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 built 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.
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: $x_i(t) = f_i(x_i(t), u_i(t))$, where $x_i(t)$ and $u_i(t)$ are the position and the control input associated with robot $i \in \{1, \ldots, N\}$. We assume that there are $W$ disjoint regions of interest in $\mathcal{W}$. The $j$-th region is denoted by $\ell_j$ 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 $wTS_i = (\mathcal{Q}_i, q_i^0, \rightarrow_i, w_i, AP_i, L_i)$ 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 $\ell_j$; (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 $u_i$, that can drive robot $i$ from location $\ell_j$ to $\ell_k$, then there is a transition from state $q_i^{\ell_j}$ to $q_i^{\ell_k}$ denoted by $(q_i^{\ell_j}, q_i^{\ell_k}) \in \rightarrow_i$; (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 state $q_i^{\ell_j}$ to state $q_i^{\ell_k}$; (d) $AP_i = \bigcup_{j=1}^W \{\pi_j^{\ell_j}\}$ is the set of atomic propositions, where $\pi_j^{\ell_j}$ is true if robot $i$ is inside region $\ell_j$ and false otherwise; and (e) $L_i : \mathcal{Q}_i \rightarrow AP_i$ is an observation/output function defined as $L_i(q_i^{\ell_j}) = \pi_j^{\ell_j}$, 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 wTSs, as follows [9]:

**Definition 2.2 (PTS):** Given $N$ transition systems $wTS_i = (\mathcal{Q}_i, q_i^0, \rightarrow_i, w_i, AP_i, L_i)$, the product transition system $PTS = wTS_1 \otimes wTS_2 \otimes \cdots \otimes wTS_N$ is a tuple $PTS = (\mathcal{Q}_{PTS}, q_{PTS}^0, \rightarrow_{PTS}, w_{PTS}, AP, L_{PTS})$ where (a) $\mathcal{Q}_{PTS} = \mathcal{Q}_1 \times \mathcal{Q}_2 \times \cdots \times \mathcal{Q}_N$ is the set of states; (b) $q_{PTS}^0 = (q_1^0, q_2^0, \ldots, q_N^0) \in \mathcal{Q}_{PTS}$ is the initial state, (c) $\rightarrow_{PTS} \subseteq \mathcal{Q}_{PTS} \times \mathcal{Q}_{PTS}$ is the transition relation defined by the rule $\lambda_{\mathcal{Q}_{\text{PTS}}}((q_1, \ldots, q_N))$, with slight abuse of notation $q_{PTS} = (q_1, \ldots, q_N) \in \mathcal{Q}_{PTS}, q_i \in \mathcal{Q}_i$. The state $q_{PTS}$ is defined accordingly. In words, this transition rule says that there exists a transition from $q_{PTS}$ to $q_{PTS}'$ if there exists a transition from $q_i$ to $q_i'$ for all $i \in \{1, \ldots, N\}$; (d) $w_{PTS} : \mathcal{Q}_{PTS} \times \mathcal{Q}_{PTS} \rightarrow \mathbb{R}_+$ is a cost function that assigns weights/cost to each possible transition in PTS, defined as $w_{PTS}(q_{PTS}, q_{PTS}') = \sum_{k=1}^N w_k((\Pi_{\mathcal{Q}_{\text{PTS}}} q_{PTS}, \Pi_{\mathcal{Q}_{\text{PTS}}} q_{PTS}'))$, where $q_{PTS}' \in \mathcal{Q}_{PTS}$ and $\Pi_{\mathcal{Q}_{\text{PTS}}} q_{PTS}$ stands for the projection of state $q_{PTS}$ onto the state space of wTSs. The state $\Pi_{\mathcal{Q}_{\text{PTS}}} q_{PTS} \in \mathcal{Q}_i$ is obtained by removing all states in $q_{PTS}$ that do not belong to $\mathcal{Q}_i$; (e) $AP = \bigcup_{i=1}^N AP_i$ is the set of atomic propositions; and, (f) $L_{PTS} : \mathcal{Q}_{PTS} \rightarrow AP$ is an observation/output function giving the set of atomic propositions that are satisfied at a state $q_{PTS} \in \mathcal{Q}_{PTS}$.

In what follows, we give definitions related to the PTS, that we will use throughout the rest of the paper. An infinite path $\tau$ of a PTS is an infinite sequence of states, $\tau = [\tau(1), \tau(2), \tau(3), \ldots]$ such that $\tau(1) = q_{PTS}^0$, $\tau(k) \in \mathcal{Q}_{PTS}$, and $(\tau(k), \tau(k+1)) \in \rightarrow_{PTS}, \forall k \in \mathbb{N}_+$, where $k$ is an index that points to the $k$-th entry of $\tau$ denoted by $\tau(k)$. The trace of an infinite path $\tau = [\tau(1), \tau(2), \tau(3), \ldots]$ of a PTS, denoted by

![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.](image-url)
trace(\(\tau\)) \in (2^{\text{AP}})^\omega$, where \(\omega\) denotes infinite repetition, is an infinite word that is determined by the sequence of atomic propositions that are true in the states along \(\tau\), i.e., trace(\(\tau\)) = L(\(\tau\)(1))L(\(\tau\)(2)) \ldots 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 \(w_{\text{PTS}}\) in Definition 2.2, the cost of a finite path \(\tau\), denoted by \(\tilde{J}(\tau)\), can be defined as

\[
\tilde{J}(\tau) = \sum_{k=1}^{|\tau|-1} w_{\text{PTS}}(\tau(k), \tau(k+1)),
\]

where \(|\tau|\) stands for the number of states in \(\tau\). In words, the cost (1) captures the total cost incurred by all robots during the execution of the finite path \(\tau\).

We assume that the robots have to accomplish a complex collaborative task encapsulated by a global LTL statement \(\phi\) defined over the set of atomic propositions \(\text{AP} = \bigcup_{i=1}^{\omega} \text{AP}_i\). 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 \(\phi\), we define the language \(\text{Words}(\phi) = \{\sigma \in (2^{\text{AP}})^\omega | \sigma \models \phi\}\), where \(|\sigma|\subseteq (2^{\text{AP}}) \times \phi\) is the satisfaction relation, as the set of infinite words \(\sigma \in (2^{\text{AP}})^\omega\) that satisfy the LTL formula \(\phi\). Any LTL formula \(\phi\) can be translated into a Nondeterministic Büchi Automaton (NBA) over \((2^{\text{AP}})^\omega\) denoted by \(B\) [21] defined as follows:

**Definition 2.3 (NBA):** A Nondeterministic Büchi Automaton (NBA) \(B\) over \((2^{\text{AP}})^\omega\) is defined as a tuple \(B = (Q_B, Q_B^0, \Sigma, \rightarrow_B, Q_B^\varepsilon)\), where \(Q_B\) is the set of states, \(Q_B^0\), \(Q_B^\varepsilon\) is a set of initial states, \(\Sigma = 2^{\text{AP}}\) is an alphabet, \(\rightarrow_B \subseteq Q_B \times \Sigma \times Q_B\) is the transition relation, and \(Q_B^\varepsilon \subseteq Q_B\) is a set of accepting/final states.

Given the PTS and the NBA \(B\) that corresponds to the LTL \(\phi\), we can now define the Product Büchi Automaton (PBA) \(P\) over the product states of the LTL description.

**Definition 2.4 (PBA):** Given the product transition system \(PTS = (Q_{\text{PTS}}, q_{\text{PTS}}^0, \rightarrow_{\text{PTS}}, w_{\text{PTS}}, \text{AP}, \text{L}_{\text{PTS}})\) and the NBA \(B = (Q_B, Q_B^0, \Sigma, \rightarrow_B, Q_B^\varepsilon)\), we can define the Product Büchi Automaton \(P\) over the product states of \(PTS\) as a tuple \(P = (Q_P, Q_P^0, \rightarrow_P, w_P, Q_P^\varepsilon)\) where (a) \(Q_P = Q_{\text{PTS}} \times Q_B\) is the set of states; (b) \(Q_P^0 = q_{\text{PTS}}^0 \times Q_B^0\) is a set of initial states; (c) \(\rightarrow_P \subseteq Q_P \times 2^{\text{AP}} \times Q_P\) is the transition relation defined by the rule: \(q_P = (q_{\text{PTS}}, q_B) \rightarrow_{P} q'_P = (q'_P, q_B)\). Transition from state \(q_P \in Q_P\) to \(q'_P \in Q_P\) is denoted by \(q_P, q'_P \in \rightarrow_P\), or \(q_P \rightarrow_{P} q'_P\); (d) \(w_P(q_P, q_B) = w_{\text{PTS}}(q_{\text{PTS}}, q_B)\), where \(q_P = (q_{\text{PTS}}, q_B)\) and \(q'_P = (q'_P, q_B)\); and (e) \(Q_P^\varepsilon = Q_{\text{PTS}} \times Q_B^\varepsilon\) is a set of accepting/final states.

Given \(\phi\) and the PBA an infinite path \(\tau\) of a PTS satisfies \(\phi\) if and only if trace(\(\tau\)) \(\in\) \(\text{Words}(\phi)\), which is equivalently denoted by \(\tau \models \phi\). Specifically, if there is a path satisfying \(\phi\), then there exists a path \(\tau\) that can be written in a finite representation, called prefix-suffix structure, i.e., \(\tau = \tau_{\text{pre}}[\tau_{\text{af}}]^{\omega}\), where the prefix part \(\tau_{\text{pre}}\) is executed only once followed by the indefinite execution of the suffix part \(\tau_{\text{af}}\). The prefix part \(\tau_{\text{pre}}\) is the projection of a finite path \(\tau_{\text{pre}}\) that lives in \(Q_P\) onto \(Q_{\text{PTS}}\). The path \(\tau_{\text{af}}\) starts from an initial state \(q_{\text{af}}^0 \in Q_{\text{af}}^0\) and ends at a final state \(q_{\text{af}}^\varepsilon \in Q_{\text{af}}^\varepsilon\), i.e., it has the following structure \(\tau_{\text{af}} = (q_{\text{PTS}}^0, q_B^0)(q_{\text{PTS}}, q_B)(q_{\text{PTS}}, q_B) \ldots (q_{\text{PTS}}, q_B)\) with \((q_{\text{PTS}}, q_B) \in Q_P\). The suffix part \(\tau_{\text{af}}\) is the projection of a finite path \(\tau_{\text{af}}\) that lives in \(Q_P\) onto \(Q_{\text{PTS}}\).

**Problem 1:** Given a global LTL specification \(\phi\), and transition systems \(wT_i\), for all robots \(i\), determine a discrete team plan \(\tau\) that satisfies \(\phi\), 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 \(\tau\) that satisfies \(\phi\), the PBA is viewed as a weighted directed graph \(G_{\phi} = \{V_{\phi}, E_{\phi}, w_{\phi}\}\) where the set of nodes \(V_{\phi}\) is indexed by the set of states \(Q_P\), the set of edges \(E_{\phi}\) is determined by the transition relation \(\rightarrow_{P}\), and the weights assigned to each edge are determined by the function \(w_{\phi}\). 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 \(G_T = \{V_T, E_T, \text{Cost}\}\) that approximately represent the state-space \(Q_P\) and the transition relation \(\rightarrow_P\) of the PBA defined in Definition 2.4. The root of \(G_T\) is denoted by \(q_{T}^0\). Also, the set of nodes \(V_T\) contains the states of \(Q_P\) that have already been sampled and added to the tree structure. The set of edges \(E_T\) captures transitions between nodes in \(V_T\) that satisfy the rule \(\rightarrow_{P}\). The function Cost : \(V_T \rightarrow \mathbb{R}_+\) assigns the cost of reaching node \(q_P \in V_T\) from the root \(q_{T}^0\) of the tree. In other words,
\[ \text{Cost}(q_T) = \hat{J}(\tau_T), \] where \( q_T \in V_T \) and \( \tau_T \) is the path in the tree \( G_T \) that connects the root to \( q_T \).

In this section we present a distributed implementation of the optimal control synthesis method presented in [12]. Specifically, each robot \( i \) builds a subtree \( G_T^i = \{V_T^i, E_T^i, \text{Cost}\} \subseteq G_T \), so that the subtrees are (i) disjoint, i.e., \( \bigcap_i V_T^i = \emptyset \) and \( \bigcap_i E_T^i = \emptyset \), (ii) the union of the subtrees comprises the global connected tree built by [12], i.e., \( \bigcup_i V_T^i = V_T, \bigcup_i E_T^i = E_T \), and (iii) for each \( q_T \in V_T \), except for the root \( q_T^* \) of \( G_T \), it holds that all successor nodes of \( q_T^* \) in \( V_T \), collected in the set \( S(q_T) \), belong to \( V_T^i \), 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 \( q_T^0 = (q_T^{\text{pre}}, q_B^0) \) to an accepting state \( q_T = (q_T^{\text{pre}}, q_B) \) in \( Q_T^P \), with \( q_B \in Q_B^F \), we can define the goal region for all trees \( G_T^i \) as [line 2, Alg. 1]

\[ X_{\text{goal}}^i = \{q_P = (q_T^{\text{pre}}, q_B) \in Q_P \mid q_B \in Q_B^F\}. \] (3)

The root \( q_T^* \) of the global tree \( \bigcup_i G_T^i = G_T \) is an initial state \( q_T^* = (q_T^{\text{pre}}, q_B^*) \) of the PBA and the following process is repeated for each initial state \( q_B^0 \) [lines 3-4, Alg. 1]. In line 4 of Algorithm 1, \( Q_B^0(b_0) \) stands for the \( b_0 \)-th state in the set \( Q_B^0 \), assuming an arbitrary enumeration of the elements of the set \( Q_B^0 \). Construction of the subtrees \( G_T^i \) 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 \( q_T^* \). 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 \( f_{i^*} : N_+ \to [0, 1] \), which we assume that is non-zero on \([1, 2, \ldots, N]\) 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 \( f_T : \mathcal{Z} \to \mathcal{X}_{goal}, \mathbf{wTS}_{1}, \ldots, \mathbf{wTS}_N, B, q_{pts}, n_{max} \)

1. Robot 1 samples robot \( i^* \) from \( f_{i^*} \) and propagates the result across the network;
2. \( \mathcal{V}_T = \{q_T^p\}, E_T^p = \emptyset, \text{Cost}(q_T^p) = 0; \)
3. \( E_T^v = E_T^p = \emptyset, \forall v \neq i^*; \)
4. for \( n = 1 : n_{\text{max}} \) do
5. \( q_{pts}^{new} = \text{Sample}(\mathcal{V}_T^v); \)
6. for \( b = 1 : |Q_B| \) do
7. Robot \( i^* \) creates \( q_{pts}^{new} = Q_B(b); \)
8. Robot \( i^* \) constructs \( q_{pts}^{new} = (q_{pts}^{prev}, q_{pts}^{new}); \)
9. State \( q_{pts}^{prev} \) is transmitted to all other robots \( i; \)
10. if \( q_{pts}^{prev} \notin \bigcup_v \mathcal{V}_T \) then
11. \( \text{CandidateParent}(q_{pts}^{prev}, \to P, G_T), \forall v; \)
12. if \( \{q_{pts}^{prev,1}, q_{pts}^{prev,2}, \ldots, q_{pts}^{prev,N}\} \neq \emptyset \) then
13. \( j = \arg\min\{C_{q_T}^{prev}\}; \)
14. if \( q_{pts}^{prev,j} \neq q_{pts}^{new} \) then
15. \( s = j; \)
16. else \( s = \arg\max\{M_{n}^{prev}\}; \)
17. \( \mathcal{V}_T = \mathcal{V}_T^v \cup \{q_{pts}^{new}\}; \)
18. \( E_T^v = E_T^v \cup \{q_{pts}^{prev,j}, q_{pts}^{new}\}; \)
19. \( \text{Cost}(q_T^{new}) = C_T^{new}; \)
20. if \( q_{pts}^{new} \in \mathcal{V}_T^{new}, \text{for some } n_{new} \) then
21. Robot \( n_{new} \) transmits to all other robots \( \text{Cost}(q_{pts}^{new}); \)
22. \( [E_T, \text{Cost}, D_i] = \)
23. \( \text{Rewire}(q_{pts}^{new}, \text{Cost}(q_{pts}^{new}), G_T^v), \forall v; \)
24. \( E_T^{new} = E_T^v \cup (\bigcup_{v \neq v_{new}} S(D_i)); \)
25. \( E_T^v = E_T^v \cup (\bigcup_{v \neq v_{new}} E_T^v); \)
26. Robot \( i^* \) samples a new robot \( i^* \) from \( f_{i^*} \);
27. \( \mathcal{Z}^v = \mathcal{V}_T^{v} \cap \mathcal{X}_{goal}, \forall v; \)

\( \mathcal{Z}^v = \mathcal{V}_T^{v} \cap \mathcal{X}_{goal}, \forall v; \)

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 \( G_P^1 \) are initialized as \( \mathcal{V}_P^1 = \{q_P^1\}, \mathcal{E}_P^1 = \emptyset \) while the cost of the root is zero [line 2, Alg. 2]. The set of nodes and edges for all other subgraphs \( G_P^i, i \neq i^*, \) are initially empty [line 3, Alg. 2].

2) Constructing state \( q_{pts}^{new} \): 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 \( q_{pts}^{new} = (q_{pts}^{new}, q_{pts}^{new}). \) This is achieved by the sampling function \( \text{Sample} \) [lines 5-8, Alg. 2]; see Algorithm 3. Specifically, robot \( i^* \) first creates a state \( q_{pts}^{new} = \Pi_{\mathbf{wTS}}(q_{pts}^{new}), \) where \( q_{pts}^{new} \) is sampled from a given discrete distribution \( f_{rand}(q_P|\mathcal{V}_T^{v}) : \mathcal{V}_T^{v} \to [0, 1] \) [lines 1-2, Alg. 3]. The probability density function \( f_{rand}(q_P|\mathcal{V}_T^{v}) \) defines the probability of selecting the state \( q_P^i \in \mathcal{V}_T^{v} \) as the state \( q_P^{rand} \) at iteration \( n \) of Algorithm 2 given the set \( \mathcal{V}_T^{v}. \) We make the following assumption for \( f_{rand}(q_P|\mathcal{V}_T^{v}) \) that is also made in [12].

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

Given a state \( q_{pts}^{rand} \), we define its reachable set in the PTS \( \mathcal{R}_{PTS}(q_{pts}^{rand}) = \{q_{pts} \in \mathcal{PTS} | q_{pts}^{rand} \to q_{pts} \} \) (4) i.e., \( \mathcal{R}_{PTS}(q_{pts}^{rand}) \subseteq \mathcal{PTS} \) collects all the states \( q_{pts} \in \mathcal{PTS} \) that can be reached from \( q_{pts}^{rand} \) in one hop. Then, we sample a state \( q_{pts}^{rand} \) from a discrete distribution \( f_{new}(q_{pts}|q_{pts}^{rand}) : \mathcal{PTS} \to [0, 1] \) [line 3, Alg. 3] that satisfies the following assumption that is also made in [12].

Assumption 3.2 (Probability density function \( f_{new} \)) (i) The probability density function \( f_{new}(q_{pts}|q_{pts}^{rand}) : \mathcal{PTS} \to [0, 1] \) is non-zero on \( \mathcal{R}_{PTS}(q_{pts}^{rand}) \). (ii) For a given \( q_{pts}^{rand} \), the distribution \( f_{new}(q_{pts}|q_{pts}^{rand}) \) remains the same for all iterations \( n \). (iii) Given a state \( q_{pts}^{rand} \), independent samples \( q_{pts}^{new} \) can be drawn from \( f_{new} \).

In order to build incrementally a graph whose set of nodes approximates the state-space \( Q_B \) we need to append to \( q_{pts}^{new} \) a state from the state-space \( Q_B \) of the NBA B. Let \( q_B^{new} = Q_B(b) \) [line 7, Alg. 2] be the candidate Büchi state that will be attached to \( q_{pts}^{new} \), where \( Q_B(b) \) stands for the \( b \)-th state in the set \( Q_B \) assuming an arbitrary enumeration of the elements of the set \( Q_B \). The following procedure is repeated for all \( q_B^{new} = Q_B(b) \) with \( b \in \{1, \ldots, |Q_B|\} \). First, we construct the state \( q_B^{new} = (q_{pts}^{new}, q_B^{new}) \in Q_B \) [line 8, Alg. 2]. Then, once \( q_{pts}^{new} \) 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

\( \text{Note that other sampling methods for } q_{pts}^{rand} \text{ and } q_{pts}^{new} \text{ can be employed that do not require the more strict conditions of Assumptions 3.2(ii) and 3.1(ii); see Remark 1.1 and Remark 2.2 in Appendix A, in [12]. Also, note that in order to obtain the state } q_{pts}^{new} \text{ we do not need to construct the reachable set } \mathcal{R}_{PTS}(q_{pts}^{new}). \) Instead, only reachable sets \( \mathcal{R}_{TS}(q_{pts}^{new}) \) for all robots \( i \) that collect all states that are reachable from the state \( q_{pts}^{new} = \Pi_{\mathbf{wTS}}(q_{pts}^{new}), Q \) in one hop need to be constructed. More details can be found in [12].
Algorithm 4: Function CandidateParent($q_i^{\text{new}}, \rightarrow_p, G_T$)

1. Collect in set $R_{V_T^i}(q_i^{\text{new}})$ all states $q_P \in V_T^i$ that abide by the following transition rule:

$$q_P, q_i^{\text{new}} \in \rightarrow_p;$$

2. if $R_{V_T^i}(q_i^{\text{new}}) \neq \emptyset$ then

3. $q_i^{\text{prev},i} = \text{argmin}_{q_P \in R_{V_T^i}(q_i^{\text{new}})} \{\text{Cost}(q_P) + w_{\text{pts}}(\Pi|\text{pts}q_P, \Pi|\text{pts}q_i^{\text{new}})\}$;

4. $C_{q_i^{\text{prev},i}}^{q_i^{\text{new}}} = \text{Cost}(q_i^{\text{prev},i}) + w_{\text{pts}}(\Pi|\text{pts}q_i^{\text{prev},i}, \Pi|\text{pts}q_i^{\text{new}})$;

5. return $q_i^{\text{prev},i}$.

that collects all states $q_P \in V_T^i$ that satisfy the transition rule $(q_P, q_i^{\text{new}}) \in \rightarrow_p$ [line 1, Alg. 4]. If this set $R_{V_T^i}(q_i^{\text{new}})$ is empty then robot $i$ does not propose any candidate parent for the sample $q_i^{\text{new}}$. In case $R_{V_T^i}(q_i^{\text{new}}) \neq \emptyset$, then robot $i$ proposes the state $q_i^{\text{prev},i} = \text{argmin}_{q_P \in R_{V_T^i}(q_i^{\text{new}})} \{\text{Cost}(q_P) + w_{\text{pts}}(\Pi|\text{pts}q_P, \Pi|\text{pts}q_i^{\text{new}})\}$ as the candidate parent of the sample $q_i^{\text{new}}$ [line 3, Alg. 4].

Algorithm 5: Function Rewire($q_i^{\text{new}}, V_T^i, E_T^i, \text{Cost}$)

1. Collect in set $R_{V_T^i}(q_i^{\text{new}})$ all states $q_P \in V_T^i$ that abide by the following transition rule:

$$q_P, q_i^{\text{new}} \in \rightarrow_p;$$

2. $D_i = \emptyset$;

3. for $q_P \in R_{V_T^i}(q_i^{\text{new}})$ do

4. if $\text{Cost}(q_P) > \text{Cost}(q_i^{\text{prev},i}) + w_{\text{pts}}(\Pi|\text{pts}q_i^{\text{new}}, \Pi|\text{pts}q_P)$ then

5. $E_T^i = E_T^i \setminus \{(\text{Parent}(q_P), q_P)\}$;

6. $\text{Cost}(q_P) = \text{Cost}(q_i^{\text{new}}) + w_{\text{pts}}(\Pi|\text{pts}q_i^{\text{new}}, \Pi|\text{pts}q_P)$;

7. Update the cost of all successor nodes of $q_P \in V_T^i$;

8. Update set of rewired nodes: $D_i = D_i \cup \{q_P\}$;

9. return $E_T^i$, Cost, $D_i$.

$q_i^{\text{prev},j}$ of $q_i^{\text{new}}$ is not the root $q_T^i$ of the tree, then robot $j$ will store $q_i^{\text{new}}$, i.e., $s = j$ [lines 14-15, Alg. 2]. On the other hand, if $q_i^{\text{prev},j} = q_T^i$, then the robot with the largest free memory at iteration $n$, denoted by $M^n_i \geq 0$, will store $q_i^{\text{new}}$, i.e., $s = \arg\max\{M^n_i\}$, 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 $q_P \in V_T^i$, except for the root $q_T^i$, belong to $V_T^i$, for all robots $i$. As it will be discussed later, this enables the parallel execution of the rewiring step across the subtrees $G_T^i$.

Finally, the robots coordinate to decide which robot will include the new state $q_i^{\text{new}}$ in its subtree. Hereafter, we denote by $s$ the robot that will store the sample $q_i^{\text{new}}$. If the parent $q_i^{\text{prev},j} \neq q_T^i$ of $q_i^{\text{new}}$ is not the root $q_T^i$, then robot $j$ will store $q_i^{\text{new}}$, i.e., $s = j$ [lines 14-15, Alg. 2]. On the other hand, if $q_i^{\text{prev},j} = q_T^i$, then the robot with the largest free memory at iteration $n$, denoted by $M^n_i \geq 0$, will store $q_i^{\text{new}}$, i.e., $s = \arg\max\{M^n_i\}$, 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 $q_P \in V_T^i$, except for the root $q_T^i$, belong to $V_T^i$, for all robots $i$. As it will be discussed later, this enables the parallel execution of the rewiring step across the subtrees $G_T^i$.

Fig. 3.  Graphical depiction of extending the subtrees. The blue square stands for the root $q_T^i$. The subtree $G_T^i$ consists of the green disks and edges and $G_T^j$ consists of the blue square, blue disks, and blue edges. The blue diamond stands for the state $q_T^i$. The dashed black arrow represents the new edge that will be added to the set $E_T^j$ after extending the subtrees.
4) Rewiring $G^T_i$ through $q^\text{new}_P$: The rewiring step occurs in two cases. First, it happens if the sample $q^\text{new}_P$ already belongs to the set of nodes $V^\text{new}_T$ of robot $i$. Second, it occurs if the sample $q^\text{new}_P$ does not already belong to $\cup_i V^\text{new}_T$, but there is a robot $s$, determined as per lines 16-17 in Algorithm 2, that extends its subtree towards $q^\text{new}_P$. Hereafter, with slight abuse of notations, we denote by $i^\text{new}$ the robot that has stored the sample $q^\text{new}_P$ for both cases. In both cases, robot $i^\text{new}$ transmits the cost $\text{Cost}(q^\text{new}_P)$ to all other robots $i$ [lines 21-22, Alg. 2]. Then, these robots $i$ along with robot $i^\text{new}$, check simultaneously if they can rewire the nodes $q_P \in V^\text{new}_T$ and $q_P \in V^\text{new}_T$ to the node $q^\text{new}_P \in V^\text{new}_T$ in order to decrease the cost $\text{Cost}(q_P)$ [lines 23, Alg. 2]. The rewiring process in $G^T_i$ is described in Algorithm 5 and is illustrated in Figure 4. Note that the rewiring process occurs in parallel across all subtrees $G^T_i$.

In Algorithm 5 we first construct the reachable set $R^-_{V^i_T}(q^\text{new}_P) \subseteq V^i_T$ defined as

$$R^-_{V^i_T}(q^\text{new}_P) = \{q_P \in V^i_T | q^\text{new}_P \rightarrow_P q_P\}, \quad (6)$$

that collects all states of $q_P \in V^i_T$ that satisfy the transition rule $(q^\text{new}_P, q_P) \in \rightarrow_P$, i.e., all states that can be directly reached by $q^\text{new}_P$ [line 1, Alg. 5]. Then, for all states $q_P \in R^-_{V^i_T}(q^\text{new}_P)$ we check if their current cost $\text{Cost}(q_P)$ is greater than their cost if they were connected to the root through $q^\text{new}_P$ [line 4, Alg. 5]. If this is the case for a node $q_P \in R^-_{V^i_T}(q^\text{new}_P)$, then the new parent of $q_P$ becomes $q^\text{new}_P$ 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 $\text{Cost}(q_P) = \text{Cost}(q^\text{new}_P) + w_{\text{PTS}}(\Pi[\text{PRE}(q_P))$ 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 $G^T_i$, collected in the set

$$S(q_P) = \{q^\text{new}_P \in V^i_T | q^\text{new}_P \text{ is connected to } q_P \text{ through a multi hop path in } G^T_i\}, \quad (7)$$

is updated to account for the change in the cost of $q_P$ [line 7, Alg. 5]. Also, all robots $i \neq j^\text{new}$ store the rewired nodes in the set $D_i$ [line 8, Alg. 5]. Then, after the rewiring step, robots $i \neq j^\text{new}$ send to robot $j^\text{new}$ the set of nodes $D_i \cup S(D_i)$, the set of edges among these nodes, denoted by $E^D_i \subseteq E^T_i$, and their respective costs, i.e., $V^\text{new}_T = V^\text{new}_T \cup (D_i \cup \cup_{j \neq i} (S(D_j)))$ and $E^\text{new}_T = E^\text{new}_T \cup (\cup_{j \neq i} (E^j))$ [lines 24-25, Alg. 2]. This way, we ensure that after the rewiring step it holds that if $q_P \in V^T_i$, then all nodes $q^\text{new}_P \in S(q^\text{pre}_P)$ belong to $V^T_i$, 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 $G^T_i$ ends after $n_{\text{max}}$ iterations, where $n_{\text{max}}$ is user specified [line 4, Alg. 2]. Then, every robot $i$ constructs the set $P^i = V^T_i \cap X^\text{goal}_{\text{pre}}$ [line 27, Algorithm 2] that collects all the states $q^\text{new}_P \in V^T_i$ that belong to the goal region $X^\text{goal}_{\text{pre}}$ defined in (3). Then, every robot $i$ finds paths that correspond to the prefix parts and connect the states $q^\text{new}_P \in P^i$ to the root of the tree $q^T_P$. In particular, the path that connects the $\alpha$-th state in the set $P^i$, denoted by $P^i(\alpha)$, to the root $q^T_P$ constitutes the $\alpha$-th prefix part found by robot $i$ and is denoted by $P^i_{\text{pre},P^i(\alpha)}$ [line 8, Algorithm 1]. Specifically, the prefix part $P^i_{\text{pre},P^i(\alpha)}$ is constructed by tracing the sequence of parents of nodes starting from the node that represents the accepting state $P^i(\alpha)$ and ending at the root of the tree. The parent of each node is computed by the function parent: $V^T_i \rightarrow V^T_i \cup \{q^T_P\}$. This function maps a node $q_P \in V^T_i$ to a unique vertex $q^T_P \in V^T_i$ if $(q^T_P, q_P) \in E^T_i$ and $q^T_P \neq q^T_P$, or to the root if $q^T_P = q^T_P$. By convention, we assume that parent$(q^T_P) = q^T_P$. 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 $q^\text{new}_P$ in Algorithm 3 is $O(\sum_i |Q_i|)$. Moreover, the time complexity of extending the tree $G^T = \bigcup_{i=1}^N G^T_i$ towards $q^\text{new}_P$ is $O(\max_i |V^i_T|(|N + 1))$; see Algorithm 4. Also, the time complexity of the rewiring step is $O(\max_i |V^i_T|(|N + 1))$; see Algorithm 5. Note that the time complexity of extending and rewiring $G^T$ using [12] is $O(\sum_i |V^i_T|(|N + 1)) > O(\max_i |V^i_T|(|N + 1))$. Similarly, the time complexity of finding a path from a node $q_P \in V^T_i$ to the root $q^T_P$ of $G^T$ is $O(|V^T_i|)$ while the respective time complexity of [12] is $O(|V^T_i|) > O(|V^T_i|)$. 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 $P^i_{\text{pre},P^i(\alpha)}$ are constructed, the corresponding suffix parts are constructed [lines 9-22, Alg. 1]. Specifically, given a prefix part $P^i_{\text{pre},P^i(\alpha)}$, the respective
suffix part \( \tau_j^{\text{suf}, P^{\alpha}(\alpha)} \) constructed by a robot \( j \) is a sequence of states in \( \mathcal{Q}_P \) that starts from the state \( P^{\alpha}(\alpha) \) and ends at the same state \( P^{\alpha}(\alpha) \), i.e., a cycle around state \( P^{\alpha}(\alpha) \) where any two consecutive states in \( \tau_j^{\text{suf}, P^{\alpha}(\alpha)} \) respect the transition rule \( \rightarrow_p \). To construct the suffix part \( \tau_j^{\text{suf}, P^{\alpha}(\alpha)} \), we build again trees \( G_j^{q} = \{\mathcal{V}_j, \mathcal{E}_j, \text{Cost}\} \) 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 \( P^{\alpha}(\alpha) \). The only differences are that: (i) the root of the tree is now \( q_p^{\tau} = P^{\alpha}(\alpha) \), 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 \( q_p^{\tau} = P^{\alpha}(\alpha) \), is defined as [line 12, Alg. 1]
\[
\mathcal{X}_{\text{goal}}(q_p^{\tau}) = \{ q_p = (q_{\text{RTS}}, q_B) \in \mathcal{Q}_P \mid (q_p, L(q_{\text{RTS}}), q_p^{\tau}) \in \rightarrow_p \},
\]
and, (iii) we first check if \( q_p^{\tau} \notin \mathcal{X}_{\text{goal}} \), i.e., if \( (\Pi_B q_p^{\tau}, L(q_{\text{RTS}}), q_p^{\tau}) \) and if the cost of such a self loop has zero cost, i.e., if \( w_p(q_p^{\tau}, q_p^{\tau}) = 0 \) [line 14, Alg. 1]. If (iii) holds, the construction of the subtrees \( G_j^{q} \) is trivial, since all the trees are empty except for the tree \( G_j^{q} \) that consists of only the root, and a loop around it with zero cost [line 15, Alg. 1].

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 \( G_j^{q} \) is terminated if a self-loop around \( q_p^{\tau} \) is detected.

Fig. 5. Graphical depiction of detecting cycles around a final/accepting state \( q_p^{\tau} = P^{\alpha}(\alpha) \) (black square) within a subtree \( G_j^{q} \). The red diamonds stand for a state \( q_B \in S_{\phi}^{\alpha} \). Solid red and blue arrows stand for two paths that connect the states in \( S_{\phi}^{\alpha} \) to the root \( P^{\alpha}(\alpha) \). The dashed red and blue arrows imply that a transition from a state \( q_B \in S_{\phi}^{\alpha} \) to \( P^{\alpha}(\alpha) \) are feasible according to the transition rule \( \rightarrow_p \) however, such a transition is not included in the set \( \mathcal{E}_j^{\tau} \). The two detected cycles around the accepting state \( P^{\alpha}(\alpha) \), denoted by \( \tau_j^{\text{suf}, P^{\alpha}(\alpha), 1} \) and \( \tau_j^{\text{suf}, P^{\alpha}(\alpha), 2} \) are illustrated by solid and dashed red and blue arrows, respectively.

C. Construction of Optimal Discrete Plans

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

Remark 3.3 (Communication rounds): For the distributed construction of the subtrees \( G_{j}^{q} \) three main communication rounds occur per sample \( q_p^{\text{new}} = (q_{\text{RTS}}^{\text{new}}, \mathcal{Q}_B(q_B)) \). The first one involves robot \( i^{*} \) that has to send the sample \( q_p^{\text{new}} \) to the remaining \( N-1 \) robots [line 9, Algorithm 2]. In the second communication round every robot \( i \) that can propose a candidate parent \( q_p^{\text{new}, i} \) for \( q_p^{\text{new}} \) has to send its respective cost \( C_{\text{new}}^{i} \) 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 \( \mathcal{D}_i \cup S({\mathcal{D}_i}) \subseteq \mathcal{V}_j \) to robot \( i^{\text{new}} \) that has stored \( q_p^{\text{new}} \). 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 \( \phi \) 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 \( G_{T_{n,i}}^{n,i} = \{\mathcal{V}_{T_{n}, i}, \mathcal{E}_{T_{n}, i}, \text{Cost}^{i}\} \) 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 \( G_{T_{n}} = \{\mathcal{V}_{T_{n}}, \mathcal{E}_{T_{n}}, \text{Cost}^{c}\} \), the tree build by [12]. Also, we denote by \( \text{Cost}^{i}(q) \) and \( \text{Cost}^{c}(q) \) 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
Lemma 4.1 (Sampling $q_0^{(n)}$): Assume that $\mathcal{V}_T^n = \cup_1^n \mathcal{V}_T^i$ for all $n \in \mathbb{N}_+$. Then if the centralized Algorithm 2 in [12] can sample a state $q_p^{(n)} = (q_{PTS}^{(n)}, Q_B(b)) \in Q_T$ then so can the proposed distributed Algorithm 2.

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

Lemma 4.2 (Set of nodes $\mathcal{V}_T^n$): For any iteration $n \geq 1$ of Algorithm 2 in [12] and the proposed Algorithm 2 it holds that $\mathcal{V}_T^n = \cup_1^n \mathcal{V}_T^i$.

Then, Lemma 4.2 yields the following results.

Lemma 4.3 (Extend): Assume $\mathcal{V}_T^n = \cup_1^n \mathcal{V}_T^i$, $\mathcal{E}_T^n = \cup_1^n \mathcal{E}_T^i$, and $\text{Cost}^i(q) = \text{Cost}^i(q)$. Then, after extending the trees $G_T^n$ and $\cup_1^n \mathcal{G}_T^i$ to a given a sample $q_p^{(n)} = (q_{PTS}^{(n)}, Q_B(b))$, with $b \in \{1, \ldots, |Q_B|\}$, as per the central-Algorithm 2 in [12] and the proposed distributed Algorithm 2, respectively, it still holds that $\mathcal{V}_T^n = \cup_1^n \mathcal{V}_T^i$, $\mathcal{E}_T^n = \cup_1^n \mathcal{E}_T^i$, and $\text{Cost}^i(q) = \text{Cost}^i(q)$.

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

Lemma 4.5 (Set of edges $\mathcal{E}_T^n$): For any iteration $n \geq 1$ of the central-Algorithm 2 in [12] and the proposed distributed Algorithm 2, it holds that $\mathcal{E}_T^n = \cup_1^n \mathcal{E}_T^i$ and $\text{Cost}^i(q) = \text{Cost}^i(q)$, where $q \in \mathcal{V}_T^n = \cup_1^n \mathcal{V}_T^n$.

Theorem 4.6 (Completeness and Optimality): The distributed Algorithm 1 preserves the probabilistic completeness and asymptotic optimality of the central-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 central-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 $9\times10^9$ states and transitions. 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,

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 $|Q_r| = 16$ 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 $\phi = [\square (\pi_1^i \land \pi_2^i \land \pi_3^i \land \pi_4^i) \land [\square (\pi_5^i \land \pi_6^i \land \pi_7^i) \land [\square (\pi_8^i \land \pi_9^i) \land [\square (\pi_{10}^i \land \pi_{11}^i) \land [\square (\pi_{12}^i \land \pi_{13}^i) \land [\square (\pi_{14}^i \land \pi_{15}^i) \land [\square (\pi_{16}^i \land \pi_{17}^i) \land [\square (\pi_{18}^i \land \pi_{19}^i)).$ In words, (a) robots 1 and 2 need to meet at location $\ell_5$ infinitely often, (b) robots 2, 3 and 4 need to meet at location $\ell_1$, infinitely often, (c) robots 4, 5, and 6 need to meet at location $\ell_7$, infinitely often, (d) robots 6 and 7 need to meet at location $\ell_8$, infinitely often, (e) robots 7 and 2 need to meet at location $\ell_{14}$, infinitely often, (f) robot 5 needs to visit location $\ell_{12}$, infinitely often (g) robots 1 and 2 should never meet at location $\ell_5$ until robot 1 visits location $\ell_7$ to collect some available information, and (h) once robots 1 and 2 meet at $\ell_5$, they should never meet again at $\ell_5$ until robots 2, 3 and 4 meet at $\ell_7$. This LTL formula corresponds to a NBA with $|Q_r| = 16$ states, $|Q_B| = 1$, $|Q_D| = 2$, and 166 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 $G^*_T$ were built for the construction of the prefix and suffix part that satisfy $\cup_i |V^*_i| = 88225$ and $\cup_i |V^*_f_i| = 109149$, respectively. Next, we run Algorithm 1 for $M = 2$ and the centralized algorithm [12] for the sequence of samples $q^n_P$ 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 $q^n_P = (q^n_P, Q_B(b))$ 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.\footnote{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 $q^n_P$, 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.}

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 $G_T = \cup^0_i G^*_T$ per iteration $n$ is $3.6982 \times 10^4$. 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 $J^* = 123.4975 + 119.0122 = 242.5097$. 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 $J^* = 257.6812$ while our method found a plan with cost $J^* = 242.5097$. 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 logic formula: $\phi = \Box (\pi_{l_6}^{f_6} \land \Box (\pi_{l_{14}}^{f_{14}})) \land (\neg \pi_{l_1}^{c_1} \land \Box (\pi_{l_{14}}^{f_{14}} \land (\neg \pi_{l_6}^{f_6} \land (\pi_{l_6}^{c_6} \land (\Box (\pi_{l_{12}}^{f_{12}}) \land (\Box (\pi_{l_2}^{c_2} \land (\pi_{l_{12}}^{c_{12}}))))))) \land (\pi_{l_7}^{c_7} \land (\Box (\pi_{l_2}^{c_2} \land (\Box (\pi_{l_4}^{c_4})))))))$ where the respective NBA has $|Q_B| = 24$ states with $|Q_{B_0}^f| = 1$, $|Q_{B_1}^f| = 4$, 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 $l_6$, (e) once robot 2 visits location $l_{14}$, it should avoid this area until robot 1 visits location $l_{14}$, (f) robot 2 to visit location $l_{12}$ eventually, and (g) robot 2 to visit location $l_{14}$ infinitely often. In this simulation study, the state space of the PBA consists of $\prod_{i=1}^N |Q_i||Q_B| = 6,144$ 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 $r^\text{max}_{\text{new}}$ and $r^\text{max}_{\text{alter}}$. Observe in Figure 9 that as we increase $r^\text{max}_{\text{new}}$ and $r^\text{max}_{\text{alter}}$, 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
in Figure 9. Figure 7(b) shows the total time required to extend and rewire the tree for $M = 1$, $M = 2$, and $M = 10$, when $n_{\text{pre}} = 1500$.

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


APPENDIX I

PROOF OF LEMMAS

A. Proof of Lemma 4.1

Recall that the distribution $f_{i,r}$ is non-zero on $[1, 2, \ldots, N]$, by definition, remains the same for all iterations $n$, and that independent 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 $q_T \in G_T$ can be selected to be the state $q_T^{\infty}$. Then, the result follows since $\mathbb{V}_T = \bigcup_{n=1}^{\infty}$.
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 \( V_T^1 = \cup_i V_{n,i}^1 \) due to the initialization of the set of nodes. Now, assuming that at the beginning of iteration \( n \geq 1 \), \( V_T^n = \cup_i V_{n,i}^n \) holds, we show that at the beginning of iteration \( n+1 \), \( V_T^{n+1} = \cup_i V_{n+1,i}^{n+1} \) holds.

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

C. Proof of Lemma 4.3

At iteration \( n \), let \( q_T^{new,n} = (q_{PTS}, Q_B(b)) \) be a state that is added to the set \( V_T^n \). Then this state \( q_T^{new,n} \) will be added to the set \( \cup_i V_{n,i}^n \), 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 \( q_T^{new,n} \). To show that, recall that by construction of the distributed Algorithm 2, the parent of a state \( q_T^{new,n} \) is the node \( q_T^{prev,n} \) that results in the minimum cost for \( q_T^{new,n} \), which is also the case in the centralized algorithm in [12]. Since \( V_T^n = \cup_i V_{n,i}^n \) by Lemma 4.2, and \( E_T^n = E_T^n \) and \( Cost_t(q) = Cost^d(q) \), \( \forall q \in V_T^n = \cup_i V_{n,i}^n \) hold by assumption, both the centralized and the distributed algorithm will select the same parent for \( q_T^{new,n} \). Therefore, the subtrees \( G_T^{n,i} \) are extended towards \( q_T^{new,n} \) in exactly the same way as the tree \( G_T^n \) does. This means that after the ‘extend’ operation towards \( q_T^{new,n} \), it still holds that \( U_i E_T^{n,i} = E_T^n \) and \( Cost^c(q) = Cost^d(q) \), \( \forall q \in V_T^n = \cup_i V_{n,i}^n \). Note that if the state \( q_T^{new,n} = (q_{PTS}, Q_B(b)) \) 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 \( U_i E_T^{n,i} = E_T^n \) and \( Cost^c(q) = Cost^d(q) \), still holds \( \forall q \in V_T^n = \cup_i V_{n,i}^n \) completing the proof.

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 rewrites all nodes \( q_T \in V_T^n \) sequentially, while Algorithm 2, rewrites all nodes \( q_T \in \cup_i V_{n,i}^n \) sequentially within \( V_{n,1}^n \) and in parallel across the sets \( V_{n,i}^n \). Therefore, it suffices to show that rewiring in parallel two states \( q_T \in V_{n,i}^n \) and \( q_T \in V_{n,j}^n \), with \( j \neq i \), returns the same result, as if \( q_T \) and \( q_T \) were rewired sequentially. More specifically, we want to show that rewiring \( q_T \) does not affect the cost of \( q_T \) 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 \( G_T^{n,i} \) and \( G_T^{n,j} \), it holds that \( q_T \notin S(q_T) \) and \( q_T \notin S(q_T) \), for all \( q_T \in V_{n,i}^n \) and \( q_T \in V_{n,j}^n \). 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 \( q_T^{new,n} \in V_{n,s}^n \), where \( s \) can be any robot and possibly robots \( i, j \). Also, let \( p \) be the path, i.e., the sequence of nodes in \( V_{n,s}^n \), that connects \( q_T^{new,n} \) to the root \( q_T^{new,n} \). To show that the nodes \( q_T^{new,n} \in G_T^{n,i} \) and \( q_T^{new,n} \in G_T^{n,j} \) can be rewired in parallel, we will consider the following cases about their existence in the path \( p \).

Assume that neither \( q_T \) nor \( q_T \) belong to the path \( p \). Next we show that rewiring \( q_T \) does not affect the cost of \( q_T \), and vice versa, which means that both \( q_T \in V_{n,i}^n \) and \( q_T \in V_{n,j}^n \) can be rewired in parallel. To show that, observe first that if the distributed Algorithm 2 rewrites one or both of the nodes \( q_T \) and \( q_T \) to \( q_T^{new,n} \), then we still have \( q_T \notin S(q_T) \) and \( q_T \notin S(q_T) \), since before rewiring neither of them belongs to the path \( p \). This means that the change in the cost of \( q_T \) does not affect the cost of \( q_T \) 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 \( q_T \in V_{n,i}^n \) and \( q_T \in V_{n,j}^n \) can be rewired in parallel. Also, since \( q_T \in V_{n,i}^n \) and \( q_T \in V_{n,j}^n \) can be rewired in parallel. Since the algorithm for \( q_T \), as well. Thus, after that rewiring \( V_T^n = \cup_i V_{n,i}^n \), \( E_T^n = \cup_i E_T^{n,i} \), and \( Cost^c(q) = Cost^d(q) \) 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 \( q_P^{\text{new},n} \in S(q_P^{n}) \subseteq V_T^{n,j} \). First, it is trivial to see that \( q_P \) will not get rewired to \( q_P^{\text{new},n} \) by either algorithm, since that would increase its cost because \( q_P^{\text{new},n} \in S(q_P) \); 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 \( q_P \in S(q_P^{\text{new},n}) \subseteq S(q_P) \), i.e., \( q_P \in S(q'_P) \) 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 \( V_T^n = \cup_i V_T^{n,i} \), \( E_T = \cup_i E_T^{n,i} \) and \( \text{Cost}^d(q) = \text{Cost}^d(q) \) holds by assumption, if \( q_P \in V_T^n \) gets rewired to \( q_P^{\text{new},n} \) by the centralized Algorithm 2 in [12], then so does \( q_P \in V_T^{n,i} \) by the distributed Algorithm 2. Hence, after this rewiring step \( V_T^n = \cup_i V_T^{n,i} \), \( E_T = \cup_i E_T^{n,i} \) and \( \text{Cost}^d(q) = \text{Cost}^d(q) \) still holds.

In the latter case, observe that if \( q_P \) gets rewired to \( q_P^{\text{new},n} \), this means that \( q_{\text{new},n} \in S(q_P) \) and, consequently, \( S(q_P^{\text{new},n}) \subseteq S(q_P) \), which means that the nodes \( \{q_P\} \cup S(q_P) \) 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 \( S(q_P) \) and at the end of the rewiring step, these nodes are transmitted to robot \( s = j \). Note that fact that the robot \( i \) rewires a set of nodes \( S(q_P) \), where \( S(q_P) \subseteq S(q_P^{\text{new},n}) \subseteq V_T^{n,j} \), 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 \( S(q_P) \) that exist in the tree \( G_{T}^{n,j} \) 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 \( S(q_P) \) and if robot \( i \) keeps rewiring the nodes \( S(q_P) \), this cannot affect the cost of any node in \( V_T^{n,j} \) that robot \( j \) is currently aware of. Thus, both the centralized and distributed algorithm perform the same rewiring steps and, therefore, \( V_T^n = \cup_i V_T^{n,i} \), \( E_T = \cup_i E_T^{n,i} \), and \( \text{Cost}^d(q) = \text{Cost}^d(q) \) 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 \( G_{T}^{n,i} \subseteq V_T^n = \cup_i V_T^{n,i} \subseteq V_T^{n,j} \) and \( \text{Cost}^d(q) = \text{Cost}^d(q) \) due to the initialization of both the centralized and the distributed algorithm. Next, assume that \( q_P^{\text{new},1,n} = (q_P^{\text{new},1,n}, Q_B(b)) \) is the first sample that can be added to the tree \( G_{T}^{n,1} \) and \( \cup_i G_{T}^{n,1,i} \), with \( b \in \{1, \ldots, |Q_B|\} \). Then, by Lemma 4.3, we have that after extending the trees \( G_{T}^{n,1} \) and \( \cup_i G_{T}^{n,1,i} \) to the sample \( q_P^{\text{new},1,n} = (q_P^{\text{new},1,n}, Q_B(b)) \), with \( b \in \{1, \ldots, |Q_B|\} \), it still holds that \( V_T^1 = \cup_i V_T^{1,i} \subseteq V_T^{n,1,i} \) and \( \text{Cost}^d(q) = \text{Cost}^d(q) \) holds. Then, due to Lemma 4.4, we have that after rewiring to \( q_P^{\text{new},1,n} = (q_P^{\text{new},1,n}, Q_B(b)) \), we get that \( V_T^1 = \cup_i V_T^{1,i} \subseteq V_T^{n,1,i} \) and \( \text{Cost}^d(q) = \text{Cost}^d(q) \) still holds. Following the same logic as above, we can show that \( V_T^{2} = \cup_i V_T^{2,i} \subseteq V_T^{n,2,i} \) and \( \text{Cost}^d(q) = \text{Cost}^d(q) \) holds as well when the next sample \( q_P^{\text{new},1,n} = (q_P^{\text{new},1,n}, Q_B(b+1)) \) is taken. Consequently, at the beginning of iteration \( n = 2 \), we have that \( V_T^{1} = \cup_i V_T^{1,i} \subseteq V_T^{n,1,i} \) and \( \text{Cost}^d(q) = \text{Cost}^d(q) \), \( \forall q \in V_T^{1} = \cup_i V_T^{1,i} \). Then, the induction step follows. Specifically, assume that at iteration \( n \), we have that \( V_T^n = \cup_i V_T^{n,i} \subseteq V_T^{n+1,i} \) and \( \text{Cost}^d(q) = \text{Cost}^d(q) \), \( \forall q \in V_T^{n} = \cup_i V_T^{n,i} \). Then, using the same logic as above, we can show that \( V_T^{n+1} = \cup_i V_T^{n+1,i} \subseteq V_T^{n+1,i} \) and \( \text{Cost}^d(q) = \text{Cost}^d(q) \), \( \forall q \in V_T^{n+1} = \cup_i V_T^{n+1,i} \) holds completing the proof.

**APPENDIX II**

**ANYTIME SAMPLING-BASED ALGORITHM**

NuSVM 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 NuSVM 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 \( p = p_{\text{pre}[p_{\text{aff}}]} \), generated by Algorithm 1, that lives in \( V_T \subseteq Q_P \) such that \( \tau = \Pi_{p_{\text{pre}}} = \phi \). 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 \( \tau_{\text{pre}}(1 \rightarrow k) \) 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 \( p_{\text{pre}}(k+1 : \text{end}) \). When the robots reach the state \( p_{\text{pre}}(k) \), they check if they have found a better path to replace \( p_{\text{pre}}(k+1 : \text{end}) \). If so, they replace the part \( p_{\text{pre}}(k+1 : \text{end}) \) 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 \( p_{\text{pre}}(\text{end}) \). The same logic can be applied for the online execution and improvement of the suffix part. For instance, given the feasible prefix part \( p_{\text{pre}} \) constructed in Section V-A, that has \( 23 \) states, we let the robots execute online only the part \( p_{\text{pre}}(1 \rightarrow 2) \) while in the meantime they run Algorithm 1 to improve the prefix part \( p_{\text{pre}}(3 \rightarrow 23) \). After running Algorithm 1 for 30 seconds and constructing
subtrees with $| \bigcup_i \mathcal{V}_i | = 4211$, the cost of the prefix part decreased from 123.4975 meters to 113.0122 meters.