# Fast Temporal Logic Mission Planning of Multiple Robots: A Planning Decision Tree Approach

Ziyang Chen, Zhangli Zhou, *Graduate Student Member, IEEE*, Shaochen Wang, Jinsong Li, and Zhen Kan, *Senior Member, IEEE*

***Abstract*—This letter develops a fast mission planning framework named planning decision tree (PDT), that can handle large-scale multi-robot systems with temporal logic specifications in real time. Specifically, PDT builds a tree incrementally to represent the task progress. The system states are modeled by both completion positions and times, which avoids sophisticated product automaton and significantly reduces the search space. By growing the tree from the root node to leaf nodes, PDT can be searched for mission plannings that satisfy the linear temporal logic (LTL) task. Rigorous analysis shows that the PDT based planning is feasible (i.e., the generated plan is applicable and satisfactory with respect to the LTL task) and complete (i.e, a feasible solution, if exits, is guaranteed to be found). We further show that PDT based planning is efficient, i.e., the solution time of finding a satisfactory plan is only linearly proportional to the robot numbers. Extensive simulation and experiment results demonstrate its efficiency and effectiveness.**

***Index Terms*—Linear temporal logic, mission planning, planning decision tree.**

## I. Introduction

LINEAR temporal logic (LTL), as a formal language, is being widely used to describe complex robotic tasks [1], [2], [3]. Although multi-robot systems with LTL specifications have shown great potentials, there is one main challenge - the timely planning - that limits their deployment in real-world applications. For instance, consider a search and rescue scenario after a natural disaster. It often requires a large group of robots (e.g., UAVs) to collaboratively search for the survivors and the mission should be planned fast to enable real time implementations. However, existing temporal logic mission planning approaches either cannot scale well for multi-robot or suffer from large computational cost limiting its applicability in real time. Hence, this work is particularly motivated to develop a fast mission planning framework that can handle a large multi-robot system.

The approaches to mission planning problems with LTL specifications can be centralized or distributed. In centralized approaches, automaton-based methods [4], [5], [6], [7] construct a product automaton based on the transition systems and the Büchi automaton generated by the LTL formula. Graph-search techniques are then applied over the product automaton to search for satisfactory plans. However, automaton-based approaches generally cannot deal with large-scale multi-robot systems, since the size of the product automaton grows substantially with the number of robots. Besides the group size, the workspace complexity also contributes significantly to the huge search space, limiting the efficiency of mission planning. Previous works heavily rely on the abstract discrete maps and the product automaton for mission planning, resulting in not only poor scalability but also poor real-time performance [8]. The method MT* in [9] constructs a reduced version of the product graph without computing the complete joint transition system. However, they still suffer from dimensional explosion with the increase of the number of robots. Recently, sampling-based algorithms [10], [11] have been developed, which can be applied in continuous maps. The solution can be obtained relatively faster than conventional approaches, and can be extended for reactive planning [12]. However, such methods are only probabilistically completed. Besides, due to the existence of a large number of useless samples, long exploration time is often needed.

Another mainstream centralized solutions are optimization-based methods. For instance, by formulating the mission planning problem as a mixed integer linear programming (MILP) problem, optimization-based methods have been investigated for metric interval temporal logic (MITL) mission planning [13], signal temporal logic (STL) mission planning [14], LTL mission planning [15], homogeneous multi-robot systems [16], and heterogeneous multi-robot systems [17]. However, MILP suffers from the high computational cost, limiting its applications in real time.

Different from centralized approaches, distributed mission planning of multi-robot systems has also been investigated. The ideas behind distributed planning can be classified as either task decomposition-based [18], [19], [20] or task coupling-based methods [21], [22], [23]. By task decomposition each robot can be planned independently, while the task coupling methods directly construct local formulas to enable global mission objectives. Distributed methods have good scalability and are suitable for real-time planning. However, the above decomposition and coupling methods have to be carefully designed case by case and need to additionally satisfy a number of restrictive hypothesis to ensure the validity and avoid deadlocks.

Manuscript received 5 January 2024; accepted 7 May 2024. Date of publication 15 May 2024; date of current version 22 May 2024. This letter was recommended for publication by Associate Editor E. Pastore and Editor C. Yan upon evaluation of the reviewers' comments. This work was supported by the National Natural Science Foundation of China under Grant U2013601 and Grant 62173314. *(Corresponding author: Zhen Kan.)*

The authors are with the Department of Automation, University of Science and Technology of China, Hefei 230026, China (e-mail: zkan@ustc.edu.cn).

Digital Object Identifier 10.1109/LRA.2024.3401166

In this letter, we develop a fast mission planning framework named planning decision tree (PDT) that can handle large-scale multi-robot systems in real time. Specifically, by considering the LTL task specifications and environment, the PDT builds a tree incrementally to represent the task progress and system states. The PDT based mission planning framework has the following advantages. First, it is applicable to large-scale multi-robot systems. To avoid the use of product automaton, the system states are modeled by both the completion position and time of each agent, which greatly reduce the search space of the multi-agent system. Such a design can deal with mission planning with multiple orders of magnitude more robots than those that existing methods can manipulate. As shown in simulation, the PDT based framework can deal with $10^4$ robots and beyond. Second, it can achieve fast planning in real time. Leveraging the tree structure, the developed PDT encodes the task and system states in a hierarchical tree, which is built incrementally during mission operation and system states iteration. Therefore, PDT based planning framework shows significant savings in terms of not only memory used to save the runtime data, but also the computational cost. The pruning in PDT can further reduce the complexity. Although only the locally optimal plan can be obtained with pruning, rigorous analysis shows that the PDT based planning is feasible (i.e., the generated plan is applicable and satisfactory with respect to the LTL task) and complete (i.e, a feasible solution, if exits, is guaranteed to be found). We further show that PDT based planning is efficient, i.e., the solution time of finding a satisfactory plan is only linearly proportional to the robot numbers. Extensive simulation and experiment results demonstrate the effectiveness of the proposed planning framework.

Notations: let $\mathbb{Z}_{\geq 0}$, $\mathbb{N}$, $\mathbb{R}^+$, and $[N]$ denote the set of non-negative integers, the set of natural numbers, the set of positive real numbers, and the shorthand notation for $\{1, \ldots, N\}$, respectively. Given a set $A$, denote by $|A|$ and $2^A$ the cardinality and power set of $A$, respectively. Given a sequence $\boldsymbol{\sigma} = \sigma_0\sigma_1 \ldots$, denote by $\boldsymbol{\sigma}[j \ldots] = \sigma_j\sigma_{j+1} \ldots$ and $\boldsymbol{\sigma}[\ldots j] = \sigma_0 \ldots \sigma_j$. Let $A \setminus B$ be the set difference.

## II. Preliminaries

An LTL formula is built on a set of atomic propositions, standard Boolean operators such as $\wedge$ (conjunction), $\neg$ (negation), and temporal operators such as $X$ (next), and $\mathcal{U}$ (until). Given a set of atomic propositions $AP$, the syntax of an LTL formula $\phi$ is defined as

$$\phi := ap \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid X\phi \mid \phi_1 \mathcal{U} \phi_2, \tag{1}$$

where $ap \in AP$ represents the atomic proposition that can be either true or false. $X\phi$ means $\phi$ is true at the next moment, and $\phi_1 \mathcal{U} \phi_2$ means $\phi_1$ is true until $\phi_2$ becomes true. Other propositional logic operators such as $\vee$ (disjunction), $\rightarrow$ (implication), and temporal operators such as $G$ (always) and $F$ (eventually) can also be defined [24].

The semantics of an LTL formula are defined over an infinite sequence $\boldsymbol{\sigma} = \sigma_0\sigma_1 \ldots$ with $\sigma_i \in 2^{AP}$, $i \in \mathbb{Z}_{\geq 0}$, where $2^{AP}$ represents the power set of $AP$. Denote by $\boldsymbol{\sigma} \models \phi$ if the word $\boldsymbol{\sigma}$ satisfies the LTL formula $\phi$. An LTL formula can be translated to a nondeterministic Büchi automaton (NBA).

*Definition 1:* An NBA is a tuple $B = (S, S_0, \Delta, \Sigma, \mathcal{F})$, where $S$ is a finite set of states; $S_0 \subseteq S$ is the set of initial states; $\Sigma = 2^{AP}$ is the input alphabet; $\Delta : S \times S \rightarrow 2^{\Sigma}$ is the transition function; and $\mathcal{F} \subseteq S$ is the set of accepting states.

Given a sequence of input $\boldsymbol{\sigma} = \sigma_0\sigma_1\sigma_2 \ldots$ over $\Sigma$, a run of $B$ generated by $\boldsymbol{\sigma}$ is an infinite sequence $\boldsymbol{s} = s_0 s_1 s_2 \cdots$ where $s_0 \in S_0$ and $\sigma_{i+1} \in \Delta(s_i, s_{i+1})$, $i \in \mathbb{Z}_{\geq 0}$. If the input $\boldsymbol{\sigma}$ can generate at least one run $\boldsymbol{s}$ that intersects the accepting states $\mathcal{F}$ infinitely many times, $B$ is said to accept $\boldsymbol{\sigma}$. For any LTL formula $\phi$ over $\Pi$, one can construct an NBA with input alphabet $\Sigma$ accepting all and only words that satisfy $\phi$. To convert an LTL formula to an NBA, readers are referred to [25] for algorithms and implementations.

## III. Problem Formulation

Consider a multi-robot system $R = \{r_1, \ldots, r_{n_a}\}$, where $n_a$ is the number of robots. The robots operate in a bounded workspace $M$, which contains $n_M \in \mathbb{N}$ non-overlapped regions of interest. Denote by $M_i$ the $i$th region of interest and denote by $M_{NI}$ the no-fly region (e.g., obstacles or the regions that the robots can not traverse or operate within), where $M_i \cap M_j = \emptyset$ and $M_i \cap M_{NI} = \emptyset$ with $\forall i \neq j$ and $i, j \in [n_M]$. For each position $p \in M$, the labeling function $L : M \rightarrow AP$ maps $p$ to the corresponding $ap \in AP$, i.e., $L(p) = ap$. Let $p_i(t) \in M$ and $v_i(t) \in \mathbb{R}$ denote the position and velocity of $r_i$ at time $t \in \mathbb{R}$, respectively. Following the works of [11], [26], [27], the agents' tasks are pre-assigned via LTL-based specifications.

*Definition 2:* The abstract task system (ATS) is defined as a tuple $T = (Q, AP, M, R, LA, LM, LR)$, where $Q$ is a finite set of abstract sub-tasks, $M$ is the workspace, $LA : Q \rightarrow AP$ is a labeling function that indicates the atomic proposition associated with the sub-task $q \in Q$, $LM : Q \rightarrow M$ is a labeling function that maps $q \in Q$ to a position $p \in M$, i.e. $LM(q) = p$, and satisfies $L(LM(q)) = LA(q)$, $LR : Q \rightarrow 2^R$ is a labeling function that maps $q \in Q$ to the set of required robots $R' \subseteq R$, i.e., $LR(q) = R'$.

The ATS $T$ in Definition 2 is developed to abstract the mission in $M$ into a set of sub-tasks. That is, each $q \in Q$ uniquely corresponds to an atomic proposition $LA(q)$, which is executed at $LM(q)$ in the workspace and the robots in $LR(q)$ get involved in performing the sub-task $q$.

*Example 1:* To illustrate the construction of ATS, consider a group of 3 robots (e.g., crazyflie) operating in the environment as shown in Fig. 1 . Suppose the mission first requires robot $r_1$ to visit the red region and robots $r_2$ and $r_3$ to visit the blue region. Then, robot $r_3$ needs to visit the green region and robots $r_1$ and $r_2$ need to visit the yellow region. Finally, all robots return to white region. Such a mission can be represented as $\phi = \phi_1 \wedge \phi_2 \wedge \phi_3$, where $\phi_1 = ((\neg ap_2 \wedge \neg ap_4)\mathcal{U} ap_3) \wedge ((\neg ap_2 \wedge \neg ap_4)\mathcal{U} ap_1)$, $\phi_2 = ((\neg ap_5)\mathcal{U} ap_2) \wedge ((\neg ap_5)\mathcal{U} ap_4)$, and $\phi_3 = Fap_5$. Since there are five areas of interest, we define $Q = \{q_1, q_2, q_3, q_4, q_5\}$, where $LA(q_i) = ap_i$, $i \in [5]$, $LR(q_1) = \{r_1\}$, $LR(q_2) = \{r_2, r_3\}$, $LR(q_3) = \{r_1, r_2\}$, $LR(q_4) = \{r_3\}$,

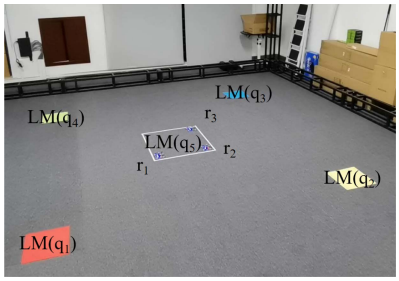



Fig. 1. The environment contains 3 robots $r_i$, $i \in [3]$, and 5 areas of interest labeled as sub-tasks $q_i$, $i \in [5]$.

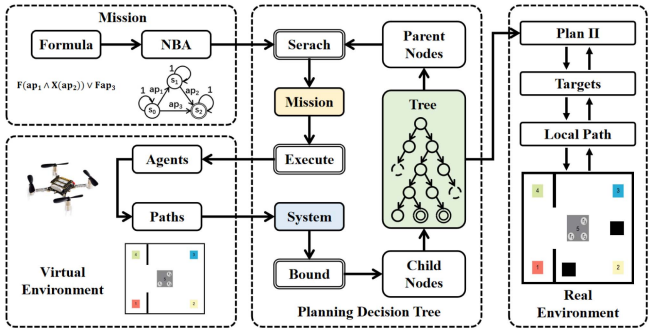



Fig. 2. The mission planning framework. The node consists of mission-related and system-related attributes. Each node searches child nodes based on sub-tasks and obtains the system states by iterations. After pruning by function Bound, child nodes are added into the tree. The robot will perform the task according to the obtained plan and adapt the path to the real environment.

$LR(q_5) = \{r_1, r_2, r_3\}$. The mappings $LM(q_i)$, $i \in [5]$, are shown in Fig. 1.

Given the workspace $M$ and the NBA $B_\phi$ generated by the LTL formula $\phi$, the plan is defined as $\Pi = (\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{\pi})$, where $\boldsymbol{q} = q_0 q_1 q_2 \ldots$ is the sequence of sub-tasks in $T$ with $q_i \in Q$, $\boldsymbol{\pi} = \pi_0 \pi_1 \pi_2 \ldots$ is the sequence of atomic proposition with $\pi_i = LA(q_i) \in AP$, and $\boldsymbol{s} = s_0 s_1 s_2 \ldots$ is the sequence of NBA states, where $s_i$ is the NBA state after completing at $\pi_i$. By denoting $\Pi_i = (q_i, s_i, \pi_i)$, $i \in \mathbb{Z}_{\geq 0}$ as a plan tuple, we can rewrite $\Pi = \Pi_0 \Pi_1 \Pi_2, \ldots$. The plan $\Pi = (\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{\pi})$ is said to satisfy the formula $\phi$, denoted as $\Pi \models \phi$, if $\boldsymbol{\pi} \models \phi$ with $LA(q_i) = \pi_i$ and $\pi_i \in \Delta(s_{i-1}, s_i)$, $\forall i \in \mathbb{Z}_{>0}$. In this work, the plan $\Pi$ is called feasible and satisfactory for $\phi$ if $\Pi \models \phi$. Based on the prefix-suffix structure [11], the plan $\Pi$ can be written in the form of $\Pi = \Pi_{pre} \Pi_{suf} \Pi_{suf} \ldots$. $\Pi_{pre}$ is the prefix part starting from an initial state and ending at an accepting. $\Pi_{suf}$ is the suffix part with the same starting and ending NBA state, which can be applied to construct a infinite word intersecting the accepting states $\mathcal{F}$ infinitely many times. Therefore, we only need to determine $\Pi_{pre}$ and $\Pi_{suf}$, which can construct an infinite plan $\Pi = \Pi_{pre} \Pi_{suf} \Pi_{suf} \ldots \models \phi$. Let $\Pi_{finite} = \Pi_{pre} \Pi_{suf}$ denoted a finite plan and the cost of $\Pi_{finite}$, denoted as $C(\boldsymbol{q})$ with $\boldsymbol{q} \in \Pi_{finite}$, is defined as the total operation time in completing the sub-tasks $\boldsymbol{q}$. Specifically, the cost of $\boldsymbol{q}$ up to the index $j$ is defined as

$$C(\boldsymbol{q}[\ldots j]) = \max\{C(\boldsymbol{q}[\ldots j-1]), t_j\} \tag{2}$$

where $t_j$ indicates the completion time for sub-task $q_j$, which satisfies $\forall r_i \in LR(q_j)$, $p_i = LM(q_j)$. Based on the defined cost, the problem is formulated as follows.

*Problem 1:* Given the map $M$, an LTL formula $\phi$, and the multi-robot system $R$, the goal is to develop a task-level plan $\Pi_{finite} = \Pi_{pre} \Pi_{suf}$ that satisfies $\phi$ while minimizing the cost $C(\boldsymbol{q})$, $\boldsymbol{q} \in \Pi_{finite}$.

## IV. Mission Planning

Since the tree-based methods such as map sampling [12] or automaton sampling [11] need to incrementally construct a dense transition system, they are limited in practice in the sense that the search space can grow substantially with the increase of agent numbers and the workspace size. To address this issue, a planning decision tree based mission planning framework is developed in this work, as shown in Fig. 2. The PDT encodes the automaton states generated by the LTL task $\phi$ and the system states in a hierarchical tree structure. By growing the tree from the root node to leaf nodes, PDT can be searched for plan $\Pi$ that satisfies the LTL task $\phi$. It is worth pointing out that the PDT based approach in this work only searches $q \in Q$, which is only related to the areas of interest. Therefore, the map size has limited influence on the algorithm complexity. In addition, the planning relies on the predicted time and positions of each robot, which can be obtained independently without searching the product automaton. Therefore, the algorithm complexity only increases linearly with the number of robots, which can be used for large-scale robots in real time.

### A. Planning Decision Tree (PDT)

As the basis of the developed mission planning framework, the PDT is defined as follows.

*Definition 3:* The planning decision tree $\mathcal{T}_\mathcal{D}$ is constructed based on a set of nodes $\{d_i\}$, $i \in \mathbb{Z}_{\geq 0}$, where $d_0$ represents the root. Each node in $\mathcal{T}_\mathcal{D}$ is defined as a tuple

$$d_i = \left(s_i^T, q_i^T, \pi_i^T, ET_i, EP_i\right)$$

where

- $s_i^T \in S$ denotes the automaton state of node $i$;
- $q_i^T \in Q$ denotes the sub-task of node $i$;
- $\pi_i^T \in AP$ denotes the atomic proposition that satisfies $\pi_i^T = LA(q_i^T)$;
- $ET_i = \{t_1^{pre}, t_2^{pre}, \ldots, t_{n_a}^{pre}\}$ denotes the predicted task completion time where $t_j^{pre}$, also denoted as $ET_i(j)$, is the predicted time that $r_j$ completes its previous sub-tasks from $d_0$ to $d_i$;
- $EP_i = \{p_1^{pre}, p_2^{pre}, \ldots, p_{n_a}^{pre}\}$ denotes the predicted robot locations for the previous task, where $p_j^{pre} \in M$, also denoted as $EP_i(j)$, is the predicted location of $r_i$ to complete its previous sub-tasks from $d_0$ to $d_i$.

To avoid notational confusion, the superscript $T$ in $s_i^T$, $q_i^T$, and $\pi_i^T$ indicates the states in $\mathcal{T}_\mathcal{D}$. The parent node of $d_i$ is denoted as $\mathsf{Ori}(d_i)$, i.e., the predecessor. Let $\mathsf{Nodes}(d_i)$ denote a sequence of nodes from the root $d_0$ to $d_i$ in $\mathcal{T}_\mathcal{D}$, i.e., $\mathsf{Nodes}(d_i) = d_{i_0} d_{i_1} \ldots d_{i_n}$ is a sequence of nodes satisfying $d_{i_0} = d_0$, $d_{i_n} =$

**Algorithm 1:** Planning Decision Tree.

**Input:** $\phi$, $M$, $d_0$
**Output:** $\Pi_{pre}\Pi_{suf}$
1 Convert $\phi$ to NBA $B_\phi$;
2 Construct the abstract task $T$ based on $M$ and $\phi$;
3 Initialize the $\mathcal{T}_\mathcal{D} = \{d_0\}$;
4 **while** $True$ **do**
5  **if** $\mathsf{Tra}(d) = 1$, $\forall d \in \mathcal{T}_\mathcal{D}$, **then**
6  break;
7  **for** $d_i \in \mathcal{T}_\mathcal{D}$ *s.t.* $\mathsf{Tra}(d) = 0$ **do**
8  $d^i_{sub} = \mathsf{Generation}(d_i, B_\phi, T)$;
9  $[d^i_{sub}, \mathcal{T}_\mathcal{D}] = \mathsf{Bound}(d^i_{sub}, \mathcal{T}_\mathcal{D})$;
10  Add $d^i_{sub}$ to $\mathcal{T}_\mathcal{D}$;
11  Set $\mathsf{Tra}(Nd) = 1$;
12  **end**
13 **end**
14 $\Pi_{pre}\Pi_{suf} = \mathsf{Get\_Plan}\,(\mathcal{T}_\mathcal{D})$;
15 Return $\Pi_{pre}\Pi_{suf}$;

$d_i$, and, $\forall j \in [n]$, $d_{i_{j-1}} = \mathsf{Ori}(d_{i_j})$. The function Plan maps the node sequence to a plan, i.e., $\mathsf{Plan}(\mathsf{Nodes}(d_i)) = \Pi_0\Pi_1 \dots \Pi_n$, where $\Pi_j = (q^T_{i_j}, s^T_{i_j}, \pi^T_{i_j})$, $j \in [n]$, is a plan tuple of node $d_{i_j} \in \mathcal{T}_\mathcal{D}$. Due to the prefix and suffix stages of the plan, the function $\mathsf{Prog}(d_i) \in \{\mathtt{pre}, \mathtt{oth}\} \cup \mathcal{F}$ is developed to record the task stage of the plan tuple $(q^T_i, s^T_i, \pi^T_i)$. Specially, $\mathsf{Prog}(d_i) = \mathtt{pre}$ indicates the plan is in the prefix stage, $\mathsf{Prog}(d_i) = s$ indicates the plan is in the suffix stage with starting state $s$, and $\mathsf{Prog}(d_i) = \mathtt{oth}$ indicates the completion of first suffix stage. Let $\mathsf{PreS}(d_i)$ denote the set of automaton states of the nodes in the same task stage, i.e., $s^T_j \in \mathsf{PreS}(d_i)$ if $d_j \in \mathsf{Nodes}(d_i)$ and $\mathsf{Prog}(d_i) = \mathsf{Prog}(d_j)$, which can be used to avoid repeated exploration. To address Problem 1, we further define $C_T(d_i) \triangleq \max\{ET_i\}$ to indicate the cost from $d_0$ to $d_i$.

The PDT $\mathcal{T}_\mathcal{D}$ can be constructed by expanding from the root node and taking into account the LTL specifications and the iterations of system states. A satisfactory plan is then generated by tracing back from the leaf node with the least cost to the root node. The general idea is outlined in Algorithm 1. Given the NBA $B_\phi$, the workspace $M$, and the ATS $T$, the tree is initialized by the root node $d_0$. Specifically, its mission states are set as $s^T_0 = s_0 \subseteq S_0$, $q^T_0 = q_0$, $\pi^T_0 = \pi_0 = \emptyset$, and $\mathsf{Prog}(d_0) = pre$. To avoid unnecessary expansion of the tree, $\mathsf{Tra}(d_i) \in \{0, 1\}$ is defined to indicate whether $d_i$ can generate its child nodes and $\mathsf{Tra}(d_0) = 0$ by default. The $EP_0$ and $ET_0$ are set according to the initial system states. In lines 4–13, if there exists a node $d_i$ satisfying $\mathsf{Tra}(d_i) = 0$, the tree is expanded by adding new child nodes in the set $d^i_{sub}$ generated by the function Generation, where the function Bound is incorporated to reduce the tree size by identifying the nodes that do not contribute to the mission planning. After all child nodes are added to the tree, the traversal flag is set to $\mathsf{Tra}(d_i) = 1$, which indicates that it has been traversed and will not generate child nodes any more. The tree stops expanding until all nodes have been traversed. Finally, the node completing the first suffix stage with the minimum cost, denoted as $d_{\min}$, is selected. The plan can then be obtained by $\Pi = \mathsf{Plan}(\mathsf{Nodes}(d_{\min}))$ and divided into $\Pi_{pre}\Pi_{suf}$ by $\mathsf{Prog}(d_i)$ for each $d_i \in \mathsf{Nodes}(d_{\min})$.

### B. Tree Expansion

The tree $\mathcal{T}_\mathcal{D}$ expands by generating feasible child nodes. Each child node contains the mission states, the system states. The mission states indicate the mission progress, which guides the execution of sub-tasks. The system states are updated based on the mission states and previous system states, which can greatly reduce the search space. The evaluation states are updated based on the system states to reduce the tree expansion.

Given a parent node $d_i$, the function Generation in Algorithm 2 is developed to generate its set of child nodes $d^i_{sub}$. Specifically, after initialization (line 1), we first identify all feasible automaton states $s$ and the next sub-tasks $q$ that satisfy $s \in S \setminus \mathsf{PreS}(d_i)$ and $LA(q) \in \Delta(s^T_i, s)$ (lines 2–3). By the selected state $s$ and sub-task $q$, the child node $d_{sub}$ is created and initialized as $q^T_{sub} = q$, $s^T_{sub} = s$, $\pi^T_{sub} = LA(q)$, $\mathsf{Ori}(d_{sub}) = d_i$, and $\mathsf{Tra}(d_{sub}) = 0$. The predicted completion system states, i.e., $EP_{sub}(j)$ and $ET_{sub}(j)$ of robot $r_j$, are updated based on the involved robots $LR(q)$ and the location of current sub-task $LM(q)$. Specifically, for each robot $r_j \in LR(q)$, we set the predicted location as $EP_{sub}(j) = LM(q)$ and the predicted arrival time as $ET_{arr}(j) = ET_i(j) + \frac{1}{v_j}\mathsf{Length}(\mathcal{P})$, where $\mathcal{P}$ is the path from $EP_i(j)$ to $LM(q)$ which can be obtained by existing path planners (e.g., PRM, RRT, RRT*) and $\frac{1}{v_j}\mathsf{Length}(\mathcal{P})$ indicates the predicted arrival time for the current sub-task $q$. For other robots $r_j \notin LR(q)$, $EP_{sub}(j)$ and $ET_{arr}(j)$ are set the same as $EP_i(j)$ and $ET_i(j)$ of their parent node $d_i$. Since the current sub-task requires the arrival of all involved robots after completing their previous sub-tasks, the predicted completion time for the agent $r_j \in LR(q)$ is designed as (3). As there exists $C_T(d_i) \le \max_{r_k \in R}\{ET_{arr}(k)\}$, (3) can be simplified as (4).

$$ET_{sub}(j) = \max\left\{\max_{r_k \in LR(q)}\{ET_{arr}(k)\}, C_T(d_i)\right\} \tag{3}$$

$$ET_{sub}(j) = \max_{r_k \in R}\{ET_{arr}(k)\} \tag{4}$$

The cost is set as $C_T(d_{sub}) = \max_{j \in [n_a]}\{ET_{sub}(j)\}$, which indicates the total completion time for all sub-tasks from $d_0$ to $d_{sub}$. The task stage $\mathsf{Prog}(d_{sub})$ indicates the current stage of the task and evolves according to

$$\mathsf{Flag}(\mathsf{Prog}, s) = \begin{cases} \mathsf{Prog}, & \text{if } s \notin \mathcal{F}, \\ s, & \text{if } s \in \mathcal{F}, \mathsf{Prog} = \mathtt{pre}, \\ \mathtt{oth}, & \text{if } \mathsf{Prog} = s. \end{cases} \tag{5}$$

Note that each NBA state can appear only once in one mission stage by line 2, which can reduce the expansion of the parent node $d_i$. To avoid the same NBA states appearing in one mission stage, the $\mathsf{PreS}(d_{sub})$ is updated based on $\mathsf{Prog}(d_{sub})$. If $\mathsf{Prog}(d_{sub})$ changes, we set $\mathsf{PreS}(d_{sub}) = \emptyset$, and $\mathsf{PreS}(d_{sub}) = \mathsf{PreS}(d_i) \cup \{s^T_{sub}\}$ otherwise. Finally, $d_{sub}$ is added to the set of child nodes $d^i_{sub}$.

In Algorithm 2, the system states are updated based on the NBA state $s^T_{sub}$ and the sub-task $q^T_{sub}$. Such a design can not only enable the robot to select feasible actions that satisfy the LTL mission, but also avoid the complex representation of system states by directly obtaining the current robot state in the search.

**Algorithm 2:** Generation.

**Input:** $d_i$, $B_\phi$, $T$
**Output:** $Children$

Initialize the set of child nodes $d^i_{sub} = \emptyset$;
**for** $s \in S \setminus \mathsf{PreS}(d_i)$ **do**
  **for** $q$ *s.t.* $LA(q) \in \Delta\left(s_i^T, s\right)$ **do**
    Initialize the child node $d_{sub} = (s^T_{sub}, q^T_{sub}, \pi^T_{sub}, ET_{sub}, EP_{sub})$;
    Set $q^T_{sub} = q$, $s^T_{sub} = s$, $\pi^T_{sub} = LA(q)$;
    Set $\mathsf{Ori}(d_{sub}) = d_i$, $\mathsf{Tra}(d_{sub}) = 0$;
    **for** $r_j \in R$ **do**
      **if** $r_j \in LR(q)$ **then**
        $EP_{sub}(j) = LM(q)$;
        $P = \mathsf{Path}(EP_i(j), LM(q))$;
        $ET_{arr}(j) = ET_i(j) + \frac{1}{v_j}\mathsf{Length}(\mathcal{P})$;
      **else**
        $EP_{sub}(j) = EP_i(j)$;
        $ET_{arr}(j) = ET_i(j)$;
    **end**
    **for** $r_j \in R$ **do**
      **if** $r_j \in LR(q)$ **then**
        $ET_{sub}(j) = \max_{k\in[n_a]}\{ET_{arr}(k)\}$;
      **else**
        $ET_{sub}(j) = ET_i(j)$;
    **end**
    Set $C_T(d_{sub}) = \max_{j\in[n_a]}\{ET_{sub}(j)\}$;
    Set $\mathsf{Prog}(d_{sub}) = \mathsf{Flag}\left(\mathsf{Prog}(d_i), s\right)$;
    **if** $\mathsf{Prog}(d_{sub}) = \mathsf{Prog}(d_i)$ **then**
      Set $\mathsf{PreS}(d_{sub}) = \mathsf{PreS}(d_i) \cup \{s^T_{sub}\}$;
    **else**
      Set $\mathsf{PreS}(d_{sub}) = \emptyset$;
    Add $d_{sub}$ to $d^i_{sub}$;
  **end**
**end**
Return $d^i_{sub}$;

**Algorithm 3:** Bound.

**Input:** $d^i_{sub}$, $\mathcal{T}_\mathcal{D}$
**Output:** the updated $d^i_{sub}$ and $\mathcal{T}_\mathcal{D}$

1 **for** $d_{sub}$ *in* $d^i_{sub}$ **do**
2  **if** $\mathsf{Prog}(d_{sub}) = \mathtt{oth}$ **then**
3  $\mathsf{Tra}(d_{sub}) = 1$;
4  **for** $d_i$ *in* $\mathcal{T}_\mathcal{D}$ **do**
5  **if** $s_i^T = s^T_{sub}$ *and* $\mathsf{Prog}(d_i) = \mathsf{Prog}(d_{sub})$ **then**
6  **if** $C_T(d_i) < C_T(d_{sub})$ **then**
7  $\mathsf{Tra}(d_{sub}) = 1$;
8  **else**
9  Delete the paths from the $d$ to all its leaf nodes in $\mathcal{T}_\mathcal{D}$;
10  **end**
11 **end**
12 Return $d^i_{sub}$, $\mathcal{T}_\mathcal{D}$;

Thus, this method greatly reduces the search space caused by the high precision of the map, the large number of robots, or the dense time stamps, which makes traversal only related to tasks.

*Remark 1:* As the robots may not all be in the areas of interest, a sparse map with only areas of interest can not fully express the system states of the multi-robot system. Therefore, when using product automaton based methods, a dense map is generally preferred over a sparse map. To address this issue, time automata are developed in [28] and [29] to indicate the waiting time, which allows the transition system with sparse states to be applied. Differently, the PDT based method in this work directly uses time and position information to describe the system states, which avoids the need of extensive exploration in each step as in the product based methods [30] and sampling based methods [11]. Therefore, our method can be used for a sparse map with only areas of interest and the system states can be obtained directly without search.

### C. Tree Pruning

Since $\mathcal{T}_\mathcal{D}$ can grow substantially due to the traversal and the iteration of system states, a pruning method is developed in this section to limit the tree expansion. In particular, the function Bound is designed, which can reduce the number of child nodes in $\mathcal{T}_\mathcal{D}$. Different from the pruning method in [10], the proposed pruning method is task-oriented, which limits the number of nodes within $(1 + |F|) \times |S|$.

As shown in Algorithm 3, for each child node, the first suffix stage of the mission has been finished if $\mathsf{Prog}(d_{sub}) = oth$ (lines 2–3), and thus we can obtain a plan by $d_{sub}$. Since this $d_{sub}$ should not be traversed to generate new child nodes, we set $\mathsf{Tra}(d_{sub}) = 1$. For the child node $d_{sub}$, the node $d_i \in \mathcal{T}_\mathcal{D}$ that has the same NBA state and the same task stage will be selected (lines 4-10). If there exists $C_T(d_i) \leq C_T(d_{sub})$, we set $\mathsf{Tra}(d_{sub}) = 1$. Otherwise, we set $\mathsf{Tra}(\hat{d}_i) = 1$ for all leaf nodes $\hat{d}_i$ generated by $d_i$, which satisfying $d_i \in \mathsf{Nodes}(\hat{d}_i)$. Later we will show that Bound can limit the number of state while speeding up the tree expansion and ensuring the solution time within the polynomial complexity.

To limit the tree expansion, the tree traversal rules are developed.

*Definition 4:* The traversal rules are defined as follows:

1) For any $d$ in $\mathcal{T}_\mathcal{D}$, if $\mathsf{Prog}(d) = \mathtt{oth}$, $d$ will no longer be traversed;
2) For any $d$ in $\mathcal{T}_\mathcal{D}$, the traversed states in $\mathsf{PreS}(d)$ will not be sampled if the plan stage (i.e., plan prefix, plan suffix) remains the same;
3) For any $d_i$ and $d_j$ in $\mathcal{T}_\mathcal{D}$, if $\exists s_i^T = s_j^T$, $\mathsf{Prog}(d_i) = \mathsf{Prog}(d_j)$ and $C_T(d_i) < C_T(d_j)$, then the node $d_j$ will no longer be traversed and the child nodes of $d_j$ will be pruned off from $\mathcal{T}_\mathcal{D}$.

The developed traversal rules in Definition 4 can effectively control the horizontal and vertical expansion of $\mathcal{T}_\mathcal{D}$. The idea behind rule 1) is to limit the length of tree as we are interested in completing the suffix loop fast. By rule 2), the depth of $\mathcal{T}_\mathcal{D}$ will be smaller than $|S|$ in each plan stage, which ensures the total length of plan is not more than $2 \times |S| + 1$. For each plan stage, only one node $d_i \in \mathcal{T}_\mathcal{D}$, $s_i^T = s$, can generate its child nodes and add them into $\mathcal{T}_\mathcal{D}$, which means that after pruning, it holds that $|\mathcal{T}_\mathcal{D}| < (1 + |\mathcal{F}|) \times |S|$. In implementation, the rules 1) and 3) are embodied in Algorithm 3 (lines 2–3 and lines 4–10) while the rule 2) is embodied in Algorithm 2 (line 2).

## V. Algorithm Analysis

This section investigates the performance of PDT in the following aspects: the feasibility, the completeness, and the optimality. The feasibility indicates if the generated plan is applicable and the algorithm completeness indicates whether or not a feasible solution, if exits, is guaranteed to be found.

### A. The Performance of PDT

We first show in Theorem 1-2 that the PDT, without using the traversal rules, is guaranteed to find a satisfactory mission plan. We then show in Lemma 1 - 4 that the traversal rules do not compromise the feasibility and completeness of PDT, which concludes in Theorem 3 that PDT with traversal rules can search for the mission planning more efficiently.

*Theorem 1 (The feasibility of PDT):* Given a multi-robot system $R$ operating in the environment $M$ with an LTL task $\phi$, if Algorithm 1 can find a plan $\Pi_{pre}\Pi_{suf}$ without using traversal rules, it is guaranteed that $\Pi_{pre}\Pi_{suf} \models \phi$.

*Proof:* Given an LTL formula $\phi$, the environment $M$, and the multi-robot system $R$, the tree $\mathcal{T}_\mathcal{D}$ can be constructed following Algorithm 1. For each node $d_i$ with a parent node $\mathsf{Ori}(d_i) = d_j$, by line 3 in Algorithm 2, one has $\pi_i^T = LA(q_i^T) \in \Delta(s_j^T, s_i^T)$. By the Algorithm 3, if $\mathsf{Prog}(d_i) = \mathtt{oth}$, then the path of $d_i$ has entered the accepting set $\mathcal{F}$ at least two times with the same accepting states. Therefore, the plan obtained by PDT can finish the first suffix loop of the mission and thus satisfies the task $\phi$.■

*Theorem 2 (The completeness of PDT):* Given a multi-robot system $R$ operating in the environment $M$ with an LTL task $\phi$, if there exists a mission planning satisfying $\phi$, the PDT in Algorithm 1 is ensured to find it without traversal rules.

*Proof:* Consider a plan $\Pi_{finite} = \Pi_{pre}\Pi_{suf}$ that satisfies the formula $\phi$ with the shortest execution time. The corresponding sequence can be represented as $\Pi_{finite} = (\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{\pi})$, which satisfies $\forall \pi_i \in \boldsymbol{\pi}$, $\pi_i \in \Delta(s_{i-1}, s_i)$ and $LA(q_i) = \pi_i$. According to definition, $s_0^T = s_0 \in \boldsymbol{s}$, $q_0^T = q_0 \in \boldsymbol{q}$, $\pi_0^T = \pi_0 \in \boldsymbol{\pi}$. Since $\mathcal{T}_\mathcal{D}$ has traversed all states when generating the child nodes, there exists a node $d_{i_1} \in \mathcal{T}_\mathcal{D}$ such that $s_{i_1}^T = s_1 \in \boldsymbol{s}$, $q_{i_1}^T = q_1 \in \boldsymbol{q}$, and $\pi_{i_1}^T = \pi_1 \in \boldsymbol{\pi}$. Then, in the following expansion, if there exists a node $d_{i_j} \in \mathcal{T}_\mathcal{D}$ with $s_{i_j}^T = s_j \in \boldsymbol{s}$, $q_{i_j}^T = q_j \in \boldsymbol{q}$, $\pi_{i_j}^T = \pi_j \in \boldsymbol{\pi}$, there must exist $d_{i_{j+1}}^T \in \mathcal{T}_\mathcal{D}$, $s_{i_{j+1}}^T = s_{j+1} \in \boldsymbol{s}$, $q_{i_{j+1}}^T = q_{j+1} \in \boldsymbol{q}$, $\pi_{i_{j+1}}^T = \pi_{j+1} \in \boldsymbol{\pi}$. Therefore, for $\forall (q_j, s_j, \pi_j) \in (\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{\pi})$, there exists $d_{i_j}^T \in \mathcal{T}_\mathcal{D}$ that satisfies $s_{i_j}^T = s_j \in \boldsymbol{s}$, $q_{i_j}^T = q_j \in \boldsymbol{q}$, $\pi_{i_j}^T = \pi_j \in \boldsymbol{\pi}$, which indicates the completeness of PDT. ■

Theorem 1-2 indicate that, without traversal rules, PDT is complete and feasible. Therefore, as long as there exists feasible plans for the mission $\phi$, the constructed tree $\mathcal{T}_\mathcal{D}$ is ensured to include such plans. Among these feasible plans, the optimal one with the minimum cost can be found. Therefore, PDT without traversal rules also has optimality. The following lemmas and theorems will show that the traversal rules only reduce the search space without compromising the feasibility and completeness of PDT. To facilitate the analysis, we use $C(\boldsymbol{q})$ to evaluate the performance of a task sequence $\boldsymbol{q} = q_0 q_1 \dots q_e$. Let $C_R(r, \boldsymbol{q})$ denote the shortest completion time for robot $r \in R$ corresponding to the task sequence $\boldsymbol{q}$, which further indicates $C(\boldsymbol{q}) = \max_{r \in R}(C_R(r, \boldsymbol{q}))$.

*Lemma 1:* Suppose there are two task sequences $\boldsymbol{q} = q_0 \dots q_e$, and $\boldsymbol{q}^* = q_0 \dots q_t \dots q_e$, where $\boldsymbol{q}^*$ is same with $\boldsymbol{q}$ but differs in containing additional task $q_t$ in the middle. It holds that $C(\boldsymbol{q}) \le C(\boldsymbol{q}^*)$.

*Proof:* Consider two task sequences $\boldsymbol{q}_1 = q_0 q_e$ and $\boldsymbol{q}_2 = q_0 q_1 q_e$, with the same initial state before executing the task. If $LR(q_1) \cap LR(q_e) = \emptyset$, one has $C(\boldsymbol{q}_2) = \max(C(q_0 q_e), C(q_0 q_1))$, and $C(q_0 q_1) \ge C(\boldsymbol{q}_1)$. If there exits a robot $r \in LR(q_1) \cap LR(q_e)$, then $C_R(r, \boldsymbol{q}_2) \ge C_R(r, \boldsymbol{q}_1)$ since the robot $r$ performs both $q_1$ and $q_e$. For the robot $r \in LR(q_1) \setminus LR(q_e)$, $C_R(r, \boldsymbol{q}_2) \ge C_R(r, \boldsymbol{q}_1)$ since robot $r$ performs one more task $q_1$ in $\boldsymbol{q}_2$. For the robot $r \in LR(q_e) \setminus LR(q_1)$, $C_R(r, \boldsymbol{q}_2) = C_R(r, \boldsymbol{q}_1)$ since robot $r$ performs the same task sequence $q_0 q_e$. Then there exists $C(\boldsymbol{q}_2) = \max_{r \in R}(C_R(r, \boldsymbol{q}_2)) \ge C(\boldsymbol{q}_1) = \max_{r \in R}(C_R(r, \boldsymbol{q}_1))$. Hence, following similar analysis above, given $\boldsymbol{q} = q_0 \dots q_e$, and $\boldsymbol{q}^* = q_0 \dots q_t \dots q_e$, it can be concluded that $C(\boldsymbol{q}) \le C(\boldsymbol{q}^*)$. ■

*Lemma 2:* Suppose $\Pi_{best} = (\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{\pi}) = \Pi_{pre}\Pi_{suf}$ is an optimal plan. The states in $\boldsymbol{s} \in \Pi_{pre}$ are all different, i.e., $s_i \ne s_j, \forall s_i, s_j \in \boldsymbol{s}, i \ne j$, and the same holds for $\boldsymbol{s} \in \Pi_{suf}$.

*Proof:* Consider an optimal plan $\Pi_{best} = \Pi_{pre}\Pi_{suf}$, whose corresponding states sequence is $s = s_0 s_1 \dots s_n$ and the propositions sequence is $\boldsymbol{\pi} = \pi_0 \pi_1 \dots \pi_n$. If $s_i = s_j \in S, i < j$, satisfying $\Delta(s_i, s_{j+1}) \ne \emptyset$ and $L(q_{j+1}) \in \Delta(s_i, s_{j+1})$, there must exist a new state sequence $\boldsymbol{s}_{new} = s_0 \dots s_i s_{j+1} \dots s_n$ and a task sequence $\boldsymbol{q}_{new} = q_0 \dots q_i q_{j+1} \dots q_n$, which satisfies $\phi$ in $M$. According to Lemma 1, one has $C(\boldsymbol{q}) \ge C(\boldsymbol{q}_{new})$, which indicates there exists a plan whose cost is smaller than $\Pi_{best}$, leading to the contradiction to the optimality assumption. ■

*Lemma 3:* Traversal rule 2) in Definition 4 does not affect the optimality and completeness of PDT.

*Proof:* According to Theorem 2, without using the traversal rules, the PDT can still obtain all feasible plans. Suppose $d_{end}$ is the leaf node with the least cost in $\mathcal{T}_\mathcal{D}$. Then $\mathcal{P} = d_{i_0} d_{i_1} \dots d_{i_n}$ is an optimal path in $\mathcal{T}_\mathcal{D}$ generated by $\mathcal{P} = \mathsf{Nodes}(d_{end})$ and the corresponding optimal plan is $\Pi = \mathsf{Plan}(\mathcal{P}) = (\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{\pi})$, $\boldsymbol{q} = q_0 \dots q_n$. If there exist two nodes $d_{i_k}, d_{i_j} \in \mathcal{P}, k < j$, such that $s_{i_k}^T = s_{i_j}^T = s_k \in \boldsymbol{s}$ and $\mathsf{Prog}(d_{i_k}) = \mathsf{Prog}(d_{i_k})$, according to Lemma 2 $C(\boldsymbol{q}^*) \le C(\boldsymbol{q})$ with $\boldsymbol{q}^* = q_0 \dots q_k q_{j+1} \dots q_n$. Therefore, $\mathcal{P}$ is not the optimal path in $\mathcal{T}_\mathcal{D}$, which is against the assumption. Hence, the traversal rule 2) does not affect the optimality and completeness of PDT.

*Lemma 4:* Traversal rule 3) in Definition 4 does not affect the completeness of PDT. ■

*Proof:* According to Theorem 2, without using the traversal rules, the PDT can still obtain all feasible plans. Suppose there exist two nodes $d_{k_1}, d_{k_2} \in \mathcal{T}_\mathcal{D}$ satisfying $s_{k_1}^T = s_{k_2}^T$, $\mathsf{Prog}(d_{k_1}) = \mathsf{Prog}(d_{k_2})$, $C_T(d_{k_1}) < C_T(d_{k_2})$. Assume that there exists a feasible plan $\mathcal{P} = d_{i_0} d_{i_1} \dots d_{i_n}$ and $d_{k_2} = d_{i_j}$, i.e. $\mathcal{P} = \mathsf{Nodes}(d_{k_2}) d_{i_{j+1}} \dots d_{i_n}$. Then, due to $s_{k_1}^T = s_{k_2}^T$ and $\mathsf{Prog}(d_{k_1}) = \mathsf{Prog}(d_{k_2})$, there exists a new node sequence $\mathcal{P}_{new} = \mathsf{Nodes}(d_{k_1}) d_{i_{j+1}} \dots d_{i_n}$ that satisfies the same task $\phi$. It indicates that if the best plan exists and is not in $\mathcal{T}_\mathcal{D}$ because

TABLE I
SOLUTION TIME FOR DIFFERENT NUMBER OF STATES

| $\|AP\|$ | $\|S\|$ | Time (s) | $\|AP\|$ | $\|S\|$ | Time (s) |
|---|---|---|---|---|---|
| 3 | 8 | 0.00532 | 6 | 56 | 0.145 |
| 4 | 16 | 0.0142 | 6 | 60 | 0.162 |
| 5 | 32 | 0.0485 | 6 | 64 | 0.187 |
| 6 | 18 | 0.0235 | 7 | 96 | 0.439 |
| 6 | 33 | 0.0617 | 7 | 128 | 0.804 |
| 6 | 40 | 0.0748 | 8 | 192 | 2.28 |
| 6 | 48 | 0.108 | 8 | 256 | 4.01 |
| 6 | 52 | 0.123 | | | |

of the traversal rules, then there must exist an approximate sub-optimal path satisfying $\phi$. Hence, the traversal rule 3) does not affect the completeness of PDT. ■

*Theorem 3:* PDT with traversal rules is feasible and complete.

*Proof:* Lemma 1-4 indicate that the traversal rules do not compromise the completeness of PDT. They only affect the node expansion of the tree, which do not compromise the selected propositions and transition of NBA states. Hence, the traversal rules do not affect the feasibility of PDT. Since PDT is feasible and complete without using traversal rules by Theorem 1-2, the PDT is also feasible and complete when incorporating the traversal rules. In addition, PDT can obtain the sub-optimal plan since traversal rule 2) does not compromise the optimality and traversal rule 3) can reserve the sub-optimal plan. ■

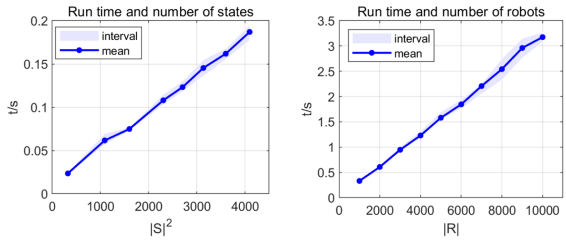
Since the search progress of PDT algorithm is only task-related, only the system states of each robot in the function Generation need to be updated. Therefore, for PDT algorithm, the time complexity related to $n_a$ is $O(n)$. As discussed before, after pruning, there are at most $(1 + |\mathcal{F}|) \times |S|$ nodes in $\mathcal{T}_\mathcal{D}$ and thus the space complexity related to $|S|$ is $O(n)$. As the upper bound length of tree is $2 \times |S| + 1$, the traversal times are smaller than $2 \times |S| + 1$. In each traversal, since at most $(1 + |\mathcal{F}|) \times |S|$ nodes can be traversed and at most $|S| \times |AP|$ child nodes can be generated by one parent node, the time complexity is at most $O(n^3)$.

## VI. NUMERICAL AND PHYSICAL EXPERIMENT

Numerical simulations are carried out in this section to evaluate the performance of PDT. Throughout this simulation, LTL2STAR is used to convert the LTL formula to an NBA [25] and Matlab 2019b is used for numerical simulations. The reported simulation results are the average of at least 20 runs.

The performance of the PDT algorithm is evaluated in terms of the computation time in finding a feasible plan for the multi-robot system $R$ operating in $M$ with an LTL task specification $\phi$. The environment $M$ consists of 8 areas of interest and several robots. The $ap_i, i = 1, \ldots, 8$, represents the tasks of visiting area $i$, respectively. The areas of interest and the initial positions of robots are randomly deployed. A total of 20 random maps have been constructed as the test environments.

We first consider a single-robot with LTL formulas of different size (i.e., various number of atomic propositions and NBA states). The average solution time of 20 runs is listed in Table I. Fig. 3(a) indicates that the solution time is approximately




(a) (b)

Fig. 3. The plots (a) and (b) show the solution time under different number of task states and robots, respectively.

TABLE II
SOLUTION TIME FOR DIFFERENT NUMBER OF ROBOTS

| $n_a$ | Time (s) | $n_a$ | Time (s) |
|---|---|---|---|
| 1 | 0.00390 | 5000 | 1.58 |
| 10 | 0.00721 | 6000 | 1.85 |
| 100 | 0.0361 | 7000 | 2.21 |
| 1000 | 0.332 | 8000 | 2.54 |
| 2000 | 0.609 | 9000 | 2.96 |
| 3000 | 0.948 | 10000 | 3.17 |
| 4000 | 1.23 | | |

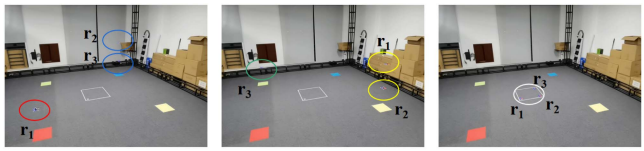
TABLE III
SOLUTION TIME FOR DIFFERENT ALGORITHMS

| $n_a$ | $\|S\|$ | Sampling(s) | MILP(s) | PDT(s) |
|---|---|---|---|---|
| 1 | 4 | 0.08043 | 2.272 | 0.01594 |
| 1 | 7 | 1.645 | 4.166 | 0.01848 |
| 1 | 11 | 3.673 | 7.463 | 0.02928 |
| 1 | 20 | 7.256 | 13.78 | 0.05389 |
| 3 | 20 | 9.923 | 26.50 | 0.05604 |
| 5 | 20 | 17.13 | 47.81 | 0.05870 |
| 10 | 20 | 25.19 | 77.68 | 0.06355 |
| 25 | 20 | 78.99 | Failure | 0.06059 |
| 100 | 20 | Failure | Failure | 0.08922 |
| 500 | 20 | Failure | Failure | 0.2440 |

linearly proportional to $|S|^2$, which is smaller than the upper bound of time complexity $O(n^3)$.

We then fix the LTL formula $\phi = GFap_1 \wedge GFap_2 \wedge GFap_3 \wedge GFap_4$ and vary the number of the robots in $R$. The average results of 20 runs are listed in Table II. Fig. 3(b) indicates that the robot numbers has little influence on the solution time when $n_a$ is small and gradually shows a linear relationship with $n_a$ when $n_a$ is large. The time complexity of PDT corresponding to the number of robots $n_a$ is $O(n)$, which is consistent with the algorithm analysis.

To show the superiority of the algorithm, the proposed PDT is compared against existing methods. For various numbers of NBA states and robots, the results using different algorithms are listed in Table III. Sampling indicates the sampling-based method in [11] and we employed exactly the same settings in the simulation section of [11] in this work for the purpose of comparison. The solution time for the first searched feasible plan is listed. The MILP methods in the works of [31], [32] are also compared with our approach in terms of the solution time and the simulation results are listed in Table III. The search step is set as 1 in a $10 \times 10$ map for the sampling-based method, MILP,




(a) (b) (c)

Fig. 4. The experiment results. (a) UAV $r_1$ arrives at region 1. UAV $r_2$ and $r_3$ arrive at region 2. (b), UAV $r_1$ and $r_2$ arrive at region 3. UAV $r_3$ arrives at region 4. (c) All UAVs return to region 5.

and ours. It can be seen from Table III that the sampling-based method and MILP can not scale well with large groups of robots and shows large variance. In contrast, the PDT based method is much faster and scales well for large-sized multi-robot systems.

Experiments are carried out for the case in Example 1. The system runs Matlab 2019b on Ubuntu18.04 and the ROS version is Melodic. PDT searches 18 nodes within $0.0056s$ and the minimum cost value is 25.61. The obtained plans are both $\boldsymbol{\pi} = ap_1ap_3ap_2ap_4ap_5$. The snapshots of the experiment are shown in Fig. 4, which indicates $\phi$ is successfully completed by the UAVs. The experiment video is provided.[1]

## VII. Conclusion

For multi-robot systems with temporal logic specifications, this letter develops a novel framework that can generate satisfactory plans not only faster than most existing methods, but also multiple orders of magnitude more robots than those that existing methods can manipulate. Additional research will consider extending the PDT based mission planning for uncertain environments or heterogeneous multi-robot systems.

## References

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

[2] A. Jones, M. Schwager, and C. Belta, "Information-guided persistent monitoring under temporal logic constraints," in *Proc. IEEE Amer. Control Conf*, 2015, pp. 1911–1916.

[3] Y. Kantaros and M. M. Zavlanos, "Distributed intermittent connectivity control of mobile robot networks," *IEEE Trans. Autom. Control*, vol. 62, no. 7, pp. 3109–3121, Jul. 2017.

[4] M. Cai, S. Xiao, Z. Li, and Z. Kan, "Optimal probabilistic motion planning with potential infeasible LTL constraints," *IEEE Trans. Autom. Control*, vol. 68, no. 1, pp. 301–316, Jan. 2023.

[5] M. Cai, M. Hasanbeig, S. Xiao, A. Abate, and Z. Kan, "Modular deep reinforcement learning for continuous motion planning with temporal logic," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7973–7980, Oct. 2021.

[6] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *Int. J. Robot. Res.*, vol. 30, no. 14, pp. 1695–1708, 2011.

[7] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimal multi-robot path planning with temporal logic constraints," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2011, pp. 3087–3092.

[8] M. Cai, H. Peng, Z. Li, and Z. Kan, "Learning-based probabilistic LTL motion planning with environment and motion uncertainties," *IEEE Trans. Autom. Control*, vol. 66, no. 5, pp. 2386–2392, May 2021.

[9] D. Gujarathi and I. Saha, "MT*: Multi-robot path planning for temporal logic specifications," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, 2022, pp. 13692–13699.

[10] Y. Kantaros and M. M. Zavlanos, "STyLuS*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *Int. J. Robot. Res.*, vol. 39, no. 7, pp. 812–836, 2020.

[11] X. Luo, Y. Kantaros, and M. M. Zavlanos, "An abstraction-free method for multirobot temporal logic optimal control synthesis," *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1487–1507, Oct. 2021.

[12] C. I. Vasile, X. Li, and C. Belta, "Reactive sampling-based path planning with temporal logic specifications," *Int. J. Robot. Res.*, vol. 39, no. 8, pp. 1002–1028, 2020.

[13] A. Nikou, J. Tumova, and D. V. Dimarogonas, "Cooperative task planning of multi-agent systems under timed temporal specifications," in *Proc. IEEE Amer. Control Conf.*, 2016, pp. 7104–7109.

[14] Y. E. Sahin, R. Quirynen, and S. Di Cairano, "Autonomous vehicle decision-making and monitoring based on signal temporal logic and mixed-integer programming," in *Proc. IEEE Amer. Control Conf.*, 2020, pp. 454–459.

[15] S. Tokuda, M. Yamakita, H. Oyama, and R. Takano, "Convex approximation for LTL-based planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot.*, 2021, pp. 9863–9869.

[16] Y. E. Sahin, P. Nilsson, and N. Ozay, "Synchronous and asynchronous multi-agent coordination with cLTL constraints," in *Proc. IEEE Conf. Decis. Control*, 2017, pp. 335–342.

[17] A. T. Buyukkocak, D. Aksaray, and Y. Yazicioğlu, "Planning of heterogeneous multi-agent systems under signal temporal logic specifications with integral predicates," *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, pp. 1375–1382, Apr. 2021.

[18] M. Guo, J. Tumova, and D. V. Dimarogonas, "Communication-free multi-agent control under local temporal tasks and relative-distance constraints," *IEEE Trans. Autom. Control*, vol. 61, no. 12, pp. 3948–3962, Dec. 2016.

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

[20] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Decomposition of finite LTL specifications for efficient multi-agent planning," in *Proc. 13th Int. Symp. Distrib. Auton. Robot. Syst.*, 2018, pp. 253–267.

[21] L. Lindemann, J. Nowak, L. Schönbächler, M. Guo, J. Tumova, and D. V. Dimarogonas, "Coupled multi-robot systems under linear temporal logic and signal temporal logic tasks," *IEEE Trans. Control Syst. Technol.*, vol. 29, no. 2, pp. 858–865, Mar. 2021.

[22] R. Peterson, A. T. Buyukkocak, D. Aksaray, and Y. Yazicioglu, "Decentralized safe reactive planning under TWTL specifications," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot.*, 2020, pp. 6599–6604.

[23] G. F. Schuppe and J. Tumova, "Multi-agent strategy synthesis for LTL specifications through assumption composition," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, 2020, pp. 533–540.

[24] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.

[25] P. Gastin and D. Oddoux, "Fast LTL to büchi automata translation," in *Proc. Int. Conf. Comput. Aided Verification*, 2001, pp. 53–65.

[26] Y. Kantaros and M. M. Zavlanos, "Sampling-based optimal control synthesis for multirobot systems under global temporal tasks," *IEEE Trans. Autom. Control*, vol. 64, no. 5, pp. 1916–1931, May, 2019.

[27] K. Leahy, A. Jones, and C. I. Vasile, "Fast decomposition of temporal logic specifications for heterogeneous teams," *IEEE Robot. Autom. Lett*, vol. 7, no. 2, pp. 2297–2304, Apr. 2022.

[28] A. Ulusoy, S. L. Smith, X. C. Ding, and C. Belta, "Robust multi-robot optimal path planning with temporal logic constraints," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2012, pp. 4693–4698.

[29] 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," *Int. J. Robot. Res.*, vol. 32, no. 8, pp. 889–911, 2013.

[30] Z. Zhou, D. J. Lee, Y. Yoshinaga, S. Balakirsky, D. Guo, and Y. Zhao, "Reactive task allocation and planning for quadrupedal and wheeled robot teaming," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, 2022, pp. 2110–2117.

[31] X. Luo and M. M. Zavlanos, "Temporal logic task allocation in heterogeneous multirobot systems," *IEEE Trans. Robot.*, vol. 38, no. 6, pp. 3602–3621, Dec. 2022.

[32] K. Leahy et al., "Scalable and robust algorithms for task-based coordination from high-level specifications (ScRATCHeS)," *IEEE Trans. Robot*, vol. 38, no. 4, pp. 2516–2535, Aug. 2022.

[1][Online]. Available: https://youtu.be/muzncQHui3w