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

Yiannis Kantaros, and Michael M. Zavlanos,

***Abstract*— This paper proposes a distributed sampling-based algorithm for optimal multi-robot control synthesis 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 our previous work, we have proposed a more tractable centralized sampling-based algorithm that builds incrementally trees that approximate the state-space and transitions of the synchronous product automaton and does not require sophisticated graph search techniques. In this work, we provide a distributed implementation of this sampling-based algorithm, whereby the robots collaborate to build subtrees that can be stored and manipulated locally decreasing the computational time significantly. We provide theoretical guarantees showing that the distributed algorithm preserves the probabilistic completeness and asymptotic optimality of its centralized counterpart. Finally, we show through numerical experiments that the proposed algorithm can synthesize optimal plans from product automata with billions of states, which is not possible using standard optimal control synthesis algorithms or off-the-shelf model checkers. To the best of our knowledge, this is the first distributed, probabilistically complete, and asymptotically optimal control synthesis for multi-robot systems under global temporal tasks.**

I. Introduction

Control synthesis for mobile robots under complex tasks, captured by Linear Temporal Logic (LTL) formulas, build upon either bottom-up approaches when independent LTL tasks are assigned to robots [1], [2] or top-down approaches when a global LTL formula describing a collaborative task is assigned to a team of robots [3], [4], as in this work. Common in the above works is that they rely on model checking theory [5] 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 [6]–[8] and [9], [10], respectively. In top-down approaches [9], [10], 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. A more tractable approach is presented in [11] that identifies independent parts of the LTL formula and builds a local product automaton for each agent. Nevertheless, this approach can be applied only to finite LTL missions and does not have optimality guarantees.

To mitigate these issues, in our previous work we have proposed a centralized optimal control synthesis algorithm that avoids the explicit construction of the product among the transition systems and the NBA. Specifically, [12] builds incrementally directed trees that approximately represent the state-space and transitions among states of the synchronous product automaton. The advantage 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, [12] scales much better compared to existing top-down approaches. Nevertheless, this approach requires a central unit to store the tree structures.

In this work, we provide a distributed implementation of [12]. In particular, the robots collaborate to build subtrees that can be stored and manipulated locally decreasing the computational time significantly while the composition of these subtrees simulates the global tree build by [12]. We also provide theoretical guarantees showing that the distributed algorithm preserves the probabilistic completeness and asymptotic optimality of its centralized counterpart [12]. We present numerical simulations that show that the proposed approach can build trees faster than [12] and 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 [13] and NuSMV [14]. Finally, in Appendix II, we discuss how the proposed algorithm can be transformed into an *anytime* sampling based algorithm [15]. This way, we can assign feasible paths to the robots which are generated offline and are optimized online, as the robots execute them.

To the best of our knowledge, the most relevant works are presented in [16]–[19]. Common in the works [16], [17] is that a discrete abstraction of the environment is built until it becomes expressive enough to generate a motion plan

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.

that satisfies the LTL specification. Both algorithms build upon the RRG algorithm to construct transition systems that capture robot mobility in the workspace. However, building arbitrary graph structures to represent transition systems compromises scalability of temporal 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. More details about comparison with these works can be found in [12]. On the other hand, our proposed sampling-based approach, given a discrete abstraction of the environment [20], 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 [17]. Moreover, we show that our proposed planning algorithm is asymptotically optimal which is not the case in [17]. Centralized sampling-based planning algorithms for multi-robot systems under global temporal goals were also proposed in our previous works [18], [19] and further extended in [12]. To the best of our knowledge, this work presents the first distributed and computationally efficient control synthesis algorithm under global temporal specifications that is probabilistically complete and asymptotically optimal.

II. 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 $i \in \{1, \dots, N\}$. We assume that there are W disjoint regions of interest in $\mathcal{W}$. 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; see e.g., [20] and the references therein. The definition of the wTS for robot i follows which is also illustrated in Figure 1.

Definition 2.1 (wTS): A *weighted Transition System* (wTS) for robot i, denoted by wTS_i is a tuple $\text{wTS}_i = (\mathcal{Q}_i, q_i^0, \rightarrow_i, w_i, \mathcal{AP}_i, L_i)$ where: (a) $\mathcal{Q}_i = \{q_i^{\ell_j}\}_{j=1}^W$ is the set of states, where a state $q_i^{\ell_j}$ indicates that robot i is at location ℓ_j; (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) $w_i : \mathcal{Q}_i \times \mathcal{Q}_i \rightarrow \mathbb{R}_+$ 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

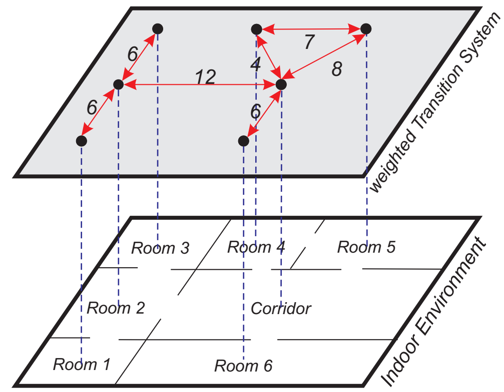


Fig. 1. Graphical depiction of a wTS that abstracts robot mobility in an indoor environment. Black disks stand for the states of wTS, red edges capture transitions among states and numbers on these edges represent the cost w_i for traveling from one state to another one.

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) $L_i : \mathcal{Q}_i \rightarrow \mathcal{AP}_i$ is an observation/output function defined as $L_i(q_i^{\ell_j}) = \pi_i^{\ell_j}$, for all $q_i^{\ell_j} \in \mathcal{Q}_i$.

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 [9]:

Definition 2.2 (PTS): Given N transition systems $\text{wTS}_i = (\mathcal{Q}_i, q_i^0, \rightarrow_i, w_i, \mathcal{AP}, L_i)$, the *product transition system* PTS $= \text{wTS}_1 \otimes \text{wTS}_2 \otimes \cdots \otimes \text{wTS}_N$ is a tuple 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) $q_{\text{PTS}}^0 = (q_1^0, q_2^0, \dots, q_N^0) \in \mathcal{Q}_{\text{PTS}}$ 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 $\frac{\bigwedge_{\forall i}(q_i \rightarrow_i q_i')}{q_{\text{PTS}} \rightarrow_{\text{PTS}} q_{\text{PTS}}'}$, where with slight abuse of notation $q_{\text{PTS}} = (q_1, \dots, q_N) \in \mathcal{Q}_{\text{PTS}}$, $q_i \in \mathcal{Q}_i$. 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 $i \in \{1, \dots, N\}$; (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 $w_{\text{PTS}}(q_{\text{PTS}}, q_{\text{PTS}}') = \sum_{i=1}^N w_i(\Pi|_{\text{wTS}_i} q_{\text{PTS}}, \Pi|_{\text{wTS}_i} q_{\text{PTS}}')$, 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) $L_{\text{PTS}} = \bigcup_{\forall i} L_i : \mathcal{Q}_{\text{PTS}} \rightarrow \mathcal{AP}$ is an observation/output function 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, $\tau = \tau(1)\tau(2)\tau(3)\dots$ such that $\tau(1) = q_{\text{PTS}}^0$, $\tau(k) \in \mathcal{Q}_{\text{PTS}}$, and $(\tau(k), \tau_i(k+1)) \in \rightarrow_{\text{PTS}}$, $\forall k \in \mathbb{N}_+$, where k is an index that points to the k-th entry of τ denoted by $\tau(k)$. The *trace* of an infinite path $\tau = \tau(1)\tau(2)\tau(3)\dots$ 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., $\texttt{trace}(\tau) = L(\tau(1))L(\tau(2))\dots$. 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 2.2, the *cost* of a finite path τ, denoted by $\hat{J}(\tau) \geq 0$, can be defined as

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

where, $|\tau|$ 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.

We assume that the robots have to accomplish a complex collaborative task encapsulated by a global LTL statement ϕ defined over the set of atomic propositions $\mathcal{AP} = \bigcup_{i=1}^{N} \mathcal{AP}_i$. Due to space limitations, we abstain from formally defining the semantics and syntax of LTL. A detailed overview can be found in [5]. Given an LTL formula ϕ, we define the *language* $\texttt{Words}(\phi) = \left\{\sigma \in (2^{\mathcal{AP}})^{\omega} | \sigma \models \phi\right\}$, where $\models \subseteq (2^{\mathcal{AP}}) \times \phi$ is the satisfaction relation, as the set of infinite words $\sigma \in (2^{\mathcal{AP}})^{\omega}$ that satisfy the LTL formula ϕ. Any LTL formula ϕ can be translated into a Nondeterministic Büchi Automaton (NBA) over $(2^{\mathcal{AP}})^{\omega}$ denoted by B [21] defined as follows:

Definition 2.3 (NBA): 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.

Given the PTS and the NBA B that corresponds to the LTL ϕ, we can now define the *Product Büchi Automaton* (PBA) $P = \text{PTS} \otimes B$ [5], as follows:

Definition 2.4 (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) $w_P(q_{\text{P}}, q'_{\text{P}}) = w_{\text{PTS}}(q_{\text{PTS}}, q'_{\text{PTS}})$, 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.

Given 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 $\tau \models \phi$. Specifically, 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}. The prefix part τ^{pre} is the projection of a finite path p^{pre} that lives in $\mathcal{Q}_P$ onto $\mathcal{Q}_{\text{PTS}}$. The path p^{pre} starts from an initial state $q_P^0 \in \mathcal{Q}_P^0$ and ends at a final state $q_P^F \in \mathcal{Q}_P^F$, i.e., it has the following structure $p^{\text{pre}} = (q_{\text{PTS}}^0, q_B^0)(q_{\text{PTS}}^1, q_B^1)\dots(q_{\text{PTS}}^K, q_B^K)$ with $(q_{\text{PTS}}^K, q_B^K) \in \mathcal{Q}_P^F$. The suffix part τ^{suf} is the projection of a finite path p^{suf} that lives in $\mathcal{Q}_P$ onto $\mathcal{Q}_{\text{PTS}}$. The path p^{suf} is a cycle around the final state $(q_{\text{PTS}}^K, q_B^K)$, i.e., it has the following structure $p^{\text{suf}} = (q_{\text{PTS}}^K, q_B^K)(q_{\text{PTS}}^{K+1}, q_B^{K+1})\dots(q_{\text{PTS}}^{K+S}, q_B^{K+S})(q_{\text{PTS}}^{K+S+1}, q_B^{K+S+1})$, where $(q_{\text{PTS}}^{K+S+1}, q_B^{K+S+1}) = (q_{\text{PTS}}^K, q_B^K)$. Then our goal is to compute a plan $\tau = \tau^{\text{pre}}[\tau^{\text{suf}}]^{\omega} = \Pi|_{\text{PTS}} p^{\text{pre}}[\Pi|_{\text{PTS}} p^{\text{pre}}]^{\omega}$, 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 $\hat{J}(\cdot)$ is defined in (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. 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, to find the optimal plan $\tau \models \phi$, shortest paths towards final states and shortest cycles around them are computed. More details about this approach can be found in [7]–[10] and the references therein.

III. Distributed Sampling-based Optimal Control Synthesis

Since the size of the PBA can grow arbitrarily large with the number of robots and complexity of the task, constructing the PBA and applying graph-search techniques to find optimal plans, as discussed in Section II-A, is resource demanding and computationally expensive. A more tractable algorithm is presented in [12] that constructs incrementally directed trees $\mathcal{G}_T = \{\mathcal{V}_T, \mathcal{E}_T, \texttt{Cost}\}$ that approximately represent the state-space $\mathcal{Q}_P$ and the transition relation $\rightarrow_P$ of the PBA defined in Definition 2.4. The root of $\mathcal{G}_T$ is denoted by q_P^r. Also, 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 $\mathcal{V}_T$ that satisfy the rule $\rightarrow_P$. The function $\texttt{Cost} : \mathcal{V}_T :\rightarrow \mathbb{R}_+$ assigns the cost of reaching node $q_P \in \mathcal{V}_T$ from the root q_P^r of the tree. In other words,

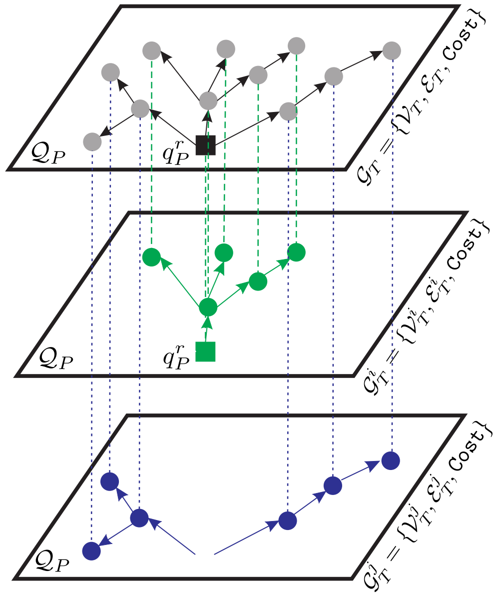


Fig. 2. Graphical depiction of the distributed construction of the global tree $\mathcal{G}_T = \{\mathcal{V}_T, \mathcal{E}_T, \texttt{Cost}\}$ by two robots i, j. Robots i and j build subtrees $\mathcal{G}_T^i$ and $\mathcal{G}_T^j$ to explore the state-space $\mathcal{Q}_P$ such that (i) $\mathcal{G}_T^i \cap \mathcal{G}_T^j = \varnothing$, (ii) $\mathcal{G}_T^i \cup \mathcal{G}_T^j = \mathcal{G}_T$, and (iii) for all $q_P \in \mathcal{V}_T^i$ $[\mathcal{V}_T^j]$, except for the root q_P^r, it holds that all nodes $q_P' \in \mathcal{S}(q_P)$ belong to $\mathcal{V}_T^i$ $[\mathcal{V}_T^j]$.

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

In this section we present a distributed implementation of the optimal control synthesis method presented in [12]. Specifically, each robot i builds a subtree $\mathcal{G}_T^i = \{\mathcal{V}_T^i, \mathcal{E}_T^i, \texttt{Cost}\} \subseteq \mathcal{G}_T$, so that the subtrees are (i) disjoint, i.e., $\bigcap_i \mathcal{V}_T^i = \varnothing$ and $\bigcap_i \mathcal{E}_T^i = \varnothing$, (ii) the union of the subtrees comprises the global connected tree built by [12], i.e., $\bigcup_{\forall i} \mathcal{V}_T^i = \mathcal{V}_T$, $\bigcup_{\forall i} \mathcal{E}_T^i = \mathcal{E}_T$, and (iii) for each $q_P \in \mathcal{V}_T^i$, except for the root q_P^r of $\mathcal{G}_T$, it holds that all successor nodes of q_P in $\mathcal{V}_T$, collected in the set $\mathcal{S}(q_P)$, belong to $\mathcal{V}_T^i$, as well, for all robots i; see Figure 2. Conditions (i)-(iii) allow for distributing the computational burden of extending, rewiring, and storing the constructed tree across the robots. The distributed construction of optimal plans is described in Algorithm 1. Specifically, the prefix parts synthesized by robot i are constructed in lines 2-8, the respective suffix parts are constructed in lines 9-22, and the optimal discrete plan is synthesized in lines 23-25.

A. Distributed Construction of Prefix Parts

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 all trees $\mathcal{G}_T^i$, as [line 2, Alg. 1]

$$\mathcal{X}_{\text{goal}}^{\text{pre}} = \{q_P = (q_{\text{PTS}},\ q_B) \in \mathcal{Q}_P \mid q_B \in \mathcal{Q}_B^F\}. \tag{3}$$

The root q_P^r of the global tree $\cup_{\forall i}\mathcal{G}_T^i = \mathcal{G}_T$ 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 $q_B^0 \in \mathcal{Q}_B^0$ [lines 3-4, Alg. 1]. In line 4 of Algorithm 1, $\mathcal{Q}_B^0(b_0)$ stands for the b_0-th

---

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

Input: Logic formula ϕ, Transition systems $\text{wTS}_1, \dots, \text{wTS}_N$, Initial location $q_{\text{PTS}}^0 \in \mathcal{Q}_{PTS}$, maximum numbers of iterations $n_{\max}^{\text{pre}}$, $n_{\max}^{\text{suf}}$
Output: Optimal plans $\tau \models \phi$
1 Convert ϕ to a NBA $B = \left(\mathcal{Q}_B, \mathcal{Q}_B^0, \rightarrow_B, \mathcal{Q}_B^F\right)$;
2 Define goal set: $\mathcal{X}_{\text{goal}}^{\text{pre}}$;
3 **for** $b_0 = 1 : |\mathcal{Q}_B^0|$ **do**
4 Initial NBA state $q_B^0 = \mathcal{Q}_B^0(b_0)$;
5 Root of the tree: $q_P^r = (q_{\text{PTS}}^0, q_B^0)$;
6 $\left[\cup_{\forall i}\mathcal{G}_T^i, \cup_{\forall i}\mathcal{P}^i\right] = \texttt{DTree}(\mathcal{X}_{\text{goal}}^{\text{pre}}, \text{wTS}_1, \dots, \text{wTS}_N, B, q_P^r, n_{\max}^{\text{pre}})$;
7 **for** $\alpha = 1 : |\mathcal{P}^i|$ **do**
8 $\tau_i^{\text{pre},\mathcal{P}^i(\alpha)} = \texttt{FindPath}(\mathcal{G}_T^i, q_P^r, \mathcal{P}^i(\alpha))$, in parallel across the robots;
9 **for** $i = 1 : N$ **do**
10 **for** $\alpha = 1 : |\mathcal{P}^i|$ **do**
11 Root of the tree: $q_P^r = \mathcal{P}^i(\alpha)$;
12 Define goal set: $\mathcal{X}_{\text{goal}}^{\text{suf}}(q_P^r)$;
13 Select robot i^*;
14 **if** $(q_P^r \in \mathcal{X}_{goal}^{suf}) \ \wedge \ (w_P(q_P^r, q_P^r) = 0)$ **then**
15 $\mathcal{G}_T^{i^*} = (\{q_P^r\}, \{q_P^r, q_P^r\}, 0)$;
16 $\mathcal{S}_\alpha^{i^*} = \{q_P^r\}$;
17 **else**
18 $\left[\cup_{\forall j}\mathcal{G}_T^j, \cup_{\forall j}\mathcal{S}_\alpha^j\right] = \texttt{DTree}(\mathcal{X}_{\text{goal}}^{\text{suf}}, \text{wTS}_1, \dots, \text{wTS}_N, B, q_P^r, n_{\max}^{\text{suf}})$;
19 **for** $e_j = 1 : |\mathcal{S}_\alpha^j|$ **do**
20 $\tau_j^{\text{suf},\mathcal{P}_i(\alpha),e_j} = \texttt{FindPath}(\mathcal{G}_T^j, q_P^r, \mathcal{S}_\alpha^j(e))$, in parallel $\forall j$;
21 Send $\tau_j^{\text{suf},\mathcal{P}_i(\alpha),e_j}$ to robot i, if $i \neq j$;
22 Robot i has collected all suffix parts for the final $\mathcal{P}^i(\alpha)$ and picks the one with the minimum cost, denoted by $\tau_i^{\text{suf},\mathcal{P}^i(\alpha)}$;
23 $[j^*, \alpha^*] = \text{argmin}_{[j,f]}(\hat{J}(\tau_j^{\text{pre},\mathcal{P}^j(\alpha)}) + \hat{J}(\tau_j^{\text{suf},\mathcal{P}^j(\alpha)}))$;
24 $\tau_{q_B^0} = \tau_{j^*}^{\text{pre},\alpha^*}[\tau_{j^*}^{\text{suf},\alpha^*}]^\omega$;
25 Optimal Plan τ is selected among all plans $\tau_{q_B^0}$ to be the one with the smallest cost;

---

state in the set $\mathcal{Q}_B^0$ assuming an arbitrary enumeration of the elements of the set $\mathcal{Q}_B^0$. Construction of the subtrees $\mathcal{G}_T^i$ is described in Algorithm 2 which requires communication between all robots.

1) Initialization: At the beginning the robots coordinate to elect a robot i^* that will store the root q_P^r. The election criteria of robot i^* can be arbitrary. In this work, the robot i^* is selected probabilistically, i.e., it is sampled from the probability density function $f_{i^*} : \mathbb{N}_+ \rightarrow [0, 1]$, which we assume that is non-zero on $[1, 2, \dots, N]$ meaning that every robot has a non-zero probability of being i^* at every iteration n of Algorithm 2. Also, we assume that the probability density function f_{i^*} remains the same for all iterations n,

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

1 Robot 1 samples robot i^* from f_{i^*} and propagates the result across the network;
2 $\mathcal{V}_T^{i^*} = \{q_P^r\}$, $\mathcal{E}_T^{i^*} = \emptyset$, $\texttt{Cost}(q_P^r) = 0$;
3 $\mathcal{V}_T^{i} = \mathcal{E}_T^{i} = \emptyset, \forall i \neq i^*$;
4 **for** $n = 1 : n_{max}$ **do**
5 $\quad q_{\text{PTS}}^{\text{new}} = \texttt{Sample}(\mathcal{V}_T^{i^*})$;
6 $\quad$ **for** $b = 1 : |\mathcal{Q}_B|$ **do**
7 $\quad\quad$ Robot i^* creates $q_B^{\text{new}} = \mathcal{Q}_B(b)$;
8 $\quad\quad$ Robot i^* constructs $q_P^{\text{new}} = (q_{\text{PTS}}^{\text{new}}, q_B^{\text{new}})$;
9 $\quad\quad$ State q_P^{new} is transmitted to all other robots i;
10 $\quad\quad$ **if** $q_P^{new} \notin \bigcup_{\forall i} \mathcal{V}_T^i$ **then**
11 $\quad\quad\quad [q_P^{\text{prev},i}, C^i_{q_P^{\text{new}}}] = \texttt{CandidateParent}(q_P^{\text{new}}, \rightarrow_P, \mathcal{G}_T^i),\ \forall i$;
12 $\quad\quad\quad$ **if** $\{q_P^{prev,1}, q_P^{prev,2}, \ldots, q_P^{prev,N}\} \neq \varnothing$ **then**
13 $\quad\quad\quad\quad j = \text{argmin}_i\{C^i_{q_P^{\text{new}}}\}$;
14 $\quad\quad\quad\quad$ **if** $q_P^{prev,j} \neq q_P^r$ **then**
15 $\quad\quad\quad\quad\quad s = j$;
16 $\quad\quad\quad\quad$ **else**
17 $\quad\quad\quad\quad\quad s = \text{argmax}_i\{M_i^n\}$;
18 $\quad\quad\quad\quad \mathcal{V}_T^s = \mathcal{V}_T^s \cup \{q_P^{\text{new}}\}$;
19 $\quad\quad\quad\quad \mathcal{E}_T^s = \mathcal{E}_T^s \cup \{q_P^{\text{prev},j}, q_P^{\text{new}}\}$;
20 $\quad\quad\quad\quad \texttt{Cost}(q_P^{\text{new}}) = C^j_{q_P^{\text{new}}}$;
21 $\quad\quad$ **if** $q_P^{new} \in \mathcal{V}_T^{i^{new}}$*, for some* i^{new} **then**
22 $\quad\quad\quad$ Robot i^{new} transmits to all other robots $\texttt{Cost}(q_P^{\text{new}})$;
23 $\quad\quad\quad [\mathcal{E}_T^i, \texttt{Cost}, \mathcal{D}_i] = \texttt{Rewire}(q_P^{\text{new}}, \texttt{Cost}(q_P^{\text{new}}), \mathcal{G}_T^i),\ \forall i$;
24 $\quad\quad\quad \mathcal{V}_T^{i^{\text{new}}} = \mathcal{V}_T^{i^{\text{new}}} \cup (\bigcup_{\forall i \neq i^{\text{new}}} \mathcal{S}(\mathcal{D}_i))$;
25 $\quad\quad\quad \mathcal{E}_T^{i^{\text{new}}} = \mathcal{E}_T^{i^{\text{new}}} \cup (\bigcup_{\forall i \neq i^{\text{new}}} \mathcal{E}_T^{\mathcal{D}_i})$;
26 $\quad$ Robot i^* samples a new robot i^* from f_{i^*};
27 $\mathcal{Z}^i = \mathcal{V}_T^i \cap \mathcal{X}_{\text{goal}}, \forall i$;

although other sampling methods for i^* can be employed; see Remark 1.1 in Appendix I. Initially, without loss of generality, we assume that robot 1 will sample from f_{i^*} the robot i^* and, then notifies the sampled robot i^* about the result [line 1, Alg. 2]. Then, the set of nodes and edges of the subgraph $\mathcal{G}_T^{i^*}$ are initialized as $\mathcal{V}_T^{i^*} = \{q_P^r\}$, $\mathcal{E}_T^{i^*} = \varnothing$ while the cost of the root is zero [line 2, Alg. 2]. The set of nodes and edges for all other subgraphs $\mathcal{G}_T^i$, $i \neq i^*$, are initially empty [line 3, Alg. 2].

2) Constructing state q_P^{new}: After each iteration n of Algorithm 2, the current robot i^* takes a sample from f_{i^*} which is the new robot i^* [line 26, Alg. 2]. Then iteration $n{+}1$ of Algorithm 2 follows. First, robot i^* is responsible for sampling a new state $q_P^{\text{new}} = (q_{\text{PTS}}^{\text{new}}, q_B^{\text{new}})$. This is achieved by the sampling function `Sample` [lines 5-8, Alg. 2]; see Algorithm 3. Specifically, robot i^* first creates 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 $f_{\text{rand}}(q_P|\mathcal{V}_T^{i^*}) : \mathcal{V}_T^{i^*} \rightarrow [0,1]$ [lines 1-2, Alg. 3]. The probability density function $f_{\text{rand}}(q_P|\mathcal{V}_T^{i^*})$ defines the

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

1 Pick a state $q_P^{\text{rand}} \in \mathcal{V}_T^{i^*}$ from a given distribution $f_{\text{rand}}(q_P|\mathcal{V}_T^{i^*}) : \mathcal{V}_T^{i^*} \rightarrow [0,1]$;
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};

probability of selecting the state $q_P \in \mathcal{V}_T^{i^*}$ as the state q_P^{rand} at iteration n of Algorithm 2 given the set $\mathcal{V}_T^{i^*}$. We make the following assumption for $f_{\text{rand}}(q_P|\mathcal{V}_T^{i^*})$ that is also made in [12].

Assumption 3.1 (Probability density function f_{rand}): (i) The probability density function $f_{\text{rand}}(q_P|\mathcal{V}_T^{i^*}) : \mathcal{V}_T^{i^*} \rightarrow [0,1]$ is non-zero on $\mathcal{V}_T^{i^*}$. (ii) The probability density function $f_{\text{rand}}(q_P|\mathcal{V}_T^{i^*}) : \mathcal{V}_T^{i^*} \rightarrow [0,1]$ remains the same for all iterations n and for a given state $q_P \in \mathcal{V}_T^{i^*}$ is monotonically decreasing with respect to the size of $|\mathcal{V}_T^{i^*}|$. This also implies that for a given $q_P \in \mathcal{V}_T^{i^*}$, the probability $f_{\text{rand}}(q_P|\mathcal{V}_T^{i^*})$ remains the same for all iterations n if the set $\mathcal{V}_T^{i^*}$ does not change. (iii) Independent samples q_P^{rand} can be drawn from f_{rand}.

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

$$\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}}) = \{q_{\text{PTS}} \in \mathcal{Q}_{\text{PTS}} \mid q_{\text{PTS}}^{\text{rand}} \rightarrow_{\text{PTS}} q_{\text{PTS}}\} \tag{4}$$

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 $f_{\text{new}}(q_{\text{PTS}}|q_{\text{PTS}}^{\text{rand}}) : \mathcal{R}_{\text{PTS}} \rightarrow [0,1]$ [line 3, Alg. 3] that satisfies the following assumption that is also made in [12].[1]

Assumption 3.2 (Probability density function f_{new}): (i) The probability density function $f_{\text{new}}(q_{\text{PTS}}|q_{\text{PTS}}^{\text{rand}}) : \mathcal{R}_{\text{PTS}} \rightarrow [0,1]$ is non-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 f_{new}.

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]. Then, once q_P^{new} is sampled, robot i^* transmits it to all other robots in the team that coordinate with each other and with robot i^* [line 9, Alg. 2] to check if there exists a robot

[1] Note that other sampling methods for q_P^{rand} and q_P^{new} can be employed that do not require the more strict conditions of Assumptions 3.2(ii) and 3.1(ii); see Remark A.1 and Remark A.2 in Appendix A, in [12]. Also, note that in order to obtain the state $q_{\text{PTS}}^{\text{new}}$ we do not need to construct the reachable set $\mathcal{R}_{\text{PTS}}(q_{\text{PTS}}^{\text{rand}})$. Instead, only 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 need to be constructed. More details can be found in [12].

Algorithm 4: Function $\texttt{CandidateParent}(q_P^{\text{new}}, \rightarrow_P, \mathcal{G}_T^i)$

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

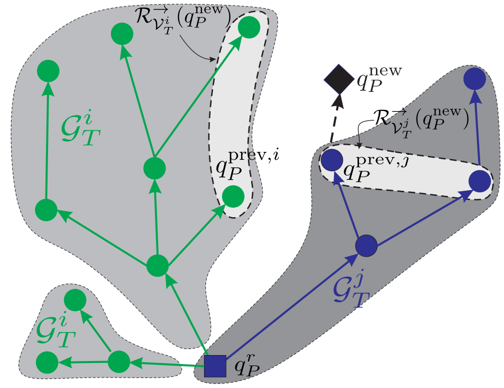


Fig. 3. Graphical depiction of extending the subtrees. The blue square stands for the root q_P^r. The subtree $\mathcal{G}_T^i$ consists of the green disks and edges and $\mathcal{G}_T^j$ consists of the blue square, blue disks, and blue edges. The blue diamond stands for the state q_P^{new}. The dashed black arrow represents the new edge that will be added to the set $\mathcal{E}_T^j$ after extending the subtrees.

i^{new} for which $q_P^{\text{new}} \in \mathcal{V}_T^{i^{\text{new}}}$.[2] If $q_P^{\text{new}} \notin \mathcal{V}_T^i$ for all robots i, then the robots check if they can extend their respective graphs $\mathcal{G}_T^i$ towards the new state q_P^{new} [lines 10-20, Alg. 2]. Otherwise, if there exists a robot i^{new} such that $q_P^{\text{new}} \in \mathcal{V}_T^{i^{\text{new}}}$, then all robots i including i^{new} check if they can rewire the nodes in $\mathcal{V}_T^i$ to q_P^{new} [lines 21-23, Alg. 2].

3) Extending subgraph $\mathcal{G}_T^i$ to q_P^{new}: If $q_P^{\text{new}} \notin \mathcal{V}_T^i$ for all robots i [lines 10-20, Alg. 2], then every robot i finds the candidate parent $q_P^{\text{prev},i}$ for q_P^{new}, selected from the set of nodes $\mathcal{V}_T^i$, that will result in the minimum cost $\texttt{Cost}(q_P^{\text{new}})$. This is accomplished by Algorithm 4 [line 11, Alg. 2] which all robots execute independently from each other. Specifically, in Algorithm 4, robot i constructs the set $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T^i}(q_P^{\text{new}}) \subseteq \mathcal{V}_T^i$ defined as

$$\mathcal{R}^{\rightarrow}_{\mathcal{V}_T^i}(q_P^{\text{new}}) = \{q_P \in \mathcal{V}_T^i | q_P \rightarrow_P q_P^{\text{new}}\}, \tag{5}$$

that collects all states $q_P \in \mathcal{V}_T^i$ that satisfy the transition rule $(q_P, q_P^{\text{new}}) \in \rightarrow_P$ [line 1, Alg. 4]. If this set $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T^i}(q_P^{\text{new}})$ is empty then robot i does not propose any candidate parent for the sample q_P^{new}. In case $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T^i}(q_P^{\text{new}}) \neq \emptyset$, then robot i proposes the state $q_P^{\text{prev},i} = \text{argmin}_{q_P \in \mathcal{R}^{\rightarrow}_{\mathcal{V}_T^i}(q_P^{\text{new}})^i}[\texttt{Cost}(q_P) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P, \Pi|_{\text{PTS}} q_P^{\text{new}})]$ as the candidate parent of the sample q_P^{new} [line 3, Alg. 4]. In words $q_P^{\text{prev},i}$ is selected among all states in $\mathcal{R}^{\rightarrow}_{\mathcal{V}_T^i}(q_P^{\text{new}})$ so that the cost of q_P^{new} is minimized. Let $C^i_{q_P^{\text{new}}} = \texttt{Cost}(q_P^{\text{prev},i}) + w_{\text{PTS}}(\Pi|_{\text{PTS}} q_P^{\text{prev},i}, \Pi|_{\text{PTS}} q_P^{\text{new}})$, be the cost of q_P^{new} if its parent is $q_P^{\text{prev},i}$ [line 4, Alg. 4]. Once Algorithm 4 has been executed by all robots, then they check if $\{q_P^{\text{prev},1}, q_P^{\text{prev},2}, \dots, q_P^{\text{prev},N}\} \neq \varnothing$, i.e., if there is at least one robot that has proposed a candidate parent for the sample q_P^{new} [line 12, Alg. 2]. If this is the case, then the parent of q_P^{new} is selected to be the node $q_P^{\text{prev},j}$, with $j = \text{argmin}_i\{C^i_{q_P^{\text{new}}}\}$, i.e., the node that results in the minimum cost for q_P^{new} among all candidate parents [line 13, Alg. 2].

Finally, the robots coordinate to decide which robot will include the new state q_P^{new} in its subtree. Hereafter, we denote by s the robot that will store the sample q_P^{new}. If the parent $q_P^{\text{prev},j}$ of q_P^{new} is not the root q_P^r of the tree, then robot j will store q_P^{new}, i.e., $s = j$ [lines 14-15, Alg. 2]. On the other hand, if $q_P^{\text{prev},j} = q_P^r$, then the robot with the largest free memory at iteration n, denoted by $M_i^n \geq 0$, will store q_P^{new}, i.e., $s = \text{argmax}_i\{M_i^n\}$, accounting in this way for an efficient use of the available resources [lines 16-17, Alg. 2]. This way we guarantee that the successor nodes of any node $q_P \in \mathcal{V}_T^i$, except for the root q_P^r, belong to $\mathcal{V}_T^i$, for all robots i. As it will be discussed later, this enables the parallel execution of the rewiring step across the subtrees $\mathcal{G}_T^i$. Next, the set of nodes and edges for the subgraph $\mathcal{G}_T^s$ are updated as $\mathcal{V}_T^s = \mathcal{V}_T^s \cup \{q_P^{\text{new}}\}$ and $\mathcal{E}_T^s = \mathcal{E}_T^s \cup \{q^j_{\text{prev}}, q_P^{\text{new}}\}$ [lines 18-19, Alg. 2], while the cost of q_P^{new} is $\texttt{Cost}(q_P^{\text{new}}) = C^j_{q_P^{\text{new}}}$ [line 20, Alg. 2]. The above procedure is illustrated in Figure 3, as well. Notice that due to the way that the robot s that stores the sample q_P^{new} is selected, it holds that at every iteration n the number of robots that have a non-empty subtree $\mathcal{G}_T^i$ is equal to $\min(|\mathcal{C}(q_P^r)|, N)$, where $\mathcal{C}(q_P^r) \subseteq \mathcal{S}(q_P^r)$ is a set that collects the *children*, i.e., the 1-hop successors, of the root q_P^r in the global tree $\mathcal{G}_T = \cup_i \mathcal{G}_T^i$.

[2]Observe that robot i^{new} is unique, since the subtrees $\mathcal{G}_T^i$ are disjoint.

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

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

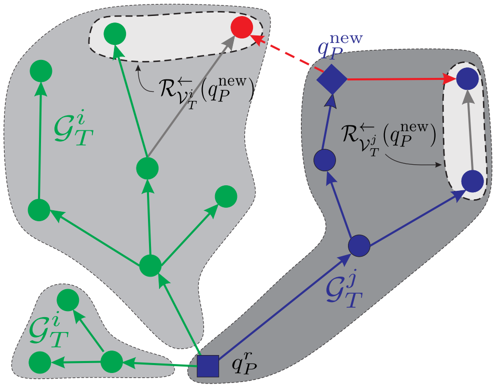


Fig. 4. Graphical depiction of Algorithm 5. The blue square stands for the root q_P^r. Gray arrows stand for the edges that will be deleted from the set $\mathcal{E}_T^i$ and $\mathcal{E}_T^j$ during the execution of Algorithm 5. Red arrows stand for the new edges that are created after rewiring within each subtree. The rewired node in $\mathcal{G}_T^i$ is represented by red color. The red node along with the red dashed edge will be removed from $\mathcal{G}_T^i$ and will be added to $\mathcal{G}_T^j$ after the end of the rewiring step since q_P^{new} (blue diamond) belongs to $\mathcal{V}_T^j$. As a result, all successor nodes of any node belong to the same subtree.

4) Rewiring $\mathcal{G}_T^i$ through q_P^{new}: The rewiring step occurs in two cases. First, it happens if the sample q_P^{new} already belongs to the set of nodes $\mathcal{V}_T^{i^{\text{new}}}$ of robot i^{new}. Second, it occurs if the sample q_P^{new} does not already belong to $\cup_i \mathcal{V}_T^i$, but there is a robot s, determined as per lines 16-17 in Algorithm 2, that extends its subtree towards q_P^{new}. Hereafter, with slight abuse of notations, we denote by i^{new} the robot that has stored the sample q_P^{new} for both cases. In both cases, robot i^{new} transmits the cost $\texttt{Cost}(q_P^{\text{new}})$ to all other robots i [lines 21-22, Alg. 2]. Then, these robots i along with robot i^{new}, check simultaneously if they can rewire the nodes $q_P \in \mathcal{V}_T^i$ and $q_P \in \mathcal{V}_T^{i^{\text{new}}}$ to the node $q_P^{\text{new}} \in \mathcal{V}_T^{i^{\text{new}}}$ in order to decrease the cost $\texttt{Cost}(q_P)$ [lines 23, Alg. 2]. The rewiring process in $\mathcal{G}_T^i$ is described in Algorithm 5 and is illustrated in Figure 4. Note that the rewiring process occurs in parallel across all subtrees $\mathcal{G}_T^i$.

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

$$\mathcal{R}^{\leftarrow}_{\mathcal{V}_T^i}(q_P^{\text{new}}) = \{q_P \in \mathcal{V}_T^i | q_P^{\text{new}} \rightarrow_P q_P\}, \tag{6}$$

that collects all states of $q_P \in \mathcal{V}_T^i$ 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 $q_P \in \mathcal{R}^{\leftarrow}_{\mathcal{V}_T^i}(q_P^{\text{new}})$ 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 4, Alg. 5]. If this is the case for a node $q_P \in \mathcal{R}^{\leftarrow}_{\mathcal{V}_T^i}(q_P^{\text{new}})$, then the new parent of q_P becomes q_P^{new} and the edge that was connecting q_P to its previous parent is deleted [line 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 $\mathcal{G}_T^i$, collected in the set

$$\begin{aligned}\mathcal{S}(q_P) =\{&q'_P \in \mathcal{V}_T^i | q'_P \text{ is connected to } q_P \text{ through} \\ &\text{a multi hop path in } \mathcal{G}_T^i\},\end{aligned} \tag{7}$$

is updated to account for the change in the cost of q_P [line 7, Alg. 5. Also, all robots $i \neq i^{\text{new}}$ store the rewired nodes in the set $\mathcal{D}_i$ [line 8, Alg. 5]. Then, after the rewiring step, robots $i \neq i^{\text{new}}$ send to robot i^{new} the set of nodes $\mathcal{D}_i \cup \mathcal{S}(\mathcal{D}_i)$, the set of edges among these nodes, denoted by $\mathcal{E}_T^{\mathcal{D}_i} \subseteq \mathcal{E}_T^i$, and their respective costs, i.e., $\mathcal{V}_T^{i^{\text{new}}} = \mathcal{V}_T^{i^{\text{new}}} \cup (\bigcup_{\forall i \neq i^{\text{new}}} \mathcal{S}(\mathcal{D}_i))$ and $\mathcal{E}_T^{i^{\text{new}}} = \mathcal{E}_T^{i^{\text{new}}} \cup (\bigcup_{\forall i \neq i^{\text{new}}} \mathcal{E}_T^{\mathcal{D}_i})$ [lines 24-25, Alg. 2]. This way, we ensure that after the rewiring step it holds that if $q_P \in \mathcal{V}_T^i$, then all nodes $q'_P \in \mathcal{S}(q_P)$ belong to $\mathcal{V}_T^i$, as well, for all robots i. In Section IV, we show that this enables the parallel execution of the rewiring step while preserving the probabilistic completeness and asymptotic optimality of the centralized algorithm [12].

5) Distributed Construction of Paths: The construction of the subtrees $\mathcal{G}_T^i$ ends after $n_{\max}$ iterations, where $n_{\max}$ is user specified [line 4, Alg. 2]. Then, every robot i constructs the set $\mathcal{P}^i = \mathcal{V}_T^i \cap \mathcal{X}_{\text{goal}}^{\text{pre}}$ [line 27, Algorithm 2] that collects all the states $q_P \in \mathcal{V}_T^i$ that belong to the goal region $\mathcal{X}_{\text{goal}}^{\text{pre}}$ defined in (3). Then, every robot i finds paths that correspond to the prefix parts and connect the states $q_P \in \mathcal{P}^i$ to the root of the tree q_P^r. In particular, the path that connects the α-th state in the set $\mathcal{P}^i$, denoted by $\mathcal{P}^i(\alpha)$, to the root q_P^r constitutes the α-th prefix part found by robot i and is denoted by $\tau_i^{\text{pre},\mathcal{P}^i(\alpha)}$ [line 8, Algorithm 1]. Specifically, the prefix part $\tau^{\text{pre},\mathcal{P}^i(\alpha)}$ is constructed by tracing the sequence of parents of nodes starting from the node that represents the accepting state $\mathcal{P}^i(\alpha)$ and ending at the root of the tree. The parent of each node is computed by the function $\texttt{parent}: \mathcal{V}_T^i \rightarrow \mathcal{V}_T^i \cup \{q_P^r\}$. This function maps a node $q_P \in \mathcal{V}_T^i$ to a unique vertex $q'_P \in \mathcal{V}_T^i$ if $(q'_P, q_P) \in \mathcal{E}_T^i$ and $q'_P \neq q_P^r$ or to the root if $q'_P = q_P^r$. By convention, we assume that $\texttt{parent}(q_P^r) = q_P^r$. Observe that communication between robots is not required for the construction of the paths, since all parent nodes of any node belong to the same subtree, except for the nodes whose parent is the root.

Next, observe that the time complexity of sampling the state $q_{\text{PTS}}^{\text{new}}$ in Algorithm 3 is $O(\sum_i |\mathcal{Q}_i|)$. Moreover, the time complexity of extending the tree $\mathcal{G}_T = \cup_{i=1}^N \mathcal{G}_T^i$ towards q_P^{new} is $O(\max_i |\mathcal{V}_T^i|(N+1))$; see Algorithm 4. Also, the time complexity of the rewiring step is $O(\max_i |\mathcal{V}_T^i|(N+1))$; see Algorithm 5. Note that the time complexity of extending and rewiring $\mathcal{G}_T$ using [12] is $O(|\bigcup_{\forall i} \mathcal{V}_T^i|(N+1)) > O(\max_i |\mathcal{V}_T^i|(N+1))$. Similarly, the time complexity of finding a path from a node $q_P \in \mathcal{V}_T^i$ to the root q_P^r of $\mathcal{G}_T$ is $O(|\mathcal{V}_T^i|)$ while the respective time complexity of [12] is $O(|\mathcal{V}_T|) > O(|\mathcal{V}_T^i|)$. More details and comparisons with state-of-the-art graph search methods can be found in [12] .

B. Distributed Construction of Suffix Parts

Once the prefix parts $\tau_i^{\text{pre},\mathcal{P}^i(\alpha)}$ are constructed, the corresponding suffix parts are constructed [lines 9-22, Alg. 1]. Specifically, given a prefix part $\tau^{\text{pre},\mathcal{P}^i(\alpha)}$, the respective

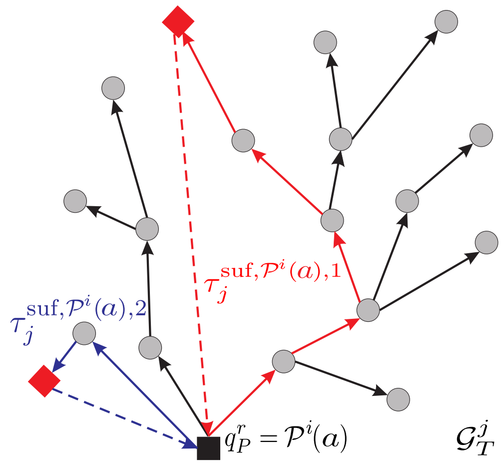


Fig. 5. Graphical depiction of detecting cycles around a final/accepting state $q_P^r = \mathcal{P}^i(\alpha)$ (black square) within a subtree $\mathcal{G}_T^j$. The red diamonds stand for a state $q_P \in \mathcal{S}_\alpha^j$. Solid red and blue arrows stand for two paths that connect the states in $\mathcal{S}_\alpha$ to the root $\mathcal{P}^i(\alpha)$. The dashed red and blue arrows imply that a transition from a state $q_P \in \mathcal{S}_\alpha^j$ to $\mathcal{P}^i(\alpha)$ are feasible according to the transition rule $\longrightarrow_P$; however, such a transition is not included in the set $\mathcal{E}_T^j$. The two detected cycles around the accepting state $\mathcal{P}^i(\alpha)$, denoted by $\tau_j^{\text{suf},\mathcal{P}^i(\alpha),1}$ and $\tau_j^{\text{suf},\mathcal{P}^i(\alpha),2}$ are illustrated by solid and dashed red and blue arrows, respectively.

suffix part $\tau_j^{\text{suf},\mathcal{P}^i(\alpha)}$ constructed by a robot j is a sequence of states in $\mathcal{Q}_P$ that starts from the state $\mathcal{P}^i(\alpha)$ and ends at the same state $\mathcal{P}^i(\alpha)$, i.e., a cycle around state $\mathcal{P}^i(\alpha)$ where any two consecutive states in $\tau_j^{\text{suf},\mathcal{P}^i(\alpha)}$ respect the transition rule $\rightarrow_P$. To construct the suffix part $\tau_j^{\text{suf},\mathcal{P}_i(\alpha)}$, we build again trees $\mathcal{G}_T^j = \{\mathcal{V}_T^j, \mathcal{E}_T^j, \texttt{Cost}\}$ that approximates the PBA P, in a similar way as in Section III-A, and implement a cycle-detection mechanism to identify cycles around the state $\mathcal{P}^i(\alpha)$. The only differences are that: (i) the root of the tree is now $q_P^r = \mathcal{P}^i(\alpha)$, i.e., it is an *accepting/final* state [line 11, Alg. 1] detected during the construction of the prefix parts, (ii) the goal region corresponding to the root $q_P^r = \mathcal{P}^i(\alpha)$, is defined as [line 12, Alg. 1

$$\begin{aligned}\mathcal{X}_{\text{goal}}^{\text{suf}}(q_P^r) =&\{q_P = (q_{\text{PTS}},\ q_B) \in \mathcal{Q}_P \mid \\ &(q_P, L(q_{\text{PTS}}), q_P^r) \in \rightarrow_P\},\end{aligned} \tag{8}$$

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 14, Alg. 1]. If (iii) holds, the construction of the subtrees $\mathcal{G}_T^j$ is trivial, since all the trees are empty except for the tree $\mathcal{G}_T^{i^*}$ that consists of only the root, and a loop around it with zero cost [line 15, Alg. 1].[3] If $q_P^r \notin \mathcal{X}_{\text{goal}}^{\text{suf}}$, then the trees $\mathcal{G}_T^j$ are constructed by Algorithm 2 [line 18, Alg. 1]. For all trees $\mathcal{G}_T^j$, we define a set $\mathcal{S}_\alpha^j \subseteq \mathcal{V}_T^j$ that collects all states $q_P \in \mathcal{V}_T^j \cap \mathcal{X}_{\text{goal}}^{\text{suf}}(q_P^r)$ [lines 16, 18, Alg. 1. Once the trees $\mathcal{G}_T^j$ are constructed, we compute for each state in the set $\mathcal{S}_\alpha^j$, denoted by $\mathcal{S}_\alpha^j(e_j)$, $e_j \in \{1, \dots, |\mathcal{S}_\alpha^j|\}$, the path that connects these states to the root [lines 19- 20, Alg. 1]. These paths correspond to possible suffix parts constructed by robot j that are associated with the prefix part $\tau_i^{\text{pre},\mathcal{P}^i(\alpha)}$ found by robot i and are denoted by $\tau_j^{\text{suf},\mathcal{P}^i(\alpha),e_j}$; see also Figure 5. Next, every robot j sends the candidate suffix parts $\tau_j^{\text{suf},\mathcal{P}^i(\alpha),e_j}$ to robot i [line 21, Alg. 1]. The robot i selects as a suffix part the one with the smallest cost. By construction of the cost functions $\texttt{Cost}$ and $\hat{J}(\cdot)$, it holds that the cost of a suffix part is $\hat{J}(\tilde{\tau}_j^{\text{suf},\mathcal{P}_i(\alpha),e_j}) = \texttt{Cost}(\mathcal{S}_\alpha^j(e_j)) + w_{\text{PTS}}(\Pi|_{\text{PTS}}\mathcal{S}_\alpha^j(e_j), \Pi|_{\text{PTS}} q_P^r)$. The selected suffix part is denoted by $\tau_i^{\text{suf},\mathcal{P}^i(\alpha)}$ [line 22, Alg. 1]. This procedure is repeated for all possible roots q_P^r, i.e., for all final states detected by any robot during the construction of the prefix parts [lines 9-11, Alg. 1].

C. Construction of Optimal Discrete Plans

By construction, any motion plan $\tau_i^{\mathcal{P}^i(\alpha)} = \tau_i^{\text{pre},\mathcal{P}_i(\alpha)}[\tau_i^{\text{suf},\mathcal{P}^i(\alpha)}]^\omega$, with $\mathcal{S}_\alpha^i \neq \emptyset$, and $\alpha \in \{1, \dots, |\mathcal{P}^i|\}$ satisfies the global LTL specification ϕ. Given an initial state $q_B^0 \in \mathcal{Q}_B^0$, among all the motion plans $\tau_i^{\mathcal{P}^i(\alpha)} \models \phi$ found by all robots, we select the one with the smallest cost $J(\tau_i^{\mathcal{P}_i(\alpha)})$ defined in (2) [line 23, Alg. 1]. The plan with the smallest cost given an initial state q_B^0 is denoted by $\tau_{q_B^0}$ [line 24, Alg. 1]. Then, among all plans $\tau_{q_B^0}$, we select again the one with smallest cost $J(\tau_{q_B^0})$, denoted by τ [line 25, Alg. 1].

Remark 3.3 (Communication rounds): For the distributed construction of the subtrees $\mathcal{G}_T^i$ three main communication rounds occur per sample $q_P^{\text{new}} = (q_{\text{PTS}}^{\text{new}}, \mathcal{Q}_B(b))$. The first one involves robot i^* that has to send the sample q_P^{new} to the remaining $N-1$ robots [line 9, Algorithm 2]. In the second communication round every robot i that can propose a candidate parent $q_P^{\text{prev},i}$ for q_P^{new} has to send its respective cost $C_{q_P^{\text{new}}}^i$ to any other robot j [lines 12-13, Alg. 2]. The third communication round concerns the rewiring step. Specifically, every robot i has to send all the nodes $\mathcal{D}_i \cup \mathcal{S}(\mathcal{D}_i) \subseteq \mathcal{V}_T^i$ to robot i^{new} that has stored q_P^{new}. Notice that during the second and the third communication round, robots can exchange of information asynchronously.

Remark 3.4: Notice that since Algorithm 1 is executed offline, it can be executed over a connected network of $M > N$ processors, instead of the network of N robots that are involved in ϕ speeding up the control synthesis.

IV. Correctness And Optimality

In this section, we show that the distributed Algorithm 1 is probabilistically complete and asymptotically optimal. In what follows, we denote by $\mathcal{G}_T^{n,i} = \{\mathcal{V}_T^{n,i}, \mathcal{E}_T^{n,i}, \texttt{Cost}^d\}$ the tree that has been built by robot i at the n-th iteration of the distributed Algorithm 2 for the construction of either a prefix or suffix part. Similarly, we denote by $\mathcal{G}_T^n = \{\mathcal{V}_T^n, \mathcal{E}_T^n, \texttt{Cost}^c\}$, the tree build by [12]. Also, we denote by $\text{Cost}^c(q)$ and $\text{Cost}^d(q)$ the cost assigned to a state q by the centralized algorithm [12] and the distributed Algorithm 2, respectively. To prove that Algorithm 1 is probabilistically

[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 22-23, Alg. 1]. For this reason, the construction of the trees $\mathcal{G}_T^j$ is terminated if a self-loop around q_P^r is detected.

complete and asymptotically optimal, we need first to state the following results.

Lemma 4.1 (Sampling $q_P^{new,n}$): Assume that $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$ for all $n \in \mathbb{N}_+$. Then if the centralized Algorithm 2 in [12] can sample a state $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b)) \in \mathcal{Q}_P$ then so can the proposed distributed Algorithm 2.

Using Lemma 4.1 we get the following result for $\mathcal{V}_T^{n,i}$.

Lemma 4.2 (Set of nodes $\mathcal{V}_T^{n,i}$): For any iteration $n \geq 1$ of Algorithm 2 in [12] and the proposed Algorithm 2 it holds that $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$.

Then, Lemma 4.2 yields the following two results.

Lemma 4.3 (Extend): Assume $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$, $\mathcal{E}_T^n = \cup_i \mathcal{E}_T^{n,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$. Then, after extending the trees $\mathcal{G}_T^n$ and $\cup_i \mathcal{G}_T^{n,i}$ to given a sample $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b))$, with $b \in \{1, \dots, |\mathcal{Q}_B|\}$, as per the centralized Algorithm 2 in [12] and the proposed distributed Algorithm 2, respectively, it still holds that $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$, $\mathcal{E}_T^n = \cup_i \mathcal{E}_T^{n,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$.

Lemma 4.4 (Rewire): Assume $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$, $\mathcal{E}_T^n = \cup_i \mathcal{E}_T^{n,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$. Then, after rewiring to a given a sample $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b))$, with $b \in \{1, \dots, |\mathcal{Q}_B|\}$ the nodes in $\mathcal{G}_T^n$ and $\cup_i \mathcal{G}_T^{n,i}$ to $q_P^{\text{new},n}$, as per the centralized Algorithm 2 in [12] and the proposed distributed Algorithm 2, respectively, it still holds that $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$, $\mathcal{E}_T^n = \cup_i \mathcal{E}_T^{n,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$.

Using Lemmas 4.3-4.4, we have the following result for the set of edges $\mathcal{E}_T^{n,i}$, which is then used in Theorem 4.6 to prove the completeness and optimality of Algorithm 2.

Lemma 4.5 (Set of edges $\mathcal{E}_T^{n,i}$): For any iteration $n \geq 1$ of the centralized Algorithm 2 in [12] and the proposed distributed Algorithm 2, it holds that $\mathcal{E}_T^n = \cup_i \mathcal{E}_T^{n,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$, where $q \in \mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$.

Theorem 4.6 (Completeness and Optimality): The distributed Algorithm 1 preserves the probabilistic completeness and asymptotic optimality of the centralized Algorithm 1 in [12].

Proof: The result is due to Lemmas 4.2 and 4.5, and the probabilistic completeness and asymptotic optimality of the centralized algorithm [12]. ■

V. Numerical Experiments

In this section, we present two case studies, implemented using MATLAB R2015b on a computer with Intel Core i7-2670QM 2.2GHz and 4Gb 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 $\Pi_{i=1}^N |\mathcal{Q}_i||\mathcal{Q}_B| = 4.295 \times 10^9$ 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 II. In fact, our implementation of the algorithm presented in Section II-A that relies on the explicit construction of the PBA cannot provide a plan for PBA with more than few tens of millions of states and transitions. This problem cannot be solved by the off-the-shelf model checker PRISM either, due to excessive memory requirements. 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 II-A. In this simulation study, we examine the performance of the proposed algorithm in terms of optimality. In both case studies, the weights w_i capture distance between states of wTS$_i$.

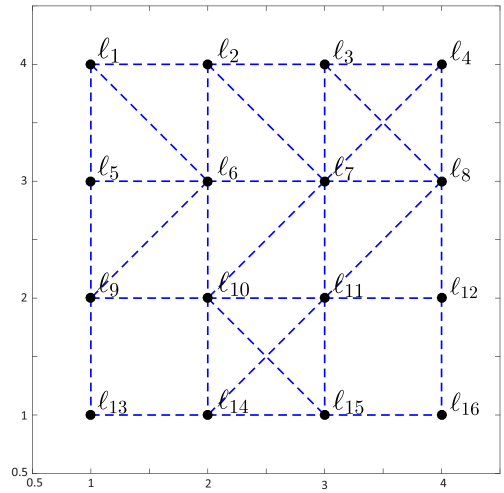


Fig. 6. Figure V-A depicts the transition systems wTS$_i$, for all robots i used in simulation study I and II. Black disks represent the states of wTS$_i$ and blue edges stand for feasible (undirected) transitions among the states. Figure 9 shows the evolution of the cost $J(\tau)$ of the resulting motion plan τ for various maximum numbers of iterations, $n_{\max}^{\text{pre}}$ and $n_{\max}^{\text{suf}}$ of Algorithm 2, for the case study II. The red line stands for the optimal cost $J^* = 14.6569$. Note that self-loops around the states are not depicted.

A. Case Study I

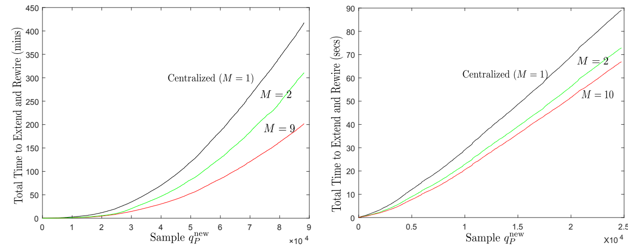
In the first simulation study, we consider a team of $N = 7$ robots. The wTS that describes the motion of each robot has $|\mathcal{Q}_i| = 16$ states and 70 transitions, including self loops around each state; see Figure V-A. The collaborative task that is assigned to the robots describes an intermittent connectivity task, defined in our previous work [22]. This intermittent connectivity requirement can be captured by a global LTL formula, which for the case study at hand takes the form $\phi = [\Box\Diamond(\pi_1^{\ell_5} \wedge \pi_2^{\ell_5})] \wedge [\Box\Diamond(\pi_2^{\ell_1} \wedge \pi_3^{\ell_1} \wedge \pi_4^{\ell_1})] \wedge [\Box\Diamond(\pi_4^{\ell_7} \wedge \pi_5^{\ell_7} \wedge \pi_6^{\ell_7})] \wedge [\Box\Diamond(\pi_6^{\ell_8} \wedge \pi_7^{\ell_8})] \wedge [\Box\Diamond(\pi_7^{\ell_{14}} \wedge \pi_2^{\ell_{14}})] \wedge [\Box\Diamond(\pi_5^{\ell_{12}})] \wedge [\neg(\pi_1^{\ell_5} \wedge \pi_2^{\ell_5}) \mathcal{U} \pi_1^{\ell_7}] \wedge [\Box((\pi_1^{\ell_5} \wedge \pi_2^{\ell_5}) \rightarrow \bigcirc(!(\pi_1^{\ell_5} \wedge \pi_2^{\ell_5}) \mathcal{U} (\pi_2^{\ell_1} \wedge \pi_3^{\ell_1} \wedge \pi_4^{\ell_1})))]$. 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 2 need to meet at location ℓ_{14}, infinitely often, (f) robot 5 needs to visit location ℓ_{12}, infinitely often (g) robots 1 and 2 should never meet at location ℓ_5 until robot 1 visits location ℓ_7 to collect some available information, and (h) once robots 1 and 2 meet at ℓ_5, they should never meet again at ℓ_5 until robots 2, 3 and 4 meet at ℓ_7. This LTL formula corresponds to a NBA with $|\mathcal{Q}_B| = 16$ states, $|\mathcal{Q}_B^0| = 1$, $|\mathcal{Q}_B^F| = 2$, and 116 transitions.

Algorithm 1 was executed over a network of $M > 1$ processors until a final state and a cycle around it are detected. When $M = 9$, subtrees $\mathcal{G}_T^i$ were built for the construction of the prefix and suffix part that satisfy $\cup_i|\mathcal{V}_T^i| = 88225$ and $\cup_i|\mathcal{V}_T^i| = 109149$, respectively. Next, we run Algorithm 1 for $M = 2$ and the centralized algorithm [12] for the sequence of samples q_P^{new} that were generated when $M = 9$. Figure 7(a) presents the the total time that the centralized algorithm [12] and the distributed Algorithm 2 for $M = 2$ and $M = 9$ have spent on extending and rewiring the trees up until the sample $q_P^{\text{new}} = (q_{\text{PTS}}^{\text{new}}, \mathcal{Q}_B(b))$ taken at iteration n for the construction of the prefix part. Observe that the distributed algorithm is at least twice as fast as the centralized algorithm and as M increases the total runtime decreases.[4] Also, the average size of subtrees assigned to each processor per iteration n of Algorithm 2, when $M = 9$, is $[0.456, 0.1156, 0.1751, 0.3199, 0.1537, 0.5173, 1.3350, 0.3353, 0.2905] \times 10^4$, while the average size of the global tree $\mathcal{G}_T = \cup_{i=1}^{9}\mathcal{G}_T^i$ per iteration n is 3.6982×10^4. The number of communication rounds per iteration n due to the distributed extend and rewire operation are shown in Figure 8; see also Remark 3.3. Observe in Figure 8, that as the size of the subtrees increases, the amount of information that robots have to exchange increases, as well. Next, given the detected final states, the construction of the suffix part follows, where similar runtimes were observed. The computation of paths over the trees associated with either the prefix or the suffix part required 0.03 seconds on average. Given the prefix and suffix part, the resulting motion plan that satisfies the considered LTL task was synthesized in 0.007 seconds and its cost is $J(\tau) = \hat{J}(\tau^{\text{pre}}) + \hat{J}(\tau^{\text{suf}}) = 123.4975 + 119.0122 = 242.5097$. Notice that the off-the-shelf model checker PRISM could not verify the considered LTL specification due to memory requirements. We also applied NuSMV to this problem that was able to generate a feasible plan in 1 second approximately with cost equal to $J(\tau) = \hat{J}(\tau^{\text{pre}}) + \hat{J}(\tau^{\text{suf}}) = 137.8406 + 137.8406 = 275.6812$ while our method found a plan with cost $J(\tau) = 242.5097$. Notice also 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 II-A failed to design a plan that satisfies the considered LTL formula and so did the algorithm presented in [18] due to excessive memory requirements.

B. Case Study II

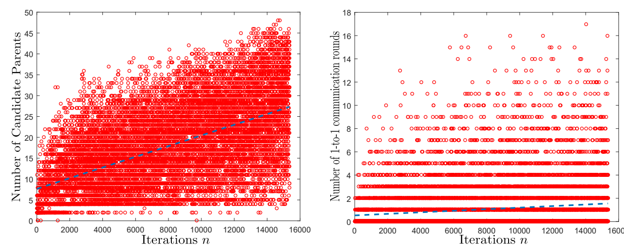
In the second simulation study, we consider a team of $N = 2$ robots with the same wTS as in the previous case study. The assigned task is expressed in the following temporal

[4]Algorithm 2 was implemented using sequential for-loops entailing that the robots extend and rewire their subtrees sequentially and not in parallel. Then, to measure the runtime of extending and rewiring the subtrees of Algorithm 2, we measure the time required by the 'slowest' robot to extend and rewire its subtree for a given sample q_P^{new}, which is reported in Figures 7(a)-7(b). Also, notice that the parallel computing toolbox of MatLab significantly slowed down our vectorized code; see e.g., `https://www.mathworks.com/help/distcomp/decide-when-to-use-parfor.html`.



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

Fig. 7. Comparison of the total time spent on extending and rewiring the graph up until the sample $q_P^{\text{new}} = (q_{\text{PTS}}^{\text{new}}, \mathcal{Q}_B(b))$ is taken at iteration n, by the centralized algorithm in [12] and the proposed distributed Algorithm 2. Algorithm 2 is executed over a network of M processors. The runtimes reported for the second simulation study pertain to the case $n_{\text{max}}^{\text{pre}} = 1500$.



(a) Extend (b) Rewire

Fig. 8. Simulation Study I: Graphical depiction of the communication rounds described in Remark 3.3 when $M = 9$ during the construction of the prefix part. Figure 8(a) depicts how many candidate parents have been proposed at iteration n in total. These candidate parents are transmitted to $M - 1$ processors. Figure 8 shows the total number of sets of nodes $\mathcal{D}_i \cup \mathcal{S}(\mathcal{D}_i)$ that are transmitted to the processor that has stored the sample q_P^{new}, at the end of the rewiring step. The blue dashed line stands for the regression line.

logic formula: $\phi = \Box\Diamond(\pi_1^{\ell_6} \wedge \Diamond(\pi_2^{\ell_{14}})) \wedge \Box(\neg\pi_1^{\ell_9}) \wedge \Box(\pi_2^{\ell_{14}} \rightarrow \bigcirc(!\pi_2^{\ell_{14}} \mathcal{U} \pi_1^{\ell_4})) \wedge (\Diamond\pi_2^{\ell_{12}}) \wedge (\Box\Diamond\pi_2^{\ell_{10}})$ 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) steps (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 II-A can be used to find the optimal plan. The cost of the optimal plan is $J^* = 14.6569$.

Algorithm 1 was run for various values of the parameters $n_{\text{max}}^{\text{pre}}$ and $n_{\text{max}}^{\text{suf}}$. Observe in Figure 9 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 4.6. 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

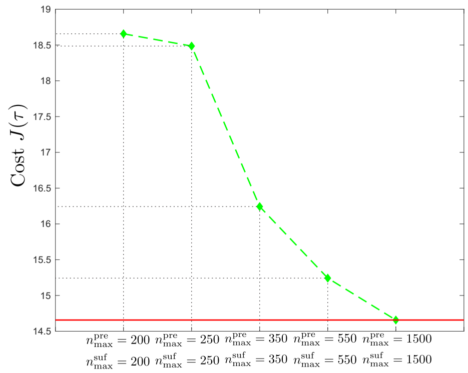


Fig. 9. Simulation Study II: Evolution of the cost $J(\tau)$ of the resulting optimal motion plan τ for various maximum numbers of iterations, $n_{\text{max}}^{\text{pre}}$ and $n_{\text{max}}^{\text{suf}}$, for Algorithm 2. The red line stands for the optimal cost J^*.

in Figure 9. Figure 7(b) shows the total time required to to extend and rewire the tree for $M=1$, $M=2$, and $M=10$, when $n_{\text{max}}^{\text{pre}}=1500$.

VI. Conclusion

In this paper we proposed the first distributed, probabilistically complete, and asymptotically optimal control synthesis algorithm for multi-robot systems under global LTL tasks. We showed through simulations that our proposed approach is computational efficient and can handle larger state-spaces than existing approaches that construct a synchronous product automaton. Future work includes experimental validation.

References

[1] 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.

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

[3] 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.

[4] 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.

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

[6] X. Ding, S. L. Smith, C. Belta, and D. Rus, "Optimal control of markov decision processes with linear temporal logic constraints," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1244–1257, 2014.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] P. Schillinger, M. Bürger, and D. Dimarogonas, "Decomposition of finite ltl specifications for efficient multi-agent planning," in *13th International Symposium on Distributed Autonomous Robotic Systems address= London, UK, year=November, 2016*.

[12] Y. Kantaros and M. M. Zavlanos, "Sampling-based optimal control synthesis for multi-robot systems under global temporal tasks," *IEEE Transactions on Automatic Control (submitted)*, 2017. [Online]. Available: https://arxiv.org/pdf/1706.04216.pdf

[13] 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.

[14] 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.

[15] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 1478–1483.

[16] 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.

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

[18] 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.

[19] Y. Kantaros and M. M. Zavlanos, "Sampling-based control synthesis for multi-robot systems under global temporal specifications," in *Cyber-Physical Systems (ICCPS), 8th International Conference on*. Pittsburgh, PA, USA: ACM/IEEE, 2017, pp. 3–13.

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

[21] 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.

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

[23] T. K. Chandra, *The Borel-Cantelli Lemma*. Springer Science & Business Media, 2012.

Appendix I
Proof of Lemmas

A. Proof of Lemma 4.1

Recall that the distribution f_{i^*} is non-zero on $[1, 2, \dots, N]$, by definition, remains the same for all iterations n, and that indepednent samples can be drawn from it. Then, using the second Borel-Cantelli lemma [23] we can show that any robot i will be selected infinitely often, with probability 1, to be the robot i^*. The proof of this part is along the lines of the proofs of Lemmas 5.4-5.5 in [12] and, therefore, is omitted. Next, following the same logic as in Corollary 5.6 in [12], we can show that show that any state $q_P \in \mathcal{G}_T^i$ can be selected to be the state q_P^{new}. Then, the result follows since $\mathcal{V}_T^n = \cup_{i=1}^N$ for all n by assumption.

Remark 1.1 (Lemma 4.1): The result shown in Lemma 4.1 holds even if the density function f_{i^*} changes with respect to iterations n, as long as it is bounded below by a sequence $\ell^n(i)$ such that $\sum_{n=0}^{\infty} \ell^n(i) = \infty$, for all i, since then the second Borel-Cantelli lemma can still be applied.

B. Proof of Lemma 4.2

To show this result we will use induction. Specifically, notice that at the beginning of iteration $n = 1$, it holds that $\mathcal{V}_T^1 = \cup_i \mathcal{V}_T^{1,i}$ due to the initialization of the set of nodes. Now, assuming that at the beginning of iteration $n \geq 1$, $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$ holds, we show that at the beginning of iteration $n+1$, $\mathcal{V}_T^{n+1} = \cup_i \mathcal{V}_T^{n+1,i}$ holds.

To show that, recall first that during iteration n, $|\mathcal{Q}_B|$ states $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b))$, with $b \in \{1, \dots, |\mathcal{Q}_B|\}$ are examined sequentially, both by the centralized and the distributed algorithm, as to whether they can be added to the tree. Let $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b))$ be the first state that is added to the set $\mathcal{V}_T^n$ for some $b \in \{1, \dots, |\mathcal{Q}_B|\}$, which can be sampled by both the centralized algorithm 2 in [12] and the distributed Algorithm 2 proposed here, as shown in Lemma 4.1. Since $q_P^{\text{new},n}$ is added to $\mathcal{V}_T^n$, it must be reachable from a state $q_P \in \mathcal{V}_T^n$ that incurs the minimum possible cost for $q_P^{\text{new},n}$. Since $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$ by assumption, it holds that $q_P \in \mathcal{V}_T^{n,s}$, for some robot s. Therefore, there is at least one candidate parent for $q_P^{\text{new},n}$ in $\cup_i \mathcal{V}_T^{n,i}$ which will be detected, since every robot i proposes a candidate parent for $q_P^{\text{new},n}$ selected from $\mathcal{V}_T^{n,i}$, by construction of Algorithm 2. Then, the distributed Algorithm 2 will select as a parent for $q_P^{\text{new},n}$ the node that results in the minimum possible cost for q_P^{new}. [5] Consequently, the state $q_P^{\text{new},n}$ will be added to the set of nodes $\mathcal{V}_T^{n,s}$, where s is determined as per lines 13-17, in Algorithm 2. All the other sets of nodes $\mathcal{V}_T^{n,i}$, with $i \neq s$ remain unaltered. Thus, after the addition of the state $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b))$, it still holds $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$. Also, if the state $q_P^{\text{new},n}$ is not added to the set $\mathcal{V}_T^n$, then it is not added to any set $\mathcal{V}_T^{n,i}$ either, since there is no candidate parent for $q_P^{\text{new},n} \in \mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$. Finally, notice that the rewiring step in both the centralized and the distributed algorithm does not affect the set of nodes. Therefore, after rewiring, it still holds that $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$. Using the same logic, we can show that this result is true for $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b+1))$ as well. Thus, we conclude that at the end of iteration n, it holds that $\mathcal{V}_T^{n+1} = \cup_i \mathcal{V}_T^{n+1,i}$ completing the proof.

C. Proof of Lemma 4.3

At iteration n, let $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b))$ be a state that is added to the set $\mathcal{V}_T^n$. Then this state $q_P^{\text{new},n}$ will be added to the set $\cup_i \mathcal{V}_T^{n,i}$, as well, due to Lemma 4.2. Now, we want to show that the edge that is constructed by the centralized Algorithm 2 in [12] is also constructed by the distributed Algorithm 2, i.e., that both algorithms select the same parent for $q_P^{\text{new},n}$. To show that, recall that by construction of the distributed Algorithm 2, the parent of a state $q_P^{\text{new},n}$ is the node $q_P^{\text{prev},j}$ that results in the minimum cost for $q_P^{\text{new},n}$, which is also the case in the centralized algorithm in [12] Since $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$ by Lemma 4.2, and $\cup_i \mathcal{E}_T^{n,i} = \mathcal{E}_T^n$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q), \forall q \in \mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$ hold by assumption, both the centralized and the distributed algorithm will select the same parent for $q_P^{\text{new},n}$. Therefore, the subtrees $\mathcal{G}_T^{n,i}$ are extended towards $q_P^{\text{new},n}$ in exactly the same way as the tree $\mathcal{G}_T^n$ does. This means that after the 'extend' operation towards $q_P^{\text{new},n}$, it still holds that $\cup_i \mathcal{E}_T^{n,i} = \mathcal{E}_T^n$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$, $\forall q \in \mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$. Note that if the state $q_P^{\text{new},n} = (q_{\text{PTS}}^{\text{new},n}, \mathcal{Q}_B(b))$ is not added to the tree constructed by the centralized algorithm then it will not be added by the distributed Algorithm 2 either, due to Lemma 4.2. In this case it is trivial to see that $\cup_i \mathcal{E}_T^{n,i} = \mathcal{E}_T^n$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$, still holds $\forall q \in \mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$ completing the proof.

[5]Note that here we do not assume that the centralized and the distributed algorithm will select the same parent node.

D. Proof of Lemma 4.4

To show this result, recall first that the only difference between the centralized Algorithm 2 in [12] and the distributed Algorithm 2 proposed here, in terms of the the rewiring step, is that the centralized algorithm rewires all nodes $q_P \in \mathcal{V}_T^n$ sequentially, while Algorithm 2, rewires all nodes $q_P \in \cup_i \mathcal{V}_T^{n,i}$ sequentially within $\mathcal{V}_T^{n,i}$ and in parallel across the sets $\mathcal{V}_T^{n,i}$. Therefore, it suffices to show that rewiring in parallel two states $q_P \in \mathcal{V}_T^{n,i}$ and $q'_P \in \mathcal{V}_T^{n,j}$, with $j \neq i$, returns the same result, as if q_P and q'_P were rewired sequentially. More specifically, we want to show that rewiring q_P does not affect the cost of q'_P and vice versa, since in this case both nodes can be rewired in parallel. To show this, we will use the following two observations throughout the proof. First, by construction of the subtrees $\mathcal{G}_T^{n,i}$ and $\mathcal{G}_T^{n,j}$, $i \neq j$, it holds that $q_P \notin \mathcal{S}(q'_P)$ and $q'_P \notin \mathcal{S}(q_P)$, for all $q_P \in \mathcal{V}_T^{n,i}$ and $q'_P \in \mathcal{V}_T^{n,j}$. Second, rewiring a node affects only the cost of all its successors or, in other words, the cost of a node is affected by rewiring one of its *predecessors* [line 7, in Algorithm 5]; note that this is the case in both the centralized and the distributed algorithm.

Let $q_P^{\text{new},n} \in \mathcal{V}_T^{n,s}$, where s can be any robot and possibly robots $i,\ j$. Also, let p be the path, i.e., the sequence of nodes in $\mathcal{V}_T^{n,s}$, that connects $q_P^{\text{new},n}$ to the root q_P^r. To show that the nodes $q_P \in \mathcal{G}_T^{n,i}$ and $q'_P \in \mathcal{G}_T^{n,j}$ can be rewired in parallel, we will consider the following cases about their existence in the path p.

Assume that neither q_P nor q'_P belong to the path p. Next we show that rewiring q_P does not affect the cost of $q'_P \in \mathcal{V}_T^{n,j}$, and vice versa, which means that both $q_P \in \mathcal{V}_T^{n,i}$ and $q'_P \in \mathcal{V}_T^{n,j}$ can be rewired in parallel. To show that, observe first that if the distributed Algorithm 2 rewires one or both of the nodes q_P and q'_P to $q_P^{\text{new},n}$, then we still have $q_P \notin \mathcal{S}(q'_P)$ and $q'_P \notin \mathcal{S}(q_P)$, since before rewiring neither of them belongs to the path p. This means that the change in the cost of q_P does not affect the cost of q'_P and vice versa, since after rewiring a node, only the cost of all its successor nodes is updated [line 7, in Algorithm 5]. Therefore, both $q_P \in \mathcal{V}_T^{n,i}$ and $q'_P \in \mathcal{V}_T^{n,j}$ can be rewired in parallel. Also, since $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$, $\mathcal{E}_T^n = \cup_i \mathcal{E}_T^{n,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$ holds by assumption, if $q_P \in \mathcal{V}_T^n$ gets rewired to $q_P^{\text{new},n}$ by the centralized Algorithm 2 in [12], then so does $q_P \in \mathcal{V}_T^{n,i}$ by the distributed algorithm. The same holds for q'_P, as well. Thus, after that rewiring $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$, $\mathcal{E}_T^n = \cup_i \mathcal{E}_T^{n,i}$, and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$ still holds.

Next, assume that either q_P or q'_P belongs to the sequence of nodes p (but not both of them). Without loss of generality, assume that q'_P belongs to p, i.e., $s = j$, which means that $q_P^{\text{new},n} \in \mathcal{S}(q'_P) \subseteq \mathcal{V}_T^{n,j}$. First, it is trivial to see that q'_P will not get rewired to $q_P^{\text{new},n}$ by either algorithms, since that would increase its cost because $q_P^{\text{new},n} \in \mathcal{S}(q'_P)$; in fact, this would also create a cycle that is disconnected from the subtree. Therefore, the cost of q'_P will not change during the rewiring step and, clearly, this cannot affect the cost of q_P. Next, we examine if the cost of q'_P can change due to possible rewiring of q_P. Specifically, if the distributed algorithm 2 rewires q_P then we have that $q_P \in \mathcal{S}(q_P^{\text{new},n}) \subseteq \mathcal{S}(q'_P)$, i.e., $q_P \in \mathcal{S}(q'_P)$ and, therefore, the cost of q'_P is not affected by that rewiring. Thus, in this case, both q_P and q'_P can be considered for rewiring in parallel, since they cannot affect the cost of each other. Also, as for q_P, we have that since $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$, $\mathcal{E}_T^n = \cup_i \mathcal{E}_T^{n,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$ holds by assumption, if $q_P \in \mathcal{V}_T^n$ gets rewired to $q_P^{\text{new},n}$ by the centralized Algorithm 2 in [12], then so does $q_P \in \mathcal{V}_T^{n,i}$ by the distributed Algorithm 2. Hence, after this rewiring step $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$, $\mathcal{E}_T^n = \cup_i \mathcal{E}_T^{n,i}$, and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$ still holds.

In the latter case, observe that if q_P gets rewired to $q_P^{\text{new},n}$, this means that $q_P \in \mathcal{S}(q_P^{\text{new},n})$ and, consequently, $\mathcal{S}(q_P) \subseteq \mathcal{S}(q_P^{\text{new},n}) \subseteq \mathcal{V}_T^{n,j}$, which means that the nodes $\{q_P\} \cup \mathcal{S}(q_P)$ should belong to robot $s = j$, since for the construction of the subtrees we require a node along with all its successors to belong to the same subtree. However, robot $s = j$ is not aware of these nodes, until all robots finish rewiring their nodes and communication between robots occurs; see lines 24-25 in Algorithm 2. In the meantime, according to Algorithm 2, robot i keeps rewiring all nodes in the set $\mathcal{S}(q_P)$ and at the end of the rewiring step, these nodes are transmitted to robot $s = j$. Note that the fact that robot i rewires a set of nodes $\mathcal{S}(q_P)$, where $\mathcal{S}(q_P) \subseteq \mathcal{S}(q_P^{\text{new},n}) \subseteq \mathcal{V}_T^{n,j}$, does not result in any inconsistency between the distributed algorithm and the centralized algorithm in terms of the set of edges and the assigned costs. The reason is that the only *predecessors* of the nodes $\mathcal{S}(q_P)$ that exist in the tree $\mathcal{G}_T^{n,j}$ belong to the path p, since the parent of q_P is the node $q_P^{\text{new},n}$, which are never rewired, as discussed before. Therefore, robot j cannot affect the cost of the nodes in $\mathcal{S}(q_P)$ and if robot i keeps rewiring the nodes $\mathcal{S}(q_P)$, this cannot affect the cost of any node in $\mathcal{V}_T^{n,j}$ that robot j is currently aware of. Thus, both the centralized and distributed algorithm perform the same rewiring steps and, therefore, $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$, $\mathcal{E}_T^n = \cup_i \mathcal{E}_T^{n,i}$, and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$ still holds. Also, notice that the case where both q_P and q'_P belong to the path p is not possible, since both nodes belong to different subtrees, by assumption. Thus, we proved that after rewiring within the subtrees, it still holds that $\mathcal{G}_T^{n,i}$, $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$, $\mathcal{E}_T^n = \cup_i \mathcal{E}_T^{n,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$ completing the proof.

E. Proof of Lemma 4.5

To prove this result we use induction. Specifically, at iteration $n = 1$, we have that $\mathcal{V}_T^1 = \cup_i \mathcal{V}_T^{1,i}$, $\mathcal{E}_T^1 = \cup_i \mathcal{E}_T^{1,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$, $\forall q \in \mathcal{V}_T^1 = \cup_i \mathcal{V}_T^{1,i}$ due to the initialization of both the centralized and the distributed algorithm. Next, assume that $q_P^{\text{new},1} = (q_{\text{PTS}}^{\text{new},1}, \mathcal{Q}_B(b))$ is the first sample that can be added to the tree $\mathcal{G}_T^1$ and $\cup_i \mathcal{G}_T^{1,i}$, with $b \in \{1, \dots, |\mathcal{Q}_B|\}$. Then, by Lemma 4.3, we have that after extending the trees $\mathcal{G}_T^1$ and $\cup_i \mathcal{G}_T^{1,i}$ to the sample $q_P^{\text{new},1} = (q_{\text{PTS}}^{\text{new},1}, \mathcal{Q}_B(b))$, with $b \in \{1, \dots, |\mathcal{Q}_B|\}$, it still holds that $\mathcal{V}_T^1 = \cup_i \mathcal{V}_T^{1,i}$, $\mathcal{E}_T^1 = \cup_i \mathcal{E}_T^{n,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$. Then, due to Lemma 4.4, we have that after rewiring to $q_P^{\text{new},1} = (q_{\text{PTS}}^{\text{new},1}, \mathcal{Q}_B(b))$, we get that $\mathcal{V}_T^1 = \cup_i \mathcal{V}_T^{1,i}$, $\mathcal{E}_T^1 = \cup_i \mathcal{E}_T^{1,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$ still holds. Following the same logic as above, we can show that $\mathcal{V}_T^1 = \cup_i \mathcal{V}_T^{1,i}$, $\mathcal{E}_T^1 = \cup_i \mathcal{E}_T^{1,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$ holds as well when the next sample $q_P^{\text{new},1} = (q_{\text{PTS}}^{\text{new},1}, \mathcal{Q}_B(b+1))$ is taken. Consequently, at the beginning of iteration $n = 2$, we have that $\mathcal{V}_T^2 = \cup_i \mathcal{V}_T^{2,i}$, $\mathcal{E}_T^2 = \cup_i \mathcal{E}_T^{2,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$, $\forall q \in \mathcal{V}_T^2 = \cup_i \mathcal{V}_T^{2,i}$. Then, the induction step follows. Specifically, assume that at iteration n, we have that $\mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$, $\mathcal{E}_T^n = \cup_i \mathcal{E}_T^{n,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$, $\forall q \in \mathcal{V}_T^n = \cup_i \mathcal{V}_T^{n,i}$. Then, using the same logic as above, we can show that $\mathcal{V}_T^{n+1} = \cup_i \mathcal{V}_T^{n+1,i}$, $\mathcal{E}_T^{n+1} = \cup_i \mathcal{E}_T^{n+1,i}$ and $\texttt{Cost}^c(q) = \texttt{Cost}^d(q)$, $\forall q \in \mathcal{V}_T^{n+1} = \cup_i \mathcal{V}_T^{n+1,i}$ holds completing the proof.

APPENDIX II
ANYTIME SAMPLING-BASED ALGORITHM

NuSMV is capable of handling large state-spaces and returning feasible, but not optimal, paths very fast. Therefore, we can initialize our trees using the prefix and suffix part returned by NuSMV and then execute Algorithm 1 to further decrease the cost of this feasible plan. Due to this initialization, Algorithm 1 can generate a feasible solution at any time. Given such a feasible solution that is generated offline, the robots can execute Algorithm 1 online to optimize the given motion plan, resulting in an *anytime* sampling-based algorithm [15]. Specifically, consider a feasible path $p = p^{\text{pre}}[p^{\text{suf}}]^\omega$, generated by Algorithm 1, that lives in $\mathcal{V}_T \subseteq \mathcal{Q}_P$ such that $\tau = \Pi|_{\text{PTS}} p \models \phi$.To improve the prefix part online, the robots execute only a segment of the prefix that involves the first k states, i.e., the path $\tau^{\text{pre}}(1 : k)$ and delete all subtrees they have constructed for the construction of p. Meanwhile, they execute Algorithm 1 to build new subtrees with the root to be the state $p^{\text{pre}}(k)$ while one of the subtrees also includes the path $p^{\text{pre}}(k+1 : \texttt{end})$, where $p^{\text{pre}}(\texttt{end})$ stands for the last state in p^{pre}. The goal set in this case includes only the final state $p^{\text{pre}}(\texttt{end})$. When the robots reach the state $p^{\text{pre}}(k)$, they check if they have found a better path to replace $p^{\text{pre}}(k+1 : \texttt{end})$. If so, they replace the part $p^{\text{pre}}(k+1 : \texttt{end})$ with the improved one. The robots repeat the same procedure for a subsequent segment of the possibly improved prefix part. This process is repeated until the robots reach the state $p^{\text{pre}}(\texttt{end})$. The same logic can be applied for the online execution and improvement of the suffix part. For instance, given the feasible prefix part p^{pre} constructed in Section V-A, that has 23 states, we let the robots execute online only the part $p^{\text{pre}}(1 : 2)$ while in the meantime they run Algorithm 1 to improve the prefix part $p^{\text{pre}}(3 : 23)$. After running Algorithm 1 for 30 seconds and constructing subtrees with $|\cup_i \mathcal{V}_T^i| = 4211$, the cost of the prefix part decreased from 123.4975 meters to 113.0122 meters.