Sampling-based Control Synthesis for Multi-Robot Systems under Global Temporal Specifications*

Yiannis Kantaros
Department of Mechanical Engineering and Materials Science
Duke University
Box 90300 Hudson Hall
Durham, North Carolina 27708-0300
ik36@duke.edu

Michael M. Zavlanos
Department of Mechanical Engineering and Materials Science
Duke University
Box 90300 Hudson Hall
Durham, North Carolina 27708-0300
mz61@duke.edu

ABSTRACT

This paper proposes a sampling-based algorithm for multi-robot control synthesis under global Linear Temporal Logic (LTL) formulas. Robot mobility is captured by transition systems whose states represent regions in the environment that satisfy atomic propositions. Existing planning approaches under global temporal goals rely on graph search techniques applied to a synchronous product automaton constructed among the robots. As the number of robots increases, the state-space of the product automaton grows exponentially and, as a result, graph search techniques become intractable. In this paper, we propose a new sampling-based algorithm that builds incrementally a directed tree that approximates 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 resources to store it and motion plans can be found by tracing the sequence of parent nodes from the leaves back to the root without the need for sophisticated graph search techniques. This significantly increases scalability of our algorithm compared to existing model-checking methods. We also show that our algorithm is probabilistically complete and asymptotically optimal and present numerical experiments that show that it can be used to model-check product automata with billions of states, which was not possible using an off-the-shelf model checker.

CCS CONCEPTS

•**Computing methodologies** →Planning and scheduling; **Multi-agent planning; Motion path planning;** *Robotic planning;* Distributed artificial intelligence; •**Theory of computation** →*Formal languages and automata theory;*

KEYWORDS

Motion Planning, Control Synthesis, Multi-agent systems, Linear Temporal Logic

*This work is supported in part by the NSF awards IIS #130228.

ICCPS, Pittsburgh, PA USA

DOI: http://dx.doi.org/10.1145/3055004.3055027

ACM Reference format:
Yiannis Kantaros and Michael M. Zavlanos. 2017. Sampling-based Control Synthesis for Multi-Robot Systems under Global Temporal Specifications. In *Proceedings of The 8th ACM/IEEE International Conference on Cyber-Physical Systems, Pittsburgh, PA USA, April 2017 (ICCPS),* 11 pages.
DOI: http://dx.doi.org/10.1145/3055004.3055027

1 INTRODUCTION

Control synthesis for mobile robots under complex tasks, captured by Linear Temporal Logic (LTL) formulas, build upon either a bottom-up approach when independent LTL expressions are assigned to robots [7, 8, 13] or top-down approaches when a global LTL formula describing a collaborative task is assigned to a team of robots [4, 16], as in our work. Top-down approaches generate a discrete high-level motion plan every robot using the individual transition systems that capture robot mobility and a Non-determisitic 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, motion plans can be derived that satisfy the global LTL specification. As the number of robots increases, the state-space of the product automaton grows exponentially and, as a result, graph-search techniques are not applicable as they become extremely resource demanding. Consequently, these motion planning algorithms scale poorly with the size of the network. To address these issues, several methods have been proposed that fall into two categories: (i) internal memory algorithms, such as partial order reduction [1] and symmetry order reduction [5] and (ii) external memory algorithms [19] that can handle state-spaces with billions of states but require external memory devices.

In this paper, we propose an internal memory algorithm which unlike existing model checking algorithms, completely avoids constructing the product among the transition systems and the NBA. Specifically, motivated by existing sampling-based algorithms [12], 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, to construct a motion plan in a prefix-suffix structure, we first build a tree incrementally 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 infinitely. The advantage of the proposed method is that approximating the product automaton by a tree rather than representing it explicitly by an arbitrary graph structure, 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, the scalability of the proposed model-checking algorithm is significantly increased compared to existing approaches. Moreover, despite the approximate representation of the global automaton, 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 be used to model-check product automata with billions of states, which was impossible using the off-the-shelf model checker PRISM [14].

To the best of our knowledge, the most relevant works are presented in [3, 10, 11, 18]. In [11], 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. In [3], a sampling-based algorithm is proposed for motion planning under temporal goals. The main goal of that work is to construct a discrete abstraction of the environment by taking into account the geometry of obstacles, the robot dynamics, and the atomic propositions that are satisfied in the workspace, to solve planning problems that involve temporal goals. Common in [3, 11] is that single-agent motion planning problems are considered, unlike our approach that is amenable to multi-robot path planning problems. In order to apply the methods proposed in [3, 11] to multi-agent motion planning problems that we consider in this paper, a product system among the agents needs to be constructed that is represented by a graph of arbitrary structure, as in [18]. Specifically, in [18] a sampling-based temporal logic path planning algorithm is proposed that also scales well for large configuration spaces. The main difference between [18] and the work proposed here is that we approximate the state-space of the product automaton by a tree which is more economical in terms of memory requirements and significantly decreases the computational cost of applying graph search techniques. This allows our method to handle large problems compared to the ones that can be solved using the approach in [18]. Moreover, we show that our proposed planning algorithm is asymptotically optimal which is not the case in [18]. Finally, common in all the above works 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, in our work, given a discrete abstraction of the workspace, we build incrementally trees that approximate the product automaton, until a motion plan is constructed. In [10], a planning algorithm for multi-agent systems under global temporal goals is proposed. The main goal of that work is to transform 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 size of the number of robots, since it relies on the construction of a product automaton among all agents, which is not the case in our work.

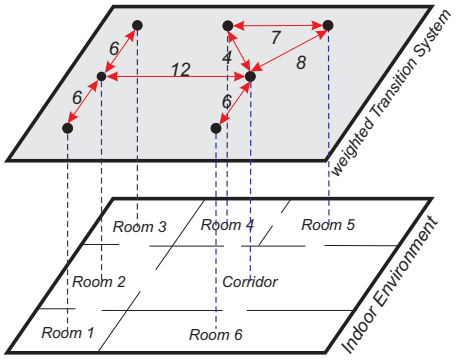


Figure 1: Graphical depiction of a wTS that abstracts robot mobility in an indoor environment. Black disks stand for the states of wTS, red edges capture transitions among states and numbers on these edges represent the cost w_i for traveling from one state to another one.

2 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, $i \in \{1, \dots, N\}$. 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. Robot mobility in the workspace is represented by a transition system defined as follows:

Definition 2.1. A *weighted Transition System* for robot i, denoted by wTS_i is a tuple $\text{wTS}_i = \left(Q_i, q_i^0, \rightarrow_i, w_i, \mathcal{AP}, L_i\right)$ where:

- $Q_i = \{q_i^{\ell_j}\}_{j=1}^{W}$ is the set of states, where a state $q_i^{\ell_j}$ indicates that robot i is at location ℓ_j;
- $q_i^0 \in Q_i$ is the initial state of robot i;
- $\rightarrow_i \subseteq Q_i \times Q_i$ 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 $(q_i^{\ell_j}, q_i^{\ell_e}) \in \rightarrow_i$;
- $w_i : Q_i \times Q_i \rightarrow \mathbb{R}_+$[1] is a cost function that assigns weights/cost to each possible transition in wTS. These costs are associated with the distance that needs to be traveled by robot i in order to move from state $q_i^{\ell_j}$ to state $q_i^{\ell_k}$;
- $\mathcal{AP} = \{\{\pi_i^{\ell_j}\}_{j=1}^{W}\}_{i=1}^{N}$ 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
- $L_i : Q_i \rightarrow 2^{\mathcal{AP}}$ is an observation/output relation giving the set of atomic propositions that are satisfied in a state.

Figure 1 illustrates Definition 2.1 for a robot that resides in an indoor environment. In what follows we give definitions related to wTS_i, that we will use throughout the rest of the paper. An *infinite path* τ_i of wTS_i is an infinite sequence of states, $\tau_i = \tau_i(1)\tau_i(2)\tau_i(3)\dots$ such that $\tau_i(1) = q_i^0$, $\tau_i(m) \in Q_i$, and

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

$(\tau_i(k), \tau_i(k+1)) \in \rightarrow_i, \forall k \in \mathbb{N}_+$, where k is an index that points to the k-th entry of τ_i denoted by $\tau_i(k)$. A *finite path* of wTS_i 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 wTS_i. Given the definition of the weights w_i in Definition 2.1, the *cost* of a finite path τ_i, denoted by $J(\tau_i)$, can be defined as follows:

$$J(\tau_i) = \sum_{k=1}^{|\tau_i|-1} w_i(\tau_i(k), \tau_i(k+1)). \quad (1)$$

In (1), $|\tau_i|$ stands for the number of states in τ_i. In words, the cost (1) captures the distance traveled by robot i during the execution of the finite path τ_i. The *trace* of an infinite path $\tau_i = \tau_i(1)\tau_i(2)\tau_i(3)\dots$ of a transition system wTS_i, denoted by $\texttt{trace}(\tau_i) \in \left(2^{\mathcal{AP}}\right)^{\omega}$, 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, i.e., $\texttt{trace}(\tau_i) = L_i(\tau_i(1))L_i(\tau_i(2))\dots$. The *language* $\texttt{Words}(\phi_i) = \left\{\sigma \in (2^{\mathcal{AP}})^{\omega} | \sigma \models \phi_i\right\}$, where $\models \subseteq (2^{\mathcal{AP}}) \times \phi_i$ is the satisfaction relation, is defined as the set of infinite words $\sigma \in (2^{\mathcal{AP}})^{\omega}$ that satisfy the LTL ϕ_i. Given an LTL formula ϕ_i, a transition system wTS_i both defined over the set of atomic propositions $\mathcal{AP}$, the infinite path τ_i of wTS_i satisfies ϕ_i if and only if $\texttt{trace}(\tau_i) \in \texttt{Words}(\phi)_i$, which is equivalently denoted by $\tau_i \models \phi_i$.

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 $\mathcal{AP} = \left\{\{\pi_i^{\ell_j}\}_{j=1}^{W}\right\}_{i=1}^{N}$. Then the problem that this paper addresses can be summarized as follows

Problem 1. *Given a global LTL specification ϕ, transitions systems wTS_i, for all robots i, determine a discrete team plan τ that satisfies ϕ, i.e., $\tau \models \phi$.*

3 PRELIMINARIES

In this section, we summarize an existing automata-based planning algorithm that solves Problem 1. First the synchronous *Product Transition System* (PTS) is constructed, which essentially captures all the possible combinations of robots' states in their respective wTS_i, and is defined as follows:

Definition 3.1. Given N transition systems $\text{wTS}_i = (Q_i, q_i^0, \rightarrow_i, w_i, \mathcal{AP}, L_i)$, the *product transition system* $\text{PTS} = \text{wTS}_1 \otimes \text{wTS}_2 \otimes \dots \otimes \text{wTS}_N$ is a tuple $\text{PTS} = (Q_{\text{PTS}}, q_{\text{PTS}}^0, \longrightarrow_{\text{PTS}}, w_{\text{PTS}}, \mathcal{AP}, L_{\text{PTS}})$ where (a) $Q_{\text{PTS}} = Q_1 \times Q_2 \times \dots \times Q_N$ is the set of states; (b) $q_{\text{PTS}}^0 = (q_1^0, q_2^0, \dots, q_N^0) \in Q_{\text{PTS}}$ is the initial state, (c) $\longrightarrow_{\text{PTS}} \subseteq Q_{\text{PTS}} \times Q_{\text{PTS}}$ is the transition relation defined by the rule[2] $\frac{\wedge_{\forall i}(q_i \rightarrow_i q_i')}{q_{\text{PTS}} \rightarrow_{\text{PTS}} q_{\text{PTS}}'}$, where with slight abuse of notation $q_{\text{PTS}} = (q_1, \dots, q_N) \in Q_{\text{PTS}}$, $q_i \in Q_i$. 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 $i \in \{1, \dots, N\}$; (d) $w_{\text{PTS}} : Q_{\text{PTS}} \times Q_{\text{PTS}} \rightarrow \mathbb{R}_+$ is a cost function that assigns weights/cost to each possible transition in PTS, defined as $w_{\text{PTS}}(q_{\text{PTS}}, q_{\text{PTS}}') = \sum_{i=1}^{N} w_i(\Pi|_{\text{wTS}_i} q_{\text{PTS}}, \Pi|_{\text{wTS}_i} q_{\text{PTS}}')$, where $q_{\text{PTS}}', q_{\text{PTS}} \in Q_{\text{PTS}}$, 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 $\Pi_{\text{wTS}_i} q_{\text{PTS}} \in Q_i$ is obtained by removing all states in q_{PTS} that do not belong to Q_i; (e) $\mathcal{AP}$ is the set of atomic propositions; and, (f) $L_{\text{PTS}} = \bigcup_{\forall i} L_i : Q_{\text{PTS}} \rightarrow 2^{\mathcal{AP}}$ is an observation/output relation giving the set of atomic propositions that are satisfied at a state $q_{\text{PTS}} \in Q_{\text{PTS}}$.

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

Any LTL formula ϕ defined over a set of atomic propositions $\mathcal{AP}$ can be translated into a Nondeterministic Büchi Automaton (NBA) over $2^{\mathcal{AP}}$ denoted by B [17], where its accepting language is $\mathcal{L}_B = \texttt{Words}(\phi)$. The NBA is defined as follows:

Definition 3.2. A *Nondeterministic Büchi Automaton* (NBA) B over $2^{\mathcal{AP}}$ is defined as a tuple $B = \left(Q_B, Q_B^0, \Sigma, \rightarrow_B, Q_B^F\right)$ where (a) Q_B is the set of states; (b) $Q_B^0 \subseteq Q_B$ is a set of initial states; (c) $\Sigma = 2^{\mathcal{AP}}$ is an alphabet; (d) $\rightarrow_B \subseteq Q_B \times \Sigma \times Q_B$ is the transition relation; and (e) $Q_B^F \subseteq Q_B$ is a set of accepting/final states.

Once the PTS and the NBA B that corresponds to the LTL ϕ are constructed, a motion plan $\tau \models \phi$ can be found by checking the non-emptiness of the language of the *Product Büchi Automaton* (PBA) $P = \text{PTS} \otimes B$ [2], which is defined as follows:

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

To check the non-emptiness of the language of P denoted by $\mathcal{L}_P = \texttt{trace}(\text{PTS}) \cap \mathcal{L}_B$, where $\texttt{trace}(\text{PTS})$ collects all words that can be generated by the PTS, and to find a motion plan τ that satisfies ϕ, existing model checking methods can be used that are based on graph search algorithms; see, e.g., [7]. Such motion plans can be written in a prefix-suffix structure $\tau = \tau^{\text{pre}}[\tau^{\text{suf}}]^{\omega}$. The prefix part τ^{pre} has the following structure $\tau^{\text{pre}} = q_{\text{PTS}}^1 q_{\text{PTS}}^2 \dots q_{\text{PTS}}^K$ and is executed only once. The suffix part τ^{suf} has the following structure $\tau^{\text{suf}} = q_{\text{PTS}}^K q_{\text{PTS}}^{K+1} \dots q_{\text{PTS}}^{K+S} q_{\text{PTS}}^{K+S+1}$, where $q_{\text{PTS}}^{K+S+1} = q_{\text{PTS}}^K$, and is repeated infinitely. The cost of such a motion plan is defined as

$$J(\tau) = \underbrace{\sum_{k=1}^{K-1} w_{\text{PTS}}(\tau^{\text{pre}}(k), \tau^{\text{pre}}(k+1))}_{\text{Cost } J(\tau^{\text{pre}}) \text{ of prefix}} + \underbrace{\sum_{k=K}^{K+S} w_{\text{PTS}}(\tau^{\text{suf}}(k)\tau^{\text{suf}}(k+1))}_{\text{Cost } J(\tau^{\text{suf}}) \text{ of suffix}} = J(\tau^{\text{pre}}) + J(\tau^{\text{suf}}), \quad (2)$$

which in fact captures the total distance traveled by all robots during the execution of the prefix and a single execution of the suffix part.

In principle, 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 Q_P, the set of edges $\mathcal{E}_P$ is determined by the transition relation $\longrightarrow_P$, and the weights w_P assigned on each edge are inherited by the function w_{PTS}. Specifically, the weight assigned on a edge that connects two nodes that represent the states q_P and q'_P is equal to $w_P(q_P, q'_P) = w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P, \Pi|_{\text{PTS}} q'_P)$. Then we find the shortest paths from the initial states to all reachable final states $q_P \in Q_P^F$ and projecting these paths onto PTS results in the prefix parts $\tau^{\text{pre},f}$, where $f = \{1, \ldots, |Q_P^F|\}$. The respective suffix parts $\tau^{\text{suf},f}$ are constructed similarly by computing the shortest cycle around the f-th final state. All the resulting motion plans $\tau^f = \tau^{\text{pre},f}[\tau^{\text{suf},f}]^\omega$ 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 $J(\tau^f)$ for all plans and picking the one with the smallest cost.

4 PROPOSED SOLUTION

The automata-based planning algorithm described in Section 3 can be employed to solve Problem 1. However, constructing the PBA and applying graph-search techniques on it, is resource demanding and scales poorly with the size of the network. Thus, in this section, we propose a sampling-based planning algorithm that scales well with the number of agents 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.[3] The procedure is based on an incremental construction of a directed tree that approximately encompasses the state-space 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 Algorithms 1 and 7, respectively.

In what follows, we denote by $\mathcal{G}_T = \{\mathcal{V}_T, \mathcal{E}_T, C_T\}$ the tree that approximately represents the PBA P. The set of nodes $\mathcal{V}_T$ contains the states of Q_P that have already been sampled and added to the tree structure. The set of edges $\mathcal{E}_T$ captures transitions among the 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 set C_T collects the cost of reaching each node $q_P \in \mathcal{V}_T$ from the root of the tree; with slight abuse of notation the cost of a node that represents a state q_P is denoted by $C_T(q_P)$.

4.1 Construction of Prefix Parts

In this Section, we describe how the prefix part is constructed. This procedure is described in Algorithm 1, as well. The set $\mathcal{V}_T$ initially contains only the initial state q_P^0 of the PBA [line 1 , Alg. 1] and therefore, the set of edges is initialized as $\mathcal{E}_T = \emptyset$ [line 2, Alg. 1]. By convention, we assume that the cost of q_P^0 is zero [line 3, Alg. 1]. The set $\mathcal{F} \subseteq \mathcal{V}_T$ [line 4, Alg. 1] collects all the final states of P that exist in the tree.

4.1.1 Sampling a state $q_P^{new} \in Q_P$. The first step for the construction of the graph $\mathcal{G}_T$ is to sample a state from the state-space of the product transition system. This is achieved by a sampling

[3]LTL specifications are satisfied by plans that are *infinite* sequences of states and, therefore, they cannot be manipulated in practice. Such an issue can be resolved by representing these plans by *finite* sequences of states, in a prefix-suffix form, called prefix-suffix form, where the prefix is executed once and suffix is executed indefinitely.

Algorithm 1: Construction of prefix parts.

Input: Initial state q_i^0, transition system wTS_i, for all robots i, NBA B, maximum number of iterations $n_{\max}^{\text{pre}}$
Output: Prefix parts $\tau^{\text{pre},f}$, set of final states $\mathcal{F}$

1 $\mathcal{V}_T = q_P^0 = \{q_{\text{PTS}}^0, q_B^0\}$, where $q_{\text{PTS}}^0 = \{q_1^0, q_2^0, \ldots, q_N^0\}$;
2 $\mathcal{E}_T = \emptyset$;
3 $C_T(q_P^0) = 0$;
4 $\mathcal{F} = \emptyset$;
5 **for** $n = 1:1:n_{max}^{pre}$ **do**
6 $\quad q_{\text{PTS}}^{\text{new}} = \texttt{Sample}(Q_1, \ldots, Q_N)$;
7 $\quad$ **for** $b = 1:|Q_B|$ **do**
8 $\quad\quad q_B^{\text{new}} = Q_B(b)$;
9 $\quad\quad q_P^{\text{new}} = (q_{\text{PTS}}^{\text{new}}, q_B^{\text{new}})$;
10 $\quad\quad$ **if** $q_P^{new} \notin \mathcal{V}_T$ **then**
11 $\quad\quad\quad [\mathcal{V}_T, \mathcal{E}_T, C_T] = \texttt{Extend}(q_P^{\text{new}}, \rightarrow_P)$;
12 $\quad\quad\quad$ **if** $q_P^{new} \in \mathcal{V}_T$ **then**
13 $\quad\quad\quad\quad$ **if** $(q_B^{new} \in Q_B^F)$ **then**
14 $\quad\quad\quad\quad\quad \mathcal{F} = \mathcal{F} \cup \{q_P^{\text{new}}\}$;
15 $\quad\quad\quad\quad [\mathcal{E}_T, C_T] = \texttt{Rewire}(q_P^{\text{new}}, \mathcal{V}_T, \mathcal{E}_T, C_T)$;
16 $\mathcal{E}_T = \texttt{OptimizeTree}(\mathcal{G}_T)$;
17 **for** $f = 1:|\mathcal{F}|$ **do**
18 $\quad \tau^{\text{pre},f} = \texttt{FindPath}(\mathcal{G}_T, q_P^0, \mathcal{F}(k))$;

function Sample that generates independent samples that belong to Q_{PTS} from a given distribution; see Algorithm 2. The sampled state is denoted by $q_{\text{PTS}}^{\text{new}}$ [line 6, Alg. 1]. Since our goal is to build incrementally a graph whose set of nodes represents the state space Q_P we need to append to $q_{\text{PTS}}^{\text{new}}$ a state from the state-space Q_B of NBA B. Let $q_B^{\text{new}} = Q_B(b)$ [line 8, Alg. 1] be the candidate Büchi state that will be attached to $q_{\text{PTS}}^{\text{new}}$, where $b \in \{1, \ldots, |Q_B|\}$. The following procedure is repeated for all $b \in \{1, \ldots, |Q_B|\}$. First, we construct the state $q_P^{\text{new}} = (q_{\text{PTS}}^{\text{new}}, q_B^{\text{new}}) \in Q_P$ [line 9, Alg. 1] and in what follows, we check if this state can be added to the tree $\mathcal{G}_T$ [lines 10-15, Alg. 1]. In case the state q_P^{new} has not already been added to the tree from a previous iteration of Algorithm 1, i.e, if $q_P^{\text{new}} \notin \mathcal{V}_T$ [line 10, Alg. 1], 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 accomplished by the function Extend described in Algorithm 3.

4.1.2 Adding a new edge to $\mathcal{E}_T$. The first step in Algorithm 3 is to construct the set $\mathcal{S}_{\rightarrow q_P^{\text{new}}} \subseteq \mathcal{V}_T$ that collects all states $q_P \in \mathcal{V}_T$ that abide by the transition rule: $(q_P, q_P^{\text{new}}) \in \rightarrow_P$ [line 1, Alg. 3]. If the resulting set is empty then the sample q_P^{new} is not added to the tree. In case $\mathcal{S}_{\rightarrow q_P^{\text{new}}} \neq \emptyset$, then the state q_P^{new} is added to the tree [lines 3-6, Alg. 3]. The parent of q_P^{new}, denoted by q_P^{prev} satisfies $q_P^{\text{prev}} = \text{argmin}_{q_P \in \mathcal{S}_{\rightarrow q_P^{\text{new}}}} [C_T(q_P) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P, \Pi|_{\text{PTS}} q_P^{\text{new}})]$, where by definition of the set C_T, $C_T(q_P) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P, \Pi|_{\text{PTS}} q_P^{\text{new}})$ stands for 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 $\mathcal{S}_{\rightarrow q_P^{\text{new}}}$ so that the incurred cost of q_P^{new} is minimized [line 3, Alg. 3]. The set of nodes and edges is updated in lines 4 and 5, respectively, as $\mathcal{V}_T = \mathcal{V}_T \cup \{q_P^{\text{new}}\}$ and $\mathcal{E}_T = \mathcal{E}_T \cup \{(q_P^{\text{prev}}, q_P^{\text{new}})\}$. Next we compute the cost of node q_P^{new}

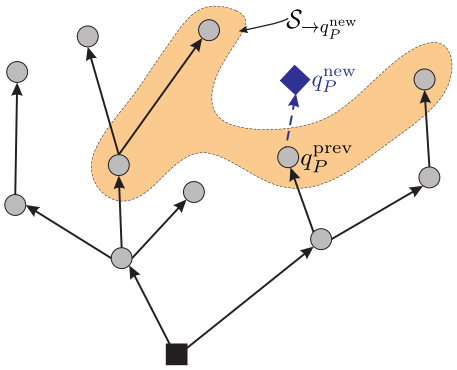


Figure 2: Graphical depiction of Algorithm 3. The black square stands for the root of the tree and the gray disks represent nodes in the set $\mathcal{V}_T$. Black arrows represent transitions captured by $\mathcal{E}_T$. The blue diamond stands for the state q_P^{new} and the dashed blue arrow represents the new edge that will be added to the set $\mathcal{E}_T$ after the execution of Algorithm 3 (line 5, Alg. 3).

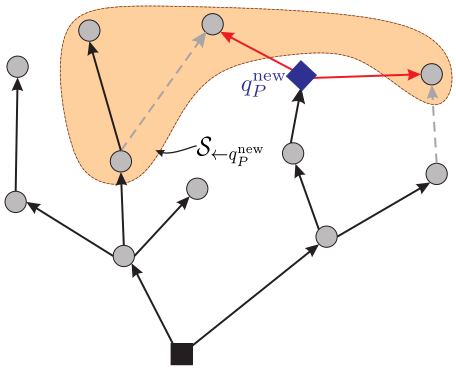


Figure 3: Graphical depiction of Algorithm 4. The black square stands for the root of the tree and the gray disks and the blue diamond represent nodes in the set $\mathcal{V}_T$. Black arrows represent transitions captured by $\mathcal{E}_T$. The blue diamond stands for the state q_P^{new}. Dashed gray arrows stand for the edges that will be deleted from the set $\mathcal{E}_T$ during the execution of Algorithm 4 (line 4, Alg. 4). Red arrows stand for the new edges that will be added to $\mathcal{E}_T$ during the execution of Algorithm 4 (line 5, Alg. 4).

as follows [line 6, Alg. 3]:

$$C_T(q_P^{\text{new}}) = \underbrace{C_T(q_P^{\text{prev}})}_{\text{Cost of reaching } q_P^{\text{prev}} \text{ from the root of the tree } \mathcal{G}_T} + \underbrace{w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P^{\text{prev}}, \Pi|_{\text{PTS}} q_P^{\text{new}})}_{\text{cost of reaching } q_P^{\text{new}} \text{ from } q_P^{\text{prev}}}. \quad (3)$$

Algorithm 3 is illustrated in Figure 2, as well.

4.1.3 Rewiring. Once a new state $q_P^{\text{new}} = [q_{\text{PTS}}^{\text{new}}, q_B^{\text{new}}]$ has been added to the tree [line 12, Alg. 1], two steps follow. First, we check if q_P^{new} is an accepting state of the PBA P. To achieve that, it suffices to check if $q_B^{\text{new}} \in Q_B^F$. If this is the case, the set $\mathcal{F}$ that collects the final states that exist in the tree is updated [lines 13-15, Alg. 1]. Second, we *rewire* the nodes in $q_P \in \mathcal{V}_T$ that can potentially get connected to the root q_P^0 of the tree through the node q_P^{new} [line 13, Alg. 1] if this will decrease the cost $C_T(q_P)$. The rewiring process is described in Algorithm 4 and is illustrated in Figure 3.

In Algorithm 4 we first construct the set $\mathcal{S}_{\leftarrow q_P^{\text{new}}} \subseteq \mathcal{V}_T$ that collects all states of $q_P \in \mathcal{V}_T$ that abide by the following transition rule: $(q_P^{\text{new}}, q_P) \in \rightarrow_P$[line 1, Alg. 4]. Then for all states $q_P \in \mathcal{S}_{\leftarrow q_P^{\text{new}}}$ we check if their current cost $C_T(q_P)$ is greater than then the cost they would have if they were connected to the root through q_P^{new} 3. If this is the case for a node $q_P \in \mathcal{S}_{\leftarrow q_P^{\text{new}}}$, then the new parent of q_P is q_P^{new}, i.e, there is a directed edge from q_P^{new} to q_P, and the edge that was connecting q_P to its previous parent is discarded [lines 4-5, Alg. 4]. The cost of node q_P is updated as $C_T(q_P) = C_T(q_P^{\text{new}}) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P^{\text{new}}, \Pi|_{\text{PTS}} q_P)$ to take into account the new path through which it gets connected to the root [line 6, Alg. 4].

4.1.4 Optimizing $\mathcal{G}_T$. The above procedure terminates after $n_{\max}^{\text{pre}}$ iterations. Once this happens, we modify the resulting set of edges $\mathcal{E}_T$, as per Algorithm 5 so that the cost of each node in $\mathcal{V}_T$ is minimized [line 16, Alg. 1]. To achieve that, it suffices to rewire all nodes in the graph $\mathcal{G}_T$ [lines 2-4, Alg. 5]. After rewiring all nodes, a new set of edges is constructed denoted by $\mathcal{E}_T^k$, where $k = 2, 3, \ldots$ and $\mathcal{E}_T^1 := \mathcal{E}_T$. This rewiring process is repeated until the set of edges stops changing, i.e, until $\mathcal{E}_T^k = \mathcal{E}_T^{k-1}$ [lines 5-8, Alg. 5]. In this way, we minimize the cost of all nodes and, consequently, of nodes that represent final states, as well, since $\mathcal{F} \subset \mathcal{V}_T$. Moreover, notice that Algorithm 5 will terminate after a finite number of iterations, since the set $\mathcal{V}_T$ is finite and there is a finite number of possible transitions among these nodes captured by the transition rule $\longrightarrow_P$.

4.1.5 Construction of Paths. After optimizing the tree structure, we compute the path that connects each final state that belongs to the set $\mathcal{F}$ to the root of the tree q_P^0 [line 17-18, Alg. 1]. The path that connects the f-th final state of the set $\mathcal{F}$ to the root is denoted by $\tau^{\text{pre},f}$ and is computed by Algorithm 6. Note that for the computation of $\tau^{\text{pre},f}$ in Algorithm 6 only the parent of each node in the graph $\mathcal{G}_T$ is required, due to the tree structure of $\mathcal{G}_T$. Specifically, the prefix part $\tau^{\text{pre},f}$ is constructed by finding the parent of a node $q_P \in \mathcal{V}_T$ starting from the node that represents the final state $\mathcal{F}(f)$, until the root of the tree is reached [lines 1-7, Alg. 6]. The parent of each node is computed by the function $\texttt{parent} : \mathcal{V}_T \rightarrow \mathcal{V}_T$ 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 $\texttt{parent}(q_P^0) = q_P^0$, where q_P^0 is the root of the tree $\mathcal{G}_T$. In line 7, $\Pi|_{\text{PTS}} p_T$ stands for the projection of the path p_T onto the state space of PTS. Moreover, in line 4 of Algorithm 6, we assume that the last node introduced in the path p_T is placed at the first entry of p_T. Thus, for the resulting prefix part $\tau^{\text{pre},f}$, it holds that $\tau^{\text{pre},f}(1) = \Pi_{\text{PTS}} q_P^0$ and $\tau^{\text{pre},f}(|\tau^{\text{pre},f}|) = \Pi_{\text{PTS}} \mathcal{F}(f)$.

Notice that the computational complexity of constructing the prefix part $\tau^{\text{pre},f}$ is $O(|\mathcal{V}_T|)$. On the other hand, if the PBA was represented by a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ of arbitrary structure, then the

Algorithm 2: Function Sample($Q_1, \ldots, Q_N$)

$q_{\text{PTS}}^{\text{new}} = [\,]$;
for $i = 1:1:N$ **do**
  Pick a state q_i from the set Q_i according to probability distribution f;
  $q_{\text{PTS}}^{\text{new}} = [q_{\text{PTS}}^{\text{new}}, q_i]$;
return q_{PTS}^{new};

Algorithm 3: Function Extend(q_P^{new})

Collect in set $\mathcal{S}_{\rightarrow q_P^{\text{new}}}$ all states $q_P \in \mathcal{V}_T$ that abide by the following transition rule: $(q_P, q_P^{\text{new}}) \in \rightarrow_P$;
if $\mathcal{S}_{\rightarrow q_P^{new}} \neq \emptyset$ **then**
  $q_P^{\text{prev}} = \text{argmin}_{q_P \in \mathcal{S}_{\rightarrow q_P^{\text{new}}}} [C_T(q_P) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P, \Pi|_{\text{PTS}} q_P^{\text{new}})]$;
  $\mathcal{V}_T = \mathcal{V}_T \cup \{q_P^{\text{new}}\}$;
  $\mathcal{E}_T = \mathcal{E}_T \cup \{(q_P^{\text{prev}}, q_P^{\text{new}})\}$;
  $C_T(q_P^{\text{new}}) = C_T(q_P) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P, \Pi|_{\text{PTS}} q_P^{\text{new}})$;
return $\mathcal{V}_T, \mathcal{E}_T, C_T$;

Algorithm 4: Function Rewire($q_P^{\text{new}}, \mathcal{V}_T, \mathcal{E}_T, C_T$)

Collect in set $\mathcal{S}_{\leftarrow q_P^{\text{new}}}$ all states of $q_P \in \mathcal{V}_T$ that abide by the following transition rule: $(q_P^{\text{new}}, q_P) \in \rightarrow_P$;
for $q_P \in \mathcal{S}_{\leftarrow q_P^{new}}$ **do**
  if $C_T(q_P) > C_T(q_P^{new}) + w_{PTS}(\Pi|_{PTS} q_P^{\text{new}}, \Pi|_{PTS} q_P)$ **then**
    $\mathcal{E}_T = \mathcal{E}_T \setminus \{(\text{Parent}(q_P), q_P)\}$;
    $\mathcal{E}_T = \mathcal{E}_T \cup \{(q_P^{\text{new}}, q_P)\}$;
    $C_T(q_P) = C_T(q_P^{\text{new}}) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P^{\text{new}}, \Pi|_{\text{PTS}} q_P)$;
return $\mathcal{E}_T, C_T$;

Algorithm 5: Function OptimizeTree($\mathcal{G}_T$)

$\mathcal{E}_T^1 = \mathcal{E}_T$;
for $q_P \in \mathcal{V}_T$ **do**
  $[\mathcal{E}_T^2, C_T] = \text{Rewire}(\mathcal{V}_T, \mathcal{E}_T^1, C_T)$;
$k = 2$;
while $\mathcal{E}_T^k \neq \mathcal{E}_T^{k-1}$ **do**
  for $q_P \in \mathcal{V}_T$ **do**
    $[\mathcal{E}_T^{k+1}, C_T] = \text{Rewire}(q_P, \mathcal{V}_T, \mathcal{E}_T^k, C_T)$;
  $k = k + 1$;
return $\mathcal{E}_T = \mathcal{E}_T^k$;

computational complexity of the Dijkstra algorithm to find the path $\tau^{\text{pre},f}$ that connects the state $\mathcal{F}(f)$ to the root with the minimum cost is $O(|\mathcal{E}| + |\mathcal{V}| \log(|\mathcal{V}|))$, where $|\mathcal{E}| + |\mathcal{V}| \log(|\mathcal{V}|) > |\mathcal{V}|$.

4.2 Construction of Suffix Parts

The construction of the suffix parts is presented in Algorithm 7. The goal of this Algorithm is to find a sequence of states, denoted by $\tau_i^{\text{suf},f}$, in Q_P that starts from state $\mathcal{F}(f)$ and ends at the same state $\mathcal{F}(f)$, i.e., a cycle around state $\mathcal{F}(f)$, where any two consecutive states in $\tau_i^{\text{suf},f}$ respect the transition rule $\rightarrow_P$, for all $f = \{1, \ldots, |\mathcal{F}|\}$. For this purpose, we build a tree $\mathcal{G}_T = \{\mathcal{V}_T, \mathcal{E}_T, C_T\}$ that approximates the PBA P, in a similar way as in Section 4.1 which combined with a cycle-detection method results in construction of $\tau_i^{\text{suf},f}$.

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

$p_T = \{q_P^{\text{goal}}\}$;
$q_P^{\text{prev}} = \text{Parent}(q_P^{\text{goal}})$;
while $q_P^{prev} \neq q_P^{initial}$ **do**
  $p_T = p_T \cup \{q_P^{\text{prev}}\}$;
  $q_P^{\text{prev}} = \text{Parent}(q_P^{\text{prev}})$;
$p_T = p_T \cup \{q_P^{\text{initial}}\}$;
$p_T = \Pi|_{\text{PTS}} p_T$;
return p_T;

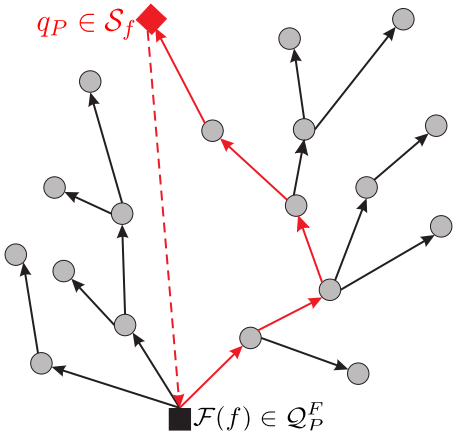


Figure 4: Graphical depiction of detecting cycles around a final state $\mathcal{F}(f)$ (black square) which acts as the root of the tree. The red diamond stands for a state $q_P \in \mathcal{S}_f$. Solid red arrows stand for the path that connects the state $q_P \in \mathcal{S}_f$ to the root $\mathcal{F}(f)$. The dashed red arrow implies that a transition from q_P to $\mathcal{F}(f)$ is feasible according to the transition rule $\rightarrow_P$; however, such a transition is not included in the set $\mathcal{E}_T$. The cycle around the final state $\mathcal{F}(f)$ is illustrated by solid and dashed red arrows.

In Algorithm 7, for a given final state $\mathcal{F}(f)$, which comes from the execution of Algorithm 1, the tree initially consists only of the state $\mathcal{F}(f)$, which by convention has zero cost [lines 2-4, Alg. 7]. Also, given a final state $\mathcal{F}(f)$, we define the set $\mathcal{S}_f = \{q_P \in \mathcal{V}_T | (q_P, \mathcal{F}(f)) \in \rightarrow_P\}$ that contains all nodes that currently exist in the set $\mathcal{V}_T$ from which a transition to the state $\mathcal{F}(f)$ is feasible. This set is initialized as $\mathcal{S}_f = \emptyset$ [line 5, Alg. 7]. Once the tree is initialized, we check if we can add the final state $\mathcal{F}(f)$ to the set $\mathcal{S}_f$ [line 6, Alg. 7]. If this is the case, then we update the set $\mathcal{S}_f$ accordingly [line 19, Alg. 7] and we terminate the construction of

the tree. Otherwise, we continue building the tree $\mathcal{G}_T$ [lines 7-17, Alg. 7], exactly as we did for the construction of the prefix part in Algorithm 1. When a new node q_P^{new} is added to the set $\mathcal{V}_T$, we check if a transition from q_P^{new} to $\mathcal{F}(f)$ is feasible according to the transition rule $\rightarrow_P$ [line 15, Alg. 7]. If so, we add the state q_P^{new} to the set $\mathcal{S}_f$. In other words, a cycle around a final state $\mathcal{F}(f)$ can be constructed by computing the path in the graph $\mathcal{G}_T$ that starts from the root $\mathcal{F}(f)$ of the tree, ends at a state $q_P \in \mathcal{V}_T \cap \mathcal{S}_f$ followed by the state $\mathcal{F}(f)$; see Figure 4. In general, since there may be more than one states in $\mathcal{S}_f$, there will be multiple possible cycles around a final state $\mathcal{F}(f)$; specifically, there will be $|\mathcal{S}_f|$ cycles around $\mathcal{F}(f)$.

Next, we optimize the structure of the tree by modifying the set of edges $\mathcal{E}_T$, as we did in Algorithm 1 [line 20, Alg. 7]. Given the optimized tree structure, among all detected cycles around $\mathcal{F}(f)$, we pick the optimal cycle for all $f \in \{1, \ldots, |\mathcal{F}|\}$ [lines 22-26, Alg. 7]. For this purpose, given a final state $\mathcal{F}(f)$, if $\mathcal{S}_f \neq \emptyset$, first we find all possible cycles around the state $\mathcal{F}(f)$ projected onto the state-space of the PTS [lines 22-24, Alg. 7]. Among them we pick the one that has the minimum cost, in terms of the cost function (1) [lines 25-26, Alg. 7]. The resulting cycle around state $\mathcal{F}(f)$ is denoted by $\tau^{\text{suf},f}$.

4.3 Construction of Optimal Discrete Plan

By construction, any motion plan $\tau^f = \tau^{\text{pre},f}[\tau^{\text{suf},f}]^\omega$, with $\mathcal{S}_f \neq \emptyset$, $f \in \{1, \ldots, |\mathcal{F}|\}$ satisfies the global LTL specification ϕ. The cost $J(\tau^f)$ of each plan τ^f is defined in (2). Among all the motion plans $\tau_f \models \phi$, our proposed method returns the solution of Problem 1 with the smallest cost $J(\tau^f)$ denoted by τ, i.e., $\tau = \tau^{f_*}$, where $f_* = \text{argmin}_f J(\tau^f)$.

5 CORRECTNESS AND OPTIMALITY

In this section we provide results pertaining to probabilistic completeness and optimality of the proposed algorithm. First, some preliminary notations are given, followed by the completeness property of the proposed algorithm. Let $\mathcal{G}_T^n = \{\mathcal{V}_T^n, \mathcal{E}_T^n, C_T^n\}$ denote the tree that has been constructed by either Algorithm 1 or Algorithm 7 at the n-th iteration. Also, let $\mathcal{X}_{\text{goal}} \subset \mathcal{Q}_P$ denote the *goal region*. Specifically, for Algorithms 1 and 7, the goal region is defined as $\mathcal{X}_{\text{goal}} = \{q_P \in \mathcal{Q}_P | q_P \in \mathcal{Q}_P^F\}$ and $\mathcal{X}_{\text{goal}} = \{q_P \in \mathcal{Q}_P | (q_P, \mathcal{F}(f)) \in \rightarrow_P, \text{for all} f \in \{1, 2, \ldots, |\mathcal{F}|\}\}$, respectively.[4]

Theorem 5.1 (Probabilistic Completeness). *If there exists a solution for Problem 1, then the proposed algorithm in Section 4.3 is probabilistically complete, i.e., it will find with probability 1 a motion plan τ that satisfies the LTL specification ϕ.*

Proof. The proof can be found in Appendix A. □

Next, we examine the optimality of the resulting motion plan τ constructed in Section 4.3.

Theorem 5.2 (Asymptotic Optimality). *The algorithm described in Section 4.3 to construct a motion plan that satisfies a given global LTL specification ϕ is asymptotically optimal, i.e., the discrete motion plan $\tau^{n_{max}^{suf}}_{n_{max}^{pre}}$ that is generated by this algorithm satisfies*

$$\mathbb{P}\left(\left\{\lim_{n_{max}^{pre}\rightarrow\infty, n_{max}^{suf}\rightarrow\infty} J(\tau^{n_{max}^{suf}}_{n_{max}^{pre}}) = J^*\right\}\right) = 1, \tag{4}$$

where J is a cost function, J^* is the optimal cost, and n_{max}^{pre} and n_{max}^{suf} are the maximum number of iterations for Algorithms 1 and 7, respectively.

Proof. The proof can be found in Appendix B. □

Using Theorem 5.2, we have the following result.

Corollary 5.3. *Given the trees constructed by Algorithms 1 and 7 within n_{max}^{pre} and n_{max}^{suf} iterations, respectively, the proposed algorithm will find the best possible plan τ in terms of the cost function* (2).

Algorithm 7: Construction of suffix parts.

Input: Set of final states $\mathcal{F}$, transition system wTS_i, for all robots i, NBA B, maximum number of iterations $n_{\max}^{\text{suf}}$
Output: Suffix Parts $\tau^{\text{suf},f}$

1 **for** $f = 1:1:|\mathcal{F}|$ **do**
2 $\mathcal{V}_T = \mathcal{F}(f)$;
3 $\mathcal{E}_T = \emptyset$;
4 $C_T(\mathcal{F}(f)) = 0$;
5 $\mathcal{S}_f = \emptyset$;
6 **if** $(\mathcal{F}(f), \mathcal{F}(f)) \notin \rightarrow_P$ **then**
7 **for** $n = 1:1:n_{max}^{suf}$ **do**
8 $q_{\text{PTS}}^{\text{new}} = \texttt{Sample}(Q_1, \ldots, Q_N)$;
9 **for** $b = 1:|\mathcal{Q}_B|$ **do**
10 $q_B^{\text{new}} = \mathcal{Q}_B(b)$;
11 $q_P^{\text{new}} = (q_{\text{PTS}}^{\text{new}}, q_B^{\text{new}})$;
12 **if** $q_P^{new} \notin \mathcal{V}_T$ **then**
13 $[\mathcal{V}_T, \mathcal{E}_T, C_T] = \texttt{Extend}(q_P^{\text{new}}, \rightarrow_P)$;
14 **if** $q_P^{new} \in \mathcal{V}_T$ **then**
15 **if** $(q_P^{new}, \mathcal{F}(f)) \in \rightarrow_P$ **then**
16 $\mathcal{S}_f = \mathcal{S}_f \cup \{q_P^{\text{new}}\}$;
17 $[\mathcal{E}_T, C_T] =$ $\texttt{Rewire}(q_P^{\text{new}}, \mathcal{V}_T, \mathcal{E}_T, C_T)$;
18 **else**
19 $\mathcal{S}_f = \{\mathcal{F}(f)\}$;
20 $\mathcal{E}_T = \texttt{OptimizeTree}(\mathcal{G}_T)$;
21 **for** $f = 1:1:|\mathcal{F}|$ **do**
22 **if** $\mathcal{S}_f \neq \emptyset$ **then**
23 **for** $e = 1:|\mathcal{S}_f|$ **do**
24 $\tilde{\tau}^{\text{suf},e} = \texttt{FindPath}(\mathcal{G}_T, \mathcal{S}_f(e), \mathcal{F}(f))$;
25 $e^* = \text{argmin}_e J(\tilde{\tau}^{\text{suf},e})$;
26 $\tau^{\text{suf},f} = \tilde{\tau}^{\text{suf},e^*}$;

6 SIMULATION RESULTS

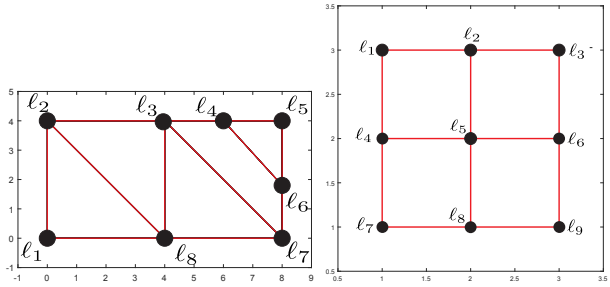
In this section, we present two case studies, implemented using MATLAB R2015b, that illustrate the efficiency and scalability of the proposed algorithm. The first case study pertains to a motion

[4] Recall that during the execution of Algorithms 1 and 7, the states that belong to the set of nodes $\mathcal{V}_T$ and to the goal region $\mathcal{X}_{\text{goal}}$ are collected in the sets $\mathcal{F}$ and $\mathcal{S}_f$, respectively.

planning problem with a PBA that has 1,073,741,824 states. Recall that the state-space of the PBA defined in Definition 3.3 has $\Pi_{i=1}^{N}|Q_i||Q_B|$ states. This problem cannot be solved by either the off-the-shelf model checker PRISM or the work in [10]. Specifically, our implementation of [10] failed to provide a motion plan for the considered case study due to the large state-space of the resulting PBA. A direct comparison with [18] cannot be made, since in that work samples for the robot positions are drawn from the continuous space, which is not the case here. Note, however, that in [18], the size of the regions that observe the atomic propositions affects the number of samples required for the construction of an expressive enough transition system that can generate a motion plan. Therefore, for small regions, more samples are needed and the state space of the resulting PBA may be too large to manipulate in practice. This issue becomes more pronounced, as the size of the NBA increases. On the other hand, our algorithm scales well to very large problems since it avoids the construction of the PBA and the application of graph search methods to find optimal motion plans altogether. The proposed tree-based approximation of the PBA requires much fewer resources to store and motion plans can be found easily by tracing the sequence of parent nodes from the leaves back to the root. In the second case study, we consider a motion planning problem with a PBA that has 5,103 states. This state-space is small enough to manipulate and construct an optimal plan using the standard method described in Section 3. In this simulation study, we examine the performance of the proposed algorithm in terms of runtime and optimality.

6.1 Case Study I

In the first simulation study, we consider a network of $N = 9$ robots residing in a workspace with $W = 8$ regions of interest. Mobility of each robot in this workspace is captured by a transition system which has $|Q_i| = 8$ states, as shown in Figure 5(a). The collaborative task that is assigned to the robots describes an intermittent connectivity problem, that was defined in our previous work [9]. In this problem setting, robots move along the edges of a mobility graph and communicate only when they meet at the vertices of this graph, giving rise to a dynamic communication network. This communication network is intermittently connected if communication occurs at the vertices of the mobility graph infinitely often. Such an intermittent connectivity requirement can be captured by a global LTL formula. In particular, in this simulation study we consider the following global LTL specification: $\phi = [\Box\Diamond(\pi_1^{\ell_5} \wedge \pi_2^{\ell_5})] \wedge [\Box\Diamond(\pi_2^{\ell_1} \wedge \pi_3^{\ell_1} \wedge \pi_4^{\ell_1})] \wedge [\Box\Diamond(\pi_4^{\ell_7} \wedge \pi_5^{\ell_7} \wedge \pi_6^{\ell_7})] \wedge [\Box\Diamond(\pi_6^{\ell_8} \wedge \pi_7^{\ell_8})] \wedge [\Box\Diamond(\pi_7^{\ell_4} \wedge \pi_8^{\ell_4})] \wedge [\Box\Diamond(\pi_8^{\ell_3} \wedge \pi_9^{\ell_3})] \wedge [\neg(\pi_1^{\ell_5} \wedge \pi_2^{\ell_5})\mathcal{U}\pi_1^{\ell_7}]$. In this LTL formula, $\Box$, $\Diamond$, and $\mathcal{U}$ stand for the temporal operators 'always', 'eventually', and 'until' respectively, and $\wedge$ and $\neg$ represent the Boolean conjunction and negation operator. In words, the considered LTL-based task in requires (a) robots 1 and 2 to meet at location ℓ_5 infinitely often, (b) robots 2, 3 and 4 to meet at location ℓ_7, infinitely often, (c) robots 4, 5, and 6 to meet at location ℓ_7, infinitely often, (d) robots 6 and 7 to meet at location ℓ_8 infinitely often, and (e) robots 7 and 8 to meet at location ℓ_4, infinitely often, (f) robots 8 and 9 to meet at location ℓ_3, infinitely often, and (g) robots 1 and 2 to never meet at location ℓ_5 until robot 1 visits location ℓ_7 to collect some available information. The considered



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

Figure 5: Graphical depiction of the transition systems wTS_i, for all robots i used in simulation study I (Figure 5(a)) and II (Figure 5(b)). Black disks represent the states of wTS_i and red edges stand for feasible transitions among the states.

LTL formula corresponds to a NBA with $|Q_B| = 8$ states.[5] Notice that the resulting NBA has only one final state, i.e., the PBA has $|Q_{\text{PTS}}| \times 1 = 8^9 = 134,217,728$ final states (either reachable or not).

Algorithms 1 and 7 were both run until a final state and cycle around it are detected, respectively. Algorithm 1 run for 14482 iterations and a tree graph $\mathcal{G}_T$ with $|\mathcal{V}_T| = 38072$ nodes was constructed, where one of these nodes corresponded to a final state, i.e., $|\mathcal{F}| = 1$. For the storage of this tree structure 1.59 Mb were utilized. Next, we optimized the tree graph as per Algorithm 5 and then the prefix part $\tau^{\text{pre},1}$ was constructed. Notice in Figure 6 that the cost of the prefix part $\tau^{\text{pre},f}$, or equivalently, the cost $C_T(\mathcal{F}(1))$ is non-increasing over iterations k of Algorithm 5, as expected by construction of this algorithm and by Corollary 5.3. Figure 7(a) depicts the number of rejected states with respect to iterations n of Algorithm 1.

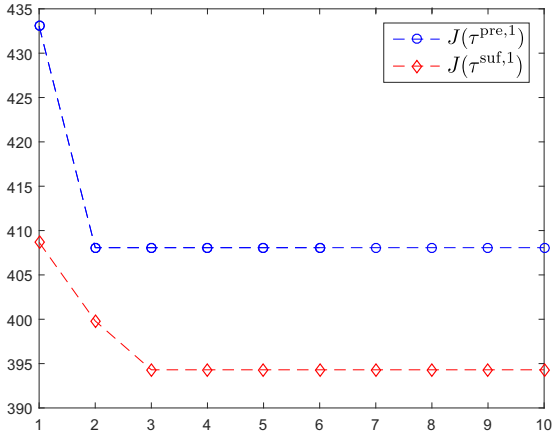
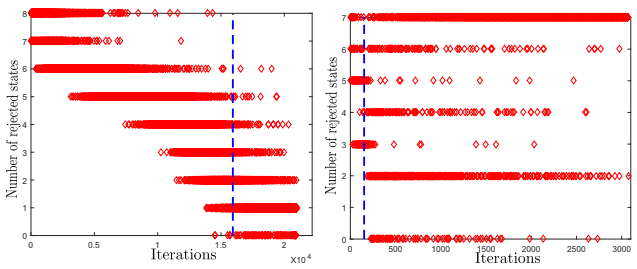


Figure 6: Simulation Study I: Evolution of the costs $J(\tau^{\text{pre},1})$ and $J(\tau^{\text{suf},1})$ over iterations κ of Algorithm 5 that optimizes the resulting tree structures.

[5] The translation of the LTL formula to a NBA was made by the tool developed in [6].



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

Figure 7: Graphical depiction of the number of rejected states per iteration after running Algorithm 1 for 21000 and 3072 iterations for simulation studies I and II. The resulting trees have 80124 and 3503 nodes, respectively. Red diamonds represent the number of rejected states at each iteration. Blue dashed line shows the first iteration at which a final state was detected. At iteration n of Algorithm 1, a state sampled from Q_{PTS} is taken. Given this state, $|Q_B|$ states that belong to Q_P are created. Consequently, at iteration n at most $|Q_B|$ states can be rejected or accepted.

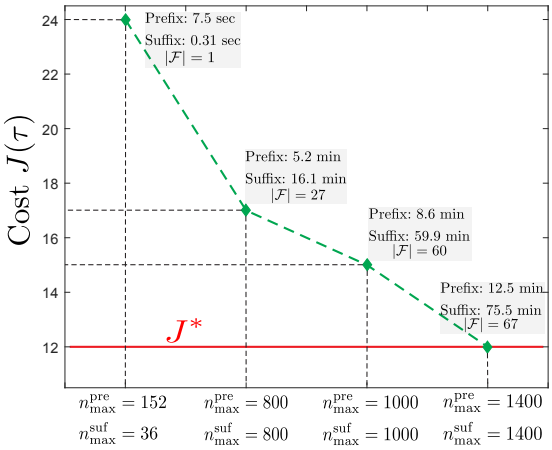


Figure 8: Simulation Study II: Evolution of the cost $J(\tau)$ of the resulting optimal motion plan τ for various maximum numbers of iterations for Algorithms 1 and 7. The time required for the construction of the optimal prefix and suffix part along with the number of detected final states for each case are included in the gray colored box. The red line stands for the optimal cost $J^* = 12$.

Next the construction of the suffix part follows. Algorithm 7 ran for 15938 iterations and constructed a tree graph with $|\mathcal{V}_T| = 37419$ nodes, with $|\mathcal{S}_f| = 1$. For the storage of this tree structure 1.49 Mb were utilized. After optimizing the tree graph, the suffix part $\tau^{\text{suf},1}$ was designed. In Figure 6 the evolution of the cost $J(\tau^{\text{suf},1})$ during the execution of Algorithm 5 is depicted, which decreases as predicted by Corollary 5.3. The total cost of the resulting motion plan $\tau^1 = \tau^{\text{pre},1}[\tau^{\text{suf},1}]^\omega$ that satisfies the considered LTL task is $J(\tau^1) = J(\tau^{\text{pre},1}) + J(\tau^{\text{suf},1}) = 408.0712 + 394.3007 = 802.3719$ meters.

Algorithm 1 detected a final state within 2.5 hours [lines 1-15, Alg. 1], the resulting tree structure was optimized within 7 hours [line 16, Alg. 1], and then the optimal prefix $\tau^{\text{pre},1}$ was constructed in 0.017 seconds. Similar runtimes were observed for the construction of the respective suffix part $\tau^{\text{suf},1}$. Notice that the off-the-shelf model checker PRISM could not generate a motion plan for this simulation scenario. Specifically, PRISM could generate a motion plan only when we considered 6 robots and a smaller part of the considered LTL formula, which was $\bar{\phi} = [\Box\Diamond(\pi_1^{\ell_5} \wedge \pi_2^{\ell_5})] \wedge [\Box\Diamond(\pi_2^{\ell_1} \wedge \pi_3^{\ell_1} \wedge \pi_4^{\ell_1})] \wedge [\Box\Diamond(\pi_4^{\ell_7} \wedge \pi_5^{\ell_7} \wedge \pi_6^{\ell_7})] \wedge [\Box\Diamond(\pi_6^{\ell_8} \wedge \pi_7^{\ell_8})]$. In this case, the size the state-space of the PBA was $|Q_P| = 10,485,760$. On the other hand, notice that in the latter case, PRISM finished the model-checking process faster than our proposed algorithm, and specifically, in 1.5 minutes.

6.2 Case Study II

In the second simulation study, we consider a network of $N = 3$ robots, where mobility of each robot is captured by the transition system depicted in Figure 5(b), which has $|Q_i| = 9$ states, for all robots i. The assigned task is expressed in the following temporal logic formula: $\phi = \Box\Diamond(\pi_1^{\ell_6} \wedge \pi_2^{\ell_4}) \wedge \neg(\pi_1^{\ell_7}) \wedge (\neg p_2^{\ell_4} \mathcal{U} p_3^{\ell_4}) \wedge (\Diamond\pi_3^{\ell_7}) \wedge (\Box\Diamond\pi_2^{\ell_2})$ where the respective NBA has $|Q_B| = 7$ states and one of them is a final state. In words, this LTL-based task requires (a) robots 1 and 2 to visit locations ℓ_6 and ℓ_4, respectively, simultaneously and infinitely often, (b) robot 1 to always avoid location ℓ_7, (c) robot 2 to avoid location ℓ_4 until robot 3 visits location ℓ_4, (d) robot 3 to visit location ℓ_7 eventually, and (e) robot 2 to visit location ℓ_2 infinitely often. In this simulation study, the state space of the PBA consists of $\Pi_{i=1}^N |Q_i||Q_B| = 5,103$ states which is small enough in order to compute the optimal plan, using the method described in Section 3. The cost of the optimal plan that satisfies the considered LTL formula is $J^* = 12$ meters.

Initially, Algorithms 1 and 7 were run until a final state and a cycle around it are detected, respectively. Algorithm 1 found a final state after 152 iterations corresponding to 7.5 seconds and Algorithm 7 detected a cycle around this final state after 36 iterations corresponding to 0.31 seconds. PRISM verified that there exists a motion plan that satisfies the considered LTL formula in 2.1 seconds. The cost of the resulting plan τ_1 is $J(\tau^1) = J(\tau^{\text{pre},1}) + J(\tau^{\text{suf},1}) = 12 + 12 = 24$ meters. Observe in Figure 8 that as we increase the number of iterations that Algorithms 1 and 7 run, the cost of the resulting plans decreases, as expected due to Theorem 5.2. The number of detected final states and runtime for each case are also depicted in the same figure. Figure 7(b) depicts the number of rejected states with respect to the iterations n of Algorithm 1.

7 CONCLUSION

In this paper we proposed a sampling-based control synthesis algorithm for multi-robot systems under global linear temporal logic (LTL) formulas. Robot mobility in the workspace was captured by transition systems whose states represented regions of the environment that satisfy atomic propositions. Existing approaches rely on graph search methods applied to a synchronous product automaton constructed among all agents, which are intractable and scale

poorly with the number of 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. Simulation studies showed that the proposed approach can be used to model-check product automata with billions of states, which was impossible using currently available methods. Finally, although the proposed sampling-based model checking algorithm was presented for robotic path planning problems, it can be employed for any LTL-based control synthesis problem, as e.g., in traffic network control [15].

REFERENCES

[1] Abdulla, P., Aronis, S., Jonsson, B., and Sagonas, K. Optimal dynamic partial order reduction. In *ACM SIGPLAN Notices* (2014), vol. 49, pp. 373–384.

[2] Baier, C., and Katoen, J.-P. *Principles of model checking*, vol. 26202649. MIT press Cambridge, 2008.

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

[4] Chen, Y., Ding, X. C., and Belta, C. 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.

[5] Chu, D.-H., and Jaffar, J. A complete method for symmetry reduction in safety verification. In *International Conference on Computer Aided Verification* (Berkeley, CA, USA, June 2012), pp. 616–633.

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

[7] Guo, M., and Dimarogonas, D. V. Reconfiguration in motion planning of single- and multi-agent systems under infeasible local LTL specifications. In *IEEE 52nd Annual Conference on Decision and Control (CDC)* (Florence, December 2013), pp. 2758–2763.

[8] Guo, M., Johansson, K. H., and Dimarogonas, D. V. 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.

[9] Kantaros, Y., and Zavlanos, M. M. Distributed intermittent connectivity control of mobile robot networks. *IEEE Transactions on Automatic Control* (2016).

[10] Kantaros, Y., and Zavlanos, M. M. 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.

[11] Karaman, S., and Frazzoli, E. Sampling-based motion planning with deterministic μ-calculus specifications. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on* (2009), IEEE, pp. 2222–2229.

[12] Karaman, S., and Frazzoli, E. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research 30*, 7 (2011), 846–894.

[13] Kress-Gazit, H., Fainekos, G. E., and Pappas, G. J. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics 25*, 6 (2009), 1370–1381.

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

[15] Sadraddini, S., and Belta, C. Model predictive control of urban traffic networks with temporal logic constraints. In *American Control Conference (ACC)* (Boston, MA, USA, July 2016), pp. 881–881.

[16] Ulusoy, A., Smith, S. L., Ding, X. C., Belta, C., and Rus, D. Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research 32*, 8 (2013), 889–911.

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

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

[19] Wu, L., Huang, H., Su, K., Cai, S., and Zhang, X. An i/o efficient model checking algorithm for large-scale systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 23*, 5 (2015), 905–915.

A PROOF OF THEOREM 5.1

To show this result, it suffices to show that for Algorithms 1 and 7 it holds that

$$\lim_{n\to\infty} \mathbb{P}\left(\{\mathcal{V}_T^n \cap \mathcal{X}_{\text{goal}} \neq \emptyset\}\right) = 1, \tag{5}$$

where $\mathbb{P}(\cdot)$ stands for the probability of an event. Let $\overline{\mathcal{S}}^n$ be a set that collects all states of P that do not belong to the set of nodes $\mathcal{V}_T^n$, i.e., $\overline{\mathcal{S}}^n = Q_P \setminus \mathcal{V}_T^n$. Let $\mathbb{P}^n(q_P^{\text{new}})$ denote the probability that the state $q_P^{\text{new}} \in \overline{\mathcal{S}}^n$ will be the next sample, which can equivalently be written as:

$$\mathbb{P}^n(q_P^{\text{new}}) = 1 - \overline{\mathbb{P}}^n(q_P^{\text{new}}), \tag{6}$$

where $\overline{\mathbb{P}}^n(q_P^{\text{new}})$ denotes the probability that the state $q_P^{\text{new}} \in \overline{\mathcal{S}}^n$ will not be the next sample. The probability $\overline{\mathbb{P}}^n(q_P^{\text{new}})$ can be written in the following equivalent form:

$$\overline{\mathbb{P}}^n(q_P^{\text{new}}) = \sum_{q_P \in \overline{\mathcal{S}}^n \setminus \{q_P^{\text{new}}\}} \mathbb{P}^n(q_P). \tag{7}$$

Combining equations (6) and (7), we get

$$\mathbb{P}^n(q_P^{\text{new}}) = 1 - \sum_{q_P \in \overline{\mathcal{S}}^n \setminus \{q_P^{\text{new}}\}} \mathbb{P}^n(q_P), \tag{8}$$

Notice that as more samples are taken, i.e., as $n \to \infty$, the cardinality of the set $\overline{\mathcal{S}}^n \setminus \{q_P^{\text{new}}\}$ goes to zero, which implies that

$$\lim_{n\to\infty} \sum_{q_P \in \overline{\mathcal{S}}^n \setminus \{q_P^{\text{new}}\}} \mathbb{P}^n(q_P) = 0. \tag{9}$$

Combining equations (8) and (9), we conclude that for the sequence $\{\mathbb{P}^n(q_P^{\text{new}})\}_{n=1}^{\infty}$ it holds that $\lim_{n\to\infty} \mathbb{P}^n(q_P^{\text{new}}) = 1$, which implies that any state $q_P^{\text{new}} \in \overline{\mathcal{S}}^n$ will eventually be sampled with probability 1, as $n \to \infty$. Moreover, since the cardinality of the set $\overline{\mathcal{S}}^n \setminus \{q_P^{\text{new}}\}$ goes to zero as $n \to \infty$, this implies that the set $\mathcal{S}_{\to q_P^{\text{new}}}$ defined in Section 4.1.2 will not be empty as $n \to \infty$, if the state q_P^{new} is reachable. [6] This means that as $n \to \infty$ any state $q_P^{\text{new}} \in \overline{\mathcal{S}}^n$ will eventually be sampled and added to the set $\mathcal{V}_T^{n+1}$. Since this result holds for any reachable state in Q_P, it holds for any reachable state state in $\mathcal{X}_{\text{goal}} \subset Q_P$, as well. Recall that since we assume that Problem 1 has a solution, then there are reachable states in $\mathcal{X}_{\text{goal}}$ for Algorithms 1 and 7. Consequently, we have that $\lim_{n\to\infty} \mathbb{P}(\{\mathcal{V}_T^n \cap \mathcal{X}_{\text{goal}} \neq \emptyset\}) = 1$, for Algorithms 1 and 7, which completes the proof.

B PROOF OF THEOREM 5.2

To show that (4) holds, we will show that as $n_{\max}^{\text{pre}} \to \infty$ and $n_{\max}^{\text{suf}} \to \infty$, the whole reachable state space of the PBA will be sampled and that every state is connected to the root of the tree through the path that has the minimum cost. Following the same logic as in the proof of Theorem 5.1, we conclude that as the number of iterations n goes to infinity for Algorithms 1 and 7, the set $\mathcal{V}_T^n$ constructed by these algorithms will contain all reachable states in Q_P, i.e.,$\mathbb{P}\left(\left\{\lim_{n\to\infty} \mathcal{V}_T^n \setminus \mathcal{R}(Q_P) = \emptyset\right\}\right) = 1$, where $\mathcal{R}(Q_P) \subseteq Q_P$

[6] For non-reachable states $q_P^{\text{new}} \in Q_P$ it always holds that $\mathcal{S}_{\to q_P^{\text{new}}} = \emptyset$ and they will never be added to $\mathcal{V}_T^n$.

is a set that collects all reachable states of Q_P. In what follows, we assume $\mathcal{V}_T^n \setminus \mathcal{R}(Q_P) = \emptyset$ for the trees constructed by Algorithms 1 and 7, for a sufficiently large n and, consequently, for a sufficiently large $n_{\max}^{\text{pre}}$ and $n_{\max}^{\text{suf}}$.

Next, we denote by $\tau^* = \tau^{\text{pre},*}[\tau^{\text{suf},*}]^\omega$ the optimal motion plan, i.e, the plan for which it holds $J(\tau^*) = J^*$. Notice that such a motion plan can be generated by the existing model checking method presented in Section 3, since this method utilizes the whole state-space of the PBA P and all transitions among the states. In what follows, we will show that the transitions among states that appear in $\tau^{\text{pre},*}$ and $\tau^{\text{suf},*}$ are captured by the set of edges $\mathcal{E}_T^n$ of the graph $\mathcal{G}_T^n$ constructed for the computation of the prefix and suffix structure, respectively, if $n_{\max}^{\text{pre}} \to \infty$ and $n_{\max}^{\text{suf}} \to \infty$. Notice that this is ensured to happen due to Algorithm 5 that optimizes the tree structure and the assumption that $\mathcal{V}_T^n \setminus \mathcal{R}(Q_P) = \emptyset$.

Specifically, since $\mathcal{V}_T^n \setminus \mathcal{R}(Q_P) = \emptyset$ we have that all states that appear in τ^* belong to the set $\mathcal{V}_T^n$, as well. Next, by construction of the prefix part $\tau^{\text{pre},*}$ it holds that it connects a final state, denoted hereafter by $q_P^F \in Q_P^F$, to the initial state q_P^0 through the shortest path in the graph $\mathcal{G}_P$ defined in Section 3. Notice that since $\mathcal{V}_T^n \setminus \mathcal{R}(Q_P) = \emptyset$, the final state $q_P^F \in Q_P^F$ belongs to $\mathcal{V}_T^n$, as well. Next, recall that after the execution of Algorithm 5, the cost $C_T(q_P)$ of any state $q_P \in \mathcal{V}_T^n$ and consequently, of the state q_P^F will be minimized. By definition of the cost $C_T(q_P)$, this means that the final state q_P^F will be connected to the root q_P^0 through a path with the minimum cost J. Consequently, the cost of the path that corresponds to the prefix part constructed by Algorithm 1 that connects the final state $q_P^F \in \mathcal{F}$ to the root q_P^0 will be $J(\tau^{\text{pre},*})$. Following the same logic, the cost of the respective suffix part, i.e., the cycle around the final state q_P^F will be $J(\tau^{\text{suf},*})$. Next, since the optimality criterion for both the algorithm described in Section 3 and our proposed algorithm is the same, defined as $J(\tau) = J(\tau^{\text{pre}}) + J(\tau^{\text{suf}})$ for a plan $\tau = \tau^{\text{pre}}[\tau^{\text{suf}}]^\omega$, the resulting plan $\tau^{n_{\max}^{\text{suf}}}_{n_{\max}^{\text{pre}}}$ given by our proposed algorithm will coincide with τ^*.

Thus, we have proved that $J(\tau^{n_{\max}^{\text{suf}}}_{n_{\max}^{\text{pre}}}) = J^*$, if $\mathcal{V}_T^n \setminus \mathcal{R}(Q_P) = \emptyset$, which is true with probability 1 if $n_{\max}^{\text{pre}} \to \infty$ and $n_{\max}^{\text{suf}} \to \infty$. Consequently, we have that

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

which completes the proof.