Distributed Task Allocation and Planning Under Temporal Logic and Communication Constraints

Ziyang Chen, Lin Li, and Zhen Kan, *Senior Member, IEEE*

***Abstract*—It is challenging to coordinately allocate and plan the tasks of a heterogeneous multi-agent system in a shared workspace. It can be even more challenging if the agents are subject to limited communication capability (i.e., exchange information with nearby agents only) and complex tasks with temporal and logic constraints. Motivated by these practical challenges, a distributed task allocation and planning method is developed, in which each agent communicates with neighboring agents about the task information (i.e., the preference of sub-tasks to be executed and the estimated task completion time) and predicts the task information of agents out of the communication range. Based on the collected task information, each agent can independently make conflict-free planning to improve the execution efficiency of the task. Rigorous analysis shows that the generated plan is guaranteed to be conflict free and numerical experiments demonstrate the effectiveness of the planned tasks.**

***Index Terms*—Linear temporal logic, distributed task allocation, task planning, heterogeneous multi-agent systems.**

I. Introduction

MULTI-AGENT systems show great potentials in various applications [1], [2]. However, allocating and planning the task of agents coordinately and efficiently in a shared workspace remains a challenge. To deal with task allocation, centralized approaches are often adopted, where a central unit collects the entire system information and determines the task assignment of agents. Despite effectiveness, for some applications, the centralized approach is not practical due to excessive information collection and potential demise/corruption of the central controller. The distributed task allocation and planning is an alternative approach in which each agent makes an independent task planning based on local information without using central task publishing or allocation. Therefore, it can effectively deal with addition/removal of agents or temporary local temporal tasks, which has been explored in the rescue missions [3] and logistics missions [4]. However, difficulties arise from limited communication and desired tasks subject to complex temporal and logic constraints, which introduce additional complexity to the task allocation problem. Hence, this work is particularly motivated to develop a distributed task allocation and planning approach subject to limited communication and temporal logic constraints.

The task allocation methods can be broadly classified as centralized and distributed approaches [5]. Representative centralized approaches include market-based methods [6] and optimization-based methods [7]. When integrated with linear temporal logic (LTL) specifications, a common approach is to formulate the task allocation as an integer linear programming (ILP) or a mixed integer linear programming (MILP) problem [8] for homogeneous robots [9], [10] or heterogeneous robots [11], [12]. In [13], [14], [15], an automaton-based decomposition-based approach was developed, which decomposes the global task into a set of independently executable sub-tasks. Such a method was extended for co-safe LTL formulas to achieve concurrent execution of sub-tasks in [16]. While centralized approaches may yield superior or even optimal task allocation and planning solutions, they often suffer from low scalability and high computational costs, making it impractical for handling complex and dynamically changing tasks. In contrast, distributed methods are more suitable for real-time applications due to the lower computational cost and higher scalability. Representative approaches include game-based methods for homogeneous or heterogeneous agents [17], [18], behaviour-based methods that leverage intention recognition and prediction for fast task allocation in a weak communication environment [5], and the methods that can handle robot failures during task execution [19], resource constrained task allocation problems [20] and communication constrained task allocation problems [21], [22], [23].

When considering temporal logic constraints, the top-down approach is to decompose the global task into a set of locally executable sub-tasks. Although decomposition-based methods enable distributed task execution, the task decomposition has to be performed in a centralized manner. On the contrary, the bottom-up approach allows each agent to have an independent local task, resulting in a series of LTL-based sub-tasks rather than a single global task to achieve group behaviors. As a result, the centralized top-down approach is more effective in terms of task decomposition, while the distributed bottom-up approach allows more flexibility in local task design. For instance, the plan reconfiguration under local LTL specifications was investigated for multi-robot systems in [3]. In [24], conflict detection was applied to ensure the security of agents and coordinate the agents' sub-tasks according to the current environment. However, existing distributed methods for multi-tasks with temporal logic

Manuscript received 18 December 2023; accepted 28 May 2024. Date of publication 3 June 2024; date of current version 7 June 2024. This letter was recommended for publication by Associate Editor T. Liu and Editor C.-B. Yan upon evaluation of the reviewers' comments. This work was supported by the National Natural Science Foundation of China under Grant 62173314 and Grant U2013601. *(Corresponding author: Zhen Kan.)*

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

Digital Object Identifier 10.1109/LRA.2024.3408486

constraints only consider the robot-robot and robot-obstacle collision avoidance. The possible task conflict, i.e. different agents target to the same sub-task at the same time, is ignored, which may lead to long waiting time.

In this letter, a distributed task allocation and planning is developed for a multi-agent system subject to communication constraints and LTL specifications. Specifically, each agent is equipped with a transceiver of limited communication range, so that the agents can only broadcast information to their immediate neighbors within certain distance. LTL is employed to specify the tasks of agents. To achieve distributed task planning, each agent communicates with neighboring agents about the task information (i.e., the preference of sub-tasks to be executed and the estimated task completion time) and predicts the task information of agents out of the communication range. Based on the collected task information, each agent can independently make conflict-free planning to improve the execution efficiency of the task.

The contributions can be summarized as follows. Most existing works for complex tasks with temporal constraints only consider path-level conflict avoidance. In this work, by introducing the task allocation, the proposed distributed planning method can consider both task-level and path-level conflict avoidance, which significantly improves the execution efficiency. Then, by leveraging local information (received from neighboring agent or predicted from out-of-communication agents), the developed task allocation and plan is guaranteed to be conflict-free for the task with temporal constraints and limited communication. Rigorous analysis and extensive experimental results demonstrate its effectiveness.

Notations: Let $\mathbb{N}$ and $\mathbb{R}^+$ denote the set of natural numbers and the set of positive real numbers, respectively. Let $[N]$ denote the shorthand notation for $\{1, \ldots, N\}$. Given a sequence $p = p_0 p_1 \ldots$, denote by $p[j \ldots] = p_j p_{j+1} \ldots$, $p[\ldots j] = p_0 \ldots p_j$, and $p[i : j] = p_i \ldots p_j$.

II. Preliminaries and Problem Formulation

A. Linear Temporal Logic

LTL is a formal language defined over a set of atomic propositions AP with Boolean and temporal operators. More details about LTL syntax, semantics, and model checking are referred to [25]. An LTL formula can be converted to a Non-deterministic Büchi Automata (NBA).

Definition 1: An NBA is a tuple $B = (S, S_0, \Sigma, \delta, S_F)$, where S is a finite set of states, $S_0 \subseteq S$ is the initial states, $\Sigma = 2^{AP}$ is the finite alphabet, $\delta : S \times \Sigma \to 2^S$ is the state transition, and $S_F \subseteq S$ is the set of accepting states.

The word $\boldsymbol{\pi} = \pi_0 \pi_1 \ldots$ is an infinite sequence with $\pi_i \in 2^{AP}$, $\forall i \in \mathbb{Z}_{\geq 0}$, where 2^{AP} represents the power set of AP. Let B_ϕ denote the NBA of the LTL formula ϕ. Let $\Delta : S \times S \to 2^\Sigma$ denote the set of atomic propositions that enables state transitions in NBA, i.e., $\forall \pi \in \Delta(s, s')$, $s' \in \delta(s, \pi)$. A valid run $\boldsymbol{s} = s_0 s_1 s_2 \ldots$ of B_ϕ generated by the word $\boldsymbol{\pi}$ with $s_i \in \delta(s_{i-1}, \pi_i)$, $\forall i \in \mathbb{N}^+$, is called accepting, if $\boldsymbol{s}$ intersects with S_F infinite often. An LTL formula can be translated to an NBA by the tool [26]. In this work, the agents' tasks are specified by LTL specifications and NBA is used to track the progress of LTL tasks.

B. Multi-Agent and Communication Models

Consider a fleet of n_a heterogeneous agents $R = \{r_1, r_2, \ldots, r_{n_a}\}$, which are classified into n_c types according to the different mobility capabilities (e.g, ground or aerial mobility) or sensing capabilities (e.g., Lidar or visual sensors). Each agent will be assigned an LTL task based on the its type. Let p_k and v_k denote the position and linear velocity of agent r_k, $\forall k \in [n_a]$, respectively. The agents are assumed to have a limited communication range $D \in \mathbb{R}^+$, i.e., two agents are able to communicate and exchange information only when their inter-distance is less than D. Let $t_n = n\Delta_t$, $n \in \mathbb{N}$, be a time sequence, where $\Delta_t \in \mathbb{R}^+$ is a sampling period. Given an agent r_k, its neighboring agents at time t_n are defined as $R_k(t_n) = \{r_j | \|p_k(t_n) - p_j(t_n)\| \leq D\}$. Similarly, the agents outside the communication range of r_k at time t_n is defined as $\bar{R}_k(t_n) = \{r_j | \|p_k(t_n) - p_j(t_n)\| > D\}$. Suppose the agents in R are assigned different tasks. For $r_k \in R$, let ϕ_k and B_k denote the task specification and the corresponding NBA, respectively.

C. Environment and Task Models

The heterogeneous multi-agent system R operates in a bounded workspace M consisting of $l \in \mathbb{N}$ non-overlapped regions of interest. Denote by M_i the ith region of interest and denote by M_{NI} the region of non-interest (e.g., obstacles or the regions that the agents can not traverse or operate within), where $M_i \cap M_j = \emptyset$ and $M_i \cap M_{NI} = \emptyset$ with $\forall i \neq j$ and $i, j \in [l]$. For each position $p \in M$, the labeling function $L : M \to AP$, indicates the associated atomic proposition $ap \in AP$ (i.e., the task to be executed) at a position $p \in M$, i.e., $L(p) = ap$.

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

Definition 2: The abstract task system is defined as a tuple $T = (Q, M, AP, LA, LM, LT, W)$, where Q is a finite set of abstract sub-tasks, each of which can be performed by an agent independently; M is the workspace; AP is the set of the atomic proposition; $LA : Q \to AP$ indicates the atomic proposition associated with the state in Q; $LM : Q \to M$ maps $q \in Q$ to a position $p \in M$; $LT : Q \to \mathbb{R}^+$ indicates the cost (e.g., the time elapsed) of performing sub-tasks; and $W : Q \times Q \to \mathbb{R}^+$ represents the transition cost, e.g., the distance $W(q_i, q_j) = \|LM(q_i) - LM(q_j)\|$.

Based on the NBA B and the abstract task system T, multiple task sequences starting from different initial sub-tasks with different cost can be obtained, such as using the method in [27]. To select the sequence with minimum cost, the sub-tasks with lower cost are preferred. To indicate the task preference, we define a partially ordered set O over the set of sub-tasks Q, i.e., for $q_i, q_j \in Q$, $q_i \succ q_j$ in O implies that q_i is preferred over q_j due to lower execution cost. If a sub-task assigned to an agent is being performed by another agent, the agent has to wait until the completion of the sub-task. To avoid unnecessary waiting time, it is also desired to estimate the completion time of each sub-task. By taking into account the task preferences and the estimated completion time of sub-task, we define a tuple $I = (O, t^{next})$, namely task information, to facilitate task allocation, where O indicates the task preference and $t^{next} : Q \to \mathbb{R}^+$ indicates the predicted timestamp of completing the sub-task $q \in Q$. Each agent calculates its own task information and exchanges the task information with neighboring agents via communication. The task information of r_j collected by r_k is denoted as $I_{k,j} = (O_{k,j}, t^{next}_{k,j})$ and $I_{k,k} = (O_{k,k}, t^{next}_{k,k})$ indicates the task information of r_k itself.

D. Task Planning

Given the workspace M and NBA B_ϕ corresponding to the LTL task ϕ, the plan is defined as the sequence of sub-tasks and the corresponding NBA states, denoted as $\Pi = (\boldsymbol{q}, \boldsymbol{s})$, where $\boldsymbol{q} = q_0 q_1 q_2 \ldots$ is the state trajectory of T with $q_i \in Q$ and q_1 indicating the initial sub-task (as q_0 is an empty task), $\boldsymbol{s} = s_0 s_1 s_2 \ldots$ is the state trajectory of B_ϕ corresponding to $\boldsymbol{q}$ where s_i indicates the automaton state after the atomic task is completed at q_i. By denoting $\Pi_i = (q_i, s_i)$, $i \in \mathbb{N}$, we can rewrite $\Pi = \Pi_0 \Pi_1 \Pi_2, \ldots$ as a series of plan tuples. Let $\boldsymbol{\pi} = LA(\boldsymbol{q})$ be the word generated by the trajectory $\boldsymbol{q}$. Given a plan $\Pi = (\boldsymbol{q}, \boldsymbol{s})$, the plan Π is said to satisfy the formula ϕ denoted as $\Pi \models \phi$, if $\boldsymbol{\pi} \models \phi$ and $\forall q_i \in \boldsymbol{q}$, $q_i \neq q_0$, $LA(q_i) \in \Delta(s_{i-1}, s_i)$.

Based on the prefix-suffix structure, the plan Π can be further written in the form of $\Pi = \Pi_{pre} \Pi_{suf} \Pi_{suf} \ldots$, where Π_{pre} and Π_{suf} are a finite prefix and a finite cyclic suffix, respectively. Since $\Pi_{pre} \Pi_{suf} \models \phi$ also indicates that $\Pi_{pre} \Pi_{suf} \Pi_{suf} \ldots \models \phi$, we only need to determine Π_{pre} and Π_{suf} in Π, denoted as $\Pi_{finite} = \Pi_{pre} \Pi_{suf}$. For the finite plan $\Pi^k_{finite} = \Pi^k_{pre} \Pi^k_{suf}$ of agent r_k, its cost value is defined as

$$\mathsf{Cost}(\Pi^k_{finite}) = \sum_{i=1}^{|\Pi^k_{finite}|} \left(\frac{1}{v_i} W(q_{i-1}, q_i) + LT(q_i) + t^i_w \right), \tag{1}$$

where t^i_w is the waiting time due to task occupation. The cost $\max_{r_k \in R} \{\mathsf{Cost}(\Pi^k_{finite})\}$ is defined as the group cost of R.

Suppose each agent is assigned an LTL task ϕ_i, $i \in [n_a]$, and the task specifications are the same if the agents are of the same type. Since each task is composed of a series of sub-tasks in AP, the agents need to perform these sub-tasks to complete their own LTL task independently. Given that each sub-task in the workspace can be performed by only one agent at each time stamp, the goal of this work is to develop a distributed task allocation and planning method, i.e., which sub-task $ap \in AP$ should be performed next to satisfy the LTL task while minimizing the execution time. Such a problem can be formulated as follows.

Problem 1: Giving a workspace M and a group of heterogeneous agents R with corresponding LTL tasks $\{\phi_1, \ldots, \phi_{n_c}\}$ for each type of agents, the goal is to develop a distributed task allocation and planning for each agent $r_k \in R$ to generate a plan Π^k_{finite} via local communication such that the group cost $\max_{r_k \in R} \{\mathsf{Cost}(\Pi^k_{finite})\}$ is minimized.

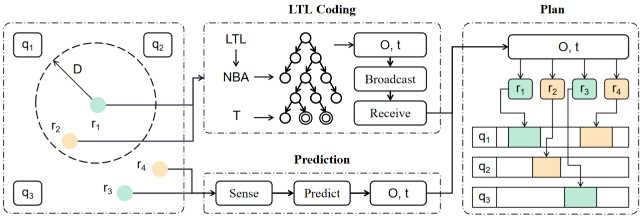


Fig. 1. The left plot shows the workspace consisting of 4 robots and 3 areas of interest. The dashed circle indicates the limited communication range of agents. Each agent $r_k \in R$ exchanges its task information $I_{k,k}(t) = \{O_{k,k}(t), t_{k,k}(t)\}$ with neighboring agents at time t. For the agent r_j out of the communication range, r_k predicts the task information $I_{k,j}(t) = \{O_{k,j}(t), t^{next}_{k,j}(t)\}$ of r_j. Based on the collected task information, the task allocation is obtained as shown in the right plot.

III. Distributed Task Allocation and Planning

This section presents a distributed task allocation and planning mechanism for multi-agent systems. As shown in Fig. 1, each agent encodes its sub-tasks with preferences and broadcasts the task information (i.e., the preference with completion time of each sub-task) to neighboring agents. Leveraging the received task information of neighboring agents or predicted task information of non-neighboring agents, a conflict-free task allocation can then be established at each time stamps. The established task allocation will construct the next plan tuple for each agent and significantly improves the execution efficiency of the whole plan.

A. Task Preference

The task preference of agents, i.e., which sub-task is preferred by the agent, is critical to task allocation. However, existing real-time task allocation methods, e.g., hedonic game [17], only consider task preference for single tasks. If each agent has an infinitely long LTL-based task to complete, besides sub-tasks sequences, the whole temporal task specification should be considered to determine the task preference.

To do so, the sets of feasible tasks for the given LTL formula are first obtained offline. Concretely, for an LTL task ϕ and its corresponding NBA B_ϕ, if there exist finite sequences $\boldsymbol{\pi} = \pi_2 \ldots \pi_n$ and $\boldsymbol{s} = s_1 s_2 \ldots s_{n-1} s_n$ such that $s_n \in S_F$, and $\pi_i \in \Delta(s_{i-1}, s_i)$, $\forall \pi_i \in \boldsymbol{\pi}, s_i \in \boldsymbol{s}$, it is said S_F is reachable from s_1. Let $FeaS(\phi)$ denote the set of automaton states from which S_F is reachable. Given a state $s \in S$, its feasible atomic proposition is defined as $ap \in \Delta(s, s')$ with $s' \in FeaS(\phi)$. The set of feasible atomic propositions of s for the LTL task ϕ is denoted by $FeaAP(\phi, s)$. For the agent $r_k \in R$, given its corresponding ϕ_k and the current NBA state s, we can obtain the plan according to $FeaAP(\phi_k, s)$. As outlined in Algorithm 1, the $\mathsf{Sampling}$ method in [28] is first leveraged to construct a planning tuple tree $\mathcal{T}_s$, comprising feasible plan tuples along with the associated transition costs. Subsequently, the tree undergoes continuous optimization through sampling and pruning and finally yields a sequence of tuples satisfying the task specification. For any $ap \in FeaAP(\phi_k, s)$, the corresponding task q, i.e., $LA(q) = ap$, is selected as the initial sub-task. A family of plan Π^* with q as the initial sub-task

Algorithm 1: Automaton_order.

```
Input: $s$, $q$, $\phi_k$
Output: $O_{k,k}$, $t^{next}_{k,k}$
Construct $FeaAP(\phi_k, s)$ and initialize $\mathcal{T}_s$ with tuple $(q_0, s)$;
Apply Sampling$(\mathcal{T}_s, B_k, T)$ ;
for all $q$ with $LA(q) \in FeaAP(\phi_k, s)$ do
    for $\Pi = [(q_0, s_0), (q_1, s_1), \ldots, (q_{|\Pi|}, s_{|\Pi|})]$ in $\mathcal{T}_s$, which
      satisfies $s_{|\Pi|} \in \mathcal{F}$, $q_1 = q$ do
        add $\Pi$ into $\Pi^*$;
    end
    Identify $\Pi_{min}$, i.e., Cost $(\Pi_{min}) \leq$ Cost $(\Pi)$ , $\forall \Pi \in \Pi^*$;
    $t^{next}_{k,k}(q) = $ Cost$(\Pi_{min}(0:1))$;
    $t^{all}_{k}(q) = $ Cost$(\Pi_{min})$;
end
$O_{k,k} = $ Sort$(Q, t^{all}_k)$;
Return $O_{k,k}$, $t^{next}_{k,k}$;
```

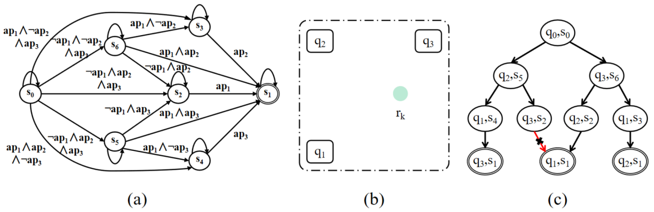


Fig. 2. (a) The NBA corresponding to the formula ϕ. (b) The workspace with a robot r_1 and three sub-tasks q_1, q_2, and q_3. (c) The tree $\mathcal{T}_s$ constructed by the sampling-based method [28], where black arrows indicate feasible transitions and red arrows indicate pruned transitions due to large cost.

can then be identified in the tree $\mathcal{T}_s$. Let $\Pi_{\min} \in \Pi^*$ be the plan with the lowest cost. Then, for agent r_k and the selected initial sub-task q, the time to complete the first loop of the suffix part can be estimated as $t^{all}_k(q) = $ Cost$(\Pi_{\min})$. Similarly, let $t^{next}_{k,k}(q) = $ Cost$(\Pi_{\min}[0:1])$ be the time for r_k to complete the next sub-task q. To reduce the whole task execution time, the sub-task q with smaller $t^{all}_k(q)$ has higher priority. Therefore, the partial orders regarding q are established by t^{all}_k, i.e., $q \succ q'$ if $t^{all}_k(q) < t^{all}_k(q')$ and the whole poset is denoted as $O_{k,k}$. The set of partial order relation $O_{k,k}$ and expected completion time $t^{next}_{k,k}$ can form the task information $I_{k,k} = (O_{k,k}, t^{next}_{k,k})$ of r_k at the current moment.

Example 1: Consider an agent r_k with an LTL task $\phi_k = Fap_1 \wedge Fap_2 \wedge Fap_3 \wedge (\neg ap_1) U (ap_3 \vee ap_2)$, whose corresponding NBA B_k is shown in Fig. 2(a). Fig. 2(b) shows the workspace, which consists of sub-tasks q_1, q_2, and q_3 with $LA(q_i) = ap_i$, $i = [3]$. Let $\mathcal{T}_s$ be the tree constructed using the Sampling method [28] and $\Pi^{k,i}$ indicates the searched ith plan in $\mathcal{T}_s$ for the agent r_k. Initially, there are two feasible atomic propositions $FeaAP(\phi_k, s_0) = \{ap_2, ap_3\}$ satisfying $LA(q_2) = ap_2$ and $LA(q_3) = ap_3$. If starting from the sub-task q_2, according to Fig. 2, there is a plan $\Pi^{k,1} = [(q_0, s_0), (q_2, s_5), (q_1, s_4), (q_3, s_1)]$ satisfying ϕ_k, i.e., $\Pi^* = \{\Pi^{k,1}\}$. Therefore, for agent r_k, $t^{next}_{k,k}(q_2) = $ Cost$(\Pi^{k,1}[0:1])$ and $t^{all}_k(q_2) = $ Cost$(\Pi^{k,1})$. Differently, if starting from the sub-task q_3, there are $\Pi^{k,2} = [(q_0, s_0), (q_3, s_6), (q_2, s_2), (q_1, s_1)]$, $\Pi^{k,3} = [(q_0, s_0), (q_3, s_6), (q_1, s_3), (q_2, s_1)]$ satisfying ϕ_k, i.e., $\Pi^* = \{\Pi^{k,2}, \Pi^{k,3}\}$. As Cost$(\Pi^{k,2}) <$ Cost$(\Pi^{k,3})$, for the agent r_k, $t^{next}_{k,k}(q_3) = $ Cost$(\Pi^{k,2}[0:1])$ and $t^{all}_k(q_3) = $ Cost$(\Pi^{k,2})$. Finally, as Cost$(\Pi^{k,2}) <$ Cost$(\Pi^{k,1})$, then $(q_3 \succ q_2) \in O_{k,k}$. Therefore, if there is no task conflict, q_3 is more preferred than q_2 as the target sub-task for r_k.

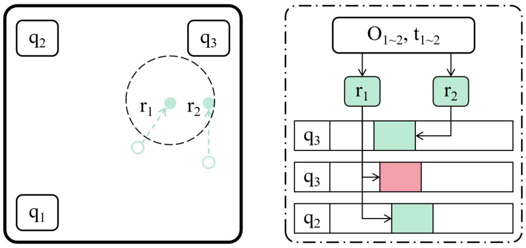
Algorithm 2: Predict_order.

```
Input: $P^{pred}_k(r_j)$
Output: $O_{k,j}$, $t^{next}_{k,j}$
for $q \in Q$ do
    if $\forall p \in P^{pred}_k$, $\|p^*_i - LM(q)\| \leq \|p - LM(q)\|$ then
        $t^{next}_{k,j}(q) = \frac{1}{v_j}$Dis$(P^{pred}_k(r_j)[\ldots i])$;
        $t_{pref}(q) = \frac{1}{v_j}\|LM(q) - p^*_i\|$;
    end
end
$O_{k,j} = $ Sort$(Q, t_{pref})$;
Return $O_{k,j}$, $t^{next}_{k,j}$;
```

B. Path Prediction

Due to limited communication range, agents can only exchange task information with neighboring agents. For agents outside the communication range, the path prediction module is developed to estimate their task information. Specifically, at time t_n, if r_j is outside the communication range of r_k, i.e. $r_j \in \bar{R}_k(t_n)$, the trajectory prediction method such as [29] is employed for r_k to predict the future m-step trajectory of r_j. That is, $P^{pred}_k(r_j) = p^{k,j}_n p^{k,j}_{n+1} \ldots p^{k,j}_{n+m-1} p^{k,j}_{n+m}$, where $p^{k,j}_n$ is the current position of r_j at time t_n and $p^{k,j}_{n+l}, l = 1, \ldots, m$, is the predicted position of r_j at time t_{n+l}. By function Predict_order in Algorithm 2 and the predicted trajectory $P^{pred}_k(r_j)$, r_k can estimate the task information of r_j, i.e. $I_{k,j} = \{O_{k,j}, t_{k,j}\}$. For sub-task q, the closest node $p^*_i \in P^{pred}_k(r_j)$ to $LM(q)$ is selected as the possible destination for r_j to perform the sub-task q, and the arrival time is estimated as $t_{k,j}(q) = \frac{1}{v_j}Dis(P^{pred}_k(r_j)[\ldots i])$, where Dis$(P^{pred}_k(r_j)[\ldots i])$ is the distance traveled by r_j along the path $P^{pred}_k(r_j)$ from $p^{k,j}_n$ to $p^{k,j}_{n+i}$. As the task that is closer to the predicted trajectory is more likely to be performed, the distance from $LM(q)$ to the trajectory $P^{pred}_k(r_j)$ is applied to measure the task preference of q. Let $t_{pref}(q) = \frac{1}{v_j}\|LM(q) - p^*_i\|$ be the deviation of $LM(q)$ and $P^{pred}_k(r_j)$. After estimation of all sub-tasks in Q, the sub-tasks are sorted to generate the partial order $O_{k,j}$, i.e., $q \succ q' \in O_{k,j}$ if $t_{pref}(q) < t_{pref}(q')$. Through the path prediction module, we can construct the task information for agents outside the communication range.

Example 2: For each agent $r_j \in \bar{R}_k(t_n)$, r_k will obtain the task information $O_{k,j}$ by prediction. We first obtain the predicted track $P^{pred}_k(r_j)$ using the method in [29]. For sub-task q_1, suppose the i_1th entry of P^{pred}_k is the nearest position to $LM(q_1)$, then, the predicted arrival time of q_1 is $t^{next}_{k,j}(q_1) = \frac{1}{v_j}Dis(P^{pred}_k[\ldots i_1])$. Similarly, for q_2, $t^{next}_{k,j}(q_2) = \frac{1}{v_j}Dis(P^{pred}_k[\ldots i_2])$, which yields $t_{pref}(q_1) = \frac{1}{v_j}\|LM(q) - p_{i_1}\|$ and $t_{pref}(q_2) = \frac{1}{v_j}\|LM(q) - p_{i_2}\|$. As $t_{pref}(q_2) < t_{pref}(q_1)$, r_j prefers to preform the sub-task q_2, i.e. $(q_2 \succ q_1) \in O_{k,j}$.



(a) (b)

Fig. 3. The selection of next sub-task. (a) r_1 and r_2 are neighboring agents within the communication range. (b) r_2 prefers q_3 and arrives at $LM(q_3)$ earlier than r_1. Due to the task conflict of performing q_3 (i.e., the red block), r_1 selects another sub-task q_2.

C. Distributed Task Allocation and Planning

After each agent r_k obtains the task information $I_{k,j}, \forall r_j \in R$, the hedonic game from [17] can be employed for distributed task allocation, i.e. adjusting the allocation in finite times (less than $\frac{n_a(n_a+1)}{2}$) based on the task preference and performance until no agent can individually increase its performance. Since shorter waiting time is preferred, for a sub-task q, the agent preferring q with earlier arrival time will finally occupy the sub-task q. Consequently, following a similar procedure as in the hedonic game, the task allocation game can be played by asking the agents with earlier arrival time to execute next sub-task, which does not change with the task allocation result and the maximum number of iterations can be reduced to n_a.

Specifically, as outlined in Algorithm 3, leveraging $O_k = \{O_{k,j} | \forall r_j \in R\}$ and $t_k^{next} = \{t_{k,j}^{next} | \forall r_j \in R\}$, each agent r_k at t_n predicts the next sub-tasks of the other agents, yielding a conflict-free task allocation $G_k(t_n) = \{g_{k,1}, \ldots, g_{k,n_a}\}$, where $g_{k,j} \in Q, \forall j \in [n_a]$, indicates the predicted next sub-task of agent r_j by agent r_k. Specifically, if there exists a $g \in G_k$ such that $g = \emptyset$, then for each sub-task $q \in Q$, all agents in $R_{pref}(q)$ are allowed to participate in the sub-task q, where $R_{pref}(q)$ represents the set of agents that prefer q according to O_k. If $r_j \in R_{pref}(q)$ arrives at q_j first, the next sub-task of r_j will be set as $g_{k,j} = q$. For the other agents, if there exists an agent r_i such that $t_{k,i}^{next}(q) \in [t_{k,j}^{next}(q), t_{k,j}^{next}(q) + LT(q)]$, it indicates r_i cannot participate in the sub-task q on time because of task conflict. Then, all preference relation with q will be removed from the poset $O_{k,i}$, which can clear the conflict of sub-task q. We then set $O_{k,j} = \emptyset$ to indicate that agent r_j will not participate in the selection of other sub-tasks. If an agent r_j has $g_{k,j} = \emptyset$ and $O_{k,j} = \emptyset$, it indicates that there is no feasible task at the current time, and the next sub-task is set to q_0, i.e., r_j is considered to stay at its current position $LM(q_0)$. Such a procedure continues until all agents complete the task selection.

Example 3: Consider two agents operating in the environment as shown in Fig. 3(a). Suppose that r_1 is aware of its own task information $I_{1,1} = \{O_{1,1}, t_{1,1}^{next}\}$ and $I_{1,2} = \{O_{1,2}, t_{1,2}^{next}\}$ of r_2 via communication. r_1 then selects its next sub-task by function Select as shown in Fig. 3(b). As there exist $(q_3 \succ q_2) \in O_{1,1}$ and $(q_3 \succ q_2) \in O_{1,2}$, there is no agent considering q_1 or q_2 as the preferred sub-task and q_3 is the preferred sub-task for both r_1 and r_2, i.e. $r_1, r_2 \in R_{pre}$. Since there exists $t_{1,2}^{next}(q_3) < t_{1,1}^{next}(q_3)$, $g_{1,2}$ is set as q_3. Besides, there exists a task conflict for r_1 performing q_3, i.e. $t_{1,1}^{next}(q_3) \in [t_{1,2}^{next}(q_3), t_{1,2}^{next}(q_3) + LT(q_3)]$, and q_3 will be removed from $O_{1,1}$. Then, q_2 is the preferred sub-task for r_1 and $g_{1,1}$ is set as q_2. Therefore, q_2 is set as the next sub-task of r_1. For the agent r_2, it will obtain the same task information and q_3 will be selected as its next sub-task. As r_2 is the neighboring agent of r_1, i.e. $r_2 \in R_1$, there exist $I_{2,1} = I_{1,1}$ and $I_{2,2} = I_{1,2}$. Therefore, $G_2 = G_1$ and q_3 is the next sub-task of r_2.

Algorithm 3: Select.

```
Input: $O_k$, $t_k^{next}$
Output: $G_k$
Initialize $\forall g \in G_k$, $g = \emptyset$;
while exists $g = \emptyset$ do
    for $q \in Q$ do
        if $\forall q_i \in Q$, $q \succcurlyeq q_i$ for $O_{k,j} \in O_k$ then
            add $r_j$ in $R_{pref}$;
        end
        if $\forall r_i \in R_{pref}$, $t_{k,j}^{next}(q) \leq t_{k,i}^{next}(q)$ then
            Set $g_{k,j} = q$;
        end
        for $r_i \in R \setminus r_j$ do
            if $t_{k,i}^{next}(q) \in [t_{k,j}^{next}(q), t_{k,j}^{next}(q) + LT(q)]$ then
                Delete $q$ from $O_{k,i}$;
            end
        end
        Set $O_{k,j} = \emptyset$;
    end
    if exists $g_{k,j} \in G$, $g_{k,j} = \emptyset$, $O_{k,j} = \emptyset$ then
        Set $g_j = q_0$;
    end
end
Return $G_k$;
```

D. System Synthesis

Based on Sections III-A–III-C, we summarize the overall architecture in this section. As outlined in Algorithm 4, r_k first obtains its own posets via function Automaton_order at each step and receives the task information of neighboring agents $r \in R_k$ via communication. For agents $r_j \in \bar{R}_k$ outside the communication range, the path prediction module is carried out to obtain P_k^{pred} by [29] and the developed function Predict_order is applied to predict the posets and arrival times of agents. The poset O of all agents and the expected time to arrive at the next sub-task are then established. The function Select is used to obtain the desired task set G_k for all agents and returns the target sub-task $g_{k,k}$ of agent r_k.

IV. Algorithm Analysis

This section investigates the performance of the distributed task allocation in the following aspects: the feasibility and the complexity. The feasibility indicates that task conflicts can be detected and properly coordinated.

A. Feasibility

If the agents are out of the communication range, their task information have to be predicted. Hence, there might be task

Algorithm 4: Distributed Task Allocation.

```
Input: $s$, $p$
Output: $g_{k,k}$
$O_{k,k}, t_{k,k}^{next} = \mathsf{Automaton\_order}(s, p)$;
for $r_j \in R_k$ do
    Receive $I_{k,j} = \{O_{k,j},\, t_{k,j}^{next}\}$ ;
end
Update $P(r_j)$ for $\forall r_j \in R$;
for $r_j \in R_k$ do
    Predict $P_k^{pred)}$ using [29];
end
for $r_j \in \bar{R}_k$ do
    $O_{k,j}, t_{k,j}^{next} = \mathsf{Predict\_order}(P_k^{pred}(r_j))$;
end
$G_k = \mathsf{Select}(O_k, t_k^{next})$;
Return $g_{k,k} \in G_k$;
```

conflicts due to inaccurate prediction. In this section, we first show that, if the prediction of task information is accurate, no task conflicts exist using our proposed task allocation scheme. Then, we show the agent can re-plan to resolve potential task conflicts if the prediction is not accurate.

Theorem 1: Given accurate task information $I_{k,j}$, $\forall r_j \in R$ (i.e., the task preference and arrival time are consistent and satisfy $I_{k,j} = I_{k',j}, \forall r_{k'} \in R$), the proposed task allocation scheme guarantees that no task conflicts exist.

Proof: If the communication range is sufficiently large that all agents can communicate to obtain others' task information, such task information is considered as accurate. According to Algorithm 3, if there exist task conflicts, the agent with longer arrival time will not be selected for this task. Instead, it will select the next task according to the task preference. If all tasks of an agent are in conflict and the arrival time is longer than other agents, this agent will stay idle to avoid conflict. Hence, if other agent task information obtained by the agent is accurate, there is no conflict in the task allocation. ■

Remark 1: In the proposed distributed algorithm, the next possible task of agent can only be predicted through a small amount of information outside the communication range. Since the actual task state cannot be fully taken into account, the real task sequence of agent cannot be obtained. Therefore, the inaccuracy of partial task information will inevitably affect the feasibility of planning.

Theorem 2: The task conflict caused by inaccuracy of partial task information in a short time can be predicted and avoided in advance.

Proof: Consider a case that agents r_k and r_j need to go to q and they are not neighboring agents, i.e., $g_{k,k} = g_{j,j} = q$ and $r_k \notin R_j$. Suppose r_k predicts that r_j will go to q', i.e., $g_{k,j} = q'$. Therefore, r_k is not aware of that the target sub-task of r_j is q at first and thus goes to q according to its predicted task information. Similarly, r_j can also misjudge the target sub-task of r_k and continues to go to q. Since both r_k and r_j are heading to q, when r_k and r_j are sufficiently close to be able to communicate, i.e., $r_k \in R_j$, r_k will discover the actual target sub-task of r_j and re-selects the target sub-task based on the current task information. It is the same for r_j. Finally, only one agent is guaranteed to perform q. Note that although r_k and r_j are not in each other's communication range, there is an area centered on the $LM(q)$, containing only $LM(q)$, r_k and r_j. Therefore, according to Algorithm 3, r_i and r_j will adjust the current target sub-task according to the expected arrival time to ensure that at most one agent reaches q. ■

The above analysis shows that there might be conflict of target sub-tasks due to inaccurate prediction of task information. However, before the agents reach the wrong common destination, once they are close enough to communicate about the true task information, such conflicts can be resolved. That is, only one agent at most will be assigned with the sub-task in this period of time. Therefore, the feasibility of the distributed real-time planning algorithm is guaranteed.

B. Complexity

At each time step, the agent updates its own task information, communicates with neighboring agents to obtain other's task information, and predicts the task information of agents outsides its communication zone. The next sub-tasks are then determined based on all collected task information. Since the transitions of NBA states only rely on previous automaton states, it only needs to update the task information according to the current position without re-sampling in the same NBA state. Re-sampling is only needed when the task states change. Since the sampling tree at previous timestamp contains the current state transition, the current sampling tree can be directly obtained as one of its sub-tree. By [30] the time complexity of planning is $O(|S|^2|Q|^2)$. The agent will obtain the corresponding poset O by sorting and the complexity is $O(|Q| \log |Q|)$. The complexity of sub-task selection is based on Algorithm 3. For n_a agents, there are at most n_a selections and the ith selection has at most $n_a - i + 1$ agents can be selected. Therefore, the time complexity of Algorithm 3 is $\frac{n(n+1)}{2}$.

V. Result

The proposed distributed task allocation and planning is evaluated via simulation and physical experiment in this section. LTL2STAR is used to convert LTL formula to NBA [26]. Matlab 2019b is used for numerical simulation. Ubuntu 18.4 and ROS melodic are used for experiment.

A. Numerical Simulation

Consider an environment as shown in Fig. 4, in which ap_i, $i = 1, \ldots, 8$, represent the areas of interest. There are 4 robots of 2 different types represented as blue and gray dots, respectively. Suppose the LTL task of r_1 and r_2 is $\phi_1 = \phi_2 = GF(ap_3 \vee ap_4) \wedge GF(ap_1 \vee ap_2) \wedge GF(ap_5 \vee ap_6 \vee ap_7 \vee ap_8)$, and the LTL task of r_3 and r_4 is $\phi_3 = \phi_4 = GF(ap_1 \vee ap_2) \wedge GF(ap_3 \vee ap_4) \wedge GF(ap_5 \vee ap_6 \vee ap_7 \vee ap_8)$. We consider the following two cases: communication only and communication with prediction.

For case 1 (i.e., communication only) and case 2 (i.e., communication with prediction), we test the performance of the proposed task allocation under different communication range D.

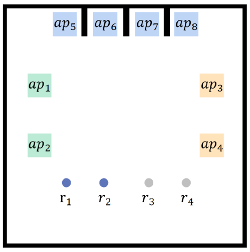


Fig. 4. The simulation environment is of size 10 × 10 and consists of 8 areas of interest represented as ap_1-ap_8. There are 4 robots of 2 different types operating in the environment, which are represented as blue (i.e., r_1 and r_2) and gray dots (i.e., r_3 and r_4).

TABLE I
PERFORMANCE UNDER DIFFERENT COMMUNICATION RANGE

D	0	1	2	3	4	5	6
Case 1	315.8	315.4	315.6	315.4	233.2	232.4	230.8
Case 2	244.6	243.6	241.4	236.0	233.4	231.8	230.0

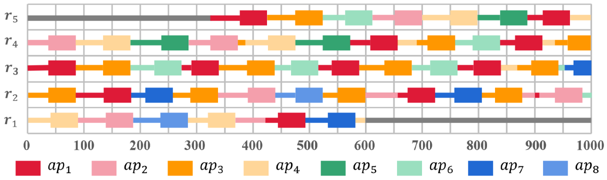


Fig. 5. The X-axis is the time stamps and the Y-axis is the agent indices. The block indicates the task that the agent is performing currently. The gray line indicates that the agent has no target sub-task.

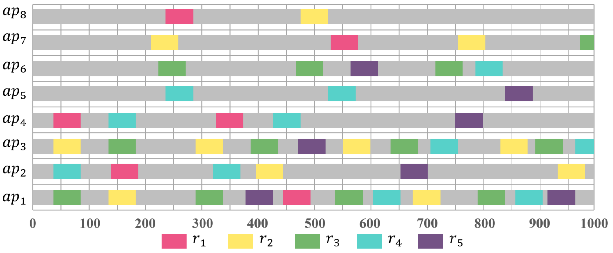


Fig. 6. The X-axis is the time stamps and the Y-axis is the atomic propositions. The gray blocks indicates that the sub-task is idle. The colored block indicates that the corresponding sub-task is occupied by an agent.

The group cost $\max_{r_k \in R}\{\mathsf{Cost}(\Pi^k_{finite})\}$ (i.e., the total elapsed time) is reported in Table I. Note that, $D = 0$ indicates that there is no communication between agents. The group cost decreases with the increase of the communication range, since the tasks can be better coordinated if more task information of other agents are available. For the same communication range, the group cost of case 2 is less than that of case 1, which demonstrates that the prediction module can effectively improve the efficiency of distributed task planning.

We next show that the proposed task allocation can successfully deal with the addition and removal of agents during task operation. Consider an additional agent r_5 with $\phi_5 = \phi_3$, which will be added to the system at the timestamp t_{300}. The agent r_1 will be removed from the timestamp t_{600}. The simulation results are shown in Figs. 5 and 6. In the first 300 time steps, the tasks are well allocated to the four initial agents. When r_5 joins in the system, the task allocation is updated with little waiting time. When r_1 breaks out, ap_1 and ap_3 are preferred for better task coordination.

TABLE II
COMPARISON METHODS

$\lvert n_a \rvert$	Task	$\lvert S \rvert$	P. Yu [24]	Ours
12	ϕ_1	16	395.8	300.2
12	$\phi_1 \vert \phi_2$	16|16	388.4	281.8
6	$\phi_1 \vert \phi_2$	16|16	232.8	172.6
24	$\phi_1 \vert \phi_2$	16|16	521.0	451.6
12	$\phi_4 \vert \phi_5$	8|8	313.8	222.0
12	$\phi_6 \vert \phi_7$	32|32	377.6	298.6
12	$\phi_1 \vert \phi_2 \vert \phi_3$	16|16|16	382.0	250.2

Our approach is then compared with [24] under different tasks (i.e., $\phi_i, i \in [7]$) with different number of agents and types. Each set is performed in five different workspaces with 8 areas of interest and the average group costs are shown in Table II, where $|S|$ indicates the size of NBA for different tasks. For instance, the last row in Table II indicates there are 12 agents of 3 types corresponding to tasks ϕ_1, ϕ_2, and ϕ_3, respectively, and the corresponding automaton sizes are all 16. Table II demonstrates that the task-level conflict avoidance can improve the execution efficiency, as [24] only considers path-level conflict avoidance.

$$\phi_1 = Fap_1 \wedge Fap_2 \wedge F(ap_3 \vee ap_4) \wedge F(ap_5 \vee ap_6),$$

$$\phi_2 = Fap_3 \wedge Fap_4 \wedge F(ap_5 \vee ap_6) \wedge F(ap_7 \vee ap_8),$$

$$\phi_3 = Fap_5 \wedge Fap_6 \wedge F(ap_7 \vee ap_8) \wedge F(ap_1 \vee ap_2),$$

$$\phi_4 = Fap_1 \wedge Fap_2 \wedge F(ap_3 \vee ap_4),$$

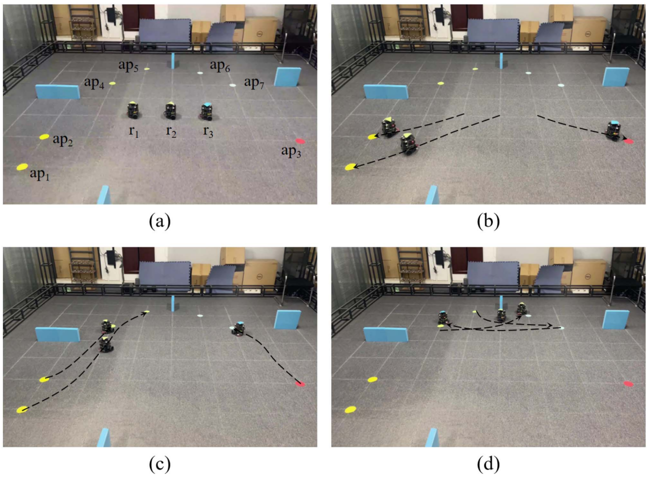
$$\phi_5 = Fap_5 \wedge Fap_6 \wedge F(ap_1 \vee ap_2),$$

$$\phi_6 = Fap_1 \wedge Fap_2 \wedge \left(\bigwedge_{i=3,5,7} F(ap_i \vee ap_{i+1}) \right)$$

$$\phi_7 = Fap_3 \wedge Fap_4 \wedge \left(\bigwedge_{i=1,5,7} F(ap_i \vee ap_{i+1}) \right)$$

B. Physical Experiment

Consider a factory environment as shown in Fig. 7(a), where ap_1 and ap_2 represent the goods shelves, ap_3 represents the letters shelf, ap_4 and ap_5 represent the recharge stations, and ap_6 and ap_7 represent the warehouses. Suppose there are 3 agents of 2 different types performing the cargo sorting task. The agents r_1 and r_2 belong to type 1, which should deliver the goods from the goods shelf to a warehouse and then return to the recharge station. Such a task for r_1 and r_2 is written in an LTL formula as $\phi_1 = \phi_2 = ((\neg ap_4 \wedge \neg ap_5 \wedge \neg ap_6 \wedge \neg ap_7) U (ap_1 \vee ap_2)) \wedge F(ap_4 \vee ap_5) \wedge F(ap_6 \vee ap_7)$. The agent r_3 belongs to type 2, which should deliver letters from the letters shelf to a warehouse and return to the recharge station. The task specification for r_3 is $\phi_3 = ((\neg ap_4 \wedge \neg ap_5 \wedge \neg ap_6 \wedge \neg ap_7) U (ap_3)) \wedge F(ap_4 \vee ap_5) \wedge F(ap_6 \vee ap_7)$. The experiment results are shown in Fig. 7. In (b), the agents r_1, r_2, r_3 are assigned with the task ap_2, ap_1, ap_3, respectively. r_2 selects ap_1 as it predicts that



(a) (b) (c) (d)

Fig. 7. The dashed lines indicate the agent trajectories. In (a), ap_1-ap_7 represents the areas of interest, respectively. There are two kinds of agents: $\{r_1, r_2\}$ and $\{r_3\}$. In (b)-(d), the agents are executing their sub-tasks.

r_1 will arrive at ap_2 before itself. In (c), agents are assigned with the task ap_4, ap_5, ap_7, respectively. r_2 selects ap_5 as it predicts that r_1 will arrive at ap_4 before itself. In (d), agents are assigned with the task ap_6, ap_7, ap_4, respectively. The experiment video is provided.[1]

VI. Conclusion

In this letter, a distributed task allocation and planning method is developed for a heterogeneous multi-agent system subject to communication constraints and linear temporal logic specifications. Each agent can obtain its own plan by the received or predicted task information. Simulation results demonstrate the effectiveness of the proposed method.

References

[1] Z. Yan, N. Jouandeau, and A. A. Cherif, “A survey and analysis of multi-robot coordination,” *Int. J. Adv. Robot. Syst.*, vol. 10, no. 12, p. 399, 2013.

[2] L. Li, Z. Chen, H. Wang, and Z. Kan, “Fast task allocation of heterogeneous robots with temporal logic and inter-task constraints,” *IEEE Robot. Automat. Lett.*, vol. 8, no. 8, pp. 4991–4998, Aug. 2023.

[3] M. Guo and D. V. Dimarogonas, “Multi-agent plan reconfiguration under local LTL specifications,” *Int. J. Robot. Res.*, vol. 34, no. 2, pp. 218–235, 2015.

[4] N. Jabeur, T. Al-Belushi, M. Mbarki, and H. Gharrad, “Toward leveraging smart logistics collaboration with a multi-agent system based solution,” *Procedia Comput. Sci.*, vol. 109, pp. 672–679, 2017.

[5] N. Seenu, K. C. RM, M. M. Ramya, and M. N. Janardhanan, “Review on state-of-the-art dynamic task allocation strategies for multiple-robot systems,” *Ind. Robot., Int. J. Robot. Res. Appl.*, vol. 47, no. 6, pp. 929–942, 2020.

[6] E. Schneider, E. I. Sklar, and S. Parsons, “Mechanism selection for multi-robot task allocation,” in *Proc. Annu. Conf. Towards Auton. Robotic Syst.*, 2017, pp. 421–435.

[7] W. P. N. dos Reis and G. S. Bastos, “An arrovian view on the multi-robot task allocation problem,” in *Proc. IEEE Int. Conf. Advan. Robot.*, 2017, pp. 290–295.

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

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

[10] Y. E. Sahin, P. Nilsson, and N. Ozay, “Multirobot coordination with counting temporal logics,” *IEEE Trans. Robot.*, vol. 36, no. 4, pp. 1189–1206, Aug. 2020.

[11] M. Cai, K. Leahy, Z. Serlin, and C.-I. Vasile, “Probabilistic coordination of heterogeneous teams from capability temporal logic specifications,” *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 1190–1197, Apr. 2022.

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

[13] J. Tumova and D. V. Dimarogonas, “Multi-agent planning under local LTL specifications and event-based synchronization,” *Automatica*, vol. 70, pp. 239–248, 2016.

[14] P. Schillinger, M. Bürger, and D. V. Dimarogonas, “Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems,” *Int. J. Robot. Res.*, vol. 37, no. 7, pp. 818–838, 2018.

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

[16] I. Hustiu, C. Mahulea, and M. Kloetzer, “Distributing co-safe LTL specifications to mobile robots,” in *Proc. IEEE Int. Conf. Syst. Theory, Control, Comput.*, 2022, pp. 306–311.

[17] I. Jang, H.-S. Shin, and A. Tsourdos, “Anonymous hedonic game for task allocation in a large-scale multiple agent system,” *IEEE Trans. Robot.*, vol. 34, no. 6, pp. 1534–1548, Dec. 2018.

[18] I. Jang, H.-S. Shin, A. Tsourdos, J. Jeong, S. Kim, and J. Suk, “An integrated decision-making framework of a heterogeneous aerial robotic swarm for cooperative tasks with minimum requirements,” *J. Aerosp. Eng.*, vol. 233, no. 6, pp. 2101–2118, 2019.

[19] W. Zhao, Q. Meng, and P. W. Chung, “A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario,” *IEEE Trans. Cybern.*, vol. 46, no. 4, pp. 902–915, Apr. 2016.

[20] W. Lee and D. Kim, “Adaptive approach to regulate task distribution in swarm robotic systems,” *Swarm Evol. Comput.*, vol. 44, pp. 1108–1118, 2019.

[21] M. Otte, M. Kuhlman, and D. Sofge, “Competitive target search with multi-agent teams: Symmetric and asymmetric communication constraints,” *Auton. Robots*, vol. 42, pp. 1207–1230, 2018.

[22] M. Otte, M. J. Kuhlman, and D. Sofge, “Auctions for multi-robot task allocation in communication limited environments,” *Auton. Robots*, vol. 44, pp. 547–584, 2020.

[23] Z. Liu, B. Wu, J. Dai, and H. Lin, “Distributed communication-aware motion planning for networked mobile robots under formal specifications,” *IEEE Trans. Control Netw. Syst.*, vol. 7, no. 4, pp. 1801–1811, Dec. 2020.

[24] P. Yu and D. V. Dimarogonas, “Distributed motion coordination for multirobot systems under LTL specifications,” *IEEE Trans. Robot.*, vol. 38, no. 2, pp. 1047–1062, Apr. 2022.

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

[26] P. Gastin and D. Oddoux, “Fast LTL to büchi automata translation,” in *Proc. Int. Conf. Comput. Aided Verif.*, 2001, pp. 53–65.

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

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

[29] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, “Social GAN: Socially acceptable trajectories with generative adversarial networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit*, 2018, pp. 2255–2264.

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

[1] [Online]. Available: https://youtu.be/lm8klvUrCqM