relies on Lemma 5.3. Let $A^{\text{rand},n+k}(q_P) = \{q_P^{\text{rand},n+k} = q_P\}$, with $k \in \mathbb{N}$, denote the event that at iteration $n+k$ of Algorithm 2 the state $q_P \in \mathcal{V}_T^n$ is selected by the function `Sample` to be the node $q_P^{\text{rand},n+k}$ [line 1, Alg. 3]. Also, let $\mathbb{P}(A^{\text{rand},n+k}(q_P))$ denote the probability of this event, i.e., $\mathbb{P}(A^{\text{rand},n+k}(q_P)) = f_{\text{rand}}(q_P|\mathcal{V}_T^{n+k})$.

Next, define the infinite sequence of events $A^{\text{rand}} = \{A^{\text{rand},n+k}(q_P)\}_{k=0}^{\infty}$, for a given node $q_P \in \mathcal{V}_T^n$. In what follows, we show that the series $\sum_{k=0}^{\infty} \mathbb{P}(A^{\text{rand},n+k}(q_P))$ diverges and then we complete the proof by applying Lemma 5.3. Recall first that the size of $\mathcal{V}_T^{n+k}$ cannot grow arbitrarily large, since it holds that $|\mathcal{V}_T^{n+k}| \leq |\mathcal{Q}_P| < \infty$, for all $k \in \mathbb{N}$. Also, by Assumption 4.1(ii), we have that for a given $q_P \in \mathcal{V}_T^n$, the probability $f_{\text{rand}}(q_P|\mathcal{V}_T^{n+k})$ decreases monotonically with respect to $|\mathcal{V}_T^{n+k}|$. From these two observations we deduce that

$$\mathbb{P}(A^{\text{rand},n+k}(q_P)) = f_{\text{rand}}(q_P|\mathcal{V}_T^{n+k}) \geq f_{\text{rand}}(q_P|\mathcal{Q}_P), \quad (19)$$

for all $k \in \mathbb{N}$, where $f_{\text{rand}}(q_P|\mathcal{Q}_P)$ is the probability assigned to selecting the state $q_P \in \mathcal{V}_T^n$ as $q_P^{\text{rand},n+k}$ when $\mathcal{V}_T^{n+k} = \mathcal{Q}_P$. Note that $f_{\text{rand}}(q_P|\mathcal{Q}_P)$ is a strictly positive term due to Assumption 4.1(i). Also, $f_{\text{rand}}(q_P|\mathcal{Q}_P)$ is constant, by Assumption 4.1(ii), since $\mathcal{Q}_P$ is a fixed set.

Next, since (19) holds for all $k \in \mathbb{N}$, we have that

$$\sum_{k=0}^{\infty} \mathbb{P}(A^{\text{rand},n+k}(q_P)) \geq \sum_{k=0}^{\infty} f_{\text{rand}}(q_P|\mathcal{Q}_P). \quad (20)$$

Since for any state $q_P \in \mathcal{Q}_P$, we have that $f_{\text{rand}}(q_P|\mathcal{Q}_P)$ is a strictly positive constant term, the infinite $\sum_{k=0}^{\infty} f_{\text{rand}}(q_P|\mathcal{Q}_P)$ diverges. Then, we conclude that

$$\sum_{k=0}^{\infty} \mathbb{P}(A^{\text{rand},n+k}(q_P)) = \infty. \quad (21)$$

Combining (21) and the fact that the events $A^{\text{rand},n+k}(q_P)$ are independent by Assumption 4.1(iii), we get that $\mathbb{P}(\limsup_{k\to\infty} A^{\text{rand},n+k}(q_P)) = 1$, due to Lemma 5.3. In other words, the events $A^{\text{rand},n+k}(q_P)$ occur infinitely often, for all $q_P \in \mathcal{V}_T^n$. This equivalently means that for every node $q_P \in \mathcal{V}_T^n$, for all $n \in \mathbb{N}_+$, there exists an infinite subsequence $\mathcal{K} \subseteq \mathbb{N}$ so that for all $k \in \mathcal{K}$ it holds $q_P^{\text{rand},n+k} = q_P$, completing the proof.[9]

Remark A.1 (Lemma 5.4): The result shown in Lemma 5.4 holds even if Assumption 4.1(ii) does not hold, i.e., if the density function $f_{\text{rand}}(q_P|\mathcal{V}_T^n)$ changes every iteration n for a fixed set $\mathcal{V}_T^n$ and a fixed node $q_P \in \mathcal{V}_T^n$, and even if it does not decrease monotonically with the cardinality of $\mathcal{V}_T^n$, as long as this varying density function $f_{\text{rand}}^n(q_P|\mathcal{V}_T^n)$ is bounded below by a sequence $g^n(q_P|\mathcal{V}_T^n)$, such that $\sum_{n=1}^{\infty} g^n(q_P|\mathcal{V}_T^n) = \infty$, for all $q_P \in \mathcal{V}_T^n$. This will ensure that $\sum_{k=0}^{\infty} \mathbb{P}(A^{\text{rand},n+k}(q_P)) = \sum_{k=0}^{\infty} f_{\text{rand}}^{n+k}(q_P|\mathcal{V}_T^{n+k}) \geq \sum_{k=0}^{\infty} g^{n+k}(q_P|\mathcal{V}_T^{n+k}) = \infty$, which replaces (20) and still yields (21).

B. Proof of Lemma 5.5

This proof relies on Lemma 5.3 and resembles the proof of Lemma 5.4. Let $q_P^{\text{rand},n} \in \mathcal{V}_T^n$ and define the infinite sequence of events $A^{\text{new}} = \{A^{\text{new},n+k}(q_{\text{PTS}})\}_{k=0}^{\infty}$, for any given state $q_{\text{PTS}} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$, where $A^{\text{new},n+k}(q_{\text{PTS}}) = \{q_{\text{PTS}}^{\text{new},n+k} = q_{\text{PTS}}\}$, for $k \in \mathbb{N}$, denotes the event that at iteration $n+k$ of Algorithm 2 the state $q_{\text{PTS}} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$ is selected by the function `Sample` to be the node $q_{\text{PTS}}^{\text{new},n+k}$ [line 3, Alg. 3], given a state $q_P^{\text{rand},n+k} = (q_{\text{PTS}}^{\text{rand},n+k}, q_B^{\text{rand},n+k}) \in \mathcal{V}_T^{n+k}$.[10] Moreover, let $\mathbb{P}(A^{\text{new},n+k}(q_{\text{PTS}}))$ denote the probability of this event, i.e., $\mathbb{P}(A^{\text{new},n+k}(q_{\text{PTS}})) = f_{\text{new}}(q_{\text{PTS}}|q_{\text{PTS}}^{\text{rand},n+k})$.

Now, consider those iterations $n+k$ with $k \in \mathcal{K}$ such that $q_P^{\text{rand},n+k} = q_P^{\text{rand},n}$ by Lemma 5.4. We will show that the series $\sum_{k\in\mathcal{K}} \mathbb{P}(A^{\text{new},n+k}(q_{\text{PTS}}))$ diverges and then we will use Lemma 5.3 to show that $q_{\text{PTS}} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n+k})$ will be selected infinitely often to be node $q_{\text{PTS}}^{\text{new},n+k}$.

Since $q_{\text{PTS}} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n+k})$ we have that $\mathbb{P}(A^{\text{new},n+k}(q_{\text{PTS}})) = f_{\text{new}}(q_{\text{PTS}}|q_{\text{PTS}}^{\text{rand},n+k})$ is a strictly positive constant for all $k \in \mathcal{K}$, by Assumption 4.2(i) and 4.2(ii). Therefore, we have that $\sum_{k\in\mathcal{K}} \mathbb{P}(A^{\text{new},n+k}(q_{\text{PTS}}))$ diverges, since it is an infinite sum of a strictly positive constant term. Using this result along with the fact that the events $A^{\text{new},n+k}(q_{\text{PTS}})$ are independent, by Assumption 4.2(iii), we get that $\mathbb{P}(\limsup_{k\to\infty} A^{\text{new},n+k}(q_{\text{PTS}})) = 1$, due to Lemma 5.3. In words, this means that the events $A^{\text{new},n+k}(q_{\text{PTS}})$ for $k \in \mathcal{K}$ occur infinitely often. Thus, for every node $q_{\text{PTS}} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$ for all $n \in \mathbb{N}_+$, there exists an infinite subsequence $\mathcal{K}' \subseteq \mathcal{K}$ so that for all $k \in \mathcal{K}'$ it holds $q_{\text{PTS}}^{\text{new},n+k} = q_{\text{PTS}}$, completing the proof.

Remark A.2 (Lemma 5.5): Lemma 5.5 holds even if Assumption 4.2(ii) does not hold, i.e., if for any given node $q_P^{\text{rand},n} \in \mathcal{V}_T^n$, the density function f_{new} changes with iterations $n+k$, where $k \in \mathcal{K}$, for which $q_P^{\text{rand},n} = q_P^{\text{rand},n+k}$, as long

[9] Note that the subsequence $\mathcal{K}$ is different across the nodes $q_P \in \mathcal{V}_T^n$.

[10] Recall that the reachable set $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$ defined in (6) remains the same for all iterations n, for a given state $q_{\text{PTS}}^{\text{rand},n}$.

as it is bounded below by a sequence $h^{n+k}(q_{\text{PTS}}|q_{\text{PTS}}^{\text{rand},n})$, for all $k \in \mathcal{K}$, such that $\sum_{k\in\mathcal{K}} h^{n+k}(q_{\text{PTS}}|q_{\text{PTS}}^{\text{rand},n}) = \infty$, for all $q_{\text{PTS}} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$.

C. Proof of Corollary 5.6

Recall that a state $q_P = (q_{\text{PTS}}, q_B)$ belongs to $\mathcal{R}_P(q_P^{\text{rand},n})$ if $q_P^{\text{rand},n} \rightarrow_P q_P$, i.e., if (i) $q_{\text{PTS}}^{\text{rand},n} \rightarrow_{\text{PTS}} q_{\text{PTS}}$ and (ii) $q_B^{\text{rand},n} \xrightarrow{L(q_{\text{PTS}}^{\text{rand},n})} q_B$. Then, to prove this result, it suffices to show that all states q_P that satisfy both conditions (i) and (ii) are sampled infinitely often, which is a direct result from Lemma 5.5.

Specifically, observe first that, due to Lemma 5.5, for all states $q_{\text{PTS}} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$, i.e., for all states that satisfy condition (i), there exists an infinite number of iterations $n+k$, at which they will selected to be the nodes $q_{\text{PTS}}^{\text{new},n+k}$ with probability 1, for all $n \in \mathbb{N}$. Second, given a state $q_{\text{PTS}}^{\text{new},n} \in \mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand},n})$, the states $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b))$, for all $b \in \{1,\dots,|\mathcal{Q}_B|\}$ are created, by construction of Algorithm 2. Therefore, given a state $q_{\text{PTS}}^{\text{new},n}$, if there exists $b \in \{1,\dots,|\mathcal{Q}_B|\}$ such that $q_B^{\text{rand},n} \xrightarrow{L(q_{\text{PTS}}^{\text{rand},n})} \mathcal{Q}_B(b)$, then the state $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b)) \in \mathcal{R}_P(q_P^{\text{rand},n})$ that satisfies (ii) will be sampled/constructed infinitely often.[11] Thus, for all states $q_P \in \mathcal{R}_P(q_P^{\text{rand},n})$ there exists an infinite number of iterations $n+k$, with $k \in \mathcal{K}' \subseteq \mathcal{K}$, at which they will be selected by Algorithm 3 to be the node $q_P^{\text{new},n+k}$ completing the proof.

D. Proof of Lemma 5.7

First note that (13) trivially holds for all states $q_P^{\text{rand},n}$ that satisfy $\mathcal{R}_P(q_P^{\text{rand},n}) = \emptyset$. Hence, in what follows we consider only states $q_P^{\text{rand},n} \in \mathcal{V}_T^n$ that satisfy $\mathcal{R}_P(q_P^{\text{rand},n}) \neq \emptyset$. The proof of this result relies on Corollary 5.6. Specifically, recall that, due to Corollary 5.6, for all states $q_P \in \mathcal{R}_P(q_P^{\text{rand},n})$, there exists an infinite number of iterations $n+k$, $k \in \mathcal{K}' \subseteq \mathcal{K}$, such that $q_P^{\text{new},n+k} = q_P$. This means that with probability 1, there exists an iteration $n+k$ of Algorithm 2 at which the state $q_P^{\text{new},n+k} = q_P$, will be sampled.

Since this iteration $n+k$ satisfies $q_P^{\text{new},n+k} \in \mathcal{R}_P(q_P^{\text{rand},n})$, we get that $q_P^{\text{rand},n} \in \mathcal{R}^{\rightarrow}_{\mathcal{V}_T^{n+k}}(q_P^{\text{new},n+k}) \subseteq \mathcal{V}_T^{n+k}$. This follows from the definition of $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T^{n+k}}(q_P^{\text{new},n+k})$ in (7) and the fact that since $q_P^{\text{rand},n}$ belongs to $\mathcal{V}_T^n$ it also belongs to $\mathcal{V}_T^{n+k}$. Therefore, $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T^{n+k}}(q_P^{\text{new},n+k}) \neq \emptyset$, which means that the state $q_P^{\text{new},n+k} = q_P \in \mathcal{R}_P(q_P^{\text{rand},n})$ will be added to the tree at iteration $n+k$ by construction of Algorithm 4; see line 2 in Algorithm 4. We conclude, that for all states $q_P \in \mathcal{R}_P(q_P^{\text{rand},n})$ there exists a subsequent iteration $n+k$ at which they will be added to the tree with probability 1.[12] In mathematical terms, this result can be written as $\lim_{k\to\infty} \mathbb{P}\left(\{\mathcal{R}_P(q_P^{\text{rand},n}) \subseteq \mathcal{V}_T^{n+k}\}\right) = 1$ completing the proof.

[11] Recall that the reachable set $\mathcal{R}_P(q_P^{\text{rand},n})$ defined in (6) remains the same for all iterations n, for a given state $q_P^{\text{rand},n}$.

[12] If the states $q_P \in \mathcal{R}_P(q_P^{\text{rand},n})$ already belong to the tree then the rewiring step follows.

E. Proof of Corollary 5.8

This result is due to Lemmas 5.4 and 5.7. Specifically, by Lemma 5.7, we have that

$$\lim_{k\to\infty} \mathbb{P}\left(\{\mathcal{R}_P(q_P^{\text{rand},n}) \subseteq \mathcal{V}_T^{n+k}\}\right) = 1, \tag{22}$$

for a given state $q_P^{\text{rand},n} \in \mathcal{V}_T^n \subseteq \mathcal{V}_T^{n+k}$ and any iteration $n \in \mathbb{N}_+$. Also, by Lemma 5.4, we have that every state $q_P \in \mathcal{V}_T^n$ will be selected infinitely often to be the node $q_P^{\text{rand},n+k}$, as $k \to \infty$. Therefore, we get that (22) holds for all states $q_P \in \mathcal{V}_T^n$, i.e.,

$$\lim_{k\to\infty} \mathbb{P}\left(\{\mathcal{R}_P(q_P) \subseteq \mathcal{V}_T^{n+k}\}\right) = 1, \ \forall q_P \in \mathcal{V}_T^n. \tag{23}$$

Since (23) holds for any iteration $n \in \mathbb{N}_+$, we can rewrite (23) as $\lim_{n\to\infty} \mathbb{P}\left(\{\mathcal{R}_P(q_P) \subseteq \mathcal{V}_T^n\}\right) = 1, \ \forall q_P \in \mathcal{V}_T^n$ completing the proof.

F. Proof of Corollary 5.10

The proof relies on Corollary 5.6 and Lemma 5.4. From Lemma 5.4, we have that for every state $q'_P \in \mathcal{V}_T^n$ there exists an infinite number of iterations $n+k$, with $k \in \mathcal{K}$, at which the state q'_P is selected by Algorithm 3 to be the node $q_P^{\text{rand},n+k}$. Also, for any iteration n and for any state $q_P^{\text{rand},n}$, we know from Corollary 5.6 that for every state $q_P \in \mathcal{R}_P(q_P^{\text{rand},n})$ there exists an infinite number of subsequent iterations $n+k$, with $k \in \mathcal{K}' \subseteq \mathcal{K}$, at which the state q_P is selected by Algorithm 3 to be the node $q_P^{\text{new},n+k}$, given a node $q_P^{\text{rand},n}$. Combining these two results, we get that for every state $q_P \in \mathcal{R}_P(q'_P)$, and for all $q'_P \in \mathcal{V}_T^n$, there exists an infinite number of subsequent iterations $n+k$, with $k \in \mathcal{K}'$, at which the state q_P is selected to be the node $q_P^{\text{new},n+k}$. Equivalently, this means that for every state $q_P \in \cup_{q'_P\in\mathcal{V}_T^n} \mathcal{R}_P(q'_P)$ there exists an infinite number of iterations $n+k$, with $k \in \mathcal{K}'$, at which the state q_P, is selected to be the node $q_P^{\text{new},n+k}$. Then, it suffices to show that the set $\mathcal{V}_T^n$ is a subset of the set $\cup_{q'_P\in\mathcal{V}_T^n} \mathcal{R}_P(q'_P)$. This would mean that for any state $q_P \in \mathcal{V}_T^n$, there exists and infinite number of iterations $n+k$, with $k \in \mathcal{K}'$, at which the state $q_P \in \mathcal{V}_T^n$, is selected by Algorithm 3 to be the node $q_P^{\text{new},n+k}$.

To show that $\mathcal{V}_T^n \subseteq \cup_{q'_P\in\mathcal{V}_T^n} \mathcal{R}_P(q'_P)$, we will show that if $q \in \mathcal{V}_T^n$ then $q \in \cup_{q'_P\in\mathcal{V}_T^n} \mathcal{R}_P(q'_P)$. Consider a node $q \in \mathcal{V}_T^n$. Then, by construction of the tree, we have that there exists another node $q' \in \mathcal{V}_T^n$, such that $\texttt{parent}(q) = q' \in \mathcal{V}_T^n$, which means $q \in \mathcal{R}_P(q')$. Observe that $\mathcal{R}_P(q') \subseteq \cup_{q'_P\in\mathcal{V}_T^n} \mathcal{R}_P(q'_P)$, since $q' \in \mathcal{V}_T^n$ by assumption. Therefore, we have that $q \in \mathcal{R}_P(q') \subseteq \cup_{q'_P\in\mathcal{V}_T^n} \mathcal{R}_P(q'_P)$, i.e., $q \in \cup_{q'_P\in\mathcal{V}_T^n} \mathcal{R}_P(q'_P)$ completing the proof.

REFERENCES

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[2] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain, April 2005, pp. 2020–2025.

[3] M. Guo and M. M. Zavlanos, "Distributed data gathering with buffer constraints and intermittent communication," in *Robotics and Automation (ICRA), IEEE International Conference on*, Singapore, May-June 2017, pp. 279–284.

[4] Y. Kantaros and M. M. Zavlanos, "Distributed intermittent connectivity control of mobile robot networks," *Transactions on Automatic Control*, vol. 62, no. 7, pp. 3109–3121, July 2017.

[5] K. Leahy, D. Zhou, C.-I. Vasile, K. Oikonomopoulos, M. Schwager, and C. Belta, "Persistent surveillance for unmanned aerial vehicles subject to charging and temporal logic constraints," *Autonomous Robots*, vol. 40, no. 8, pp. 1363–1378, 2016.

[6] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press Cambridge, 2008.

[7] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.

[8] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[9] ——, "Where's waldo? sensor-based temporal logic motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, Roma, Italy, April 2007, pp. 3116–3121.

[10] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, AL, May 2010, pp. 2689–2696.

[11] Y. Chen, X. C. Ding, and C. Belta, "Synthesis of distributed control and communication schemes from global LTL specifications," in *50th IEEE Conference on Decision and Control and European Control Conference*, Orlando, FL, USA, December 2011, pp. 2718–2723.

[12] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta, "Formal approach to the deployment of distributed robotic teams," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 158–171, 2012.

[13] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[14] 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.

[15] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic motion specifications," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 48–61, 2010.

[16] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.

[17] A. Ulusoy, S. L. Smith, and C. Belta, "Optimal multi-robot path planning with ltl constraints: guaranteeing correctness through synchronization," in *Distributed Autonomous Robotic Systems*. Springer, 2014, pp. 337–351.

[18] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[19] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic symbolic model checker," in *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, London, UK, April 2002, pp. 200–204.

[20] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *International Conference on Computer Aided Verification*. Springer, 2002, pp. 359–364.

[21] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning with deterministic μ-calculus specifications," in *American Control Conference (ACC)*, Montreal, Canada, June 2012, pp. 735–742.

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

[23] Y. Kantaros and M. M. Zavlanos, "Intermittent connectivity control in mobile robot networks," in *49th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, USA, November, 2015, pp. 1125–1129.

[24] ——, "Sampling-based control synthesis for multi-robot systems under global temporal specifications," in *Proc. 8th ACM/IEEE International Conference on Cyber-Physical Systems*, Pittsburgh, PA, USA, April 2017, pp. 3–13.

[25] D. C. Conner, A. A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, NV, USA, October 2003, pp. 3546–3551.

[26] C. Belta and L. Habets, "Constructing decidable hybrid systems with velocity bounds," in *43rd IEEE Conference on Decision and Control (CDC)*, Nassau, Bahamas, December 2004, pp. 467–472.

[27] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environments," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 864–874, 2005.

[28] M. Kloetzer and C. Belta, "Reachability analysis of multi-affine systems," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2006, pp. 348–362.

[29] D. Boskos and D. V. Dimarogonas, "Decentralized abstractions for multi-agent systems under coupled constraints," in *54th IEEE Conference on Decision and Control (CDC)*, Osaka, Japan, December 2015, pp. 7104–7109.

[30] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in *1st Symposium in Logic in Computer Science (LICS)*. IEEE Computer Society, 1986.

[31] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013, pp. 5025–5032.

[32] R. Sedgewick and K. Wayne, *Algorithms*. Addison-Wesley Professional, 2011.

[33] R. E. Korf, "Best-first frontier search with delayed duplicate detection," in *AAAI*, vol. 4, 2004, pp. 650–657.

[34] S. Russell and P. Norvig, "Artificial intelligence: A modern approach," *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, p. 27, 1995.

[35] G. Grimmett and D. Stirzaker, *Probability and random processes*. Oxford university press, 2001.

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

Yiannis Kantaros received the Diploma in Electrical and Computer Engineering in 2012 from the University of Patras, Patras, Greece. He is currently working toward the Ph.D. degree in the Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC, USA. His current research interests include distributed control, distributed optimization, multi-agent systems and robotics. He received the Best Student Paper Award at the 2nd IEEE Global Conference on Signal and Information Processing in 2014.

Michael M. Zavlanos (S'05M'09) received the Diploma in mechanical engineering from the National Technical University of Athens (NTUA), Athens, Greece, in 2002, and the M.S.E. and Ph.D. degrees in electrical and systems engineering from the University of Pennsylvania, Philadelphia, PA, in 2005 and 2008, respectively.

He is currently an Associate Professor in the Department of Mechanical Engineering and Materials Science at Duke University, Durham, NC. He also holds a secondary appointment in the Department of Electrical and Computer Engineering and the Department of Computer Science. Prior to joining Duke University, Dr. Zavlanos was an Assistant Professor in the Department of Mechanical Engineering at Stevens Institute of Technology, Hoboken, NJ, and a Postdoctoral Researcher in the GRASP Lab, University of Pennsylvania, Philadelphia, PA. His current research interests include networked systems, distributed control and optimization, and formal methods and control synthesis, with applications in robotics, wireless networking, and sensing and estimation.

Dr. Zavlanos is a recipient of various awards including the 2014 Naval Research Young Investigator Program (YIP) Award and the 2011 National Science Foundation Faculty Early Career Development (CAREER) Award.