Original Article

The International Journal of Robotics Research
2024, Vol. 0(0) 1–25

DOI: 10.1177/02783649241278372
journals.sagepub.com/home/ijr

Real-time reactive task allocation and planning of large heterogeneous multi-robot systems with temporal logic specifications

Ziyang Chen and Zhen Kan

Abstract

Existing methods for the task allocation and planning (TAP) of multi-robot systems with temporal logic specifications mainly rely on optimization-based approaches or graph search techniques applied to the product automaton. However, these methods suffer from high computational cost and scale poorly with the number of robots and the complexity of temporal logic tasks, thus limiting the applicability in real-time implementation, especially for large multi-robot systems. To address these challenges, this work develops a novel TAP framework that can solve reactive temporal logic planning problems for large-scale heterogeneous multi-robot systems (HMRS) in real time. Specifically, we develop a planning decision tree (PDT) to represent the task progression and task allocation specialized for HMRS with temporal logic specifications. Based on the PDT, we develop two key search algorithms—the planning decision tree search (PDTS) and the interactive planning decision tree search (IPDTS)—where PDTS generates an offline plan which will be modified online by PDTS and IPDTS jointly to enable fast reactive planning if environmental changes or temporary tasks occur. Such a design can generate satisfying plan for HMRS with multiple orders of magnitude more robots than those that existing methods can manipulate. Rigorous analysis shows that the PDT-based planning is feasible (i.e., the generated plan is applicable) and complete (i.e., a feasible plan, if exits, is guaranteed to be found). The algorithm complexity further indicates that the solution time is only linearly proportional to the robot numbers and types. Simulation and experiment results demonstrate that reactive plan can be generated for large HMRS in real-time, which outperforms the state-of-the-art methods.

Keywords

Task allocation and planning, temporal logic, heterogeneous multi-robot systems, planning decision tree

Received 15 December 2023; Revised 30 June 2024; Accepted 8 August 2024

1. Introduction

Recent advances in robotics have enabled robots to operate beyond structured manufacturing environments into more unstructured and dynamic workspaces (Garrett et al., 2021; Yu and Dimarogonas, 2022). In these applications, robots with distinct sensing and manipulation capabilities can form a heterogeneous multi-robot system (HMRS) to accomplish specialized tasks with great versatility. It is highly desired that the HMRS can engage in collaborative tasks with prompt responsiveness to environmental and task-related changes. For instance, consider a fire-fighting scenario that requires a group of heterogeneous robots with different sensors (e.g., visual, infrared, and lidar sensors) to provide persistent surveillance over possible fire spots and search for survivors. If detected, such requests need to be promptly handled (e.g., extinguish the fire and rescue the survivor). A crucial step toward its success is a timely and reactive task allocation and planning (TAP), which is concerned with assigning the tasks (e.g., the fire-fighting and rescue tasks) to the robots based on their capabilities and current states and planning the task collaboration accordingly. Despite recent progress of multi-robot TAP, below are the significant challenges to be addressed.

1.1. Large-scale system

Temporal logics, such as linear temporal logic (LTL), are capable of expressing complex robotic tasks beyond point-to-point navigation. However, building upon the transition system and the automaton, the size of the product automaton grows exponentially with the number of robots. Thus, conventional model checking methods suffer from the curse

Department of Automation, University of Science and Technology of China, Hefei, China

Corresponding author:
Zhen Kan, Department of Automation, University of Science and Technology of China, No. 443, Huangshan Road, Hefei 230026, China.
Email: zkan@ustc.edu.cn

of dimensionality, rendering them arduous to effectively address a large number of robots with LTL specifications. This issue is further exacerbated if robots with different capabilities and mutually dependent goals are involved. Hence, there is a growing need for online scalable methods for the TAP of a large HMRS with LTL specifications.

1.2. Reactive planning

Since the workspace is often dynamic and not fully known a priori, the offline plan is significantly restricted, as the pre-assigned tasks and planning can be found prohibitive to the robots during the runtime. For instance, in the fire-fighting case, the fire spots are time-varying and hard to predict, necessitating the robot to adapt to environmental changes. Hence, reactive planning that can actively respond to the change of environment (e.g., time-varying fire spots) and dynamic task request (e.g., the detection of survivors) is desired.

1.3. Timely planning

Obtaining a feasible solution to the TAP of HMRS, especially for a large group, can be time consuming and computationally expensive. For reactive planning, it is arguable that a timely TAP with shorter solution time (possibly sub-optimal) is more preferable, especially for temporary local task requests that require the robots to react quickly and effectively. For instance, once a survivor is detected, the time window to rescue is short. Hence, a timely coordination and planning of HMRS is highly desired.

Motivated by the above practical challenges, this work considers a heterogeneous multi-robot system where the robots are of different types to capture different capabilities. Due to rich expressivity of temporal logic, LTL is employed to specify the collective tasks, where each task may require robots of multiple types to collaborate to satisfy the specification. Since the TAP problem for heterogeneous multi-robot systems is NP-hard (Luo and Zavlanos, 2022), existing approaches to this problem become intractable for large-scale applications. To address this challenge, we develop a novel framework that can solve TAP problems in real time for large-scale HMRS with potential reactive tasks. In this work, by reactive tasks, we mean local events, such as agent failures, task and environmental changes, or complex temporary tasks, that require the agents to adapt to. Specifically, we develop a novel planning decision tree (PDT) to record the task progress and the system state without sophisticated product automaton. Based on the PDT, we then develop a real-time TAP framework for timely reactive planning of HMRS. The planning decision tree search (PDTS) is developed for fast TAP and can be applied to offline planning and re-planning. The interactive planning decision tree search (IPDTS) is then developed for multi-task planning, which can be applied to unexpected temporary tasks.

The PDT-based TAP framework has the following advantages. First, it is applicable to large-scale HMRS. Instead of constructing sophisticated product automaton as in many existing works, the developed PDT encodes the sub-tasks and task progression in a tree incrementally. Such a design can generate satisfying plan for HMRS 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 HMRS with more than 200 task states, 100 robot types, and 10^4 robots and beyond. Second, without searching in system states, the PDT-based TAP framework shows significant savings in resources, not only in terms of memory to save the runtime data, but also in terms of computational cost compared with optimization-based or graph search techniques. Therefore, it can achieve timely planning in real-time. Finally, it enables reactive TAP for HMRS. To account for possible agent failure, task and environmental change, and complex temporary task, the developed PDTS and IPDTS are proven to be feasible (i.e., the generated TAP solution is applicable) and complete (i.e., a feasible TAP solution, if exits, is guaranteed to be found). Rigorous analysis further indicates that PDTS and IPDTS are computationally efficient, that is, the solution time of finding a satisfying plan is only linearly proportional to the robot numbers and types. Extensive simulation and experiment results demonstrate that the PDT-based TAP outperforms the state-of-the-art methods.

2. Related works

2.1. Task specifications and allocation

Temporal logics (TL), such as linear temporal logic (LTL), computation tree logic (CTL), and signal temporal logic (STL), recently gain popularity in robotics due to its ability in specifying a rich class of robotic tasks (Belta et al., 2007). By logically reasoning about the temporal ordering of events at the task level, temporal logics can describe complex tasks beyond traditional go-to-goal navigation for robotic systems. For instance, tasks such as visiting targets sequentially (e.g., reach A, and then B, and then C) or surveillance (e.g., visit A and then B infinitely often) can be conveniently expressed by temporal logics. By specifying tasks as TL formulas, tools from formal verification (Baier and Katoen, 2008) can be used to generate task allocation and motion plans. Given that robots operating in complex environments are often subjected to various uncertainties, such as stochastic motion behaviors and uncertain environment properties, there has been increasing research attention devoted to addressing Markov decision processes (MDPs) with linear temporal logic (LTL) specifications through probabilistic model checking, such as co-safe LTL tasks (Lacerda et al., 2019; Ulusoy et al., 2014; Jagtap et al., 2020), stochastic game-based approaches (Ramasubramanian et al., 2020), and learning-based approaches (Shah et al., 2023; Cai et al., 2021a,b). In the works of Kloetzer and Belta (2010), Guo and Dimarogonas (2015), and Ulusoy et al. (2013), LTL was exploited to describe the complex collective tasks for multi-robot systems, where the LTL tasks were

assigned locally to the robots. However, the construction of a team product for robots with individual goals can lead to state explosion. Hence, such approaches cannot scale well for large multi-robot systems.

An alternative is to assign a global LTL specification to the team, where the global LTL can be either explicitly (Kantaros and Zavlanos, 2020; Luo et al., 2021; Smith et al., 2011) or implicitly (Luo and Zavlanos, 2022; Lacerda and Lima, 2019) assigned to individual robots. A bottom-up task coordination strategy was developed in Guo and Dimarogonas (2017), which contains an offline initial plan synthesis and an online coordination scheme. The works of Guo et al. (2017) and Schuppe and Tumova (2020) specify individual TL formulas for each agent and complete the group task through coupling between the formulas. For global LTL specifications that are not explicitly assigned to the robots, task decomposition is typically used. For instance, the global specification was decomposed into local specifications and then allocated to the robots in Kantaros and Zavlanos (2016). Similarly, by exploiting the structure of the automata, the global specification was decomposed into multiple sub-tasks in Schillinger et al. (2018a). The cross-entropy temporal logic optimization was developed in Banks et al. (2020) to optimally allocate tasks to a team of robots given a global task specification. However, the aforementioned works do not account for heterogeneous capabilities of the multi-robot system.

By exploiting different capabilities of robots, the decomposition of graph-based automaton was investigated in Schillinger et al. (2018b) and Luo and Zavlanos (2022). In Sahin et al. (2019), the counting LTL was proposed to capture rich heterogeneous specifications and an integer linear program (ILP) approach was developed to search for feasible plans. In recent works of Leahy et al. (2022, 2021), a variant of STL, namely, CaTL, was introduced to express the task specifications in a more compact form and mixed integer linear program (MILP) was formulated for task coordination. Unfortunately, ILP/MILP is in general computationally expensive and thus cannot scale well and provide timely planning if a large number of heterogeneous robots is considered. As an extension of CaTL, CaTL+ was developed in Liu et al. (2023), which can be solved by a two-step optimization method. Despite recent progress, few existing methods can generate timely planning for large HMRS. Instead, leveraging the tree structure and the developed prune algorithm and traversal rules, the PDT-based TAP framework significantly improves the computational time and search efficiency in finding a satisfying plan. Hence, our approach has much less complexity and computational time than MILP and can be applied in real-time.

2.2. Reactive planning

Reactive temporal logic planning can account for environment uncertainties or local temporary tasks (Cai et al., 2020; Partovi et al., 2018; Vasilopoulos et al., 2021). In general, reactive planning can be environment-specific, task-specific, agent-specific, or a combination thereof.

The environment-specific reactive planning focuses on unknown or dynamic environments. For instance, in such environments, an agent will re-plan a new local path (Ulusoy and Belta, 2014). Otte and Frazzoli (2016) developed the fast re-planning RRTX algorithm, a variant of RRT, which accelerates the speed of path re-planning. Then, He et al. (2017) developed an approach enabling the agent to modify its entire task plan based on observed environmental states rather than just the local path. Leveraging an incremental method with rapid solution speed, Kantaros et al. (2020) applied environment-specific reactive planning in multi-agent systems, enabling fast re-planning in unknown environments during task performance.

The agent-specific reactive planning addresses failures such as loss of capabilities (e.g., manipulation or perception) or complete removal of the agent (e.g., due to malfunction or battery depletion). For instance, the simultaneous task allocation and planning developed by Faruq et al. (2018) provides probabilistic safety guarantees on performance and reallocates tasks if individual agents fail. Considering heterogeneous agents and unexpected failures in agent capabilities, Kalluraya et al. (2023) proposed a resilient mission planning algorithm that reallocates sub-tasks to robots based on their current functioning skills, minimally disrupting existing team motion plans. To further enhance the autonomy of reactive planning, Zhao et al. (2023) devised a failure-aware task planning approach to address internal robot failures, which cannot be directly measured but must be inferred from agent actions.

The task-specific reactive planning primarily addresses changes in tasks. For instance, Cai et al. (2023) considered task changes through soft LTL constraints, which handle partially infeasible tasks in dynamic environments by allowing certain tasks to be relaxed. Beyond focusing on changes in global tasks, it is also crucial to address additional local tasks. Vasile et al. (2020) demonstrated that independent local tasks can be integrated with global tasks to achieve a joint plan. However, this approach is not suitable for complex and coupled local tasks. To manage the convergence of multiple tasks, Finkbeiner et al. (2021) proposed LiveLTL, which facilitates joint planning for both global and local tasks.

To address reactive planning, both fast solution times and the ability to record task progress are essential. For heterogeneous agent systems, optimization-based approaches such as ILP/MILP are the primary solutions (Leahy et al., 2021; Liu et al., 2023). However, these approaches struggle with dynamic requests (e.g., local temporary tasks or task changes) during task operations. If the current task is interrupted, ILP/MILP must be reformulated to obtain a feasible plan. Additionally, optimization-based approaches are generally computationally expensive, making them challenging to apply for real-time reactive planning. Although some search-based methods have lower complexity (Kantaros and Zavlanos, 2020), they cannot be directly

applied since the assignment is not determined. Luo and Zavlanos (2022) used search for task decomposition before applying MILP. Zhou et al. (2022) proposed a hierarchical approach. Despite its ability to adapt to locomotion failures and human interventions, it does not scale well due to the use of product automaton.

In contrast, the PDT-based planning in this work builds a tree to facilitate reactive task allocation and planning without requiring sophisticated graph products, which can be applied to large-scale HMRS in real-time. In our previous work (Chen et al., 2024), we developed a fast mission planning framework to manage large-scale multi-robot systems with temporal logic specifications in real-time. However, task allocation was not addressed in Chen et al. (2024), as each sub-task was pre-assigned in the task specification. Additionally, this work addresses joint planning for both temporary reactive and global tasks, an aspect not covered in Chen et al. (2024).

3. Preliminaries

Due to the rich expressivity of temporal logic in formulating robotic tasks, LTL is employed in this work for task specifications of HMRS. An LTL formula is built on a set of atomic propositions AP, standard Boolean operators such as $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation), and temporal operators such as F (eventually), X (next), G (always), and $\mathcal{U}$ (until). The syntax of an LTL formula Φ is defined as

$$\Phi := ap \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \mid F\Phi \mid X\Phi \mid G\Phi \mid \Phi_1 \mathcal{U} \Phi_2,$$

where the atomic proposition $ap \in AP$ indicates the properties of system states that can be either true or false, $G\Phi$ means Φ is true for all future moments, $F\Phi$ means Φ is true at some future moments, $X\Phi$ means Φ is true at the next moment, and $\Phi_1 \mathcal{U} \Phi_2$ means Φ_1 is true until Φ_2 becomes true.

The semantics of an LTL formula are defined over an infinite sequence $\boldsymbol{\sigma} = \sigma_0\sigma_1\ldots$ with $\sigma_i \in 2^{AP}$ for all $i \geq 0$, where 2^{AP} represents the power set of AP. Denote by $\boldsymbol{\sigma} \models \Phi$ if the word $\boldsymbol{\sigma}$ satisfies the LTL formula Φ. Detailed descriptions of the syntax and semantics of LTL can be found in Baier and Katoen (2008).

An LTL formula can be translated to a nondeterministic Büchi automaton (NBA).

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

Let Δ: $S \times S \rightarrow 2^{\Sigma}$ denote the set of atomic propositions that enables state transitions in $\mathcal{B}$, i.e., $\forall\sigma \in \Delta(s, s')$, $s' \in \delta(s, \sigma)$. A valid run $\boldsymbol{s} = s_0 s_1 s_2\ldots$ of $\mathcal{B}$ generated by the word $\boldsymbol{\sigma} = \sigma_0\sigma_1\sigma_2\ldots$ with $\sigma_{i+1} \in \Delta(s_i, s_{i+1})$ is called accepting, if $\boldsymbol{s}$ intersects with $\mathcal{F}$ infinitely often (Clarke et al., 1999). In this paper, NBA will be used to track the progress of the satisfaction of LTL tasks. To convert an LTL formula to an NBA, readers are referred to Gastin and Oddoux (2001) for algorithms and implementations.

4. Problem formulation

To facilitate problem formulation, we first introduce the notations used throughout this article. Let $\mathbb{N}$ and $[N]$ denote the set of natural 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 $\boldsymbol{\sigma}[\ldots j] = \sigma_0\ldots\sigma_j$.

4.1. Environment and heterogeneous multi-robot systems

Consider a fleet of robots $\mathcal{R} = \{r_i\}_{i=1}^{n_a}$ operating in a bounded workspace $M \subset \mathbb{R}^2$, where n_a is the number of robots. Suppose the workspace M contains $l \in \mathbb{N}$ non-overlapped regions of interest. Denote by M_I and M_{NI} the regions of interest and the regions of non-interest (e.g., obstacles or the regions that the robots cannot traverse or operate within), respectively, where $M_i \cap M_j = \emptyset$ and $M_i \cap M_{NI} = \emptyset$, $\forall i \neq j$ and $i, j \in [l]$. It is assumed that each M_i, $i \in [l]$, is associated with an atomic proposition.

Suppose that the robots are heterogeneous and of n_c different types (i.e., with different capabilities such as ground mobility, aerial mobility, and object manipulation). It is further assumed that each robot belongs to exactly one type. We denote by r_i^j the robot $i \in [n_a]$ of type $j \in [n_c]$ and denote by $\mathcal{K}_j$ the set that contains all robots of type j, i.e., $\sum_j |\mathcal{K}_j| = n_a$ and $\mathcal{K}_i \cap \mathcal{K}_j = \emptyset$, $\forall i \neq j$. Let $p_i(k) \in M$ denote the position of robot i in the workspace at time $k \in \mathbb{N} \cup \{0\}$. In particular, $p_i(0)$ represents the initial position of robot i.

Example 1. Imagine that a fleet of robots is tasked to operate in a farm environment as illustrated in Figure 1. The colored areas represent the regions of interest M_i and the thick white lines represent areas that the robots cannot traverse (i.e., M_{NI}). It is assumed that the robots are of different types (e.g., wheeled mobile robots, and quadrotors) and different M_i may require a group of robots with different capabilities to serve.

4.2. Global collaborative tasks

Before formulating the global collaborative tasks, we first introduce a set of atomic propositions $ap_i \in AP$, $i \in [n_p]$, where n_p denotes the number of atomic propositions. For instance, $ap \in AP$ can represent the task of visiting a desired region of interest by a sub-group of robots of particular types. Let L: $M \rightarrow AP$ be a labeling function that indicates the atomic proposition associated with a position in M. To specify the task allocation, define a set $TK = \{tk_1, \ldots, tk_{n_p}\}$,

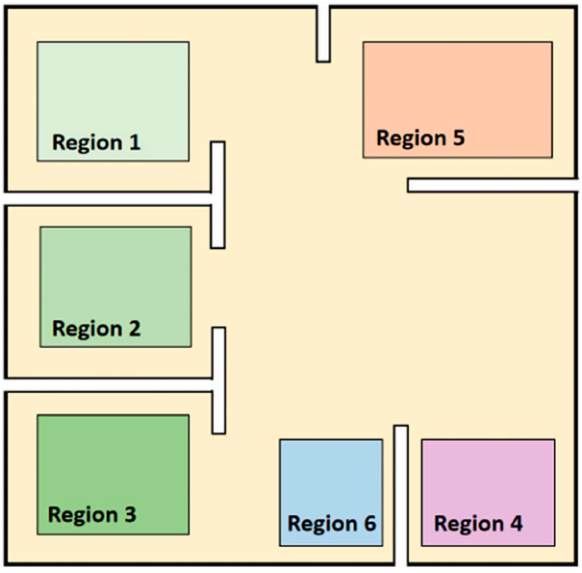


Figure 1. A farm environment, where regions 1, 2, and 3 represent the planting areas 1, 2, and 3, respectively. Region 5 represents the livestock area. Region 4 represents the warehouse. Region 6 represents the other facilities. The thick white lines represent the fence that the robots cannot traverse.

where $tk_i = \left(n_1^{tk_i}, n_2^{tk_i}, \ldots, n_{n_c}^{tk_i}\right)$, $i \in [n_p]$, and the entry $n_j^{tk_i}$ of tk_i, $j \in [n_c]$, indicates the total number of robots of type j. That is, tk_i specifies the numbers and types of robots required to perform the atomic task ap_i. By specifying LTL formulas over AP with TK, robots can perform various collaborative temporal–spatial tasks using their versatile capabilities.

Example 2. Continue with Example 1. To collaboratively serve in the workspace, consider a set of atomic propositions $AP = \{ap_1, ap_2, ap_3, ap_4, ap_5\}$, where ap_i, $i \in \{1, \ldots, 5\}$ represents the sub-task of reaching the region i, respectively. The associated task requirements are $TK = \{tk_i\}_{i=1}^{5}$, where $tk_1 = (2, 2, 1)$, $tk_2 = (3, 2, 3)$, $tk_3 = (2, 2, 2)$, $tk_4 = (5, 5, 5)$, and $tk_5 = (2, 2, 2)$. For instance, tk_1 indicates that ap_1 requires two robots of type 1, two robots of type 2, and one robot of type 3. An example global task is $\Phi = GFap_1 \wedge GFap_2 \wedge GFap_3 \wedge GFap_4$, which requires the fleet of robots to repeatedly visit these regions while satisfying the task requirement TK (i.e., for each ap_i there are $n_j^{tk_i}$ agents in $\mathcal{K}_j$, $j \in [n_c]$, reaching the region i according to TK).

To facilitate task allocation and motion planning, we construct an abstract task system (ATS) to relate the sub-tasks, the temporal logic, the map, and the task requirements.

Definition 2. The abstract task is defined as a tuple $T = (Q, AP, M, TK, LA, LM, LT)$, where Q is a finite set of abstract sub-tasks, AP is the set of the atomic propositions, M is the workspace, LA: $Q \to AP$ indicates the atomic proposition associated with the state in Q, LM: $Q \to M$ maps $q \in Q$ to a position $p \in M$, which satisfies $L(LM(q)) = LA(q) = ap$, and LT: $Q \to TK$ indicates the task requirements of an atomic task, i.e., $LT(q_i) = tk_i$, $q_i \in Q$, $tk_i \in TK$.

The abstract task T in Def. 2 is developed to abstract the task in the workspace into a finite set of abstract sub-tasks. That is, each abstract task $q \in Q$ uniquely corresponds to an atomic proposition $LA(q)$, which is executed at $LM(q)$ in the workspace requiring robots $LT(q)$. For instance, given Example 2, we can construct the abstract states $Q = \{q_1, q_2, q_3, q_4, q_5\}$, where $LA\ (q_i) = ap_i \in AP$ indicates that ap_i is associated with the abstract state q_i, $LM(q_i)$ indicates the position in M that ap_i can be executed, and $LT\ (q_i) = tk_i$ indicates the task requirement of q_i.

4.3. Reactive tasks

During task operation, the ability to react to unexpected environmental and task changes is highly desirable. In this work, environmental changes may entail previously executable propositions becoming invalid (e.g., an accessible area becomes inaccessible) so that re-planning is required for the robots to adapt to the changed environment. It should be noted that, if such environmental changes conflict with the task specification (i.e., no feasible plan exists), our approach will yield no solution. Since the task allocation and planning within a potential infeasible environment is beyond the scope of this work, in the subsequent development, we assume that the environmental changes will not conflict with the task specifications. Apart from environmental changes, the task change also includes temporary tasks (e.g., rerouting for locally encountered events), change of task requirement, and unexpected robot failures. Such tasks are considered as reactive tasks in this work, which require robots to respond reactively. Specifically, we mainly focus on the following types of reactive tasks.

- *Changes of environment* refer to the task re-planning of HMRS to adapt to environmental changes (e.g., the areas of interest that were previously accessible may occasionally become restricted to the agent due to the presence of moving obstacles).
- *Changes of task requirements* refer to the cases that the robots are assigned with new task assignment TK while the task specifications remain unchanged.
- *Local temporary tasks* refer to the cases that the robots are asked to deal with temporary tasks (e.g., an urgent local service) while respecting the global task.
- *Failure recovery* refers to the ability of the HMRS to adapt to unexpected robot failures (e.g., removal from the team due to malfunction or hardware issues) during mission operation.

It is assumed that robot failure can be detected either by observation (e.g., no longer moving as commanded) or via wireless communication to report hardware failures.

Both co-safe LTL (Kupferman and Vardi, 2001; Vasile et al., 2017) and LTLf (Tabajara and Vardi, 2020) can describe finite-length tasks. In this paper, co-safe LTL is employed to represent the local reactive tasks. As indicated in Lacerda et al. (2014), given an LTL formula Φ and a word $\boldsymbol{\sigma} = \sigma_0\sigma_1\ldots \in (2^{AP})^{\omega}$ satisfying $\boldsymbol{\sigma} \models \Phi$, $\boldsymbol{\sigma}$ is said to have a good prefix if there exists $n \in \mathbb{N}$ and a truncated finite sequence $\boldsymbol{\sigma}[\ldots n]$ such that $\boldsymbol{\sigma}[\ldots n]\boldsymbol{\sigma}[n\ldots] \models \Phi$ for any infinite sequence $\boldsymbol{\sigma}[n\ldots] \in (2^{AP})^{\omega}$. Such a formula Φ is called a co-safe LTL formula, which can be translated into a NFA (Cho et al., 2017). The NFA is defined as $\mathcal{A} = (S_A, S_{A0}, \Delta_A, \Sigma, \mathcal{F}_A)$, where S_A, S_{A0}, Σ, $\mathcal{F}_A$ indicate the set of states, the initial states, the words, the accepting states, respectively, and $\Delta_A : S_A \times S_A \to 2^{\Sigma}$ records the feasible words from one NFA state to another. Though defined similarly to NBA, NFA accepts finite words only while the accepted language of NBA is a ω-language. More explanations about NFA can be found in Baier and Katoen (2008).

4.4. Planning model

Given the workspace M and the NBA $\mathcal{B}_\Phi$ generated by the LTL formula Φ, the plan is defined as a tuple $\Pi = (\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{C})$, where $\boldsymbol{q} = q_0q_1q_2\ldots$ is the trajectory of the states in T with $q_i \in Q$, $\boldsymbol{s} = s_0s_1s_2\ldots$ is the trajectory of $\mathcal{B}_\Phi$ states corresponding to $\boldsymbol{q}$ where s_i indicates the automaton state after the atomic task is applied at q_i, and $\boldsymbol{C} = C_0C_1C_2, \ldots$ indicates the task allocations along the state trajectories. Specifically, $C_i = (c_1^i, c_2^i, \ldots, c_{n_a}^i)$, $i \in \mathbb{N} \cup \{0\}$, where the jth entry indicates whether or not the robot r_j is involved in the task $LA(q_i)$, that is, $c_j^i = 1$ if involved and $c_j^i = 0$ otherwise. $\boldsymbol{C}$ is said to satisfy TK, denoted as $\boldsymbol{C} \models TK$, if $\forall C_i = (c_1^i, \ldots, c_{n_a}^i) \in \boldsymbol{C}$ and $tk = LT(q_i) = \left(n_1^{tk}, n_2^{tk}, \ldots, n_{n_c}^{tk}\right)$ it holds that $n_j^{tk} = \sum_{r_k \in \mathcal{K}_j} c_k^i$, $\forall j \in [n_c]$. Note that C_0 indicates an empty task allocation, since the initial state q_0 corresponds to an empty task. By denoting $\Pi_i = (q_i, s_i, C_i)$, $i \in \mathbb{N} \cup \{0\}$, we can rewrite $\Pi = \Pi_0\Pi_1\Pi_2, \ldots$ as a series of plan tuples. Given a plan $\Pi = (\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{C})$, let $\boldsymbol{\pi} = LA(\boldsymbol{q})$ be the word generated by the trajectory $\boldsymbol{q}$. The plan Π is said to satisfy Φ and the task requirements TK, denoted as $\Pi \models (\Phi, TK)$, if $\boldsymbol{\pi} \models \Phi$ and $\boldsymbol{C} \models TK$.

4.5. Problem

Given the NBA $\mathcal{B}_\Phi$, the accepting run $\boldsymbol{s} = s_0s_1s_2\ldots$ of $\mathcal{B}_\Phi$ can be generally written in a prefix–suffix structure, where the prefix part starting from an initial state and ending at an accepting state is traversed only once and the suffix part, a cyclic path of accepting states, is traversed infinitely often (Baier and Katoen, 2008). Based on the prefix–suffix structure, the plan Π can be written in the form of $\Pi = \Pi_{pre}\Pi_{suf}\Pi_{suf}\ldots$, where Π_{pre} and Π_{suf} are finite prefix stage and finite cyclic suffix stage of the plan, respectively. Since the generated suffix part may not start at the end state of the identified prefix part, or the end state of local temporary task does not match with that of the global task, a transition stage Π_{tra} is introduced to bridge the gap between the end state of prefix part and the starting state of the newly identified suffix part. Consequently, the infinite plan can be written in the form of $\Pi = \Pi_{pre}\Pi_{tra}\Pi_{suf}\Pi_{suf}\ldots$ with three types of task stages. Denote by $\Pi_{finite} = \Pi_{pre}\Pi_{tra}\Pi_{suf}$ the finite plan and since, $\Pi_{finite} \models (\Phi, TK)$ also indicates that $\Pi \models (\Phi, TK)$, we only need to determine Π_{pre}, Π_{tra}, and Π_{suf}.

Problem 1. Given a heterogeneous multi-robot system operating in a workspace M with an LTL formula Φ and task requirement TK, the goal is to develop a real-time reactive task allocation and planning strategy that satisfies the task Φ and TK, that is, $\Pi_{finite} \models (\Phi, TK)$, in the environment with reactive tasks.

5. Task allocation and planning

This section presents a real-time reactive TAP framework for heterogeneous multi-robot systems. An overview of the approach is shown in Figure 2, which consists of an off-line planning module, an on-line planning module, and a path planning module. At the basis of these modules is the development of a planning decision tree (PDT). In the subsequent development, we will first present the developed PDT, and then explain how PDT can be leveraged to develop PDTS and IPDTS for real-time reactive TAP of HMRS in an environment with reactive tasks.

5.1. Planning decision tree (PDT)

The tree-search methods are promising candidates for the decision-making of robotic systems (Kiesbye et al., 2022). Despite its recent progress, new challenges arise when considering the TAP of HMRS. When the cooperative tasks are specified by complex temporal logic constraints, the TAP is generally performed over the product of the automaton that represents the LTL tasks and the transition system that represents the system dynamics. Due to the large state space of the product, the tree-search based approach soon becomes intractable to search over such a product graph. For instance, consider a relatively small multi-robot system, for example, 15 robots with 16 NBA states operating in a 10×10 grid world. The size of the product automaton can reach $16 \times (10 \times 10)^{15}$ and the upper bound of the tree size without pruning can reach $(16 \times (10 \times 10)^{15})!$, which makes tree-search based methods impractical.

To prevent the issue of state explosion, we develop the planning decision tree (PDT) to avoid product automaton, denoted by $\mathcal{T}_\mathcal{D} = \{\mathcal{V}_T, \mathcal{E}_T, Cost\}$, where $\mathcal{V}_T = \{Nd_i\}$, $i = 0, 1, 2\ldots$, represents the set of nodes as defined in Def. 3, $\mathcal{E}_T$ represents the set of edges capturing the hierarchical transitions among the nodes in $\mathcal{V}_T$, that is, $\left(Nd, Nd'\right) \in \mathcal{E}_T$ if Nd' is the child node of Nd, and $Cost : \mathcal{V}_T \to \mathbb{R}^+$ measures

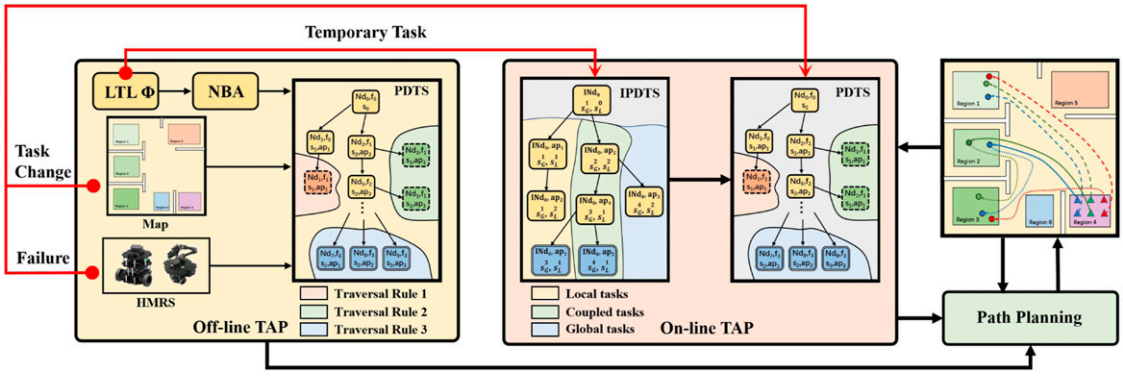


Figure 2. The overview of the real-time reactive TAP framework, which consists of an off-line and on-line TAP module and a path planning module. Given the map M, the LTL task Φ, and the task requirement TK, the PDTS first generates an initial off-line plan, which is fed to the path planning module for real-time implementation. During task operation, if environment and task changes are encountered, reactive plan will be obtained by PDTS and IPDTS jointly. Existing methods (e.g., probabilistic roadmap or receding horizon control) are used in the path planning module.

the cost of sub-tasks from Nd_0 to the current node. The key idea behind $\mathcal{T}_\mathcal{D}$ is to restrict the search within the automaton states and the abstract sub-tasks, rather than the whole state space of the system, to enable fast and effective search of satisfying and executable plan. It is worth pointing out that, for the aforementioned example, as system states are directly obtained by iteration without searching or sampling in the product automaton, the state space size in PDT can be reduced to (16)! and after proper pruning, the space size can even be reduced to 16^2.

Definition 3. The planning decision tree $\mathcal{T}_\mathcal{D}$ is constructed based on a set of nodes $\{Nd_i\}$, $i \in \mathbb{N} \cup \{0\}$, where Nd_0 represents the root of $\mathcal{T}_\mathcal{D}$ and each node is defined as a tuple

$$Nd_i = \left(B_s, T_s, C_s, ET_{pre}, EP_{pre}\right)$$

where

- $B_s \in S$ denotes the automaton state of node i;
- $T_s \in Q$ denotes the sub-task of node i;
- $C_s = (c_1, c_2, c_3, \ldots, c_{n_a})$ denotes the task allocation of the corresponding sub-task T_s;
- $EP_{pre} = \left\{p_{pre\_1}, \ldots, p_{pre\_na}\right\}$ denotes the predicted robot locations after completing the preceding and current tasks, where $p_{pre\_i} \in M$ is the predicted location of r_i, denoted as $EP_{pre}(i)$;
- $ET_{pre} = \left\{t_{pre\_1}, \ldots, t_{pre\_na}\right\}$ denotes the predicted timestamp after completing the preceding and current tasks, where t_{pre_i} is the predicted timestamp of r_i, denoted as $ET_{pre}(i)$.

By Def. 3, we define the cost of Nd_i as the predicted maximum task completion time of all agents, that is, $Cost(Nd_i) = \max\left\{Nd_i.ET_{pre}\right\}$. To facilitate the following analysis, we denote by $Parent(Nd_i)$ the parent node of Nd_i (i.e., $(Parent(Nd), Nd) \in \mathcal{E}_T$) and denote by $PreAuSt(Nd_i)$ the automaton states of nodes that have been traversed in the same task stage from Nd_0 to Nd_i. To indicate the tree expansion status, we define $FlagTra : \mathcal{V}_T \rightarrow \{0, 1\}$, where $FlagTra(Nd) = 1$ if $Nd \in \mathcal{V}_T$ cannot generate its child nodes and 0 otherwise. To indicate the task stages of Π_{finite}, we define $Prog : \mathcal{V}_T \rightarrow \{pre, tra, s, oth\}$ to specify the task stages of nodes, where *pre* and *tra* represent the prefix and the transition stage, respectively. As the cyclic suffix part requires to start and end at the same state, $s \in S$ in *Prog* is such a state verifying whether the first suffix stage ends. Once the first suffix stage completes, due to the consideration of Π_{finite}, the task stage proceeds to *oth*, which indicates the end of search.

In Def. 3, both time and position information in performing sub-tasks are incorporated in the modeling of system states in the tree $\mathcal{T}_\mathcal{D}$. Such a design can conveniently model the agents performing different sub-tasks at different time, avoiding dense discretization of unnecessary planning details as in many existing approaches. Therefore, compared with search or sampling-based methods, PDTS only need to consider the areas of interest, which greatly reduces the search space. For the aforementioned multi-robot example, the searching space can be reduced to $(16 \times Q)!$.

The tree $\mathcal{T}_\mathcal{D}$ encodes the automaton states and system states in a hierarchical tree structure to enable fast and effective TAP for HMRS tasks with temporal logic constraints. The tree $\mathcal{T}_\mathcal{D}$ grows incrementally from the root node by searching sub-tasks satisfying LTL task specifications. The best task allocation for each sub-task is determined based on the system states, which are continuously updated along with the task evolution. A feasible plan is then generated by tracing back from the leaf node with the least cost to the root node. Specifically, as indicated in Def. 3, each node contains task planning related states (i.e., T_s, B_s, and C_s) and system related states (i.e., EP_{pre}, ET_{pre}). Given a node $Nd = \left(B_s, T_s, C_s, ET_{pre}, EP_{pre}\right)$, $\Pi_i = (T_s, B_s, C_s)$ represents a plan tuple in $\Pi = \Pi_0\Pi_1\Pi_2\ldots$. The performance of Nd_i is then evaluated via *Cost*, which is determined based on the predicted task completion location *EPpre* and the predicted task completion timestamp ET_{pre}. Throughout this

work, we will adopt the data structure to represent the node components, for example, $Nd_i.B_s$ represents the automaton states associated with node i. Note that, for any node $Nd_i \in \mathcal{V}_T$, it holds that $LA(Nd_i.T_s) \in \Delta(Parent(Nd_i).B_s, Nd_i.B_s)$, where $\Delta \in \mathcal{B}_\phi$ is defined in Def. 1 ensuring that the sequence of sub-tasks satisfies the temporal logic specification.

5.2. Planning decision tree search (PDTS)

As discussed in the overview in Figure 2, the first step is to generate an offline task allocation and planning to Prob. 1. That is, given the workspace M, a global LTL task specification Φ_G, and the task allocation TK, the goal is to obtain a feasible and satisfying plan $\Pi_{finite} \models (\Phi_G, TK)$ such that the task can be completed efficiently, that is, completing the first loop of the suffix stage with minimum cost. Such an offline global plan will be considered as a base solution, which will be modified online to enable reactive planning by taking into account potential environmental changes and local temporary tasks in Section 5.3.

This section presents how the tree $\mathcal{T}_\mathcal{D}$ in Def. 3 is constructed and exploited to develop the PDTS algorithm for an offline plan that satisfies Φ_G. In particular, as outlined in Alg. 1, the algorithm is initialized in lines 1–3 by constructing the automaton $\mathcal{B}_G$, the abstract task T, and the root Nd_0 of PDT. The tree $\mathcal{T}_\mathcal{D}$ is expanded by adding child nodes using Children_Generation in lines 4–14. The tree $\mathcal{T}_\mathcal{D}$ stops growing if all nodes have been traversed, that is, $\forall Nd \in \mathcal{V}_T$, $FlagTra\ (Nd) = 1$. The function Get_Plan is then used to obtain the plan $[\Pi_{pre}, \Pi_{tra}, \Pi_{suf}]$. The following running examples are used to explain the key functions in Alg. 1.

Algorithm 1 Planning Decision Tree Search

Input: LTL task Φ, environment M, Nd_0
Output: $\Pi_{pre}, \Pi_{tra}, \Pi_{suf}$
1: Convert Φ to NBA $\mathcal{B}_\Phi$
2: Construct the abstract task T based on M
3: Initialize the $\mathcal{T}_\mathcal{D} = \{Nd_0\}$
4: **while** 1 **do**
5: **if** $FlagTra(Nd) = 1, \forall Nd \in \mathcal{V}_T$, **then**
6: break
7: **end if**
8: **for** $Nd \in \mathcal{V}_T$ s.t. $FlagTra(Nd) = 0$ **do**
9: $Children =$ Children_Generation$(Nd, \mathcal{B}_\Phi, T)$
10: $[Children, \mathcal{T}_\mathcal{D}] =$ Bound$(Children, \mathcal{T}_\mathcal{D})$
11: Add $Children$ to $\mathcal{T}_\mathcal{D}$
12: Set $FlagTra(Nd) = 1$
13: **end for**
14: **end while**
15: $[\Pi_{pre}, \Pi_{tra}, \Pi_{suf}] =$ Get_Plan $(\mathcal{T}_\mathcal{D})$
16: Return $\Pi_{pre}, \Pi_{tra}, \Pi_{suf}$

Example 3. Consider a workspace as in Figure 3(a), which consists of three areas of interest and four robots of two types (i.e., $\{r_1, r_2\}$ and $\{r_3, r_4\}$). The robots need to visit region 1 once and repeatedly visit region 2 and 3. The visit of each region requires two robots in total, one of each type. The ATS T is then constructed with $Q = \{q_1, q_2, q_3\}$, and for $i \in \{1, 2, 3\}$, $LA(q_i) = ap_i$, $LT(q_i) = (1, 1)$, and $LM(q_i)$ indicates the corresponding region of ap_i. The LTL task is formulated as $\Phi = Fap_1 \wedge GFap_2 \wedge GFap_3$, which can be converted into the NBA $\mathcal{B}_\Phi$ as shown in Figure 3(b). The root node Nd_0 is set as $Nd_0.B_s = s_0 \in S_0$, $Nd_0.ET_{pre}(i) = 0$, $Nd_0.EP_{pre}(i) = p_i(0)$, $i \in \{1, 2, 3\}$ with empty task and allocation. The tree structure related functions are initialized as $PreAuSt(Nd_0) = \{s_0\}$, $FlagTra(Nd_0) = 0$, $Prog(Nd_0) = pre$. The tree grows from the root Nd_0 by generating child nodes $Children$ using Children_Generation and Bound. After adding child nodes, it is set $FlagTra(Nd) = 1$ which indicates the parent node has been traversed and will no longer generate child nodes.

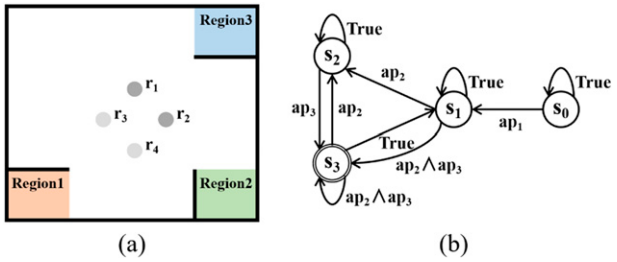


Figure 3. (a) The operating environment with three areas of interest (i.e., ap_1, ap_2, and ap_3) and four robots r_i, $i \in [4]$, of two types. (b) The NBA $\mathcal{B}_\Phi$ corresponding to $\Phi = Fap_1 \wedge GFap_2 \wedge GFap_3$ includes four NBA states and one accepting state $s_3 \in \mathcal{F}$.

5.2.1. Function Children_Generation. A key component in Alg. 1 is the function Children_Generation, which grows $\mathcal{T}_\mathcal{D}$ by adding possible child nodes. The function Children_Generation is task-oriented, that is, only the sub-tasks and NBA states will be searched. The system states will be updated iteratively by the searched sub-task, task allocation, and previous system states rather than searching in product automaton and time automaton (modeling the waiting time for cooperation). Therefore, for the aforementioned multi-robot example, the searching space can be further reduced to (16)!. Specifically, suppose Nd is the current node in $\mathcal{T}_\mathcal{D}$ and Alg. 2 explains how the child nodes of Nd can be constructed. Particularly, for each NBA state $s \in S \setminus PreAuSt\ (Nd)$, we enumerate all possible sub-task $q \in Q$ satisfying $LA(q) \in \Delta(Nd.B_s, s)$, where $\Delta(Nd.B_s, s)$ is the set of words over 2^Σ enabling the transition from $Nd.B_s$ to s. Then, for each pair of s and q, we can create a new child node $Child$ and set its sub-task, automaton state, and parent node as q, s, and Nd, respectively. The function Request_Response is then invoked to obtain the task allocation $Child.C_s$, which together with $Nd.EP_{pre}$ and $Nd.ET_{pre}$ can obtain the child node's predicted completion position $Child.EP_{pre}$ and predicted completion time $Child.ET_{pre}$. To evaluate the performance of the node, the timestamp that completes the current task is treated as the cost in this work, that is, the cost $Cost(Child)$ is set as $\max\{Child.ET_{pre}\}$ to

indicate the worst case completion time among all robots. The task stage *Prog*(*Child*) is updated by $\mathsf{Flag}(Prog(Nd), s)$, where the function Flag is defined as

$$\mathsf{Flag}(Prog, s) = \begin{cases} Prog, & \text{if } s \notin \mathcal{F}, \\ tra, & \text{if } s \in \mathcal{F}, Prog = pre, \\ s, & \text{if } s \in \mathcal{F}, Prog = tra, \\ oth, & \text{if } Prog = s. \end{cases} \quad (1)$$

If *Child* is in the same stage with *Nd*, its *PreAuSt* is updated by adding *s* to *PreAuSt* (*Nd*), otherwise it is set as empty. Finally, *Child* is added to the set *Children* to expand $\mathcal{T}_{\mathcal{D}}$.

Algorithm 2 Children_Generation

Input: $Nd, \mathcal{B}_{\Phi}, T$
Output: $Children$
1: Initialize the set of child nodes $Children = \emptyset$
2: **for** $s \in S \setminus PreAuSt(Nd)$ **do**
3: **for** q s.t. $LA(q) \in \Delta(Nd.B_s, s)$ **do**
4: Initialize $Child = \emptyset$
5: Set $Child.T_s = q$, $Child.B_s = s$
6: Obtain $[Child.C_s, Child.EP_{pre}, Child.ET_{pre}] = \mathsf{Request\_Response}(q, Nd.EP_{pre}, Nd.ET_{pre})$
7: Set $Cost(Child) = \max\{Child.ET_{pre}\}$
8: Set $Prog(Child) = \mathsf{Flag}(Prog(Nd), s)$
9: **if** $Prog(Child) = Prog(Nd)$ **then**
10: Set $PreAuSt(Child)$ by adding s to $PreAuSt(Nd)$
11: **else**
12: Set $PreAuSt(Child) = \emptyset$
13: **end if**
14: Add $Child$ to $Childeren$
15: **end for**
16: **end for**
17: Return $Childeren$

Example 4. Continue with Example 3. Starting from the root Nd_0, the function Children_Generation is invoked to generate a child node Nd_1, where $Nd_1.B_s$ = 1 (i.e., the automation state in $\mathcal{B}_{\Phi}$), $Nd_1.T_s = q_1$ with $LA(q_1) = ap_1$, and the task allocation $Nd_1.C_s$ = (1, 0, 1, 0). The tree structure function is set as $Parent(Nd_1) = Nd_0$. Based on $Nd_1.T_s$, $Nd_1.C_s$, and preceding system states $Nd_0.ET_{pre}$ and $Nd_0.EP_{pre}$, the current system states $Nd_1.ET_{pre}$ and $Nd_1.EP_{pre}$ can be obtained by the function Request_Response. The cost is set as $Cost(Nd_1) = \max_{i\in[n_a]}\{Nd_1.ET_{pre}(i)\}$, that is, the maximum predicted task completion time. Since the automaton state $Nd_1.B_s \notin \mathcal{F}$, the task stage is in the prefix stage, that is, $Prog(Nd_1) = Prog(Nd_0) = pre$. Then, $PreAuSt(Nd_1) = \{s_0, s_1\}$ and the traversal flag is set as $FlagTra(Nd_1) = 0$ for subsequent expansions.

5.2.2. Function Request_Response. Given the sub-task $q \in Q$, and the associated task requirement $tk = LT(q) \in TK$, the goal of Request_Response in Alg. 3 is to determine a task allocation for *q*. The underlying idea is to predict the completion time of tasks for each robot and select those with the least completion time. Recall that EP_{pre} and ET_{pre} indicate the predicted robot locations in *M* and the predicted completion time of preceding and current sub-tasks. Let $\tilde{ET}_{pre}$ denote the predicted arrival time of robots for the current task *q*. Hence, given the position *LM*(*q*) and the allowed maximum traveling velocity $v_{max} \in \mathbb{R}^+$, assuming that r_i participates in the current task, the predicted arrival time at *LM*(*q*) after completing preceding sub-tasks is estimated as

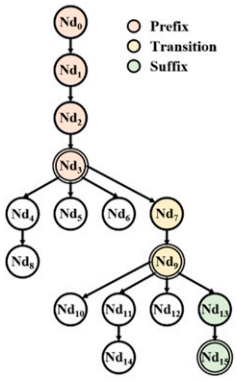
$$\tilde{ET}_{pre}(i) = ET_{pre}(i) + \frac{1}{v_{max}} \left\| LM(q) - EP_{pre}(i) \right\|. \quad (2)$$

As the completion of sub-task *q* requires arrival of all selected robots and the completion of the last sub-task *Parent* (*Child*).T_s, based on task requirement *LT*(*q*), we select the fastest arriving agents for the sub-task *q*. Specifically, for $\forall j \in [n_c]$, the robots in $\mathcal{K}_j$ are sorted from small to large according to $\tilde{ET}_{pre}$, where $\mathcal{K}_j$ is the set of robots of type *j*. Let $n_j = tk(j)$ denote the number of robots of type *j* required for *q*. Then, for each type *j*, the first n_j robots from $\mathcal{K}_j$ are fastest arriving robots and are selected in the task allocation.

Lines 5–13 explain how *Nd* is updated to $\tilde{Nd}$. The predicted positions of selected agents are updated to *LM*(*q*) and the predicted completion time of selected agents are firstly updated to their arrival time. Then, $\tilde{Nd}.ET_{pre}$ is further updated in lines 16–19 to indicate the predicted completion time of the sub-task *q* for selected agents, which is the maximum of predicted completion time of all agents.

Algorithm 3 Request_Response

Input: $q, Nd.EP_{pre}, Nd.ET_{pre}$
Output: $\tilde{Nd}.EP_{pre}, \tilde{Nd}.ET_{pre}, \tilde{Nd}.C_s$
1: $\tilde{ET}_{pre}(i) = ET_{pre}(i) + \frac{1}{v_{max}} \|LM(q) - EP_{pre}(i)\|$, $\forall r_i \in \mathcal{R}$
2: **for** $j \in [n_c]$ **do**
3: Sort the robots in $\mathcal{K}_j$ from small to large by $\tilde{ET}_{pre}$
4: Determine $tk = LT(q)$ for the sub-task q
5: **for** $i \in [n_a]$ **do**
6: **if** $r_i \in \mathcal{R}$ is in the first n_j agents of $\mathcal{K}_j$ **then**
7: Set $\tilde{Nd}.C_s(i) = 1$
8: Set $\tilde{Nd}.EP_{pre}(i) = q$
9: Set $\tilde{Nd}.ET_{pre}(i) = \tilde{ET}_{pre}(i)$
10: **else**
11: Set $\tilde{Nd}.C_s(i) = 0$
12: Set $\tilde{Nd}.EP_{pre}(i) = Nd.EP_{pre}(i)$
13: Set $\tilde{Nd}.ET_{pre}(i) = Nd.ET_{pre}(i)$
14: **end if**
15: **end for**
16: **for** $i \in [n_a]$ **do**
17: **if** $r_i \in \mathcal{R}$ is in the first n_j agents of $\mathcal{K}_j$ **then**
18: Set $\tilde{Nd}.ET_{pre}(i) = max\left\{\tilde{Nd}.ET_{pre}\right\}$
19: **end if**
20: **end for**
21: **end for**
22: Return $\tilde{Nd}.C_s, \tilde{Nd}.EP_{pre}, \tilde{Nd}.ET_{pre}$



(a)

Stage	S	T	C
Prefix	1	q_1	(1,0,1,0)
	2	q_2	(0,1,0,1)
	3	q_3	(0,1,0,1)
Transition	2	q_2	(1,0,1,0)
	3	q_3	(0,1,0,1)
Suffix	2	q_2	(1,0,1,0)
	3	q_3	(0,1,0,1)

(b)

Figure 4. (a) The generated planning decision tree $\mathcal{T}_\mathcal{D}$ for the task in Example 3, where the colored nodes indicate the identified path that satisfies Φ and the task requirement *TK*. The node with double circles indicates that it is in the accepting state of NBA and will proceed to the next task stage. (b) The sequence of NBA state $Nd.B_s$, the sub-task $Nd.T_s$, and the task allocation state $Nd.C_s$ associated with the feasible plan.

Example 5. Continue with Example 4. The new child node Nd_2 is generated from Nd_1. Since r_1 and r_3 have been selected for the sub-task $Nd_1.T_s = q_1$, the predicted completion position and time of r_1 and r_3 are updated as $Nd_1.EP_{pre}(i) = LM(q_1)$, $Nd_i.ET_{pre}(i) = \max_{j\in\{1,3\}}\frac{1}{v_j}\|LM(q_1) - p_j(0)\|$, $i \in \{1, 3\}$. Then, based on the function Request_Response, we can obtain the task allocation, predicted completion position and time for Nd_2. For each robot $r_i \in R$, the earliest arrival time $\tilde{ET}_{pre}(i)$ to sub-task $Nd_2.T_s = q_2$ is $Nd_1.ET_{pre}(i) + \frac{1}{v_i}\|LM(q_2) - Nd_2.EP_{pre}(i)\|$. Since r_2 has the smallest arrival time and the sub-task q_2 requires a robot of type 1, r_2 is selected. Similar procedure applies for the selection of r_4, which yields $Nd_2.C_s = (0, 1, 0, 1)$. Based on the task allocation, the predicted completion position and time of each robot are updated. Since r_1 and r_3 have not been selected, their predicted completion position and time do not change. For r_2 and r_4, the predicted position is updated to $LM(q_2)$ and the predicted completion time is updated to the completion time of the sub-task q_2. When all selected robots arrive at $LM(q_2)$ and the preceding sub-task $Nd_1.T_s = q_1$ has been finished, the sub-task q_2 can be executed. Therefore, the completion time is the predicted maximum arrival time of each robot, that is, $\max_{i\in[n_a]}\{\tilde{ET}_{pre}(i)\}$.

5.2.3. Function Bound. To avoid unnecessary expansion of $\mathcal{T}_\mathcal{D}$, Bound is developed in Alg. 4 to reduce the search space of $\mathcal{T}_\mathcal{D}$ by filtering out the nodes that do not contribute to the planning. By pruning according to the NBA states, the search space can be further reduced without affecting the completeness of proposed method. For the aforementioned multi-robot example, the searching space can be further reduced to $(16)^2$. Specifically, if *Child* has the same B_s and *Prog* with a node *Nd* in $\mathcal{T}_\mathcal{D}$ and *Cost* (*Nd*) < *Cost* (*Child*), the *Child* will no longer be traversed, otherwise the paths from the *Nd* to its leaf nodes in $\mathcal{T}_\mathcal{D}$ will all be pruned out. Since *Prog* (*Child*) = *oth* indicates that the first *suf* phase has been completed, the *Child* will also no longer be traversed. Otherwise, *Child* will be kept in $\mathcal{T}_\mathcal{D}$ and can be traversed.

Algorithm 4 Bound

Input: $Children, \mathcal{T}_\mathcal{D}$
Output: the updated $Children$ and $\mathcal{T}_\mathcal{D}$

```
Initialize the set of child nodes Children = ∅
for Child in Children do
    for Nd in T_D do
        if Nd.B_s = Child.B_s and Prog(Nd) = Prog(Child) then
            if Cost(Nd) < Cost(Child) then
                FlagTra(Child) = 1
            else
                Delete the paths from the Nd to all its leaf nodes in T_D
            end if
        end if
    end for
    if Prog(Child) = oth then
        FlagTra(Child) = 1
    end if
end for
Return Children, T_D
```

Example 6. Continue with Example 5. After tree expansion, Nd_3 has four child nodes as shown in Figure 4(a). Since $Nd_4.B_s = Nd_5.B_s = Nd_6.B_s = s_1$ and $Prog(Nd_4) = Prog(Nd_5) = Prog(Nd_6) = tra$, according to the function Bound, only one node among Nd_4, Nd_5, and Nd_6 can be expanded subsequently. Given that Nd_4 has the minimum cost, only Nd_4 will be expanded and we set $FlagTra(Nd_5) = FlagTra(Nd_6) = 1$ to indicate that Nd_5 and Nd_6 will no longer be considered in the following planning. The other child of Nd_3 is Nd_7, which is set as $Nd_7.B_s = s_2$ and $Prog(Nd_7) = tra$. The nodes Nd_4 and Nd_7 are then selected as the parent nodes for the subsequent expansion. Then, $Prog(Nd_{15}) = oth$ as shown in Figure 4(a). By function Bound, Nd_{15} has finished the first suffix stage and should not generate child nodes, that is, $FlagTra(Nd_{15}) = 1$.

5.2.4. Function Get_Plan. After constructing the tree $\mathcal{T}_\mathcal{D}$, the function Get_Plan is invoked to generate a feasible plan $\Pi_{finite} = \Pi_{pre}\Pi_{tra}\Pi_{suf}$. Let $Nd_{min} \in \mathcal{V}_T$ denote the node with the least cost and $Prog(Nd_{min}) = oth$. Note that, by the definition of the pruning function Bound, it indicates that $Nd_{\min}$ is the leaf node of $\mathcal{T}_\mathcal{D}$ that has finished the first suffix stage. As outlined in Alg. 4, the general idea of generating the plan is to trace back from $Nd_{\min}$ to the root node Nd_0 over $\mathcal{T}_\mathcal{D}$. Specifically, starting from the node *Nd* which is initiated as $Nd_{\min}$, at each iteration, the parent node *Parent* (*Nd*) is exploited to update the corresponding Π_{pre}, Π_{tra}, Π_{suf}. Repeating the above process until the root node Nd_0 is reached.

Example 7. Continue with Example 6. Once the tree $\mathcal{T}_\mathcal{D}$ is constructed as shown in Figure 4(a), it is observed that there is one node completing the first suffix stage of the task, that is, $Prog(Nd_{15}) = oth$. Note that Nd_{15} also has the minimum cost. Hence, we can identify a path from $\mathcal{T}_\mathcal{D}$ as [Nd_0, Nd_1, Nd_2, Nd_3, Nd_7, Nd_9, Nd_{13}, Nd_{15}], from which the plan can be obtained as $\boldsymbol{\pi}_{pre}$ = [ap_1, ap_2, ap_3] and $\boldsymbol{\pi}_{tra}$ = $\boldsymbol{\pi}_{suf}$ = [ap_2, ap_3]. The plan can be found in Figure 4(b).

Algorithm 5 Get_Plan

Input: $\mathcal{T}_\mathcal{D}$
Output: The plan $\Pi_{pre}, \Pi_{tra}, \Pi_{suf}$
1: Obtain $Nd_{min} \in \mathcal{V}_T$ s.t. $Prog(Nd_{min}) = oth$ and $Cost(Nd_{min}) \leq Cost(Nd), \forall Nd \in \mathcal{V}_T$
2: Initialize $\Pi_{tra} = \Pi_{pre} = \Pi_{suf} = \emptyset$
3: Set $Nd = Nd_{min}$
4: **while** 1 **do**
5: **if** $Nd \neq Nd_0$ **then**
6: **if** $Prog(Parent(Nd)) \in S$ **then**
7: add $(Nd.T_s, Nd.B_s, Nd.C_s)$ to Π_{suf}
8: **end if**
9: **if** $Prog(Parent(Nd)) = tra$ **then**
10: add $(Nd.T_s, Nd.B_s, Nd.C_s)$ to Π_{tra}
11: **end if**
12: **if** $Prog(Parent(Nd)) = pre$ **then**
13: add $(Nd.T_s, Nd.B_s, Nd.C_s)$ to Π_{pre}
14: **end if**
15: Set $Nd = Parent(Nd)$
16: **else**
17: break
18: **end if**
19: **end while**
20: Return Π_{pre},Π_{tra}, Π_{suf}

5.2.5. Tree traversal rules. In Alg. 2, $\mathcal{T}_\mathcal{D}$ grows by adding all possible child nodes. A potential issue is that it can lead to dimension explosion. To mitigate this issue, the following traversal rules are developed to reduce the search space by focusing more on the automaton states.

Definition 4. The traversal rules are defined as follows:

(1) For any *Nd* in $\mathcal{T}_\mathcal{D}$, if *Prog* (*Nd*) = *oth*, *Nd* will no longer be traversed;
(2) For any *Nd* in $\mathcal{T}_\mathcal{D}$, the traversed states in *PreAuSt* (*Nd*) will not be sampled if the task stage (i.e., prefix stage, transition stage, and suffix stage) remains the same;
(3) For any Nd_i and Nd_j in $\mathcal{T}_\mathcal{D}$, if $\exists Nd_i.B_s = Nd_j.B_s$, *Prog* ($Nd_i$) = *Prog*($Nd_j$), and *Cost*($Nd_i$) < *Cost*($Nd_j$), then the node Nd_j will no longer be traversed and the child nodes of Nd_j will be pruned off from $\mathcal{T}_\mathcal{D}$.

The developed traversal rules in Def. 4 can effectively control the horizontal and vertical expansion of $\mathcal{T}_\mathcal{D}$. Specifically, given that a feasible plan is in the form of $\Pi = \Pi_{pre}\Pi_{tra}(\Pi_{suf})^w$, we only need to determine the finite plans Π_{pre}, Π_{tra}, and Π_{suf}, respectively. The traversal rule 1 will stop the tree expansion once the first suffix plan has completed, which limits the depth of $\mathcal{T}_\mathcal{D}$. The traversal rule 2 aims at avoiding duplicate searching, which limits both the depth and width of $\mathcal{T}_\mathcal{D}$. As will be proven in Lemma 3, given an obtained plan, if there exists identical NBA states in the same task stage, there must exist another plan with less or the same cost. Therefore, for a node sequence within the same task stage, the length of node sequence is less than or equal to $|S|$. The traversal rules 1 and 2 jointly ensure that the tree expansion can be completed in finite time, that is, $3 \times |S| + 1$. Finally, the traversal rule 3 indicates that, for nodes with the same task progress (the same task stage and NBA state), the node with larger cost cannot generate child nodes, which limits the width of the tree. That is, in the prefix stage, transition stage, each suffix stage with different starting NBA states, and other stage, only one node $Nd \in \mathcal{V}_T$ with $Nd.B_s = s$ will be saved in $\mathcal{T}_\mathcal{D}$, which means, after pruning, the number of nodes in $\mathcal{T}_\mathcal{D}$ is at most $(3 + |\mathcal{F}|) \times |S|$. In implementation, the rules 1 and 3 are embodied in Algorithm 4 (line 13 and lines 5–9), while the traversal rule 2 is embodied in Algorithm 2 (line 2).

5.3. Interactive planning decision tree search (IPDTS) for reactive planning

When operating in an environment with unpredictable factors, for example, environmental change, hardware failures, or changes of task requirements, a new root node Nd_0 can be constructed according to the current system and NBA state, and PDTS can then be leveraged for the re-planning. However, when encountering a local temporary task coupled with the global task, a re-planning satisfying both global and local tasks is required. To do so, the plan generated by PDTS is no longer applicable and the interactive planning decision tree search (IPDTS) is developed based on PDTS to synthesize new plans based on global and local tasks.

The major challenge in reactive planning is that the local task can be coupled with the global task, resulting in that the re-planning cannot be realized by PDTS. For instance, consider a global task $\Phi_G = G\,(F\,(ap_1 \wedge X\,(F\,(ap_2 \wedge X\,(Fap_3)))))$, which requires the robots to consistently monitor a region by sequentially visiting the areas 1, 2, and 3. If unexpected events occur in areas 1 (e.g., the detection of an intruder), the local task will be triggered to handle such an emergency while respecting the constraints imposed by Φ_G. An example local task is $\Phi_L = F(ap_4 \wedge X(F(ap_1 \wedge X(Fap_3))))$, which requires the robots to visit area 4 first (e.g., report the findings to the base station), and then visit the areas 1 and 3 sequentially. Since Φ_G requires to visit area 2 before visiting area 3, Φ_L may conflict with Φ_G due to the coupling among ap_1, ap_2, and ap_3.

To address this issue, IPDTS is developed to take into account the constraints in both Φ_L and Φ_G. In this work, the local tasks Φ_L are specified by co-safe LTL formulas to represent finite-length temporary tasks while the global task Φ_G is specified by a general LTL formula to capture a richer class of tasks. Let AP_G and AP_L denote the atomic propositions related to Φ_G and Φ_L,

and let $\mathcal{B}_G = (S_G, S_{G0}, \Delta_G, \Sigma_G, \mathcal{F}_G)$ and $\mathcal{A}_L = (S_L, S_{L0}, \Delta_L, \Sigma_L, \mathcal{F}_L)$ represent the corresponding NBA and NFA, respectively. Let $\Pi_{tem} = (\boldsymbol{s}_G, \boldsymbol{s}_L, \boldsymbol{q}, \boldsymbol{C})$ denote the plan of the local tasks Φ_L, where $\boldsymbol{s}_G = s_0^G s_1^G \ldots s_{n_\Pi}^G$ and $\boldsymbol{s}_L = s_0^L s_1^L \ldots s_{n_\Pi}^L$ represent the sequence of automaton states in $\mathcal{B}_G$ and $\mathcal{A}_L$, respectively, $\boldsymbol{q}$ represents the sequence of sub-tasks, and $\boldsymbol{C}$ represents the sequence of tasks allocations.

Before introducing IPDTS, the following assumption is made.

Assumption 1. The temporal local task Φ_L is compatible with the global task Φ_G.

Assumption 1 is mild, since it only requires that Φ_L does not conflict with Φ_G. Otherwise, there does not exist a plan that satisfy Φ_L while respecting Φ_G.

In IPDTS, the tree $\mathcal{T}_\mathcal{D}$ is extended to an interactive planning decision tree, denoted by $\mathcal{IT}_\mathcal{D}$, where the node set is changed to $\mathcal{IV}_T = \{INd_i\}$, $i = 0, 1, 2\ldots$.

Definition 5. The $\mathcal{IT}_\mathcal{D}$ is constructed based on a set of nodes $\{INd_i\}$, $i \in \mathbb{N} \cup \{0\}$, where INd_0 represents the root and each node is defined as a tuple

$$INd_i = \left(GB_s, LB_s, T_s, C_s, ET_{pre}, EP_{pre}\right)$$

where $GB_s \in S_G$ and $LB_s \in S_L$ denote NBA and NFA states of $\mathcal{B}_G$ and $\mathcal{A}_L$, respectively. Similar to the function $PreAuSt$, $PreS_G$ (INd_i), and $PreS_L$ (INd_i) denote the GB_s and LB_s of nodes in the same task stage from INd_0 to INd_i. $Prog(INd_i)$ indicates the global task stage of INd_i and the other functions for INd_i are defined the same as in $\mathcal{T}_\mathcal{D}$.

The IPDTS is developed to construct $\mathcal{IT}_\mathcal{D}$ in real-time and search over it for online reactive task allocation and planning. The general idea of IPDTS is to traverse the states and atomic propositions related to Φ_L first to search for a task plan compatible with Φ_G. If a compatible plan does not exist due to the couplings between Φ_L and Φ_G, IPDTS will traverse the states and atomic proposition related to Φ_G and modify the plan to adapt to Φ_L. The temporary plan Π_{tem} can then be obtained by searching over $\mathcal{IT}_\mathcal{D}$ and then resume the global task using PDTS for the remaining plan.

Specifically, as outlined in Alg. 6, given tasks Φ_G, Φ_L, and the workspace M, NBA $\mathcal{B}_G$, NFA $\mathcal{A}_L$, and ATS T are constructed and the root node INd_0 is initialized based on the current system and sub-tasks (lines 1–2). The search terminates if all nodes in $\mathcal{IT}_\mathcal{D}$ have been traversed, that is, $FlagTra(INd) = 1$, $\forall INd \in \mathcal{IV}_T$ (lines 3–6). For nodes INd that have not been traversed, the function Get_Local, similar to Children_Generation in Alg. 1, generates a set of child candidates of INd, namely, $LNodes$, based on $\mathcal{A}_L$ and $\mathcal{B}_G$, and then determines the flag F_C to indicate if there is conflict between Φ_L and Φ_G (line 8). To deal with the potential conflict, Get_Local searches over all $ap \in AP_L$ and their corresponding automaton states in $\mathcal{A}_L$ and $\mathcal{B}_G$. If there exists an $ap \in AP_L$ feasible with $\mathcal{A}_L$ (i.e., $\exists s \in S_L$ such that $ap \in \Delta_L(INd.LB_s, s)$) but not with $\mathcal{B}_G$ (i.e., $\forall s \in S_G$ such that $ap \notin \Delta_G(INd.GB_s, s)$), we set $F_C = 1$, otherwise $F_C = 0$. If there exists conflict between Φ_L and Φ_G, Get_Global is further invoked to generate another set of child candidates, namely, $GNodes$, based on $\mathcal{B}_G$ (lines 9–13). The goal of Get_Global is to find all feasible $ap \in AP_G$, $ap \notin AP_L$ (i.e., $\exists s \in S_G$ such that $ap \in \Delta_G(INd.GB_s, s)$) and their corresponding automaton states in $\mathcal{A}_L$ and $\mathcal{B}_G$. Due to $ap \notin AP_L$, the LB_s of $GNodes$ is the same with the parent node $INd.LB_s$. Similar to Bound, IBound is developed to generate the child nodes of INd based on the candidates $LNodes$ and $GNodes$ while avoiding unnecessary expansion of $\mathcal{IT}_\mathcal{D}$ (lines 14–16). The constructed set $Children$ is then added to $\mathcal{IT}_\mathcal{D}$. Different from Bound, the traversal rule 3 of $IBound$ is re-defined as $FlagTra(Child) = 1$ if $INd.GB_s = Child.GB_s$, $INd.LB_s = Child.LB_s$, $Prog(INd) = Prog(Child)$, and $Cost(INd) \leq Cost(Child)$. Besides, the traversal rule 1 is re-defined as $FlagTra(INd) = 1$ if $INd.LB_s \in \mathcal{F}_L$.

After $\mathcal{IT}_\mathcal{D}$ is constructed, the node $INd_{\min}$ can be obtained which has the minimum cost to complete Φ_L. Note that the accepting states of Φ_G are reachable[1] from the global NBA state of $INd_{\min}$. The local plan Π_{tem} is then constructed by tracing back from the root node INd_0 to $INd_{\min}$ over $\mathcal{IT}_\mathcal{D}$. Once Φ_L is completed, the robots should continue on with Φ_G, which needs to be re-planned (Lines 29–31). As the accepting NBA states of Φ_G are reachable from the end global NBA state of the local task, the PDTS can be invoked to obtain the plan $\Pi_{pre}\Pi_{tra}\Pi_{suf}$ that starts from the end of local plan and return $\Pi_{tem}\Pi_{pre}\Pi_{tra}\Pi_{suf}$ as the plan of Φ_L and Φ_G.

Algorithm 6 Interactive Planning Decision Tree Search

Input: Φ_G, Φ_L, M, INd_0
Output: $\Pi_{tem}, \Pi_{pre}, \Pi_{tra}, \Pi_{suf}$

1: Construct NBA $\mathcal{B}_G$, NFA $\mathcal{A}_L$, T from Φ_G, Φ_L, M
2: Initialize the tree $\mathcal{IT}_\mathcal{D} = \{INd_0\}$
3: **while** 1 **do**
4: **if** $FlagTra(INd) = 1$ for all $INd \in \mathcal{IV}_T$ **then**
5: break
6: **end if**
7: **for** $INd \in \mathcal{IV}_T$ s.t. $FlagTra(INd) = 0$ **do**
8: $[LNodes, F_C] =$ Get_Local$(INd, \mathcal{A}_L, \mathcal{B}_G, T)$
9: **if** $F_C = 1$ **then**
10: $GNodes =$ Get_Global$(INd, \mathcal{B}_G, T)$
11: **else**
12: $GNodes = \emptyset$
13: **end if**
14: $Children = LNodes + GNodes$
15: $[Children, \mathcal{IT}_\mathcal{D}] =$ IBound$(Children, \mathcal{IT}_\mathcal{D})$
16: add $Children$ to $\mathcal{IT}_\mathcal{D}$
17: **end for**
18: **end while**
19: Select the node $INd_{min} \in \mathcal{IV}_T$ s.t. $Prog(INd_{min}) = oth$ and $Cost(INd_{min})$ is the minimum
20: Initialize $\Pi_{tem} = \emptyset$
21: Trace INd from INd_0 to INd_{min} and add $(INd.GB_s, INd.LB_s, INd.T_s, INd.C_s)$ into Π_{tem}
22: Initialize Nd_0 by the end states of Π_{tem}
23: $[\Pi_{pre}, \Pi_{tra}, \Pi_{suf}] = PDTS(\Phi, M, Nd_0)$
24: Return $\Pi_{tem}, \Pi_{pre}, \Pi_{tra}, \Pi_{suf}$

Remark 1. For the cases of unexpected robot failures, the developed PDTS can be immediately invoked to generate a feasible plan. The updated plan $\Pi_{pre}\Pi_{tra}\Pi_{suf}$ can be obtained by running Alg. 1 using the updated Nd_0 as the new root node. In particular, if a robot is no longer functional, we reset the root node properties, that is, set $Nd_0.B_s$ as the current task state s in the NBA, and set $Nd_0.T_s$, $PreAuSt(Nd_0)$, $Parent(Nd_0)$, and $Nd_0.C_s$ as empty. Then, $Prog(Nd_0)$ satisfies the following condition

$$Prog(Nd_0) = \begin{cases} pre, & Prog = pre, \\ tra, & \text{otherwise}. \end{cases}$$

It indicates that the stage of Φ_G will fall back at least to *tra*, and *Prog* is the current task stage when the robot fails in the real-time system. For each agent r_i, we set $Nd_0.EP_{pre}(i)$ as its current positions and $Nd_0.ET_{pre}(i) = 0$. In addition, if the environment or the task requirement changes that requires re-planning, we set Nd_0 as before and re-plan by PDTS. Thus, if the task stage falls back to *tra*, Π_{pre} will be an empty sequence. Finally, we delete all plan sequences after the current stage in the original plan Π and add $\Pi_{pre}\Pi_{tra}\Pi_{suf}$ to the end of Π, where Π_{suf} is the new suffix loop.

After obtaining a feasible and satisfying plan for HMRS in an environment with the reactive task by Alg. 1 and Alg. 6, the next step is the path planning to realize the designed TAP. The controller records the current automaton state and the completion of each sub-task. When certain sub-task has been completed, the automaton state will be updated and the controller will send the next sub-task for each robot based on TAP. Since the destination of the robots at the next moment can be determined by the plan Π, many existing methods (e.g., potential field based approaches, RRT, probability road map, and receding horizon control) can be employed to realize point-to-point navigation. When encountering obstacles (e.g., non-convex obstacles or specific types of obstacle geometry), methods such as Hauser (2021) and Sawant et al. (2023) can be applied for collision avoidance.

5.4. Discussion

The predicted completion time for robots in Alg. 1 and Alg. 6 are calculated based on the robots' maximal linear velocity and the distances between the robots' current positions and the goal positions corresponding to the current node task. Such an estimate time can be conservative, as robots may not move along straight lines in the environment. Since we do not assume full knowledge of the environment and have no clues about the future robot paths before task allocation, we use the estimated times as a lower bound of the estimated arrival time in this work, based on which the task allocation is then explored over the PDT. Hence, the estimated time can be considered as a lower bound of the task operation times, rather than the actual operation times of robots, which is used to determine the costs of task allocations and solve task allocations. Although various practical factors (e.g., slippery of terrain and disturbances) can affect the predicted completion time, $Nd.C_s$ and $Nd.T_s$ are guaranteed by selection from sequentially feasible task allocations and sub-tasks (via Alg. 2, lines 2–3 and Alg. 3, lines 16–19). Therefore, disturbances do not compromise the feasibility of the proposed methods. Then, the real-time framework assigns current target sub-tasks and paths to each agent based on the generated plan and current task state. The current task state, tracked through $\boldsymbol{s} \in \Pi$, is updated using real-time feedback from system states rather than relying on predicted completion times in T_D. Hence, the impact of disturbances during task execution can be mitigated.

6. Algorithm analysis

This section investigates the performance of PDTS and IPDTS in the following aspects: the feasibility, the completeness, and the algorithm complexity. The feasibility indicates if the generated plan is applicable and the algorithm completeness indicates a feasible solution, if exits, is guaranteed to be found.

6.1. The performance of PDTS

Before investigating PDTS, we first show in Lemma 1 that Request_Response in Alg. 3 is feasible and complete in terms of task allocation. Theorem 1 shows the PDTS, without using the traversal rules, is guaranteed to find a feasible task allocation and plan for the heterogeneous multi-robot system. We then show in Lemma 2–4 that the traversal rules do not compromise the feasibility and completeness of PDTS, which concludes in Theorem 2 that PDTS with traversal rules can search for the task allocation and plan more efficiently.

Lemma 1. *The* Request_Response *in Alg.* 3 *is feasible and complete, and the task allocation is locally optimal.*

Proof. Given the current tree $\mathcal{T}_\mathcal{D}$, suppose $q \in Q$ is the atomic task to be performed. To allocate the atomic task for robots, Request_Response in Alg. 3 evaluates each robot based on the updated ET_{pre} (i.e., the estimated time to perform q). Since the robots of each type $j \in [n_c]$ are sorted based on ET_{pre} and selected according to $tk \in TK$, where $tk = LT(q)$ specifies the numbers and types of robots required to perform q, the task requirement TK is always satisfied, which indicates the feasibility of task allocation using Request_Response is guaranteed. Due to the proposed task allocation mechanism, as long as there are sufficient number and types of robots for the required tk of q (i.e., the existence of solution), the task allocation is guaranteed to be found and thus Request_Response is complete.

To show the task allocation generated by Request_Response is locally optimal, we show that the robots can perform the current atomic task with a minimum cost based on the current estimated time and position. Recall

that, for $\forall j \in [n_c]$, the robots in $\mathcal{K}_j$ are sorted from small to large according to $\tilde{ET}_{pre}$. Since the first n_j robots from $\mathcal{K}_j$ are selected, where $n_j = tk(j)$ denotes the number of robots of type j required for ap, it indicates the task allocation for robots of type j is locally optimal with respect to (2). Following similar analysis, the same conclusion applies to any type $j \in [n_c]$. In addition, since each robot is of only one type, it can only be assigned for a particular $tk(l)$, $l \in [n_c]$. That is, each entry in tk can be individually determined. Hence, due to the use of greedy algorithm in each step, the task allocation by Request_Response is locally optimal with respect to the metric (2).

Theorem 1. *Give the LTL formula* Φ, *the task requirement TK, and the environment M, if there exists a task allocation and plan satisfying* Φ *in prefix-transition-suffix structure, the PDTS in Alg. 1 is ensured to find it without using traversal rules.*

Proof. Both task allocation and mission planning are considered in PDTS. Since the feasibility and completeness of the task allocation will not affect that of the task planning as indicated in Alg. 1, they can be individually investigated. Given that the task allocation has been proven to be feasible and complete in Lemma 1, we only need to show that the task plan is also feasible and complete.

If there exists a plan $\Pi_{pre}\Pi_{tra}\Pi_{suf}$ satisfying the task Φ in the given environment M, the corresponding trajectories of task points, automation states, and allocation states are $\boldsymbol{q}$, $\boldsymbol{s}$, $\boldsymbol{C}$, respectively, and the atomic task $\pi_i = LA(q_i)$ satisfies $\pi_i \in \Delta(s_{i-1}, s_i)$, $\forall \pi_i \in \boldsymbol{\pi}$. Note that a core idea of *PDTS* is to build a tree $\mathcal{T}_{\mathcal{D}}$. By definition, $Nd_0.B_s = s_0 \in \boldsymbol{s}$,, $Nd_0.T_s = q_0 \in \boldsymbol{q}$, $LA(Nd_0.T_s) = \pi_0 \in \boldsymbol{\pi}$. Since $\mathcal{T}_{\mathcal{D}}$ traverses all possible nodes, there exists a node $Nd \in \mathcal{V}_T$ such that $Nd.B_s = s_1 \in \boldsymbol{s}$, $Nd.T_s = q_1 \in \boldsymbol{q}$, $LA(Nd.T_s) = \pi_1 \in \boldsymbol{\pi}$. Then, for $\forall (q_k, s_k, C_k) \in (\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{C})$, there always exists a node $\tilde{Nd} \in \mathcal{V}_T$, which satisfies $\tilde{Nd}.B_s = s_k$, $\tilde{Nd}.T_s = q_k$. Hence, by tracing back the path of nodes until the root Nd_0, PDTS is ensured to find a feasible task allocation and plan.

Theorem 1 shows that, without using traversal rules, the PDTS in Alg. 1 is complete and feasible. The following lemmas and theorems show that the traversal rules only reduce the search space without compromising the feasibility and completeness of PDTS. To facilitate the analysis, we define a cost function $W : \boldsymbol{q} \times \boldsymbol{C} \to \mathbb{R}$ that evaluates the performance of the task sequence $\boldsymbol{q}$ and the task allocation sequence $\boldsymbol{C}$ and define a function $bestC : \boldsymbol{q} \to \boldsymbol{C}$ that maps the task sequence to an optimal allocation with the least cost.

Lemma 2. *Suppose there are two task sequences* $\boldsymbol{q} = q_0 \ldots q_i q_{i+1} \ldots q_e$ *and* $\boldsymbol{q}^* = q_0 \ldots q_i q_t q_{i+1} \ldots q_e$, *where q* is the same with q but differs in containing an additional temporary task* q_t *in the middle. It holds that* $W(\boldsymbol{q}, bestC(\boldsymbol{q})) \le W(\boldsymbol{q}^*, bestC(\boldsymbol{q}^*))$.

Proof. Given $q_0, q_t, q_e \in Q$, consider two task sequences $\boldsymbol{q}^1 = q_0 q_e$ and $\boldsymbol{q}^2 = q_0 q_t q_e$. Let $\boldsymbol{C}^1 = bestC(\boldsymbol{q}^1) = C_0^1 C_e^1$ and $\boldsymbol{C}^2 = bestC(\boldsymbol{q}^2) = C_0^t C_t^t C_e^t$ denote the optimal task allocation for $\boldsymbol{q}^1$ and $\boldsymbol{q}^2$, respectively. Since C^1 is the optimal task allocation for $\boldsymbol{q}^1$, the task allocation $C_0^1 C_e^1$ must have a smaller cost than any other allocation scheme (e.g., $C_0^2 C_e^2$). Therefore, it holds that $W(\boldsymbol{q}^1, C_0^2 C_e^2) \ge W(\boldsymbol{q}^1, C_0^1 C_e^1)$.

Comparing $\boldsymbol{q}^1$ and $\boldsymbol{q}^2$, there is an additional task q_t in $\boldsymbol{q}^2$. By definition, the predicted arrival time of each agent is the time that the agent arrives at its last participating task. After the sub-task q_0 with C_0^2 has been completed, for the agent not in C_t^2, the predicted arrival time of $\boldsymbol{q}^2$ is equal to $\boldsymbol{q}^1$. For the agent in C_t^2, suppose its predicted completion position and time after q_0 are p_0 and t_0, respectively. In this case, if the agent is in $c_{2,e}$, the predicted arrival time of $\boldsymbol{q}^2$ is $t_0 + t_w + \frac{1}{v}(\|p_0 - LM(q_t)\| + \|LM(q_t) - LM(q_e)\|)$, where $t_w \ge 0$ is the waiting time for the arrival of other agents. It is larger than the predicted arrival time of $\boldsymbol{q}^1$, that is, $t_0 + \frac{1}{v}(\|p_0 - LM(q_e)\|)$. If the agent is not in $c_{2,e}$, the predicted arrival time of $\boldsymbol{q}^2$ is $t_0 + \frac{1}{v}(\|p_0 - LM(q_t)\|)$, which is larger than the predicted arrival time of $\boldsymbol{q}^1$, that is, t_0. Therefore, for each agent, the predicted arrival time of $\boldsymbol{q}^2$ is larger than $\boldsymbol{q}^1$. The predicted completion time of agents in $c_{2,e}$ is the maximum predicted arrival time of all agents. As the cost is the maximum predicted completion time of all agents, it always holds $W(\boldsymbol{q}^2, \boldsymbol{C}^2) = W(\boldsymbol{q}^2, C_0^2 C_t^2 C_e^2) \ge W(\boldsymbol{q}^1, C_0^2 C_e^2)$. Since $W(\boldsymbol{q}^1, C_0^2 C_e^2) \ge W(\boldsymbol{q}^1, C_0^1 C_e^1) = W(\boldsymbol{q}^1, \boldsymbol{C}^1)$, it must hold $W(\boldsymbol{q}^1, \boldsymbol{C}^1) \le W(\boldsymbol{q}^2, \boldsymbol{C}^2)$.

Following similar analysis, if there are two task sequences $\boldsymbol{q} = q_0 \ldots q_i q_{i+1} \ldots q_e$ and $\boldsymbol{q}^* = q_0 \ldots q_i q_t q_{i+1} \ldots q_e$, where $\boldsymbol{q}^*$ is the same with $\boldsymbol{q}$ but differs in containing an additional temporary task q_t in the middle. It holds that $W(\boldsymbol{q}, bestC(\boldsymbol{q})) \le W(\boldsymbol{q}^*, bestC(\boldsymbol{q}^*))$.

Lemma 3. *Given an optimal plan* $\Pi_{best} = \Pi_{pre}\Pi_{tra}\Pi_{suf}$, *in any stage of the optimal plan* $\Pi = (\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{C}) \in \{\Pi_{pre}, \Pi_{tra}, \Pi_{suf}\}$, *the states in* $\boldsymbol{s}$ *are all different, that is.,* $s_i \ne s_j$, $\forall s_i, s_j \in \boldsymbol{s}$, $i \ne j$.

Proof. Given an optimal plan $\Pi_{pre}\Pi_{tra}\Pi_{suf}$, suppose its corresponding states sequence is $\boldsymbol{s} = s_0 s_1 \ldots s_n$ and the propositions sequence is $\boldsymbol{\pi} = \pi_0 \pi_1 \ldots \pi_n = LA(\boldsymbol{q})$. If there exist $s_i = s_j \in S$, $i \le j$, satisfying $\Delta(s_i, s_{j+1}) \ne \emptyset$ and $\pi_{j+1} \in \Delta(s_i, s_{j+1})$, there must exist a new state sequence $\boldsymbol{s}_{new} = s_0 \ldots s_i s_{j+1} \ldots s_n$ and a new propositions sequence $\boldsymbol{\pi}_{new} = LA(\boldsymbol{q}_{new}) = \pi_0 \ldots \pi_i \pi_{j+1} \ldots \pi_n$ that satisfy the same task Φ in the environment M. According to Lemma 2, there exists a plan whose cost is smaller than Π_{best}, that is, $W(\boldsymbol{q}_{new}, bestC(\boldsymbol{q}_{new})) \le W(\boldsymbol{q}, bestC(\boldsymbol{q}))$, which contradicts the optimality of Π. Hence, the states in $\boldsymbol{s}$ are all different.

To facilitate the analysis, let Get_Path denote a function that takes the tree $\mathcal{T}_{\mathcal{D}}$ and a node $Nd \in \mathcal{V}_T$ as input and output a sequence of nodes from the root Nd_0 to Nd in $\mathcal{T}_{\mathcal{D}}$, that is, $\mathcal{P} = \text{Get\_Path}(Nd, \mathcal{T}_{\mathcal{D}})$, where $\mathcal{P} = Nd_0 Nd_1 \ldots Nd_{|\mathcal{P}|}$ is a sequence of nodes satisfying $Nd_{i-1} = Parent(Nd_i)$, $i = 1, 2, \ldots, |\mathcal{P}|$, $Nd_{|\mathcal{P}|} = Nd$. Let Planning denote a

function that maps the node sequence to a plan, that is, $\Pi = \Pi_0\Pi_1\ldots\Pi_{|\mathcal{P}|} = \mathsf{Planning}(\mathcal{P})$, where $\Pi_i = (Nd_i.B_s, Nd_i.T_s, Nd_i.C_s)$, $i = 0, \ldots, |\mathcal{P}|$.

Lemma 4. *The traversal rule 2 does not affect the local optimality and completeness of PDTS and the traversal rule 3 does not affect its completeness.*

Proof. According to Theorem 1, even without using the traversal rules, the PDTS can still obtain a feasible task allocation and plan. Suppose Nd_{end} is the leaf node with the least cost in $\mathcal{T}_\mathcal{D}$. Then $\mathcal{P} = Nd_0Nd_1\ldots Nd_{end}$ is an optimal path in $\mathcal{T}_\mathcal{D}$ generated by $\mathcal{P} = \mathsf{Get\_Path}(Nd_{end}, \mathcal{T}_\mathcal{D})$ and the corresponding optimal plan is $\Pi = \mathsf{Planning}(\mathcal{P}) = (\boldsymbol{q}, \boldsymbol{s}, \boldsymbol{C})$ with $\boldsymbol{\pi} = LA(\boldsymbol{q}) = \pi_0\pi_1\ldots\pi_{|\Pi|}$. If there exists two nodes $Nd_i, Nd_j \in \mathcal{P}$, $i < j$, such that $Nd_i.B_s = Nd_j.B_s$ and $Prog(Nd_i) = Prog(Nd_j)$, according to Lemma 3 $W(\boldsymbol{q}_{new}, bestC(\boldsymbol{q}_{new})) \leq W(\boldsymbol{q}, bestC(\boldsymbol{q}))$ with $\boldsymbol{\pi}_{new} = LA(\boldsymbol{q}_{new}) = [\pi_0, \ldots, \pi_i, \pi_{j+1}, \ldots, \pi_n]$.

Therefore, $\mathcal{P}$ is not the optimal path in $\mathcal{T}_\mathcal{D}$, which contradicts to the assumption. Hence, the traversal rule 2 does not affect the local optimality and completeness of PDTS.

For the traversal rule 3, suppose there exist two nodes $Nd_i \notin \mathcal{P}$ and $Nd_j \in \mathcal{P}$ in $\mathcal{T}_\mathcal{D}$ such that $Nd_i.B_s = Nd_j.B_s$, $Prog(Nd_i) = Prog(Nd_j)$, $Cost(Nd_i) \leq Cost(Nd_j)$. Then, there exists a new node sequence $\mathcal{P}_{new} = Nd_0\ldots Nd_iNd_{j+1}\ldots Nd_{end}$ that satisfies the same task Φ. It indicates that if the optimal plan exists and is not in $\mathcal{T}_\mathcal{D}$ because of the traversal boundary rules, there must exist an approximate sub-optimal path satisfying Φ. Hence, we can always find a satisfying plan (e.g., $\mathcal{P}_{new}$) after applying the traversal rule 3.

Theorem 2. *PDTS with Traversal Rules has feasibility and completeness.*

Proof. By Theorem 1 and Lemma 4, the PDTS with the traversal rules is feasible and complete. Since the function Request_Response uses greedy algorithm to ensure that each sub-task in each step is locally optimal, the task allocation and planning in PDTS is thus also locally optimal.

Remark 2. For each node $Nd \in \mathcal{V}_T$, PDTS selects the task allocation based on the current sub-task $Nd.T_s$ and the system states $Nd.EP_{pre}$ and $Nd.ET_{pre}$, without considering the potential impact of current decision on the subsequent future sub-tasks. In other word, the task allocation might be myopic. Therefore, PDTS compromises the global optimality and is more suitable for quickly obtaining a satisfying plan.

6.2. The performance of IPDTS

This section investigates the feasibility and completeness of IPDTS. Since the same Request_Response is used, by Lemma 1 the task allocation in IPDTS is feasible with the task requirement *TK*. Therefore, we focus on the task planning of IPDTS.

First, we divide tasks by coupling and obtain state transitions of temporary and global tasks to verify the feasibility of IPDTS. We then show that IPDTS can find a satisfying plan as long as it exits. Similar to the analysis of PDTS, let Get_IPath denote a function that takes the tree $\mathcal{IT}_\mathcal{D}$ and a node $INd \in \mathcal{IV}_T$ as input and output a sequence of nodes from the root INd_0 to INd in $\mathcal{IT}_\mathcal{D}$, that is, $\mathcal{P} = \mathsf{Get\_IPath}(INd, \mathcal{IT}_\mathcal{D})$ where $\mathcal{P} = INd_0INd_1\ldots INd_{|\mathcal{P}|}$ is a sequence of nodes satisfying $INd_{i-1} = Parent(INd_i)$, $i = 1, 2, \ldots, |\mathcal{P}|$, and $INd_{|\mathcal{P}|} = INd$. Let IPlanning denote a function that maps the node sequence to a plan, that is, $\Pi = \Pi_0\Pi_1\ldots\Pi_{|\mathcal{P}|} = \mathsf{Planning}(\mathcal{P})$, where $\Pi_i = (INd_i.GB_s, INd_i.GB_s, INd_i.T_s, INd_i.C_s)$, $i = 1, \ldots, |\mathcal{P}|$.

Theorem 3. *The IPDTS in Alg. 6 is feasible, that is, the generated task allocation and plan satisfies both Φ_L and Φ_G.*

Proof. In Alg. 6, for each parent node *INd*, there may exist two classes of child nodes. The first class is *LNodes* generated by Get_Local, which searches over all sub-tasks $ap \in AP_L$ and their corresponding global NBA states s_G and local NFA states s_L that satisfy $ap \in \Delta_L(INd.LB_s, s_L) \cup \Delta_G(INd.GB_s, s_G)$. The other class is *GNodes* generated by Get_Global, which searches over all sub-tasks $ap \in AP_G$ and $ap \notin AP_L$, and their corresponding global NBA states s_G that satisfy $ap \in \Delta_G(INd.GB_s, s_G)$. Therefore, there are three possible cases for each node $Child \in LNodes \cup GNodes$, which are (1) $LA(Child.T_s) \in \Delta_L(INd.LB_s, Child.LB_s)$, or (2) $LA(Child.T_s) \notin AP_L$ and $LA(Child.T_s) \in \Delta_G(INd.GB_s, Child.LB_s)$, or (3) $LA(Child.T_s) \notin AP_G$.

If there exists $Prog(INd) = oth$, then there exists a plan $\Pi_{tem} = (\boldsymbol{s}_G, \boldsymbol{s}_L, \boldsymbol{q}, \boldsymbol{C}) = \mathsf{Planning}(\mathcal{P})$, where $\mathcal{P}$ is generated by $\mathsf{Get\_IPath}(INd, \mathcal{IT}_\mathcal{D})$. For any $INd_i \in \mathcal{P}\backslash\backslash INd_0$, one has $LA(INd_i.T_s) \in \Delta_L(Parent(INd_i).LB_s, INd_i.LB_s)$ or $LA(INd_i.T_s) \notin AP_L$, $LA(INd_i.T_s) \in \Delta_G(Parent(INd_i).GB_s, INd_i.GB_s)$ or $LA(INd_i.T_s) \notin AP_G$. Furthermore, $\forall q_i \in \boldsymbol{q}, i \neq 0$, if $LA(q) \in AP_L$, then $LA(q) \in \Delta_L(s^L_{i-1}, s^L_i)$ and if $LA(q) \in AP_G$, then $LA(q) \in \Delta_G(s^G_{i-1}, s^G_i)$. Therefore, if there exists $INd.GB_s$ such that $\mathcal{F}_G$ is reachable, Π_{tem} satisfies Φ_L and does not violate Φ_G. Then, in Alg. 6, PDTS can find the new plan $\Pi_{pre}\Pi_{tra}\Pi_{suf}$, which satisfies Φ_G. Therefore, The IPDTS in Alg. 6 is feasible, that is, the generated task allocation and plan satisfies both Φ_L and Φ_G.

Lemma 5. *Consider a feasible temporary plan $\Pi_{tem} = \Pi^{tem}_0\Pi^{tem}_1\ldots\Pi^{tem}_{n_\Pi} = (\boldsymbol{s}_G, \boldsymbol{s}_L, \boldsymbol{q}, \boldsymbol{C})$, where $\boldsymbol{q} = q_0q_1\ldots q_{n_\Pi}$, $\boldsymbol{s}_G = s^G_0\ldots s^G_{n_\Pi}$, $\boldsymbol{s}_L = s^L_0\ldots s^L_{n_\Pi}$. For $\forall\Pi^{tem}_i \in \Pi_{tem}$ with $s^L_i \in \boldsymbol{s}_L$ and $s^G_i \in \boldsymbol{s}_G$, if there exists a node $INd \in \mathcal{IV}_T$ with $INd.LB_s = s^L_i$, then there must exist a node $INd^* \in \mathcal{IV}_T$ with $INd^*.LB_s = s^L_i$ and $s^G_i \in R(INd^*.GB_s)$, that is, s^G_i is reachable from $INd^*.GB_s$.*

Proof. Since Π_{tem} is feasible, $LA(q_i) \notin AP_G$ or $LA(q_i) \in \Delta_G(s^G_{i-1}, s^G_i)$ for any $q_i \in \boldsymbol{q}$. As indicated in Alg. 6,

by traversing $ap \in AP_L \cup AP_G$ to construct the tree $\mathcal{IT}_\mathcal{D}$, all feasible pairs of (s_i^G, s_i^L) can be obtained. Then, by traversing $ap \in AP_L$, the transitions of states in $\mathcal{B}_G$ will be further reduced. Therefore, if there exists $INd.LB_s = s_i^L$, then there exists a node $INd* \in \mathcal{IV}_T$ such that $INd*.LB_s = s_i^L$ and $INd*.GB_s \in (s_0^G, \ldots, s_i^G)$. Since $\forall s_i^G \in \boldsymbol{s}_G$ and $s_i^G \neq s_0^G$, one has $R(s_i^G) \subset R(s_{i-1}^G)$, which further indicates that s_i^G is reachable from $INd*.GB_s$.

Theorem 4. *The IPDTS in Alg.* 6 *is complete, that is, if exist, IPDTS is guaranteed to find a feasible task allocation and plan that satisfies* Φ_L *without violating* Φ_G.

Proof. Suppose $\Pi_{tem} = \Pi_0^{tem}\Pi_1^{tem}\ldots\Pi_{n_\Pi}^{tem} = (\boldsymbol{s}_G, \boldsymbol{s}_L, \boldsymbol{q}, \boldsymbol{C})$ is a feasible temporary plan that satisfies Φ_L without violating Φ_G. To prove that Π_{tem} is ensured to be found by the IPDTS in Alg. 6, recall that the root node INd_0 of $\mathcal{IT}_\mathcal{D}$ is initialized as the current automaton and system states. That is, Π_0^{tem} is set according to the properties of INd_0. The idea of the following proof is to show that, for the node INd in $\mathcal{IT}_\mathcal{D}$ corresponding to Π_i^{tem}, $\forall i \in [n_\Pi - 1] \cup \{0\}$, there always exists a child node of INd corresponding to Π_{i+1}^{tem}.

Specifically, suppose the sequence $\boldsymbol{s}_L = s_1^L\ldots s_{n_\Pi}^L$ in Π_{tem} satisfies that $s_i^L \notin \mathcal{F}_L$, $\forall i \in [n_\Pi - 1]$, and $s_{n_\Pi}^L \in \mathcal{F}_L$, where $\mathcal{F}_L$ is the accepting set of $\mathcal{A}_L$, which indicates Π_{tem} completes the local task Φ_L in the last step. Consider a state $s_i^L \in \boldsymbol{s}_L$, $i \in [n_\Pi - 1]$, and let INd denote the node in $\mathcal{IT}_\mathcal{D}$ such that $INd.LB_s = s_i^L$. As indicated in Alg. 6, to generate the child nodes of INd, the state $s \in S_L$ and $ap \in AP_L$ related to the local task Φ_L are first traversed, and the states $s \in S_G$ and $ap \in AP_G$ will be traversed only when there is conflict between Φ_L and Φ_G. Hence, there are two possible cases.

(1) If $F_C = 0$, that is, there is no conflict between Φ_L and Φ_G, the transitions along the sequences $\boldsymbol{s}_G$ and $\boldsymbol{s}_L$ in the tree $\mathcal{IT}_\mathcal{D}$ are feasible with Φ_L and Φ_G. Therefore, given that $INd.LB_s = s_i^L$ and $INd.GB_s = s_i^G$, there must exist a node $INd_s \in LNodes$ that satisfies $INd_s.LB_s = s_{i+1}^G$, which indicates the state transitions over $\mathcal{B}_G$ will not be affected during the task operation of Φ_L.

(2) If $F_C = 1$, due to the conflicts between Φ_L and Φ_G, the existence of $INd_j \in LNodes$ that satisfies $INd_s.LB_s = s_{i+1}^L$ depends on $INd.GB_s$. If $INd.GB_s = s_i^G$, there must exist a node $INd_j \in LNodes$ that satisfies $INd_s.LB_s = s_{i+1}^L$, since $LA(q_{i+1}) \notin AP_G$ or $LA(q_{i+1}) \in \Delta_G(s_i^G, s_{i+1}^G)$. If $INd.GB_s \neq s_i^G$, by Lemma 5 and $INd.LB_s = s_i^L$, there must exist a node $INd* \in \mathcal{IV}_T$ such that s_i^G is reachable from $INd*.GB_s$ over $\mathcal{B}_G$ and $INd*.LB_s = INd.LB_s = s_i^L$. For the expansion of $INd*$, if there does not exist a node $INd_s^* \in LNodes$ that satisfies $INd_s^*.LB_s = s_{i+1}^L$, then $\mathcal{IT}_\mathcal{D}$ will generate nodes by the global task. Therefore, for the following expansion, if there does not exist a node $INd_s^{**} \in LNodes$, which satisfies $INd_s^{**}.LB_s = s_{i+1}^L$, the global state will change and keep the same local NFA state. As s_i^G is reachable from $INd*.GB_s$ over $\mathcal{B}_G$, for the following expansion, there may exist a node $INd**$, $INd* \in \mathsf{Get\_IPath}(INd**, \mathcal{IT}_\mathcal{D})$, $INd**.LB_s = s_i^L$, $INd**.GB_s = s_i^G$. Then, for the next expansion of $INd**$, there exists $INd_s^{**} \in LNodes$, which satisfies $INd_s^{**}.LB_s = s_{i+1}^L$.

In summary, there always exists $INd_0 \in \mathcal{IV}_T$ with $INd_0.LB_s = s_0^L$ and if there exists $INd_{parent} \in \mathcal{IV}_T$, $INd_{parent}.LB_s = s_i^L$, then there exists $INd_{child} \in \mathcal{IV}_T$, $INd_{child}.LB_s = s_{i+1}^L$. Therefore, there always exists a node $INd \in \mathcal{IV}_T$ such that $INd.LB_s = s_{n_\Pi}^L \in \mathcal{F}_L$. By Lemma 5, there exists a node $INd*$ such that $INd*.LB_s = s_{n_\Pi}^L \in \mathcal{F}_L$ and $s_{n_\Pi}^G$ is reachable from $INd*.GB_s$. As $\mathcal{F}_G$ is reachable from $s_{n_\Pi}^G$, $\mathcal{F}_G$ also reachable from $INd*.GB_s$, which indicates that Π_{tem} does not violate Φ_G. Then $\Pi = \mathsf{Planning}(\mathsf{Get\_IPath}(INd, \mathcal{IT}_\mathcal{D}))$ is a feasible solution that can complete Φ_L without violating Φ_G. Since $\mathcal{F}_G$ is reachable from the ending NBA states of Φ_G, if there exist feasible plannings, PDTS can obtain one of them by Theorem 2. Therefore, the IPDTS in Alg. 6 is complete, that is, if exist, IPDTS is guaranteed to find a feasible task allocation and plan that satisfies both Φ_L and Φ_G.

Remark 3. Theorem 3 and 4 proves feasibility and completeness of IPDTS. Nevertheless, the task allocation is obtained based on the previous plan without considering future sub-tasks, and thus is only locally optimal.

6.3. Complexity of PDTS

Based on the traversal rules, the number of nodes can be further reduced after tree pruning. According to the traversal rule 3, there is only one node with the same NBA state and stage flag. Then, if the extra nodes are pruned by the traversal rule 3, there exist at most $|S|$ nodes in the prefix, transition, and other stages and there are at most $|\mathcal{F}| \times |S|$ nodes in the suffix stage. Therefore, the maximum number of nodes in $\mathcal{T}_\mathcal{D}$ is $(3 + |\mathcal{F}|) \times |S|$ and the space complexity of $|S|$ is $O(n)$.

According to the traversal rules 1 and 2, the traversal time is finite. In each stage of a path of nodes, the NBA states can be different. Therefore, the length of node path in each stage is not larger than $|S|$. The maximum number of nodes in prefix, transition, and suffix stages with the last node in the other stage is at most $3 \times |S| + 1$. Therefore, the traversal time is at most $3 \times |S|$. For each traversal, according to rule (3), among the nodes with the same node status and phase flag, only the one with the minimum cost value can be traversed. Specially, there are only $|S|$, $|S|$, and $|\mathcal{F}| \times |S|$ parent nodes in the prefix, transition, and suffix stages, respectively. Therefore, the number of parent nodes is at most $(2 + |\mathcal{F}|) \times |S|$. According to Children_Generation, at most $|S| \times |Q|$ nodes can be generated for a parent node.

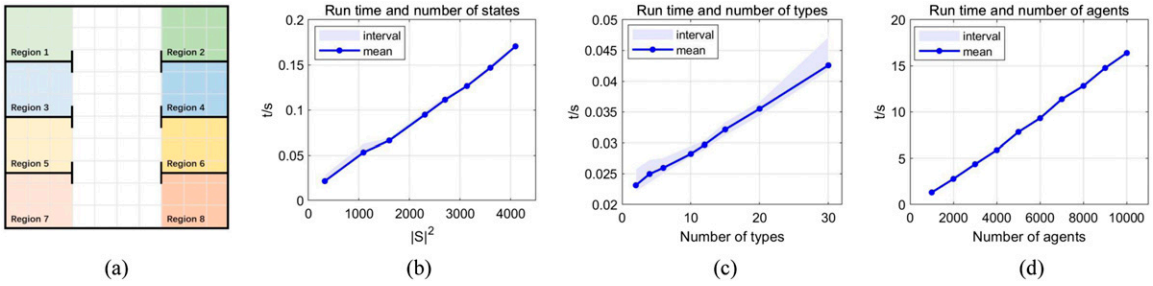


Figure 5. (a) The environment contains eight regions of interest labeled as ap_i, $i \in [8]$. The robots are randomly deployed in the environment initially. (b) The plot of solution time that varies with the number of NBA states $|S|^2$. (c) The plot of solution time that varies with the number of robot types. (d) The plot of solution time that varies with the number of robots.

Hence, the maximum number of total exploration is $3 \times |S| \times ((2 + |\mathcal{F}|) \times |S|) \times (|S| \times |Q|)$ and the upper bound of time complexity is $O(n^3)$ for the size of NBA $|S|$. Since the number of child nodes is in general linearly proportional to $|Q|$, the actual time complexity is close to $O(n^2)$.

Noted that the agents may not all be in the areas of interest at the same time, a dense map is often employed to fully express the system states in the product-based methods (Ulusoy et al., 2013) or sampling-based methods (Kantaros and Zavlanos, 2020). In this work, the system states are modeled by the predicted completion time and position, allowing each agent has different predicted timestamps for the current sub-task. Therefore, a sparse map with only areas of interest can be applied and the system states can be obtained directly by the states iteration without searching. The task allocation can be obtained by calculating the predicted arrival time of agents and selecting the agents with smaller predicted arrival time for each type. Therefore, the time and space complexity of PDTS are both $O(n)$ for n_a agents of n_c types.

6.4. Complexity of IPDTS

In IPDTS, the traversal rule 3 is changed as follows. If there exist nodes $INd_i, INd_j \in \mathcal{IV}_T$ such that $INd_i.GB_s = INd_j.GB_s$, $INd_i.LB_s = INd_j.LB_s$, $Prog(INd_i) = Prog(INd_j)$, and $Cost(INd_i) < Cost(INd_j)$, INd_j and all of its child nodes are removed. Then, similar with the analysis for $\mathcal{T}_\mathcal{D}$, it holds in $\mathcal{IT}_\mathcal{D}$ that $|\{(INd.GB_s, INd.LB_s) | INd \in \mathcal{IV}_T\}| \leq (3 + |\mathcal{F}_G|) \times |S_G| \times |S_L|$. Therefore, the maximum number of nodes of $\mathcal{IT}_\mathcal{D}$ is $(3 + |\mathcal{F}_G|) \times |S_G| \times |S_L|$, that is, its space complexity is $O(n^2)$.

In each traversal, at least one automaton state of the local and global tasks changes. According to the traversal rule 2 for IPDTS, there is no same automaton states in the same stage in both local and global tasks. Therefore, for the local task with one stage and the global task with three stages, the maximum number of traversal is $1 \times |S_L| + 3 \times |S_G|$. According to the traversal rule 3, at most $|S_G| \times |S_L|$ nodes can generate child nodes in the prefix stage of the global task. Then, there are at most $|S_G| \times |S_L|$ parent nodes for the transition stage and $|\mathcal{F}_G| \times |S_G| \times |S_L|$ parent nodes for the suffix stage. Therefore, the number of parent nodes is at most $(2 + |\mathcal{F}_G|) \times |S_G| \times |S_L|$. Since at most $|S_G| \times |S_L| \times |Q|$ nodes can be generated for a single parent node, the maximum number of total exploration is $(|S_G| + 3 \times |S_L|) \times (2 + |\mathcal{F}_G|) \times (|S_G| \times |S_L|)^2 \times |Q|$. Therefore, the time complexity of Φ_L and Φ_G is both $O(n^3)$. Due to the use of Request_Response, the time and space complexity of IPDTS remains $O(n)$ for n_a agents of n_c types.

7. Numerical simulations

Numerical simulations are carried out in this section to evaluate the performance of the developed TAP framework. Since PDTS and IPDTS are the core planning algorithms, we first evaluate their performance under different atomic propositions, agent numbers up to 10^4, and agent classes that range from dozens to hundreds in Sections 7.1 and 7.3, respectively. The comparison of PDTS and previous method is provided in Section 7.2 to show the superiority of our approach. An example is then provided in Section 7.4 to demonstrate the effectiveness of TAP in handling a variety of reactive tasks, such as task changes, agent failures, and local temporary tasks. Throughout this simulation, LTL2STAR is used to convert an LTL formula to NBA (Gastin and Oddoux, 2001) and Matlab 2019 is used for numerical simulation.

7.1. Performance of PDTS

We first evaluate the performance of PDTS in terms of the solution time used to find a satisfying plan. Consider an environment consisting of eight areas of interest in Figure 5. Suppose there are two types of robots and each type has four robots, that is, $n_c = 2$, $n_a = 8$. The atomic task ap_i, $i = 1, \ldots, 8$, represents the task of visiting area i, respectively. For each atomic task, the task requirement is defined as $tk = (2, 2)$. The solution time under different LTL tasks is shown in Table 1. Figure 5(b) indicates that the solution time is approximately linearly proportional to $|S|^2$, which is less than the upper bound of the time complexity in Section 6.3.

Table 1. Solution time for different LTL tasks.

$\|AP\|$	$\|S\|$	Time (/s)	$\|AP\|$	$\|S\|$	Time (/s)
3	8	0.00330	6	56	0.126
4	16	0.0113	6	60	0.148
5	32	0.0402	6	64	0.170
6	18	0.0214	7	96	0.410
6	33	0.0532	7	128	0.754
6	40	0.0661	8	196	1.88
6	48	0.0947	8	256	3.53
6	52	0.111			

Table 2. Solution time for different robot types.

n_c	n_a	Time (/s)	n_c	n_a	Time (/s)
2	2 × 30	0.0231	12	12 × 5	0.0297
4	4 × 15	0.0250	15	15 × 4	0.0321
6	6 × 10	0.0260	20	20 × 3	0.0355
10	10 × 6	0.0282	30	30 × 2	0.0426

To show how the performance of PDTS varies with the robot types, we consider a group of 60 robots performing an LTL task $\Phi_G = Fap_1 \wedge Fap_2 \wedge Fap_3 \wedge Fap_4$. Suppose each type has the same number of robots. The simulation results are listed in Table 2. Figure 5(c) shows that the solution time is linearly proportional to the number of robot types. Since the total number of agents remains unchanged during the simulation and only the number of robot type changes, according to the complexity analysis in Section 6.3, the time complexity of the PDTS is $O(n)$, which is consistent with the simulation results.

To show how the performance of PDTS varies with the number of robots, we consider the same LTL task Φ_G and fix the number of robot types, that is, $n_c = 100$ when $n_a \geq 1000$. Suppose each type has the same number of agents and each task requires half group of each type of robots. The simulation results with various robot numbers are listed in Table 3. Figure 5(d) shows that the solution time is linearly proportional to the robot number, which is consistent with the analysis in Section 6.3 that the algorithm's time complexity is $O(n)$.

7.2. Performance comparison

Since MILP is a SOTA method for TAP (Leahy et al., 2021, 2022), our approach is further compared with MILP. We consider an environment with four areas of interest and vary the number of NBA states, the number of robots, and the number of robot types, respectively. As summarized in Table 4, PDTS outperforms MILP in terms of the solution time and shows improved scalability to the robot number.

Since PDTS can only obtain locally optimal plans, we compare the locally optimal plan with the optimal plan obtained by MILP in 50 different environments. The LTL task is set as $\Phi = Fap_1 \wedge Fap_2 \wedge Fap_3 \wedge Fap_4$ with 16 NBA states and four types of agents. As shown in Table 5, the average cost of PDTS is close to the optimal plans of MILP.

Table 3. Solution time for different robot numbers.

n_a	n_c	Time (/s)	n_a	n_c	Time (/s)
10	5	0.0130	5000	100	7.83
100	10	0.0387	6000	100	9.32
1000	100	1.32	7000	100	11.4
2000	100	2.78	8000	100	12.8
3000	100	4.37	9000	100	14.8
4000	100	5.90	10,000	100	16.4

Table 4. Comparison of solution time with MILP.

$\|S\|$	$\|n_a\|$	$\|n_c\|$	MILP (/s)	PDTS (/s)
16	24	4	9.903	0.01561
12	24	4	12.15	0.01215
10	24	4	16.25	0.007986
8	24	4	12.41	0.006037
16	24	3	9.813	0.01647
16	24	2	9.841	0.01445
16	24	1	9.946	0.01666
16	100	4	10.32	0.03333
16	200	4	11.35	0.05620
16	400	4	30.84	0.1016

Table 5. Comparison of the average cost of plans.

$\|n_a\|$	TK	Cost of MILP	Cost of PDTS
12	(1,1,1,1)	9.96	10.18
24	(2,2,2,2)	9.82	11.12
36	(3,3,3,3)	9.60	11.60
48	(4,4,4,4)	9.82	11.44

When using the task decomposition and MILP to address the TAP of HMRS as in Leahy et al. (2022), if 50 robots are considered, the solution time to obtain a feasible plan takes 11 s and the solution time to obtain a local optimal plan is more than 2 min. The approach developed in Messing et al. (2022) takes about 10 min to find a feasible plan for a group of 45 robots with 30 goals. By combining automaton and MILP, the work of Luo and Zavlanos (2022) takes about 4000 s for a group of 30 robots, while the work of Leahy et al. (2021) obtains the initial motion plan in 160 s and re-planning in 47 s when considering 19 automaton states and 10 robots. In contrast, the PDTS in this work can solve similar TAP problems in nearly 0.01 s, which is at least 10^3 times faster than existing methods. It can also solve the TAP problems with over 200 automaton states and over 10^4 robots.

Table 6. Solution time for different temporary tasks.

Temporary task Φ_L	$\lvert S_L\rvert$	Coupled	Uncoupled	IPDTS (/s)	PDTS (/s)	Total (/s)
Fap_4	2	1	0	0.00161	0.00613	0.00774
Fap_5	2	0	1	0.000560	0.00736	0.00792
$Fap_3 \wedge Fap_4$	4	2	0	0.00223	0.00574	0.00797
$Fap_4 \wedge Fap_5$	4	1	1	0.00408	0.00578	0.00986
$Fap_5 \wedge Fap_6$	4	0	2	0.00133	0.00736	0.00869
$Fap_2 \wedge Fap_3 \wedge Fap_4$	8	3	0	0.00263	0.00577	0.00840
$Fap_3 \wedge Fap_4 \wedge Fap_5$	8	2	1	0.00638	0.00585	0.0122
$Fap_4 \wedge Fap_5 \wedge Fap_6$	8	1	2	0.0120	0.00556	0.0176
$Fap_5 \wedge Fap_6 \wedge Fap_7$	8	0	3	0.00396	0.00705	0.0110
$Fap_1 \wedge Fap_2 \wedge Fap_3 \wedge Fap_4$	16	4	0	0.00318	0.00589	0.00907
$Fap_2 \wedge Fap_3 \wedge Fap_4 \wedge Fap_5$	16	3	1	0.00856	0.00594	0.0145
$Fap_3 \wedge Fap_4 \wedge Fap_5 \wedge Fap_6$	16	2	2	0.0194	0.00566	0.0251
$Fap_4 \wedge Fap_5 \wedge Fap_6 \wedge Fap_7$	16	1	3	0.0436	0.00637	0.0500
$Fap_5 \wedge Fap_6 \wedge Fap_7 \wedge Fap_8$	16	0	4	0.0143	0.00844	0.0227

7.3. Performance of IPDTS

To evaluate the performance of IPDTS, we consider three types of robots and each type has five robots. Suppose the global task is $\Phi_G = GFap_1 \wedge GFap_2 \wedge GFap_3 \wedge GFap_4$, and for each atomic task ap_i, $i = 1, \ldots, 4$, the task requirement is $tk = (2, 2, 2)$. To show how the performance of IPDTS varies with the local temporary tasks, a set of various Φ_L is considered in Table 6. The results in Table 6 indicate that (1) when there is coupling between Φ_G and Φ_L and the size of Φ_L is the same, the higher the coupling degree is, the less the computation time required using IPDTS; (2) when there is no coupling between Φ_G and Φ_L, the computation time is small; (3) when the number of coupled atomic propositions in Φ_G and Φ_L is the same or the number of uncoupled atomic propositions is the same, the larger the size of Φ_L is, the longer the computation time is.

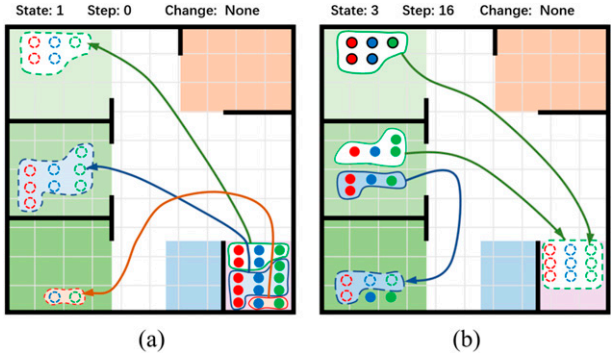


Figure 6. (a) There are 15 robots of three types (red: type 1, blue: type 2, green: type 3) starting from the warehouse (region 4). The robots with desired types and numbers go to region 1 and 2 to complete ap_1 and ap_2, while the remaining robots go to region 3 and wait for the other agents to complete ap_3. (b) After completing ap_1 and ap_2, some robots go to region 3 to complete ap_3 while the others return to the warehouse and wait to perform ap_4.

7.4. Reactive TAP

To show the reactive planning, we consider the following cases: local temporary tasks, agent failures, task changes, and environmental changes. Consider the farm environment in Figure 1. Suppose there are three types of robots and each type has five robots. The robots of type 1, 2, and 3 are represented by red, blue, and green dots, respectively. We consider the following atomic proposition: (1) ap_1: visit the planting area 1 with $tk_1 = (2, 2, 1)$; (2) ap_2: visit the planting area 2 with $tk_2 = (3, 2, 3)$; (3) ap_3: visit the planting area 3 with $tk_3 = (2, 2, 2)$; and (4) ap_4: visit the warehouse for maintenance by all robots, that is, $tk_4 = (5, 5, 5)$. The collaborative global task is specified as $\Phi_G = GFap_1 \wedge GFap_2 \wedge GFap_3 \wedge GFap_4$ with the task requirement $TK = \{tk_1, tk_2, tk_3, tk_4\}$. The simulation video is provided.[2]

7.4.1. Global task. Initially, the global task Φ_G with task requirement TK is assigned to the robots. Figure 6(a) shows that all robots are initially in the warehouse (i.e., region 4). As the system evolves, ap_1 and ap_2 are completed. Due to the lack of robots to perform ap_3, a sub-group of robots is rerouted to perform ap_3 after completing ap_2 in Figure 6(b). The remaining robots return to the warehouse and wait to perform ap_4. The average time of global planning is 0.0137 s.

7.4.2. Local temporary tasks. As shown in Figure 7(a), the robots in planting area 3 plan to return to the warehouse after completing ap_3. Suppose there is a local temporary task of visiting the livestock area and then return to the warehouse at the moment. Such a local task is specified as $\Phi_L = Fap_4 \wedge Fap_5 \wedge ((\neg ap_4)\mathcal{U}ap_5)$, where ap_5 indicates the task of visiting the livestock area with $tk_5 = (2, 2, 2)$. To deal with (Φ_L, TK), the re-plan is triggered. We run re-planning 20 times and the average planning time of Φ_L is

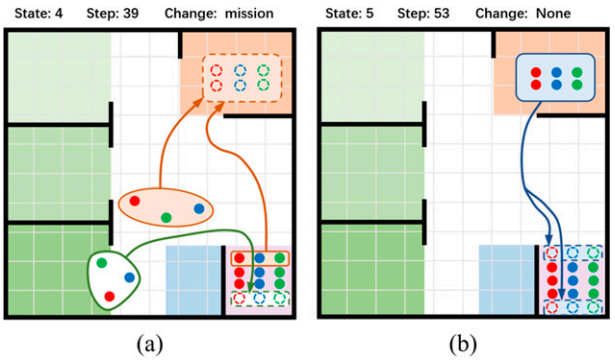


Figure 7. (a) After completing ap_3, on the way to the warehouse (region 4), the local task Φ_L is triggered. Thus some robots are assigned with the new task of ap_5 while the remaining robots return to the warehouse. (b) After completing ap_5, all agents return to the warehouse to complete ap_4.

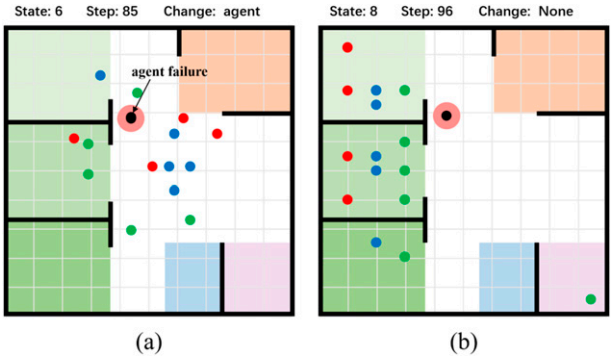


Figure 8. (a) A robot of type 1 (i.e., the black dot) fails during task operation. (b) The failed robot will no longer participate in the following tasks. Thus, a new plan is obtained and executed.

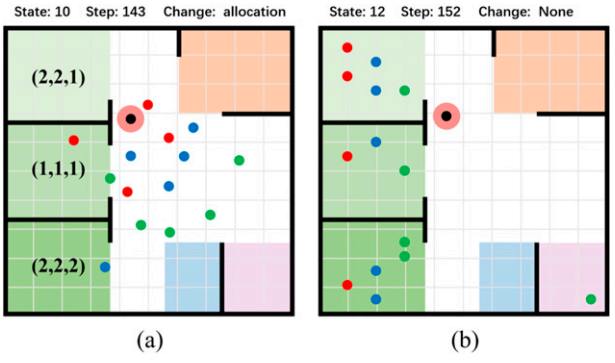


Figure 9. (a) The task requirement of q_2 changes from (3,2,3) to (1,1,1) during task operation. (b) Since four robots of type 3 are needed, one robot of type 3 returns to the warehouse while the others complete the updated task.

0.00919 s. Figure 7(b) shows that after completing ap_5, the robots will return to the warehouse.

7.4.3. Agent failure. During task operation, suppose a robot fails, that is, the robot r_1^1 fails at the time step 85 in Figure 8(a). The re-planning is triggered. We run re-planning 20 times and the average planning time is 0.00934 s. Figure 8(b) shows that ap_1 is completed. Since there are only two robots of type 1 in the planting area 2, one robot of type 1 in the planting area 1 is assigned to the planting area 2 to replace the failed robot.

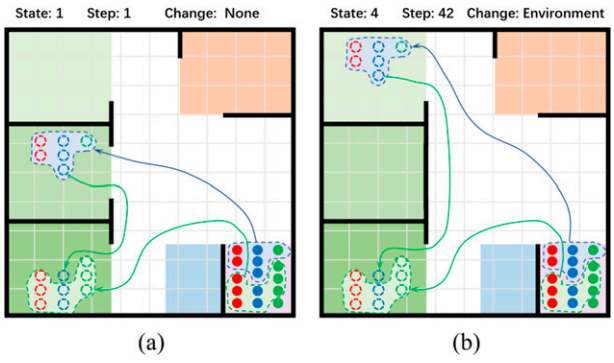


Figure 10. (a) The initial plan. (b) The modified plan when region 2 becomes inaccessible.

7.4.4. Task change. Figure 9(a) shows that the system continues on the task (Φ_G, TK). Suppose at time step 143 the task requirement is changed to $TK = \{(2,2,1),(1,1,1),(1,2,2),(5,5,5)\}$ and thus the re-planning is triggered. We run re-planning 20 times and the average planning time is 0.00838 s. As shown in Figure 9(b), since only four robots of type 3 are needed for the tasks ap_1, ap_2, and ap_3, one robot of type 3 returns to the warehouse and waits to perform ap_4.

7.4.5. Environmental change. Consider a global mission $\Phi_G = GF\ (ap_1 \vee ap_2) \wedge GF\ (ap_3) \wedge GF\ (ap_4)$ with task requirement TK = ((2, 3, 1), (2, 3, 1), (3, 3, 3), (5, 5, 5)). Figure 10(a) shows the generated plan $\boldsymbol{\pi}_{pre} = \boldsymbol{\pi}_{tra} = \boldsymbol{\pi}_{suf} = ap_2ap_3ap_4$, which indicates that a sub-group of robots travels to region 2 to perform ap_2 while another sub-group of robots travels to region 3 prepared for performing ap_3. After completing ap_2, one agent of type 2 will travel to region 3 and joins the robots there to collaboratively perform ap_3. The robots then return to region 4 to perform ap_4 completing the prefix plan. Suppose an environmental change occurs then, that is, region 2 becomes inaccessible, as shown in Figure 10(b). A new plan $\boldsymbol{\pi}_{tra} = \boldsymbol{\pi}_{suf} = ap_1ap_3ap_4$ is then generated. The sub-group of robots previously going to region 2 is rerouted to region 1 instead.

8. Experiments

Experiments are carried out in this section to demonstrate the developed TAP framework in a real world environment. A fleet of turtlebots is used in the experiment. Each experiment is performed at least five times and the average performance is reported. The system runs Matlab (2019b) on Ubuntu18.04 and the ROS version is Melodic. The

experiments are performed on a laptop with Intel Core i7-7500U 2.70 GHz and 16 Gb RAM.

8.1. Tower defense game

This section considers a real-time strategy game, namely, tower defense, to demonstrate the capability of heterogeneous robots in handling both global and local tasks in an environment with the reactive task. As shown in Figure 11, the operating environment consists of a base, the guard tower 1, the guard tower 2, and an alarm tower. Suppose there are two types of robots: the defense robots (i.e., type 1) and the reconnaissance robots (i.e., type 2). Recall that r_i^j denotes the ith robot of type j. We consider three types of atomic tasks: (1) ap_1: visit the guard tower 1 by one robot of type 1 and one robot of type 2; (2) ap_2: visit the guard tower 2 by one robot of type 1 and one robot of type 2; (3) ap_3: visit the base to report the findings by two robots of type 2. The experiment video is provided.[3]

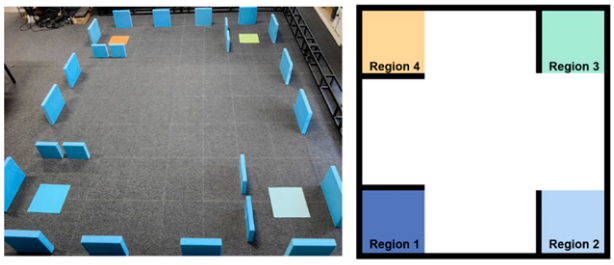


Figure 11. (a) The experiment environment. (b) The corresponding environment in simulation. The upper left and right corners in (a) (i.e., the orange and green blocks in (b)) represent the alarm tower (region 4) and the base (region 3), respectively. The lower left and right corners in (a) (i.e., the dark blue and light blue blocks in (b)) represent the guard tower 1 (region 1) and the guard tower 2 (region 2), respectively. The plastic blocks in (a) (i.e., the thick black lines in (b)) represent the wall that the robots cannot traverse.

8.1.1. Global task. Consider a global task $\Phi_G = GFap_1 \wedge GFap_2 \wedge GFap_3$, which requires the robots to consistently monitor the areas of interest by performing ap_i, $i = 1, 2, 3$. The corresponding task requirement is $TK = \{(1,1),(1,1),(0,2)\}$. As shown in Figure 12, Φ_G is successfully carried out. The construction of the automaton $\mathcal{B}_G$ takes 0.278 s and the generation of the task allocation and planning using PDTS takes 0.00697 s. The generated plan scheme is $\boldsymbol{\pi}_{pre} = \boldsymbol{\pi}_{tra} = \boldsymbol{\pi}_{suf} = [ap_1, ap_2, ap_3]$, $\boldsymbol{C}_{pre} = [(0, 1, 0, 1), (1, 0, 1, 0), (0, 0, 1, 1)]$, and $\boldsymbol{C}_{tra} = \boldsymbol{C}_{suf} = [(0, 1, 1, 0), (1, 0, 0, 1), (0, 0, 1, 1)]$. Figure 12(b) shows that r_2^1 and r_4^2 move to the guard tower 1 to complete ap_1. Figure 12(c) shows that r_1^1 and r_3^2 move to the guard tower 2 to complete ap_2. After completing their tasks, r_3^2 and r_4^2 then head to the base to complete ap_3 in Figure 12(d). Upon completing the current tasks, the next loop of performing ap_i, $i = 1, 2, 3$, starts according to $\boldsymbol{\pi}_{tra}$ and $\boldsymbol{C}_{tra}$.

8.1.2. Agent failures. To deal with unexpected agent failures, suppose r_1^1 fails during task operation. Let r_1^0 denote that r_1^1 no longer functions. The re-planning takes 0.00530 s using PDTS and the plan scheme is $\boldsymbol{\pi}_{pre} = \emptyset$, $\boldsymbol{\pi}_{tra} = \boldsymbol{\pi}_{suf} = [ap_1, ap_2, ap_3]$, $\boldsymbol{C}_{pre} = \emptyset$, and $\boldsymbol{C}_{tra} = \boldsymbol{C}_{suf} = [(0, 1, 1, 0), (0, 1, 0, 1), (0, 0, 1, 1)]$. Figure 13(a) shows that two robots of type 1 are on standby at the guard tower 1 and 2, respectively. Two robots of type 2 move to the guard tower for ap_3. The robot r_1^1 suddenly fails and cannot participate in the following tasks. The robot r_2^1 goes to the guard tower 2 to replace r_1^1 to perform ap_2 after completing ap_1 in Figure 13(b). Then, the robot r_2^1 returns to the guard tower 1 and waits to perform ap_1.

8.1.3. Task changes. Suppose one robot of type 1 and one robot of type 2 need to return to the base to report findings.

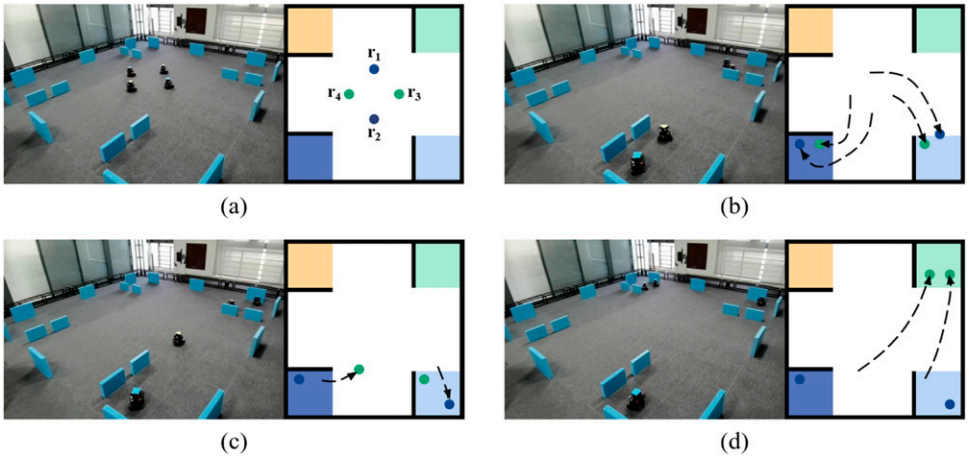


Figure 12. The snapshots of robots performing the global task Φ_G. (a) The initial positions of the robots (blue: type 1, green: type 2). (b) A robot of type 1 and a robot of type 2 perform ap_1 by visiting the guard tower 1. (c) A robot of type 1 and a robot of type 2 perform ap_2 by visiting the guard tower 2. (d) Two robots of type 2 return to the base to complete ap_3.

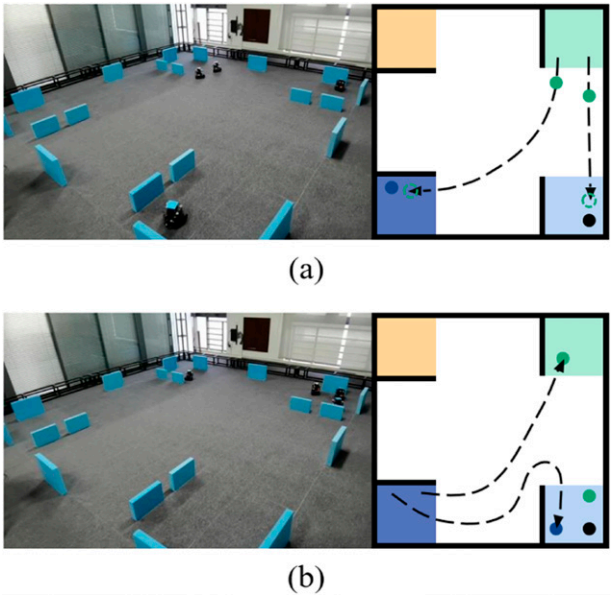


Figure 13. Agent failure. (a) Two robots of type 1 are on standby at the guard tower 1 and 2, respectively, while two robots of type 2 perform their tasks. Then, one robot of type 1 fails and the re-planning is initiated. (b) After completing ap_1, the remaining robot of type 1 goes to the guard tower 2 to complete ap_2.

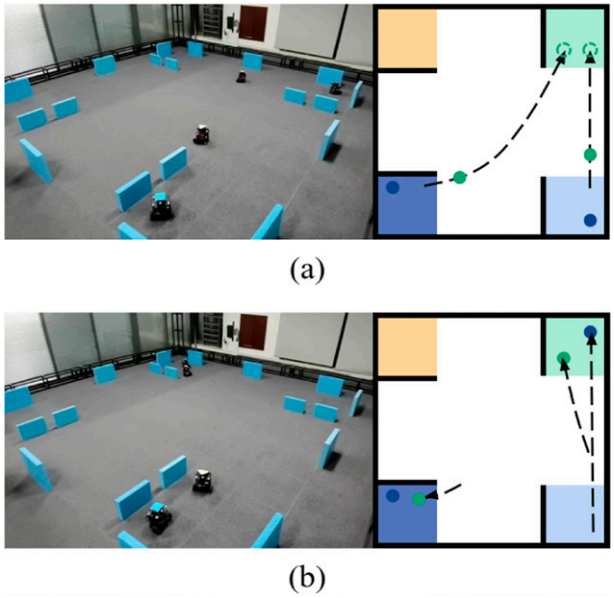


Figure 14. Task change. (a) The robot has performed ap_1 and ap_2. Then, the task requirement changes to $tk_3 = (1, 1)$. (b) One robot of type 1 and one robot of type 2 return to the base to complete ap_3 and the remaining agents wait for ap_1 to be executed at the guard tower 1.

Thus, the task requirement changes to $tk_3 = (0, 2) \rightarrow (1, 1)$, that is $TK = \{(1, 1), (1, 1), (1, 1)\}$. The re-planning takes 0.00496 s using PDTS and the plan scheme is $\boldsymbol{\pi}_{pre} = [ap_3]$, $\boldsymbol{\pi}_{tra} = \boldsymbol{\pi}_{suf} = [ap_1, ap_2, ap_3]$, $\boldsymbol{C}_{pre} = [(1, 0, 1, 0)]$, $\boldsymbol{C}_{tra} = [(0, 1, 0, 1), (1, 0, 1, 0), (0, 1, 0, 1)]$, and $\boldsymbol{C}_{suf} = [(1, 0, 1, 0), (0, 1, 0, 1), (1, 0, 1, 0)]$. Figure 14(a) shows that ap_1 and ap_2 have been completed and two robots of type 2 go to the base to complete ap_3. Then, the task requirement changes, that is, $tk_3 = (0, 2) \rightarrow (1, 1)$. The robot r_1^1 and r_3^2 go to the base to perform ap_3 and r_4^2 returns to the guard tower 1 to wait to perform ap_1 in Figure 14(b). When ap_1 is completed in the stage of tra, r_2^1 and r_4^2 at the guard tower 1 go to the base to wait for the execution of ap_3, and r_1^1 and r_3^2 in the base go to the guard tower 2 to perform ap_2.

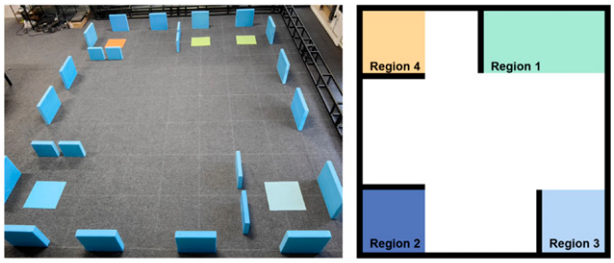


Figure 15. The corresponding simulated hospital environment. The orange and green blocks in (a) and (b) represent the hospital warehouse (region 4) and the robot warehouse (region 1), respectively. The dark blue and light blue blocks in (a) and (b) represent the sterile area (region 2) and the disinfection area (region 3), respectively. The plastic blocks in (a) (the black lines in (b)) represent the wall that the robots cannot traverse.

8.2. Experiment 2: Hospital cleaning

Consider a hospital environment consisting of four areas: the sterile area, the disinfected area, the robot warehouse, and the hospital warehouse, as shown in Figure 15. Suppose there are four robots operating in the hospital environment: three robots of type 1 (e.g., the Burger) and one robot of type 2 (e.g., the Waffle with a manipulator). The robots are required to clean the sterile area and the disinfected area and then return to the robot warehouse for recharging and self-cleaning. Consider the following atomic propositions: ap_1: visit the robot warehouse; ap_2: clean the sterile area; ap_3: clean the disinfected area. The above daily mission is expressed as an LTL formula: $\Phi_G = GFap_1 \wedge GFap_2 \wedge GFap_3$ with task requirement $TK = \{(3, 1), (2, 0), (2, 0)\}$. If the robot finds infected objects, they need to return to the robot warehouse for self-cleaning and the robot of type 2 will be activated to handle the infected object (e.g., pick it up and trash it). Hence, we consider the following atomic propositions for the robot of type 2: ap_4: pick the object up; and ap_5: visit the hospital warehouse and trash the object.

Such a local temporary task is expressed as $\Phi_L = Fap_4 \wedge Fap_5 \wedge Fap_1 \wedge ((\neg ap_4)\mathcal{U}ap_1) \wedge (\neg ap_5)\mathcal{U}ap_4$ with the task requirement $TK = \{(3, 1), (2, 0), (2, 0), (0, 1), (0, 1)\}$. The experiment video is provided.[4]

8.2.1. Global task. Turtlebot3 burger and waffle are used for the robot of type 1 and 2 in the experiment, respectively. The construction of automaton takes 0.262 s and the PDTS

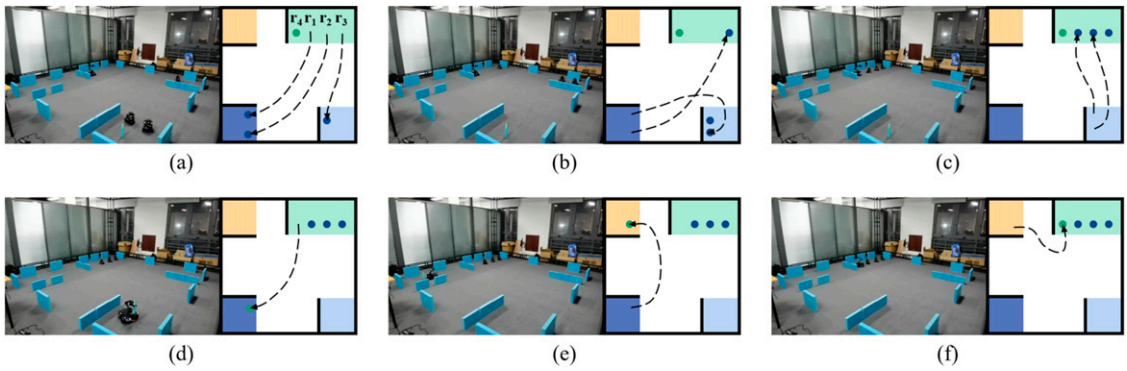


Figure 16. The experiment results of performing the local temporary task. (a) An infected object is detected by two robots (i.e., blue dots) when performing ap_2 and thus the complex temporary task Φ_L is triggered. (b) The robots perform ap_3. (c) The robots perform ap_1. (d) One robot of type 2 performs ap_4 by fetching the infected object. (e) One robot of type 2 performs ap_5 by placing the infected object in the hospital warehouse. (f) Φ_L is completed and all robots return to the robot warehouse to perform ap_1.

planning takes 0.00725 s. The obtained task allocation and plan is $\boldsymbol{\pi}_{pre} = \boldsymbol{\pi}_{tra} = \boldsymbol{\pi}_{suf} = [ap_1, ap_2, ap_3]$ and $\boldsymbol{C}_{pre} = \boldsymbol{C}_{tra} = \boldsymbol{C}_{suf} = [(1, 1, 1, 1), (1, 1, 0, 0), (1, 0, 1, 0)]$.

8.2.2. Complex temporary task. As shown in Figure 16, once an infected object is detected, the temporary task Φ_L is triggered. The construction of automaton takes 0.283 s. The IPDTS and PDTS planning take 0.00982 s and 0.00357 s, respectively. The plan scheme is $\boldsymbol{\pi}_{tem} = [ap_3, ap_1, ap_4, ap_5, ap_1]$, $\boldsymbol{\pi}_{pre} = \emptyset$, $\boldsymbol{\pi}_{tra} = [ap_2, ap_3]$, $\boldsymbol{\pi}_{suf} = [ap_1, ap_2, ap_3]$ and $\boldsymbol{C}_{tem} = [(0, 1, 1, 0), (1, 1, 1, 1), (0, 0, 0, 1), (0, 0, 0, 1), (1, 1, 1, 1)]$, $\boldsymbol{C}_{pre} = \emptyset$, $\boldsymbol{C}_{tra} = [(1, 1, 0, 0), (1, 0, 1, 0)]$, and the suffix stage allocation $\boldsymbol{C}_{suf} = [(1, 1, 1, 1), (1, 1, 0, 0), (1, 0, 1, 0)]$. To not violate the constraints of Φ_G, r_2^1 and r_3^1 continue to perform ap_3 in Figure 16(b). All agents then return to the robot warehouse to perform ap_1 in Figure 16(c). Next, r_4^2 will go to the sterile area in Figure 16(d) and the hospital warehouse in Figure 16(e) to perform ap_4 and ap_5 and finally return to the robot warehouse to complete the complex temporary task in Figure 16(f). The system continues to execute tasks according to $\boldsymbol{\pi}_{tra}$ and $\boldsymbol{C}_{tra}$. The experimental results show that the solution time of a single round of low-level planning for four robots takes about 0.7 s, while the solution time of high-level reactive plan only takes 0.0134 s. Thus, it can be used in real-time for reactive planning.

9. Conclusions

This work develops a novel TAP framework that can solve reactive temporal logic planning problems for large-scale heterogeneous multi-robot systems in real-time. It can generate satisfying plan for HMRS with multiple orders of magnitude more robots than those that existing methods can manipulate. Rigorous analysis shows that the PDT based TAP is not only feasible and complete, but also has lower complexity (i.e., the solution time is only linearly proportional to the robot numbers and types) and can be applied in real-time. In this work, the detection of robot failures and the assignment of planning are conducted via wireless communication. Given that wireless communication can be subject to various constraints, such as limited range, interference, and bandwidth limitations, future research will explore task allocation and planning under communication constraints. Additional research will also consider the time interval, varying-time requirement, and other complex task for HMRS.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported in part by the National Natural Science Foundation of China under Grant 62173314 and U2013601.

ORCID iD

Zhen Kan https://orcid.org/0000-0003-2069-9544

Notes

1. For a NBA $\mathcal{B} = (S, S_0, \Delta, \Sigma, \mathcal{F})$, if there exist finite sequences $\boldsymbol{\pi} = \pi_2 \ldots \pi_n$ and $\boldsymbol{s} = s_1 s_2 \ldots s_{n-1} s_n$ that satisfy $\pi_i \in \Delta(s_{i-1}, s_i)$, $\forall \pi_i \in \boldsymbol{\pi}$ and $s_i \in S$, it is said s_n is reachable from s_1. Denote by $R(s_i)$ the set of states in S reachable from s_i. Apparently, one has $R(s_i) = \bigcup_{s_j \in R(s_i)} R(s_j)$ and thus $R(s_j) \subset R(s_i)$ if $s_j \in R(s_i)$.
2. https://youtu.be/37hp_ouverQ
3. https://youtu.be/_g2iaIRturM
4. https://youtu.be/v3nVmqeI6MM

References

Baier C and Katoen JP (2008) *Principles of Model Checking*. Cambridge, MA: MIT Press.

Banks C, Wilson S, Coogan S, et al. (2020) Multi-agent task allocation using cross-entropy temporal logic optimization.

2020 IEEE International Conference on Robotics and Automation (ICRA). Paris: IEEE, 7712–7718.

Belta C, Bicchi A, Egerstedt M, et al. (2007) Symbolic planning and control of robot motion [Grand Challenges of Robotics]. *IEEE Robotics and Automation Magazine* 14(1): 61–70.

Cai M, Peng H, Li Z, et al. (2020) Receding horizon control based motion planning with partially infeasible LTL constrains. *IEEE Control Syst. Lett* 5(4): 1279–1284.

Cai M, Hasanbeig M, Xiao S, et al. (2021a) Modular deep reinforcement learning for continuous motion planning with temporal logic. *IEEE Robotics and Automation Letters* 6(4): 7973–7980.

Cai M, Peng H, Li Z, et al. (2021b) Learning-based probabilistic ltl motion planning with environment and motion uncertainties. *IEEE Transactions on Automatic Control* 66(5): 2386–2392.

Cai M, Xiao S, Li Z, et al. (2023) Optimal probabilistic motion planning with potential infeasible LTL constraints. *IEEE Transactions on Automatic Control* 68(1): 301–316.

Chen Z, Zhou Z, Wang S, et al. (2024) Fast temporal logic mission planning of multiple robots: a planning decision tree approach. *IEEE Robotics and Automation Letters* 9(7): 6146–6153. DOI: 10.1109/LRA.2024.3401166.

Cho K, Suh J, Tomlin CJ, et al. (2017) Cost-aware path planning under co-safe temporal logic specifications. *IEEE Robotics and Automation Letters* 2(4): 2308–2315.

Clarke EM, Grumberg O and Peled D (1999) *Model Checking*. Cambridge, MA: MIT Press.

Faruq F, Parker D, Laccrda B, et al. (2018) Simultaneous task allocation and planning under uncertainty. *IEEE International Conference on Intelligent Robots and Systems (IROS)*. Paris: IEEE, 3559–3564.

Finkbeiner B, Klein F and Metzger N (2021) Live synthesis. *International Symposium on Automated Technology for Verification and Analysis*. Berlin: Springer, 153–169.

Garrett CR, Chitnis R, Holladay R, et al. (2021) Integrated task and motion planning. *Annu. Rev. Control Robot. Auton. Syst* 4(4): 265–293.

Gastin P and Oddoux D (2001) Fast LTL to Büchi automata translation. *Computer Aided Verification. CAV 2001. Lecture Notes in Computer Science*. Berlin: Springer, 53–65.

Guo M and Dimarogonas DV (2015) Multi-agent plan reconfiguration under local LTL specifications. *The International Journal of Robotics Research* 34(2): 218–235.

Guo M and Dimarogonas DV (2017) Task and motion coordination for heterogeneous multiagent systems with loosely coupled local tasks. *IEEE Transactions on Automation Science and Engineering* 14(2): 797–808.

Guo M, Bechlioulis CP, Kyriakopoulos KJ, et al. (2017) Hybrid control of multiagent systems with contingent temporal tasks and prescribed formation constraints. *IEEE Trans. Control Network Syst* 4(4): 781–792.

Hauser K (2021) Semi-infinite programming for trajectory optimization with non-convex obstacles. *The International Journal of Robotics Research* 40(10-11): 1106–1122.

He K, Lahijanian M, Kavraki LE, et al. (2017) Reactive synthesis for finite tasks under resource constraints. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC: IEEE, 5326–5332.

Jagtap P, Soudjani S and Zamani M (2020) Formal synthesis of stochastic systems via control barrier certificates. *IEEE Transactions on Automatic Control* 66(7): 3097–3110.

Kalluraya S, Pappas GJ and Kantaros Y (2023) Resilient temporal logic planning in the presence of robot failures. *2023 62nd IEEE Conference on Decision and Control (CDC)*. Singapore: IEEE, 7520–7526.

Kantaros Y and Zavlanos MM (2016) Distributed communication-aware coverage control by mobile sensor networks. *Automatica* 63: 209–220.

Kantaros Y and Zavlanos MM (2020) Stylus*: a temporal logic optimal control synthesis algorithm for large-scale multi-robot systems. *The International Journal of Robotics Research* 39(7): 812–836.

Kantaros Y, Malencia M, Kumar V, et al. (2020) Reactive temporal logic planning for multiple robots in unknown environments. *2020 IEEE International Conference on Robotics and Automation (ICRA)*. Paris: IEEE, 11479–11485.

Kiesbye J, Grover K, Ashok P, et al. (2022) Planning via model checking with decision-tree controllers. *2022 International Conference on Robotics and Automation (ICRA)*, Philadelphia, PA: IEEE, 4347–4354.

Kloetzer M and Belta C (2010) Automatic deployment of distributed teams of robots from temporal logic motion specifications. *IEEE Transactions on Robotics* 26(1): 48–61.

Kupferman O and Vardi MY (2001) Model checking of safety properties. *Formal Methods in System Design* 19(3): 291–314.

Lacerda B and Lima PU (2019) Petri net based multi-robot task coordination from temporal logic specifications. *Robotics and Autonomous Systems* 122: 103289.

Lacerda B, Parker D and Hawes N (2014) Optimal and dynamic planning for markov decision processes with co-safe ltl specifications. *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Chicago, IL. IEEE, 1511–1516.

Lacerda B, Faruq F, Parker D, et al. (2019) Probabilistic planning with formal performance guarantees for mobile service robots. *The International Journal of Robotics Research* 38(9): 1098–1123.

Leahy K, Serlin Z, Vasile CI, et al. (2021) Scalable and robust algorithms for task-based coordination from high-level specifications (scratches). *IEEE Transactions on Robotics* 38(4): 2516–2535.

Leahy K, Jones A and Vasile CI (2022) Fast decomposition of temporal logic specifications for heterogeneous teams. *IEEE Robotics and Automation Letters* 7(2): 2297–2304.

Liu W, Leahy K, Serlin Z, et al. (2023) Robust multi-agent coordination from catl+ specifications. *2023 American Control Conference (ACC)*, San Diego, CA: IEEE, 3529–3534.

Luo X and Zavlanos MM (2022) Temporal logic task allocation in heterogeneous multirobot systems. *IEEE Transactions on Robotics* 38(6): 3602–3621.

Luo X, Kantaros Y and Zavlanos MM (2021) An abstraction-free method for multirobot temporal logic optimal control synthesis. *IEEE Transactions on Robotics* 37(5): 1487–1507.

Messing A, Neville G, Chernova S, et al. (2022) Grstaps: graphically recursive simultaneous task allocation, planning, and scheduling. *The International Journal of Robotics Research* 41(2): 232–256.

Otte M and Frazzoli E (2016) RRTX: asymptotically optimal single-query sampling-based motion planning with quick replanning. *The International Journal of Robotics Research* 35(7): 797–822.

Partovi A, da Silva RR and Lin H (2018) Reactive integrated mission and motion planning for mobile robotic manipulators. *2018 Annual American Control Conference (ACC)*, Milwaukee, WI: IEEE, 3538–3543.

Ramasubramanian B, Niu L, Clark A, et al. (2020) Secure control in partially observable environments to satisfy ltl specifications. *IEEE Transactions on Automatic Control* 66(12): 5665–5679.

Sahin YE, Nilsson P and Ozay N (2019) Multirobot coordination with counting temporal logics. *IEEE Transactions on Robotics* 36(4): 1189–1206.

Sawant M, Tayebi A and Polushin I (2023) Hybrid feedback for autonomous navigation in environments with arbitrary non-convex obstacles. *arXiv preprint arXiv:2304.10598*.

Schillinger P, Bürger M and Dimarogonas DV (2018a) Hierarchical LTL-task mdps for multi-agent coordination through auctioning and learning. *The International Journal of Robotics Research* 37(7): 818–838.

Schillinger P, Bürger M and Dimarogonas DV (2018b) Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. *The International Journal of Robotics Research* 37(7): 818–838.

Schuppe GF and Tumova J (2020) Multi-agent strategy synthesis for LTL specifications through assumption composition. *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, Hong Kong: IEEE, 533–540.

Shah A, Kamath P, Li S, et al. (2023) Supervised bayesian specification inference from demonstrations. *The International Journal of Robotics Research* 42(14): 1245–1264.

Smith SL, Tumova J, Belta C, et al. (2011) Optimal path planning for surveillance with temporal-logic constraints. *The International Journal of Robotics Research* 30(14): 1695–1708.

Tabajara LM and Vardi MY (2020) Ltlf synthesis under partial observability: from theory to practice. *arXiv preprint arXiv: 2009.10875*.

Ulusoy A and Belta C (2014) Receding horizon temporal logic control in dynamic environments. *The International Journal of Robotics Research* 33(12): 1593–1607.

Ulusoy A, Smith SL, Ding XC, et al. (2013) Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research* 32(8): 889–911.

Ulusoy A, Wongpiromsarn T and Belta C (2014) Incremental controller synthesis in probabilistic environments with temporal logic constraints. *The International Journal of Robotics Research* 33(8): 1130–1144.

Vasile CI, Tumova J, Karaman S, et al. (2017) Minimum-violation scLTL motion planning for mobility-on-demand. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Singapore: IEEE, 1481–1488.

Vasile CI, Li X and Belta C (2020) Reactive sampling-based path planning with temporal logic specifications. *The International Journal of Robotics Research* 39(8): 1002–1028.

Vasilopoulos V, Kantaros Y, Pappas GJ, et al. (2021) Reactive planning for mobile manipulation tasks in unexplored semantic environments. *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi'an: IEEE, 6385–6392.

Yu P and Dimarogonas DV (2022) Distributed motion coordination for multirobot systems under LTL specifications. *IEEE Transactions on Robotics* 38(2): 1047–1062.

Zhao J, Wang S and Yin X (2023) Failure-aware self-diagnostic task planning under temporal logic specifications. *IFAC-PapersOnLine* 56(2): 4582–4588.

Zhou Z, Lee DJ, Yoshinaga Y, et al. (2022) Reactive task allocation and planning for quadrupedal and wheeled robot teaming. *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, Mexico City: IEEE, 2110–2117.