Time Minimization and Online Synchronization for Multi-agent Systems under Collaborative Temporal Logic Tasks

Zesen Liu, Meng Guo and Zhongkui Li[1]

State Key Laboratory for Turbulence and Complex Systems,Department of Mechanics and Engineering Science, College of Engineering, Peking University, Beijing 100871, China.

Abstract

Multi-agent systems can be efficient when solving a team-wide task in a concurrent manner. However, without proper synchronization, the correctness of the combined behavior is hard to guarantee, such as to follow a specific ordering of subtasks or to perform a simultaneous collaboration. This work addresses the minimum-time task planning problem for multi-agent systems under complex global tasks stated as syntactically co-safe Linear Temporal Logic (sc-LTL) formulas. These tasks include the temporal and spatial requirements on both independent local actions and direct sub-team collaborations. The proposed solution is an anytime algorithm that combines the partial-ordering analysis of the underlying task automaton for task decomposition, and the branch and bound (BnB) search method for task assignment. We analyze the soundness, completeness and optimality regarding the minimal completion time. We show that a feasible and near-optimal solution is quickly reached while the search continues within the time budget. Furthermore, to handle fluctuations in task duration and agent failures during online execution, we propose an adaptation algorithm to synchronize execution status and re-assign unfinished subtasks dynamically to maintain correctness and optimality. Both algorithms are validated over systems of more than 10 agents via numerical simulations and hardware experiments, against several established baselines.

Keywords: Networked Robots, Linear Temporal Logic, Task Coordination, Anytime Search.

1. Introduction

Multi-agent systems consist of a fleet of homogeneous or heterogeneous robots, such as autonomous ground vehicles and aerial vehicles. They are often deployed to accomplish tasks that are otherwise too inefficient or even infeasible for a single robot in (Arai et al., 2002). First, by allowing the robots to move and act concurrently, the overall efficiency of the team can be significantly improved. For instance, a fleet of delivery vehicles can drastically reduce the delivery time (Toth & Vigo, 2002); and a team of drones can surveil a large terrain and detect poachers (Cliff et al., 2015). Second, by enabling multiple robots to directly collaborate on a task, capabilities of the team can be greatly extended. For instance, several mobile manipulators can transport objects that are otherwise too heavy for one (Fink et al., 2008); and a team of mobile vehicles can collaboratively herd moving targets via formation (Varava et al., 2017). Furthermore, to specify these complex tasks, many recent work propose to use formal languages such as Linear Temporal Logic (LTL) formulas (Baier & Katoen, 2008), as an intuitive yet powerful way to describe both spatial and temporal requirements on the team behavior; see (Ulusoy et al., 2013; Kantaros & Zavlanos, 2020; Schillinger et al., 2018; Guo & Zavlanos, 2018) for examples.

However, coordination of these robots to accomplish the desired task can result in great complexity, as the set of possible task assignments can be combinatorial with respect to the number of robots and the length of tasks (Toth & Vigo, 2002; Lavaei et al., 2022). This is particularly the case when certain metrics should be minimized, such as the completion time or the summed cost of all robots. There are many recent work obtain onsiderable results, e.g., via optimal planning algorithms such as mixed integer linear programming (MILP) in (Luo & Zavlanos, 2022; Sahin et al., 2019; Jones et al., 2019), and search algorithms over state or solution space as in (Kantaros & Zavlanos, 2020; Schillinger et al., 2018; Luo et al., 2021). Whereas being sound and optimal, many existing solutions are designed from an offline perspective, thus lacking in one of the following aspects that could be essential for robotic missions: (I) *Real-time requirements*. Optimal solutions often take an unpredictable amount of time to compute, without any intermediate feedback. However, for many practical applications, good solutions that are generated fast and reliably are often more beneficial; (II) *Synchronization* during plan execution. Many of the derived plans are assigned to the robots and executed independently without any synchronization among them, e.g., no coordination on the start and finish time of a subtask. Such procedure generally relies on a precise model of the underly-

[1]This work was supported by the National Natural Science Foundation of China under grants U2241214, T2121002.

E-mail: 1901111653@pku.edu.cn (Zesen Liu), meng.guo@pku.edu.cn (Meng Guo), zhongkli@pku.edu.cn (Zhongkui Li).

Table 1: Comparison of related work as discussed in Sec. 2.2, regarding the problem formulation and key features.

Ref.	Syntax	Collaboration	Objective	Solution	Anytime	Synchronization	Adaptation
(Guo & Dimarogonas, 2015; Tumova & Dimarogonas, 2016)	Local-LTL	No	Summed Cost	Dijkstra	No	Event-based	Yes
(Guo & Dimarogonas, 2016)	Local-LTL	Yes	Summed Cost	Dijkstra	No	Event-based	Yes
(Kantaros & Zavlanos, 2020; Luo et al., 2021)	Global-LTL	No	Summed Cost	Sampling	Yes	All-time	No
(Schillinger et al., 2018)	Global-LTL	No	Balanced	Martins' Alg.	No	None	No
(Luo & Zavlanos, 2022)	Global-LTL	Yes	Summed Cost	MILP	No	All-time	No
(Sahin et al., 2019; Jones et al., 2019)	Global-cLTL	No	Summed Cost	MILP	No	Partial	No
Ours	Global-LTL	Yes	Min Time	BnB	Yes	Event-based	Yes

ing system such as traveling time and task execution time, and the assumption that no direct collaborations are required. Thus, any mismatch in the given model or dependency in the subtasks would lead to erroneous execution; (III) *Online adaptation*. An optimal but static solution can not handle changes in the workspace or in the team during online execution, e.g., certain paths between subtasks are blocked, or some robots break down.

To overcome these challenges, this work takes into account the minimum-time task coordination problem of a team of heterogeneous robots. We specify the team-wise task as LTL formulas over the desired actions to perform at the desired regions of interest in the environment. Such action can be independent that it can be done by one of these robots alone, or collaborative where several robots should collaborate to accomplish it. The objective is to find an optimal task policy for the team such that the completion time for the task is minimized. Due to the NP-completeness of this problem, the focus here is to design an anytime algorithm that returns the best solution within the given time budget. More specifically, the proposed algorithm interleaves between the partial ordering analysis of the underlying task automaton for task decomposition, and the branch and bound (BnB) search method for task assignment, Each of these two sub-routines is anytime-algorithm. The proposed partial relations can be applied to non-instantaneous subtasks, thus providing a more general model for analyzing concurrent subtasks. Furthermore, the proposed lower and upper bounding methods during the BnB search significantly reduces the search space. We prove the completeness and soundness of the overall algorithm for the considered objective, and we also show empirically that algorithm can quickly reach a feasible and near-optimal solution. Besides, we propose an online synchronization protocol to handle fluctuations in the execution time, while ensuring that the partial ordering constraints are still respected. Lastly, to handle agent failures during the task execution, we propose an adaptation algorithm to synchronize execution status and re-assign unfinished subtasks to maintain correctness and optimality. The effectiveness and advantages of the proposed algorithm are demonstrated rigorously via numerical simulations and hardware experiments, particularly over large-scale systems of more than 20 robots and 10 subtasks.

The main contribution of this work is threefold: (I) it extends the existing work on temporal task planning to allow collaborative subtasks and the practical objective of minimizing task completion time; (II) it proposes an anytime algorithm that is sound, complete and optimal, which is particularly suitable for real-time applications where computation time is restricted; and (III) it provides a novel theoretical approach that combines the partial ordering analysis for task decomposition and the BnB search for task assignment.

The rest of the paper is organized as follows: Sec. 2 reviews related work. The formal problem description is given in Sec. 4. Main components of the proposed framework are presented in Secs. 5-7. Experiment studies are shown in Sec. 8, followed by conclusions and future work in Sec. 9.

2. Related Work

2.1. Multi-agent Task Planning

Planning for multi-agent systems have two distinctive characteristics: high-level task planning in the discrete task space, and low-level motion planing in the continuous state space. In particular, given a team-wise task, task planning refers to the process of first decomposing this task into subtasks and then assigning them to the team, see (Torreño et al., 2017; Gini, 2017) for comprehensive surveys. Such tasks can have additional constraints, such as time windows (Luo et al., 2015), and ordering constraints (Nunes & Gini, 2015). The optimization criteria can be single or multiple, two of which are the most common: MinSUM that minimizes the sum of robot costs over all robots (Gini, 2017; Luo et al., 2015), and MinMAX that minimizes the maximum cost of a robot over all robots (Nunes & Gini, 2015), similar to the *makespan* of all tasks. Typical solutions can be categorized into centralized methods such as Mixed Integer Linear Programming (MILP) (Torreño et al., 2017) and search-based methods (Toth & Vigo, 2002); and decentralized methods such as market-based methods (Luo et al., 2015), distributed constraint optimization (DCOP) (Nunes & Gini, 2015). However, since many task planning problems are in general NP-hard or even NP-complete (Hochba, 1997), meta-heuristic approaches are used to gain computational efficiency, e.g., local search (Hoos & Stützle, 2004) and genetic algorithms (Khamis et al., 2015). One type of problem is particularly of relevance to this work, namely the Multi-Vehicle Routing Problem (MVRP) in operation research (Gini, 2017), where a team of vehicles are deployed to provide services at a set of locations, and the MinSum objective above is optimized. Despite of the similarity, this work considers a significantly more general specification of team-wise tasks, which can include as special cases

the vanilla formulation and its variants with temporal and spatial constraints (Nunes & Gini, 2015). Moreover, collaborative tasks and synchronization during online execution are often neglected in the aforementioned work, where planning and execution are mostly decoupled.

2.2. Temporal Logic Tasks

Temporal logic formulas can be used to specify complex robotic tasks, such as Probabilistic Computation Tree Logic (PCTL) in (Lahijanian et al., 2011), Linear Temporal Logics (LTL) in (Kantaros & Zavlanos, 2020; Schillinger et al., 2018; Guo & Dimarogonas, 2015; Chen et al., 2011), and counting LTL (cLTL) in (Sahin et al., 2019). As summarized in Table 1, the most related work can be compared in the following four aspects: (i) *collaborative tasks*. In a bottom-up fashion, (Guo & Dimarogonas, 2015; Tumova & Dimarogonas, 2016; Guo & Dimarogonas, 2016) assume local LTL tasks and dynamic environment, where collaborative tasks are allowed in (Guo & Dimarogonas, 2016). In a top-down fashion, (Kantaros & Zavlanos, 2020; Schillinger et al., 2018; Luo & Zavlanos, 2022; Sahin et al., 2019; Jones et al., 2019) consider team-wise tasks, but no direct collaboration among the agents; (ii) the *optimization criteria*. Most aforementioned work (Kantaros & Zavlanos, 2020; Guo & Dimarogonas, 2016; Luo & Zavlanos, 2022; Sahin et al., 2019; Jones et al., 2019) optimizes the summed cost of all agents, while (Schillinger et al., 2018) evaluates a weighted balance between this cost and the task completion time. Even though both objectives are valid, we emphasize in this work the achievement of maximum efficiency by minimizing solely the completion time; (iii) the *synchronization requirement*. Synchronization happens when two or more agents communicate regarding the starting time of next the next subtask. The work in (Kantaros & Zavlanos, 2020; Luo et al., 2021; Sahin et al., 2019) requires full synchronization before each subtask due to the product-based solution, while (Schillinger et al., 2018) imposes no synchronization by allowing only independent subtasks and thus limiting efficiency. This work however proposes an online synchronization strategy for subtasks that satisfies both the strict partial ordering and the simultaneous collaboration. As illustrated in Fig. 1, this can improve greatly the concurrency and thus the efficiency of the multi-agent execution even further; and lastly (iv) the *solution basis*. Solutions based on solving a MILP as in (Luo & Zavlanos, 2022; Sahin et al., 2019; Jones et al., 2019) often can not guarantee a feasible solution within a time budget, while this work proposes an anytime algorithm that could return quickly a feasible and near-optimal solution. Last but not least, instead of generating only a static team-wise plan as in the aforementioned work, the proposed online adaptation algorithm can handle fluctuations in the task duration and possible agent failures during execution.

3. Preliminaries

3.1. Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) formulas are composed of a set of atomic propositions AP in addition to several Boolean and temporal operators. Atomic propositions are Boolean variables that can be either true or false. Particularly, the syntax (Baier & Katoen, 2008) is given as follows: $\varphi \triangleq \top \mid p \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \mathsf{U} \varphi_2$, where $\top \triangleq \texttt{True}$, $p \in AP$, $\bigcirc$ (*next*), U (*until*) and $\bot \triangleq \neg\top$. For brevity, we omit the derivations of other operators like $\Box$ (*always*), $\Diamond$ (*eventually*), $\Rightarrow$ (*implication*). The full semantics and syntax of LTL are omitted here for brevity; see e.g., (Baier & Katoen, 2008).

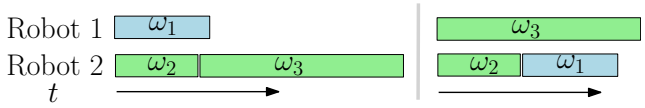

Figure 1: Comparison of the planning results based on decompositional states in (Schillinger et al., 2018) (**left**) and the partial ordering in this work (**right**). Note that subtask ω_3 has to be completed after ω_2, while ω_1 is independent.

An infinite word w over the alphabet 2^{AP} is defined as an infinite sequence $w = \sigma_1\sigma_2\cdots, \sigma_i \in 2^{AP}$. The language of φ is defined as the set of words that satisfy φ, namely, $L_\varphi = Words(\varphi) = \{\boldsymbol{w} \mid w \models \varphi\}$ and $\models$ is the satisfaction relation. However, there is a special class of LTL formula called *syntactically co-safe* formulas (sc-LTL) (Belta et al., 2017), which can be satisfied by a set of finite sequence of words. They only contain the temporal operators $\bigcirc$, U and $\Diamond$ and are written in positive normal form where the negation operator $\neg$ is not allowed before the temporal operators including $\bigcirc, \mathsf{U}, \Diamond$.

3.2. Nondeterministic Büchi Automaton

Given an LTL formula φ mentioned above, the associated Nondeterministic Büchi Automaton (NBA) (Baier & Katoen, 2008) can be derived with the following structure.

Definition 1 (NBA)**.** A NBA $\mathcal{B}$ is a 5-tuple: $\mathcal{B} = (Q, Q_0, \Sigma, \delta, Q_F)$, where Q is the set of states; $Q_0 \subseteq Q$ is the set of initial states; $\Sigma = AP$ is the allowed alphabet; $\delta : Q \times \Sigma \to 2^Q$ is the transition relation; $Q_F \subseteq Q$ is the set of *accepting* states. ■

Given an infinite word $w = \sigma_1\sigma_2\cdots$, we can get a set of resulting *runs* within $\mathcal{B}$. Each of them is an infinite sequence $\rho = q_0q_1q_2\cdots$ such that $q_0 \in Q_0$, and $q_i \in Q$, $q_{i+1} \in \delta(q_i, \sigma_i)$ hold for all index $i \geqslant 0$. The set of all words that correspond to *accepting* runs is called the *language* generated by the NBA. The language can be expressed in the prefix-suffix structure (Baier & Katoen, 2008), while the prefix part can result in a run from an initial state to an accepting state, and the suffix part can result in a cyclic run that contains the same accepting state. In addition, the satisfaction of a sc-LTL formula can be achieved in a finite time, i.e., each word satisfying an sc-LTL formula consists of a satisfying prefix and an arbitrary suffix.

3.3. Partially Ordered Set

A partial order (Simovici & Djeraba, 2008) defined on a set S is a relation $\rho \subseteq S \times S$, which is reflexive, antisymmetric, and transitive. Then, the pair (S, ρ) is referred to as a partially ordered set (or simply *poset*). A generic partial order relation is given by $\leqslant$. Namely, for $x, y \in S$, $(x, y) \in \rho$ if $x \leqslant y$. The set S is totally ordered if $\forall x, y \in S$, either $x \leqslant y$ or $y \leqslant x$ holds.

Two elements $x,\ y$ are incomparable if neither $x \leqslant y$ nor $y \leqslant x$ holds, denoted by $x \parallel y$.

4. Problem Formulation

4.1. Collaborative Multi-agent Systems

Consider a team of N agents operating in a workspace $W \subset \mathbb{R}^3$. Within the workspace, there are M regions of interest, denoted by $\mathcal{W} \triangleq \{W_1, W_2, \cdots, W_M\}$, where $W_m \in W$. Each agent $n \in \mathcal{N} = \{1, \cdots, N\}$ can navigate within these regions following its own transition graph $\mathcal{G}_n = (\mathcal{W}, \rightarrow_n, d_n)$, where $\rightarrow_n \subseteq \mathcal{W} \times \mathcal{W}$ is the allowed transition for agent n; and $d_n : \rightarrow_n \rightarrow \mathbb{R}_+$ maps each transition to its time duration.

Moreover, similar to the action model used in (Guo & Dimarogonas, 2016), each robot $n \in \mathcal{N}$ is capable of performing a set of actions: $\mathcal{A}_n \triangleq \mathcal{A}_n^{\mathrm{l}} \cup \mathcal{A}_n^{\mathrm{c}}$, where $\mathcal{A}_n^{\mathrm{l}}$ is a set of *local* actions that can be independently performed by agent n itself; $\mathcal{A}_n^{\mathrm{c}}$ is a set of *collaborative* actions that can only be performed in collaboration with other agents. Moreover, denote by $\mathcal{A}^c \triangleq \bigcup_{n\in\mathcal{N}} \mathcal{A}_n^c$, $\mathcal{A}^l \triangleq \bigcup_{n\in\mathcal{N}} \mathcal{A}_n^l$. More specifically, there is a set of collaborative behaviors pre-designed for the team, denoted by $\mathcal{C} \triangleq \{C_1, \cdots, C_K\}$. Each behavior $C_k \in \mathcal{C}$ consists of a set of collaborative actions that should be accomplished by different agents, namely:

$$C_k \triangleq \{a_1,\ a_2, \cdots, a_{\ell_k}\}, \tag{1}$$

where $\ell_k > 0$ is the number of collaborative actions required; $a_\ell \in \mathcal{A}^c,\ \forall \ell = 1, \cdots, \ell_k$. To execute a collaborative behavior C_k, each collaborative action $a_\ell \in C_k$ should be performed by a different agent n that is capable, i.e., $a_\ell \in \mathcal{A}_n$. Moreover, each collaborative action has a fixed duration as $d : \mathcal{C} \rightarrow \mathbb{R}_+$.

Remark 1. Different from (Luo & Zavlanos, 2022; Sahin et al., 2019; Jones et al., 2019), the definition of collaborative behavior above does not specify explicitly the agent identities or types, rather their capabilities. This subtle difference can improve the flexibility of the underlying solution, e.g., no hard-coding of the agent identities or types is necessary; any capable agent can be recruited for the collaborative behavior. ■

4.2. Task Specification

First, the following three types of atomic propositions can be introduced: (i) p_m is *true* when *any* agent $n \in \mathcal{N}$ is at region $W_m \in \mathcal{W}$; Let $\mathbf{p} \triangleq \{p_m, \forall W_m \in \mathcal{W}\}$; (ii) a_k^m is *true* when a *local* action a_k is performed at region $W_m \in \mathcal{W}$ by agent n, where $a_k \in \mathcal{A}_n^{\mathrm{l}}$. Let $\mathbf{a} \triangleq \{a_k^m, \forall W_m \in \mathcal{W}, a_k \in \mathcal{A}^l\}$; (iii) c_k^m is *true* when the collaborative behavior C_k in (1) is performed at region W_m. Let $\mathbf{c} \triangleq \{c_k^m, \forall C_k \in \mathcal{C}, \forall W_m \in \mathcal{W}\}$.

Given these propositions, a team-wide task specification can be specified as a sc-LTL formula:

$$\varphi = \text{sc-LTL}(\{\mathbf{p}, \mathbf{a}, \mathbf{c}\}), \tag{2}$$

where the syntax of sc-LTL is introduced in Sec. 3.1. Denote by $t_0,\ t_f > 0$ the time instants when the system starts and satisfies executing φ, respectively. Thus, the total time taken for the multi-agent team to satisfy φ is given by

$$T_\varphi = t_f - t_0. \tag{3}$$

Since a sc-LTL can be satisfied in finite time, this total duration is finite and thus can be optimized.

Remark 2. As described previously in Sec. 1, the minimum time cost in (3) is significantly different from the summed time cost of all agents, i.e., $\sum_i T_i$, where T_i is the total time agent i spent on executing task φ, as in (Kantaros & Zavlanos, 2020; Schillinger et al., 2018; Guo & Dimarogonas, 2016; Luo & Zavlanos, 2022). The main advantage of concurrent execution can be amplified via the objective of time minimization, since concurrent or sequential execution of the same task assignment could be equivalent in terms of summed cost. ■

Example 1. Consider a team of UAVs and UGVs deployed for maintaining a remote Photovoltaic power plant. One collaborative task considered in Sec. 8 is given by:

$$\begin{aligned}\varphi_1 =& \Diamond(\texttt{repair}_{\texttt{p}_3} \wedge \neg\texttt{scan}_{\texttt{p}_3} \wedge \Diamond\texttt{scan}_{\texttt{p}_3}) \wedge \Diamond(\texttt{wash}_{\texttt{p}_{21}} \wedge \\ & \Diamond\texttt{mow}_{\texttt{p}_{21}} \wedge \Diamond\texttt{scan}_{\texttt{p}_{21}}) \wedge \Diamond(\texttt{sweep}_{\texttt{p}_{21}} \wedge \neg\texttt{wash}_{\texttt{p}_{21}} \wedge \\ & \Diamond\texttt{mow}_{\texttt{p}_{21}}) \wedge \Diamond(\texttt{fix}_{\texttt{t}_5} \wedge \neg\texttt{p}_{18}) \wedge \neg\texttt{p}_{24}\, U\, \texttt{sweep}_{\texttt{p}_{27}} \\ & \wedge \Diamond(\texttt{wash}_{\texttt{p}_{34}} \wedge \bigcirc\texttt{scan}_{\texttt{p}_{34}}),\end{aligned} \tag{4}$$

which means to repair and scan certain PV panel $\texttt{p}_3$ in a given order, deeply cleaning the panel $\texttt{p}_{21}$, $\texttt{p}_{34}$, fix transformers $\texttt{t}_5$, sweep $\texttt{p}_{27}$ with safety request. ■

4.3. Problem Statement

Problem 1. Given the task specification φ, synthesize the motion and action sequence for each agent $n \in \mathcal{N}$ such that T_φ in (3) is minimized. ■

Even through the above problem formulation is straightforward, it is can be shown that this problem belongs to the class of NP-hard problems (Hochba, 1997), as its core coincides with the makespan minimization problem of flow-shop scheduling problems. Various approximate algorithms have been proposed in (Khamis et al., 2015). However, the combination of dynamic vehicles within a graph-like environment, linear temporal constraints, and collaborative tasks has not been addressed.

In the following sections, we present the main solution of this work. As shown in Fig. 2, the optimal plan synthesis which is performed offline is first described in Sec. 5, and then the online adaptation strategy to handle dynamic changes in Sec. 6. The overall solution is summarized in Sec. 7 with analyses for completeness, optimality and complexity.

5. Optimal Plan Synthesis

The optimal plan synthesis aims to solve Problem 1 offline. As mentioned previously in Sec. 2.2, most related work requires the synchronized product between models of all agents, thus subject to exponential complexity. Instead, we propose an anytime algorithm that combines seamlessly the partial-ordering analyses of the underlying task automaton for task decomposition, and the branch and bound (BnB) search method for task assignment. In particular, it consists of three main components: (i) the pre-processing of the NBA associated with the global task; (ii) the partial ordering analyses of the processed task automaton; (iii) the BnB search algorithm that searches in the plan space given the partial ordering constraints.

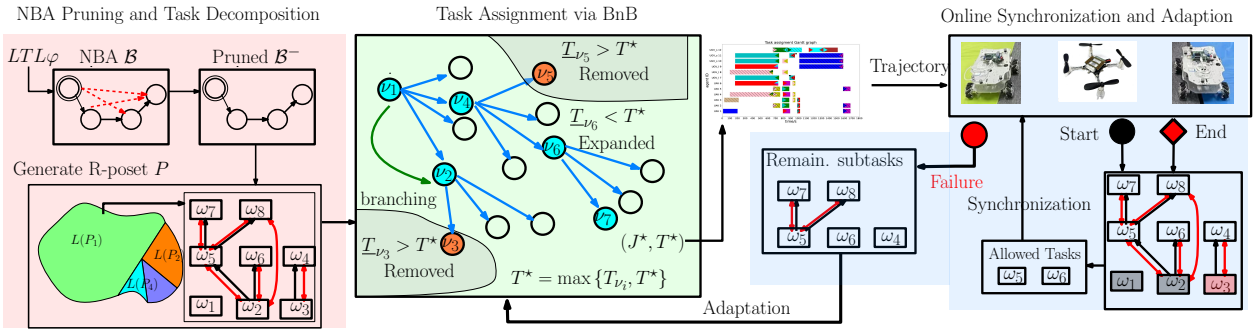

Figure 2: Overall structure of the proposed framework, which consists of three main parts: the computation of R-posets, BnB search, and online execution.

5.1. Büchi Automaton Pruning

To begin with, the NBA $\mathcal{B}_\varphi$ associated with the task φ is derived, e.g., via translation tools in (Gastin & Oddoux, 2001). Note that $\mathcal{B}_\varphi$ has the structure as defined in Def. 1, which however can be overly redundant. For instance, the required input alphabets for some transitions are infeasible for the whole team; or some transitions can be decomposed equivalently into other transitions. Via detecting and removing such transitions, the size of the underlying NBA can be greatly reduced, thus improving the efficiency of subsequent steps. More specifically, pruning of $\mathcal{B}_\varphi$ consists of three steps:

(i) Remove infeasible transitions. Given any transition $q_j \in \delta(q_i, \sigma)$ in $\mathcal{B}_\varphi$, it is infeasible for the considered system if no subgroup of agents in $\mathcal{N}$ can generate σ. It can be easily verified by checking whether there exist an agent that can navigate to region W_m and perform local action a_k; or several agents that can *all* navigate to region W_m and perform collaborative action a_k. If infeasible, this transition is removed.

(ii) Remove invalid states. Any state $q \in Q$ in $\mathcal{B}_\varphi$ is called invalid if it can not be reached from any initial state; or it can not reach any accepting state that is in turn reachable from itself. An invalid state can not be part of an accepting path thus removed in the pruned automaton.

(iii) Remove decomposable transitions. Due to the distributed nature of multi-agent systems, it is unrealistic to enforce the fulfillment of two or more subtasks to be *exactly* at the same instant in real time. Therefore, if possible, any transition that requires the simultaneous satisfaction of several subtasks is decomposed into equivalent transitions. Decomposable transitions are formally defined in Def. 2 below. In other words, the input alphabets of a decomposed transition can be mapped to two other transitions that connect the *same* pair of states, but via another intermediate state, as illustrated in Fig. 3. Thus, all decomposable transitions in $\mathcal{B}_\varphi$ are removed in the pruned automaton. Algorithmically, decomposability can be checked by simply composing and comparing the propositional formulas associated with each transition.

Definition 2 (Decomposable Transition)**.** Any transition from state q_i to q_j in $\mathcal{B}_\varphi$ is decomposable if there exists another state q_k such that $q_j \in \delta(q_i, \sigma_{ik} \cup \sigma_{kj})$ holds, $\forall \sigma_{ik}, \sigma_{kj} \subseteq \Sigma$ that $q_k \in \delta(q_i, \sigma_{ik})$ and $q_j \in \delta(q_k, \sigma_{kj})$ hold. ■

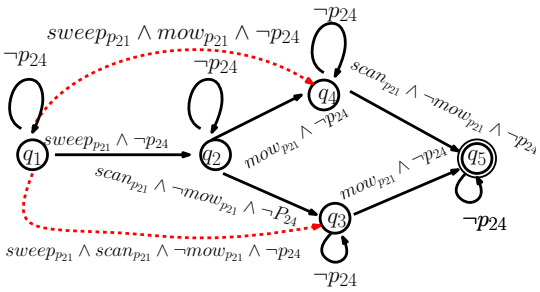

Figure 3: Example of decomposable transitions. Transitions in red dashed lines are all decomposable satisfied by Def. 2.

An example of decomposable transitions is shown in Fig. 3. To summarize, the pruned NBA, denoted by $\mathcal{B}_\varphi^-$, has the same structure as $\mathcal{B}_\varphi$ but with much fewer states and edges. In our experience, this pruning step can reduce up to 60% states and edges for typical multi-agent tasks. More details can be found in the experiment section.

Lemma 1. *If there exists a word w that is accepted by $\mathcal{B}$, then an equivalent word w' can be found that is accepted by $\mathcal{B}^-$.*

Proof. Consider an accepting word $w = \cdots\{\sigma_n\}\cdots$, and its resulting accepting run in $\mathcal{B}$ is given by $\rho = \cdots q_i q_j \cdots$, with $q_j = \delta(q_i, \sigma_n)$. For the first case, if no edges in ρ are *Decomposable Transitions*, all edges in ρ can be found in $\mathcal{B}^-$. In other words, ρ is accepting in $\mathcal{B}^-$. Namely, an equivalent word $w' = w$ can be found that is accepted by $\mathcal{B}^-$. For the second case, if the transition from q_i to q_j is removed as a decomposable transition, there exist a state q_k and σ_{kj}, σ_{ik} satisfying that $q_k = \delta(q_i, \sigma_{ik}), q_j = \delta(q_k, \sigma_{kj}), \sigma_{ik} \cup \sigma_{kj} = \sigma_n$ based on Def. 2. Then, an equivalent word $w' = \cdots\{\sigma_n\}\{\sigma_n\}\cdots$ can be created by inserting a time step in $\{\sigma_n\}$, of which the associated run is $\rho' = \cdots q_i q_k q_j \cdots$. If the transitions from q_k to q_j and from q_i to q_k are not removed in $\mathcal{B}^-$, this satisfies Case 1 and w' is the associated accepting word in $\mathcal{B}^-$. If any transition is removed as a *Decomposable Transition*, it satisfies Case 2 and the same insertion process described above can be applied. According to Def. 2, it eventually happens in Case 2 that the new created run ρ' has no *Decomposable Transition*, thus satisfying the Case 1. This completes the proof. □

Example 2. The NBA $\mathcal{B}_\varphi$ associated with the task formula in (4) has 707 states and 16044 edges with more than 1.29×10^7 accepting *words*, translated via (Gastin & Oddoux, 2001). After the pruning process described above, the pruned automaton $\mathcal{B}_\varphi^-$ has 707 states and 2423 edges with 174469 accepting *words*. This step reduces 84.9% edges and 98.6% accepting *words*. ■

5.2. Task Decomposition

Since the team-wise task in this work is given as a compact temporal task formula, a prerequisite for optimal task assignment later is to decompose this task into suitable subtasks. Moreover, different from simple reachability task, temporal tasks can impose strict constraints on the ordering of subtasks. For instance, the task $\Diamond(\alpha_1 \wedge \Diamond(\alpha_2 \wedge \Diamond\alpha_3))$ specifies that $\alpha_1, \alpha_2, \alpha_3$ should be satisfied in sequence, while $\Diamond\alpha_1 \wedge \Diamond\alpha_2 \wedge \Diamond\alpha_3$ does not impose any ordering constraint. Thus, it is essential for the overall correctness to abstract such ordering constraints among the subtasks. This part describes how the subtasks and their partial orderings are abstracted from the pruned automaton $\mathcal{B}_\varphi^-$.

Definition 3 (Decomposition and Subtasks)**.** Consider an accepting run $\rho = q_0 q_1 \cdots q_L$ of $\mathcal{B}_\varphi^-$, where $q_0 \in Q_0$ and $q_L \in Q_F$. One *possible* decomposition of φ into subtasks is defined as a set of 3-tuples:

$$\Omega_\varphi = \{\omega_\ell = (\ell,\, \sigma_\ell,\, \sigma_\ell^s),\, \forall \ell = 1, \cdots, L\}, \tag{5}$$

where ℓ is the index of subtask ω_ℓ; labels $\sigma_\ell \subseteq \Sigma$ satisfies two conditions: (i) $q_\ell \in \delta(q_{\ell-1},\, \sigma_\ell)$, and (ii) $q_\ell \notin \delta(q_{\ell-1},\, \sigma_\ell^-)$, where $\sigma_\ell^- = \sigma_\ell \backslash \{z\},\, \forall z \in \sigma_\ell$; and label $\sigma_\ell^s \subseteq \Sigma$ also satisfies two conditions: (i) $q_{\ell-1} \notin \delta(q_{\ell-1},\, \sigma)$, for all $\sigma \in \Sigma, \sigma \cap \sigma_\ell^s \neq \varnothing$, and (ii) $q_{\ell-1} \in \delta(q_{\ell-1},\, \sigma)$, for all $\sigma \in \Sigma, \sigma \cap \sigma_\ell^s = \varnothing$. ■

Example 3. As shown in Fig. 3 , the subtasks associated with run $\rho = q_1 q_2 q_4 q_5$ are given by $\Omega_\varphi = \{(1, \{\texttt{sweep}_{\texttt{p}_{21}}\}, \{\texttt{p}_{24}\}), (2, \{\texttt{mow}_{\texttt{p}_{21}}\}, \{\texttt{p}_{24}\}), (3, \{\texttt{scan}_{\texttt{p}_{21}}\}, \{\texttt{p}_{24}\})\}$. For subtask ω_1, $\ell = 1$, $\sigma_1 = \{\texttt{sweep}_{\texttt{p}_{21}}\}$ and $\sigma_1^s = \{\texttt{p}_{24}\}$, σ_1^- can be $\{\}$, which satisfies that $q_2 \notin \delta(q_1, \sigma_1^-)$, $q_2 \in \delta(q_1, \sigma_1)$. Furthermore, for $\sigma = \{\texttt{sweep}_{\texttt{p}_{21}}, \texttt{p}_{24}\}$ and $\sigma \cap \sigma_1^s \neq \varnothing$, there exists $q_1 \notin \delta(q_1, \{\texttt{sweep}_{\texttt{p}_{21}}, \texttt{p}_{24}\})$. For $\sigma = \{\texttt{sweep}_{\texttt{p}_{21}}\}$ and $\sigma \cap \sigma_1^s = \varnothing$, there exists $q_1 \in \delta(q_1, \{\texttt{sweep}_{\texttt{p}_{21}}\})$.

In other words, a subtask $\omega_\ell = (\ell, \sigma_\ell, \sigma_\ell^s)$ consists of its index, a set of action propositions labels and a set of self-loop requirement labels. The index should *not* be neglected as the same set of propositions, namely subtasks, can appear multiple times in the run. It is important to distinguish them by their indices. The labels of self loop should be satisfied before executing. Moreover, the two conditions of label σ_ℓ in the above definition are required for each label σ_ℓ of subtask ω_ℓ. Thus, every element inside σ_ℓ needs to be fulfilled for the subtask to be fulfilled. The every element in σ_ℓ^s should be forbidden before executing the σ_ℓ. Specially, the existence of self loops is not strictly required, and we denote $\sigma_\ell^s = \{Null\}$ for the case of no self loop. Note that the decomposition Ω_φ imposes directly a strict and complete ordering of the subtasks within, namely, it requires that the subtasks should be fulfilled in the exact order of their indices. This, however, can be overly restrictive as it prohibits the concurrent execution of several subtasks by multiple agents. Thus, we propose a new notion of *relaxed and partial ordering* of the decomposition, as follows.

Definition 4 (Partial Relations)**.** Given two subtasks in $\omega_h, \omega_\ell \in \Omega_\varphi$, the following two types of relations are defined:

(I) "less equal": $\preceq_\varphi \subseteq \Omega_\varphi \times \Omega_\varphi$. If $(\omega_h, \omega_\ell) \in \preceq_\varphi$ or equivalently $\omega_h \preceq_\varphi \omega_\ell$, then ω_h has to be started before ω_ℓ is started.

(II) "opposed": $\neq_\varphi \subseteq 2^{\Omega_\varphi}$. If $\{\omega_h, \ldots, \omega_\ell\} \subseteq \neq_\varphi$ or equivalently $\omega_h \neq_\varphi \cdots \neq_\varphi \omega_\ell$, then all subtask $\omega_h, \cdots$ cannot all be executed simultaneously. ■

Remark 3. Note that most related work in (Kantaros & Zavlanos, 2020; Guo & Dimarogonas, 2015; Tumova & Dimarogonas, 2016; Luo et al., 2021; Luo & Zavlanos, 2022; Sahin et al., 2019; Jones et al., 2019) treats the fulfillment of robot actions as *instantaneous*, i.e., the associated proposition becomes True once the action is finished. Moreover, most of the above approaches only take into account the *essential sequence* (Schillinger et al., 2016) associated with an accepting run, while ignoring the potential conflicts among simultaneous actions. Therefore, the above two relations are often simplified into one "less equal" relation without the "opposed" relation, see e.g., (Luo & Zavlanos, 2022). On the contrary, as described in Sec. 4.1, each action has a duration when its proposition is True during the whole period. Thus, it is essential to distinguish these two relations defined above, namely, whether one subtask should be started or finished before another subtask. ■

The above definition is illustrated in Fig. 4. Intuitively, the relation $\preceq_\varphi$ represents the ordering constraints among subtasks, while the relation $\neq_\varphi$ represents the concurrent constraints. Given these partial relations above, we can formally introduce the poset of subtasks in Ω_φ as follows.

Definition 5 (R-poset of Subtasks)**.** One *relaxed and partially* ordered set (*R-poset*) over the decomposition Ω_φ is given by

$$P_\varphi = (\Omega_\varphi,\, \preceq_\varphi,\, \neq_\varphi), \tag{6}$$

where $\preceq_\varphi, \neq_\varphi$ are the partial relations by Def. 4. ■

Similar to the original notion of poset in (Simovici & Djeraba, 2008), the above relation is irreflexive and asymmetric, however only partially transitive. In particular, it is easy to see that if $\omega_1 \preceq_\varphi \omega_2$ and $\omega_2 \preceq_\varphi \omega_3$ hold for $\omega_1, \omega_2, \omega_3 \in \Omega_\varphi$, then $\omega_1 \preceq_\varphi \omega_3$ holds. However, $\{\omega_1, \omega_2\} \subseteq \neq_\varphi$ and $\{\omega_2, \omega_3\} \subseteq \neq_\varphi$ can not imply $\{\omega_1, \omega_3\} \subseteq \neq_\varphi$. Due to similar reasons, $\{\omega_1, \omega_2, \omega_3\} \in \neq_\varphi$ can not imply $\{\omega_1, \omega_2\} \subseteq \neq_\varphi$. Clearly, given a fixed set of subtasks Ω_φ, the more elements the relations $\preceq_\varphi$ and $\neq_\varphi$ have, the more temporal constraints there are during the execution of these subtasks. This can be explained by two extreme cases: (i) no partial relations in Ω_φ, i.e., $\preceq_\varphi = \varnothing$ and $\neq_\varphi = \varnothing$. It means that the subtasks in Ω_φ can be executed in any temporal order; (ii) total relations in Ω_φ, e.g., $\omega_h \preceq_\varphi \omega_\ell$ and $\{\omega_h, \omega_\ell\} \subseteq \neq_\varphi$, for all $h < \ell$. It means that each subtask

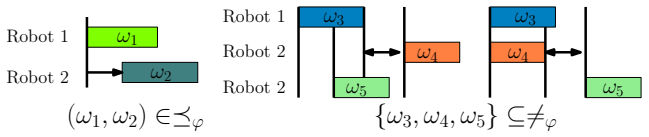

Figure 4: Illustration of the two partial relations contained in the R-poset. **Left**: $\omega_1 \preceq_\varphi \omega_2$ requires that task ω_2 is started after task ω_1. **Middle&Right**: $\{\omega_3, \omega_4, \omega_5\} \subset \neq_\varphi$ requires that $\omega_3, \omega_4, \omega_5$ are not executing simultaneously.

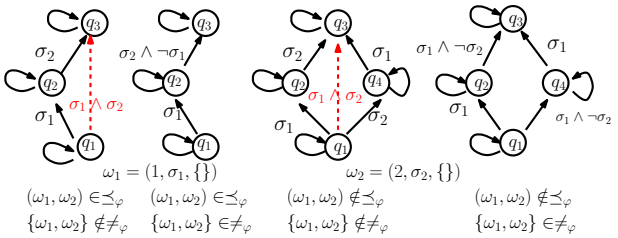

Figure 5: $\preceq_\varphi$, $\neq_\varphi$ relations in pruned NBAs, where the red dashed arrows are removed transitions.

in Ω_φ should only start after its preceding subtask finishes according to their indices in the original accepting run. For convenience, we denote by $\preceq_\varphi \triangleq \mathbb{F}$ and $\neq_\varphi \triangleq 2^{\Omega_\varphi}$ for this case, where $\mathbb{F} \triangleq \{(i, j), \forall i, j \in [0, L] \text{ and } i < j\}$. As discussed in the sequel, less temporal constraints implies more concurrent execution of the subtasks, thus higher efficiency of the overall system. Thus, it is desirable to find one decomposition and the associated R-poset that has few partial relations.

Remark 4. The above two relations, i.e., $\preceq_\varphi$ and $\neq_\varphi$, are chosen in the definition of R-posets due to following observations: as illustrated in Fig. 4 and explained in Remark 3, these two relations can describe any possible temporal relation between non-instantaneous subtasks. More importantly, they can abstract the key information contained in the structure of NBA. Specifically, we can describe the temporal orders between two subtasks ω_1, ω_2 as follows: (i) ω_2 should be executed after ω_1 has begun, i.e., $(\omega_1, \omega_2) \in \preceq_\varphi, \{\omega_1, \omega_2\} \notin \neq_\varphi$; (ii) ω_2 should be executed after ω_1 has ended, i.e., $(\omega_1, \omega_2) \in \preceq_\varphi, \{\omega_1, \omega_2\} \in \neq_\varphi$; (iii) ω_1 and ω_2 cannot be executed at the same time, i.e., $(\omega_1, \omega_2) \notin \preceq_\varphi, \{\omega_1, \omega_2\} \in \neq_\varphi$; (iv) ω_1 is parallel to ω_2, i.e., $(\omega_1, \omega_2) \notin \preceq_\varphi, \{\omega_1, \omega_2\} \notin \neq_\varphi$. These cases are summarized in Fig. 5. ■

Remark 5. It is worth noting that the proposed notion of R-Posets contains the "decomposable states" proposed in (Schillinger et al., 2018) as a *special case*. More specifically, the set of decomposable states divide an accepting run into fully independent segments, where (i) any two alphabets within the same segment are fully ordered; (ii) any two alphabets within different segments are not ordered thus independent. In contrast, the proposed R-poset allows also independent alphabets within the same segment. This subtle difference leads to more current executions not only by different segments but also within each segment, thus increases the overall efficiency. ■

To satisfy a given R-poset, the language of the underlying system is much more restricted. In particular, the language of an R-poset is defined as follows.

Definition 6 (Language of R-poset)**.** Given an R-poset $P_\varphi = (\Omega_\varphi, \preceq_\varphi, \neq_\varphi)$, its language is defined as the set of all finite words that can be generated by the subtasks in Ω_φ while satisfying the partial constraints. More concretely, the language is given by $L(P_\varphi) = \{w_\varphi\}$, where w_φ is a finite *word* constructed with the set of subtasks in P_φ, i.e.,

$$w_\varphi = (t_1, \omega_1)(t_2, \omega_2) \cdots (t_L, \omega_L), \tag{7}$$

where the subtask $\omega_\ell \in \Omega_\varphi$ and t_ℓ is the starting time of subtask ω_ℓ. Furthermore, w_φ should satisfy the partial relations in P_φ, namely: (i) $t_\ell \leqslant t_{\ell'}$ holds, $\forall(\omega_\ell, \omega_{\ell'}) \in \preceq_\varphi$; (ii) $\forall \hat{\boldsymbol{\omega}} \subseteq \neq_\varphi$, $\exists \omega_\ell, \omega_{\ell'} \in \hat{\boldsymbol{\omega}}, t_\ell + d_\ell < t_{\ell'}$ holds, where d_ℓ and $d_{\ell'}$ are the durations of subtasks $\omega_\ell, \omega_{\ell'} \in \Omega_\varphi$. With a slight abuse of notations, w_φ can also denote the simple sequence of alphabets $\omega_1 \omega_2 \cdots \omega_L$. ■

Given the above definition, a R-poset P_φ is called *accepting* if its language satisfies the original task specification, i.e., $L(P_\varphi) \subseteq L_\varphi$. In other words, instead of directly searching for the accepting word of φ, we can focus on finding the accepting R-poset that requires the least completion time. In the rest of this section, we present how this can be achieved efficiently in real-time. First of all, it is worth pointing out that it is computationally expensive to generate the *complete* set of all accepting R-poset and then select the optimal one. More precisely, even to generate *all* accepting runs in $\mathcal{B}_\varphi^-$, the worst-case computational complexity is $\mathcal{O}(|Q^-|!)$. Instead, we propose an anytime algorithm that can generate the first valid R-poset quickly, while adding more R-posets as time allows.

As summarized in Alg. 1, the proposed algorithm builds upon the modified depth first search (DFS) algorithm with local visited sets (Sedgewick, 2001). Given the pruned automaton $\mathcal{B}_\varphi^-$, the modified DFS can generate an accepting run ρ given the chosen pair of initial and final states. Given ρ, the associated set of subtasks Ω and word w can be derived by following Definition 3 , see Line 5 . Then, a R-poset P is initialized as $P = (\Omega, \mathbb{F}, 2^\Omega)$ in Line 7 , namely, a fully-ordered R-poset as described after Definition 5 . Furthermore, to reduce the partial relations, we introduce a "swapping" operation to change the order of adjacent alphabets, and then check if the resulting new word can lead to an accepting run in the circle of Line 9 - 19 . If so, it means the relative ordering of this two adjacent subtasks can potentially be relaxed or removed from $\preceq_\varphi$ as in Line 19 . On the contrary, for any other word within $L(P)$, if such swapping does not result in an accepting run, it is definitively kept in the partial ordering. The queue Que is used to store the set of accepting words w and the iteration ends when $|Que| = 0$ in Line 9, which indicates that all words in $L(P)$ have been found. The resulting R-poset is a new and valid R-poset that have less partial ordering constraints. Furthermore, for any subtasks set that belong to the "opposed" relation, a new word is generated by allowing all subtasks to be fulfilled simultaneously in Line 22 . If this new word is accepting, it means that this subtasks set do not belong to the relation $\neq_\varphi$ in Line 24 . After that, labels of self loop σ_ℓ^s is calculated by checking all feasible word in line 27 . Note that the resulting P is only one of the R-posets and the associated language is given

Algorithm 1: `compute_poset(·)`: Anytime algorithm to compute accepting R-posets.

Input : Pruned NBA $\mathcal{B}_\varphi^-$, time budget t_0.
Output: R-posets $\mathcal{P}_\varphi$, language $\mathcal{L}_\varphi$.

1 Choose initial and final states $q_0 \in Q_0^-$ and $q_f \in Q_F^-$;
2 Set $\mathcal{L}_\varphi = \mathcal{P}_\varphi = \varnothing$;
3 Begin modified DFS to find an accepting run ρ;
4 **while** $time < t_0$ **do**
`/* Subtask decomp. by Def. 3 */`
5 Compute Ω and word w given ρ;
6 **if** $w \notin \mathcal{L}_\varphi$ **then**
7 Set $P = (\Omega, \preceq_\varphi = \mathbb{F}, \neq_\varphi = 2^\Omega)$, and $\preceq_\varphi = \mathbb{F}$;
8 Set $L(P) = \{w\}$, $Que = [w]$, $I_1 = I_2 = \varnothing$;
`/* Reduce partial relations */`
9 **while** $|Que| > 0$ *and* $time > t_0$ **do**
10 $w \leftarrow Que.pop()$;
11 **for** $i = 1, 2, \cdots, |w| - 1$ **do**
12 $\omega_1 = w[i], \omega_2 = w[i+1]$;
13 $w' \leftarrow$ Switching ω_1 and ω_2 within w;
14 **if** w' *is accepting* **then**
15 Add (ω_1, ω_2) to I_1;
16 Add w' to Que, $L(P)$ if not in $L(P)$;
17 **else**
18 Add (ω_1, ω_2) to I_2;
19 Remove $\{I_1 \setminus I_2\}$ from $\preceq_\varphi$, update $time$;
20 If $time > t_0$, **return** $\mathcal{P}_\varphi, \mathcal{L}_\varphi$.
21 **for** $\hat{\boldsymbol{\omega}} \subseteq \neq_\varphi$ **do**
22 $w' \leftarrow$ Replace $\omega_i \in \hat{\boldsymbol{\omega}}$ in w by $\bigcup_{\omega_i \in \hat{\boldsymbol{\omega}}} \omega_i$;
23 **if** w' *is accepting* **then**
24 Remove $\hat{\boldsymbol{\omega}}$ from $\neq_\varphi$;
`/* Self loop calculation */`
25 **for** w *in* $L(P)$ **do**
26 Get path ρ' associated with σ_ℓ of w';
27 **for** $i = 0, 1, \cdots, |\rho'|$ **do**
28 $\sigma^p_{\ell_i} = \sigma^p_{\ell_i} \cap \delta^{-1}(\rho'[i-1], \rho'[i-1])$;
29 Add P to $\mathcal{P}_\varphi$, add $L(P)$ to $\mathcal{L}_\varphi$;
30 Continue the modified DFS, and update ρ, $time$;
31 **return** $\mathcal{P}_\varphi, \mathcal{L}_\varphi$;

by $L(P)$ as defined in Definition 6 . Lastly, as time allows, the DFS continues until a new accepting run is found, which is used to compute new R-posets with same steps.

Example 4. Continuing from example 3, we can get an R-poset with $\preceq_\varphi = \{(1,2),(1,3)\}, \neq_\varphi = \{\{2,3\}\}$. ■

Lemma 2. *Any R-poset within $\mathcal{P}_\varphi$ obtained by Alg. 1 is accepting.*

Proof. Due to the definition of accepting R-poset, it suffices to show that the language $L(P)$ derived above for reach P is accepting. To begin with, as shown in Line 16, any word w added to $L(P)$ is accepting. Secondly, assume that there exist a word $w \in L_\varphi$ that satisfies P but $w \notin L(P)$, i.e., w satisfies the partial ordering constraints in P but does not belong to $L(P)$. Regarding the ordering relation $\preceq_\varphi$, due to the iteration process of Que in Line 9-18, any accepting word w that can be generate by a sequence of switching operation will be added to $L(P)$. Assume that $w = \sigma_j \sigma_k \cdots$ where $w[1]$ is associated with the subtask ω_j. Then, $(\omega_i, \omega_j) \notin \preceq_\varphi$ holds if $i < j$ and w cannot satisfy the "less equal" constraints otherwise. Similar to the bubble sorting algorithm (Astrachan, 2003), $w_o[j]$ is switched sequentially with all the preceding terms $w_o[j-1], \cdots, w_o[0]$. The resulting words w' after each switch are accepting, because if any w' is not accepting, then the switched pairs of subtasks (ω_i, ω_j) are kept in $\preceq_\varphi$ for all $i < j$ as in Line 19. Afterwards, the resulting word is given by $w_1 = \sigma_j \sigma_1 \sigma_2 \cdots$. This operation can be applied to relocate σ_k in w_1 to the second place as $w_2 = \sigma_j \sigma_k \cdots$, and so on until $w_n = w$ holds. Thus, every word generated in the process is accepting, and the resulting w is added to $L(P)$. Respecting to each ordering relation within $\neq_\varphi$, it is simpler as any word within $L(P)$ satisfies this relation and is verified to be accepting after augmenting the alphabets with the union of all alphabets. Thus, if w satisfies the constraints of P, it will be first added to Que in Line 9-18 and then verified in Line 22, as $w \in L(P)$. This completes the proof. □

Since Alg. 1 is an anytime algorithm, its output $\mathcal{P}_\varphi$ within the time budget could be much smaller than the actual complete set of accepting R-posets. Consequently, if a word w does not satisfy any R-poset $P \in \mathcal{P}_\varphi$, i.e., $w \notin \mathcal{L}_\varphi$, it can still be accepting. Nonetheless, it is shown in the sequel for completeness analyses that given enough time, Alg. 1 can generate the complete set of R-posets. In that case, any word that does not satisfy any R-poset within $\mathcal{P}_\varphi$ is surely not accepting. It means that the complete set of R-posets $\mathcal{P}_\varphi$ is equally expressive as the original NBA $\mathcal{B}^-$. The above analysis is summarized in lemma 3.

Lemma 3. *The outputs of Alg. 1 satisfy that $L(P_i) \cap L(P_j) = \varnothing$, $\forall i \neq j$, and $L(P_i) \subset L_\varphi$, $\forall P_i \in \mathcal{P}_\varphi$. Moreover, given enough time $t_0 \to \infty$, the complete set of R-posets can be returned, i.e., $L_\varphi = \cup_{P_i \in \mathcal{P}_\varphi} L(P_i) = L_\varphi$.*

Proof. The first part can be proven by contradiction. Assume that there exists two R-posets $P_1, P_2 \in \mathcal{P}_\varphi$ and one accepting word $w \in L_\varphi$ such that $w \in L(P_1) \cap L(P_2)$ holds. Since the set of subtasks within the R-poset is simply the union of all subtasks within each word, it implies that $\Omega_1 = \Omega_2$. Then, as discussed in Lemma 2, $w \in L(P_1)$ implies that there exists a sequence of switching operations that maps the original word w_0 to w, all of which satisfy the partial relations in P_1. The same applies to $w \in L(P_2)$. Since the set of subtasks are identical, it implies that the relations in P_1 is a subset of those in P_2, or vice versa. However, since the Que in Alg. 1 iterates through all accepting words of the same Ω, the partial relations are the maximum given the same Ω, Thus both partial relations in P_1, P_2 can only be equal and $P_1 = P_2$ holds. Regarding the second part, the underlying DFS search scheme in Alg. 1 is guaranteed to exhaustively find all accepting runs of $\mathcal{B}_\varphi^-$. Namely, the complete set of R-posets $\mathcal{P}_\varphi$ returned by the algorithm after full termination is ensured to cover all accepting words of the NBA. As discussed earlier, the pruning procedure does not effect the complete set of accepting words. Thus, it can be concluded that the returned language set L_φ is equivalent to the original task specification. □

Similar to the Hasse diagram in (Simovici & Djeraba, 2008), the following graph can be constructed given one R-poset P_φ.

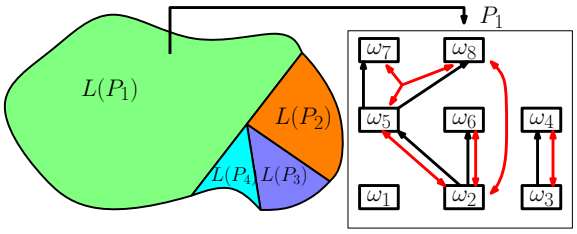

Figure 6: **Left**: an illustration of the relations between the accepting language of different R-posets $L(P_i)$ and the accepting language of the task L_φ. **Right**: an example of the R-posets graph $\mathcal{G}_{P_\varphi}$, where the relations $\preceq_\varphi$ and $\neq_\varphi$ are marked by black and red arrows, respectively.

Definition 7 (R-posets Graph)**.** The R-poset graph of $P_\varphi = (\Omega_\varphi, \preceq_\varphi, \neq_\varphi, \Omega_0)$ is a digraph $\mathcal{G}_{P_\varphi} = (\Omega, E, R)$, where Ω is the set of nodes; $E \subset \Omega \times \Omega$ is the set of directed edges; $R \subset 2^\Omega$ is the set of undirected special 'edges' which connect multiple nodes instead of only two. A edge $(\omega_1, \omega_2) \in E$ if two conditions hold: (i) $(\omega_1, \omega_2) \in \preceq_\varphi$; and (ii) there are no intermediate nodes ω_3 such that $\omega_1 \preceq_\varphi \omega_3 \preceq_\varphi \omega_2$ holds; lastly, $\Omega_0 \subseteq \Omega_\varphi$ is set of root nodes that have no incoming edges. An undirected 'edge' $(\omega_1, \omega_2, \dots) \in R$, if $\{\omega_1, \omega_2, \dots\} \in \neq_\varphi$. ■

The R-poset graph $\mathcal{G}_{P_\varphi}$ provides a straightforward representation of the partial ordering among subtasks, i.e., from low to high in the direction of edges. As shown in Fig. 6 , $\mathcal{G}_{P_\varphi}$ can be dis-connected with multiple root nodes.

5.3. Task Assignment

Given the set of R-posets $\mathcal{P}_\varphi$ derived from the previous section, this section describes how this set can be used to compute the optimal assignment of these subtasks. More specifically, we consider the following sub-problems of task assignment.

Problem 2. Given any R-poset $P = (\Omega, \preceq_\varphi, \neq_\varphi)$ where $P \in \mathcal{P}_\varphi$, find the optimal assignment of all subtasks in Ω to the multi-agent system $\mathcal{N}$ such that (i) all partial ordering requirements in $\preceq_\varphi$, $\neq_\varphi$ are respected; (ii) the maximum completion time of all subtasks is minimized. ■

To begin with, even without the constraints of partial ordering and collaborative actions, the above problem includes the multi-vehicle routing problem (Khamis et al., 2015), and the job-shop scheduling problem (Morrison et al., 2016) as special instances. Both problems are known to be NP-hard. Thus, the above problem is also NP-hard and its most straightforward solution is to formulate a Mixed Integer Linear Program (MILP). However, there are two major drawbacks of MILP: (i) the computation complexity grows exponentially with the problem size; (ii) there is often no intermediate solution before the optimal solution is generated via a MILP solver, e.g., GLPK (Makhorin, 2008). Both drawbacks hinder the usage of this approach in large-scale real-time applications, where a timely good solution is more valuable than the optimal solution.

Motivated by these observations, an anytime assignment algorithm is proposed in this work based on the Branch and Bound (BnB) search method (Morrison et al., 2016). It is not only complete and optimal, but also anytime, meaning that a good solution can be inquired within any given time budget. As shown in Fig. 7, the four typical components of a BnB algorithm are the node expansion, the branching method, and the design of the upper and lower bounds. These components for our application are described in detail below.

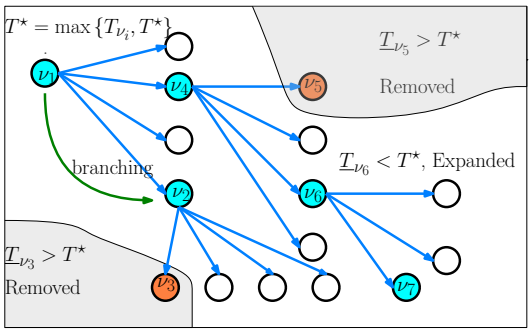

Figure 7: Illustration of the main components in the BnB search, i.e., the node expansion and branching to generate and explore new nodes (in green arrow); and the lower and upper bounding to avoid undesired branches (in orange).

Node expansion. Each node in the search tree stands for one partial assignment of the subtasks, i.e.,

$$\nu = (\tau_1, \tau_2, \cdots, \tau_N), \tag{8}$$

where τ_n is the ordered sequence of tasks assigned to agent $n \in \mathcal{N}$. To give an example, for a system of three agents, $\nu = ((\omega_1, \omega_2), (), ())$ means that two subtasks ω_1, ω_2 are assigned to agent 1, whereas no tasks to agents 2 and 3.

To avoid producing infeasible nodes, we assign only the next task that satisfies the partial order. In other words, subtask ω_i cannot be assigned to the current node ν if there exists ω_j such that $(\omega_j, \omega_i) \in \preceq_\varphi$ has not been assigned to ν. This validation process can avoid the expansion to infeasible nodes with $(\cdots \omega_i \cdots \omega_j \cdots) \in \nu'$, $(\omega_j, \omega_i) \in \preceq_\varphi$, thus improving efficiency. Let ν be the current node of the search tree. The *next* subtask ω to be assigned is chosen from the R-poset $\mathcal{G}_{P_\varphi}$ defined in Def. 7 if *all* of its parent subtasks are already assigned, i.e.,

$$\omega' \in \Omega_\nu, \ \forall \omega' \in \text{Pre}(\omega), \tag{9}$$

where $\text{Pre}(\omega)$ is the set of preceding or parenting subtasks of ω in $\mathcal{G}_{P_\varphi}$; and Ω_ν is the set of assigned subtasks as

$$\Omega_\nu = \{\omega \in \tau_n, \forall n \in \mathcal{N}\}, \ \Omega_\nu^- = \Omega \backslash \Omega_\nu, \tag{10}$$

where Ω is the set of subtasks from P_φ; Ω_ν is the set of subtasks already assigned in node ν; and Ω_ν^- are the remaining unassigned subtasks. Once this subtask ω is chosen, the succeeding or child node ν^+ of ν in the search tree is created by assigning local action to any agent or cooperative task combination of agents which has capable functions. If subtask ω is associated with a collaborative behavior C_k, an agent n may be chosen if it is capable of performing the required action $a_\ell \in \mathcal{A}_n \cap C_k$.

Branching. Given the set of nodes to be expanded, the branching method determines the order in which these child nodes are visited. Many search methods such as breadth first search (BFS), depth first search (DFS) or $A^\star$ search can be

Algorithm 2: `BnB`$(\cdot)$: Anytime BnB algorithm for task assignment

Input : Agents $\mathcal{N}$, R-poset P_φ, time budget t_0.
Output: Best assignment $J^\star$ and makespan $T^\star$.

1 Initialize root node ν_0 and queue $Q = \{(\nu_0, 0)\}$;
2 Set $T^\star = \infty$ and $J^\star = ()$;
3 **while** *(Q not empty) and ($time < t_0$)* **do**
4 Take node ν off Q;
5 $\overline{J}_\nu, \overline{T}_\nu, \overline{T}_\Omega = \texttt{upper\_bound}(\nu, P_\varphi)$
6 **if** $T^\star > \overline{T}_\nu$ **then**
7 Set $T^\star = \overline{T}_\nu$ and $J^\star = \overline{J}_\nu, T^\star_\Omega = \overline{T}_\Omega$;
8 Expand child nodes $\{\nu^+\}$ from ν ;
9 **forall** $\nu^+ \in \{\nu^+\}$ **do**
10 $\underline{T}_\nu = \texttt{lower\_bound}(\nu^+, P_\varphi)$
11 **if** $\underline{T}_\nu \leqslant T^\star$ **then**
12 Add $(\nu^+, \underline{T}_\nu)$ into Q.
13 **Return** $J^\star, T^\star, T^\star_\Omega$;

used. We propose to use $A^\star$ search here as the heuristic function matches well with the lower bounds introduced in the sequel. More specifically, the set of child nodes is expanded in the order of estimated completion time of the whole plan given its current assignment.

Lower and upper bounding. The lower bound method is designed to check whether a node fetched by *branching* has the potential to produce a better solution. The upper bound method is tried for each chosen node to update the current best solution. More specifically, given a node ν, the upper bound of all solutions rooted from this node is estimated via a greedy task assignment policy encapsulated as

$$\overline{J}_\nu, \overline{T}_\nu, \overline{T}_\Omega = \texttt{upper\_bound}(\nu, P_\varphi), \tag{11}$$

where $\overline{T}_\nu$ is the upper bound, and $\overline{J}_\nu$ is the associated complete assignment with the same structure of ν, while its Ω^- is empty; $\overline{T}_\Omega$ is the beginning time of each subtasks. From node ν, any task $\omega \in \Omega^-_\nu$ is assigned to any allowed agent set $\{n\} \subset \mathcal{N}$, thus generating a set of child nodes $\{\nu^+_{n,\omega}\}$. Then, for each node $\nu \in \{\nu^+_{n,\omega}\}$, its *concurrency* level η_ν is estimated as follows:

$$T_\nu = \max_{n \in \mathcal{N}} \{T_{\tau_n}\},\ T^{\mathrm{s}}_\nu = \sum_{\omega \in \Omega_\nu} D_\omega N_\omega,\ \eta_\nu = \frac{T^{\mathrm{s}}_\nu}{T_\nu}, \tag{12}$$

where node $\nu = (\tau_1, \cdots, \tau_N)$; T_{τ_n} is the execution time of all subtasks in τ_n by agent n; T_ν is the max current makespan; and T^{s}_ν is the total execution time of all subtasks ω given its duration D_ω and the number of participants N_ω. The makespan T_ν is calculated given the temporal constraints $\preceq_\varphi, \neq_\varphi$ of the R-Poset, the motions constraints of agents, and the collaborative subtasks. In particular, $(\omega_i, \omega_j) \in \preceq_\varphi$ requires that ω_j should start after ω_i has started; $\{\omega_i, \omega_j\} \in \neq_\varphi$ requires that ω_i, ω_j can not be executed at the same time; the dynamic constraints require that agent n needs to follow the transition cost between different regions in graph $\mathcal{G}_n$; and the collaborative constraints require that all actions of the same cooperative behavior should be executed simultaneously. We choose η_ν instead of T_ν as the index to sort child nodes, because T_ν can describe the efficiency between nodes with different assigned subtasks. Thus, the child node with the highest η_ν is chosen as the next node to expand. This procedure is repeated until no subtasks remain unassigned. Afterwards, once a complete assignment J_ν is generated, its makespan T_v and start time T_Ω are obtained by solving a linear program the same as T_ν.

Furthermore, the lower bound of the makespan of all solutions rooted from this node is estimated via two separate relaxations of the original problem: one is to consider only the partial ordering constraints while ignoring the agent capacities; another is vice versa. Details of optimization functions for these bounds can be found in (Liu et al., 2022).

Remark 6. The computation of both the upper and lower bounds are designed to be free from any integer optimization. This is intentional to avoid unpredictable solution time caused by external integer optimization solvers. ■

Given the above components, the complete BnB algorithm can be stated as in Alg. 2. In the initialization step in Line 1-2, the root node ν_0 is created as an empty assignment, the estimated optimal cost $T^\star$ is set to infinity, and the queue Q to store un-visited nodes and the lower bound as indexes contains only $(\nu_0, 0)$. Then, within the time budget, a node ν is taken from Q for expansion with smallest lower bound. We calculate the upper bound of ν in line 5 and update the optimal value $T^\star, J^\star, T^\star_\Omega$ in line 6-7. After that, we expand the child nodes $\{\nu^+\}$ of current node ν. Finally, we calculate the lower bound of each new node ν^+ in line 10 and store the node with the potential to get a better solution into Q in line 11,12. This process repeat until time elapsed or the whole search tree is exhausted.

Lemma 4. *Any task assignment $J^\star$ obtained from Alg. 2 satisfies the partial ordering constraints in P_φ.*

Proof. Since the assignment $J^\star$ belongs to the set of solutions obtained from the upper bound estimation in (11) at certain node in the search tree, it suffices to show that any solution of (11) satisfies the partial ordering constraints. Regardless of the current node ν, the set of remaining subtasks in Ω^-_ν is assigned strictly following the preceding order in the R-poset graph as defined in (9). In other words, for any pair $(\omega_1, \omega_2) \in \preceq_\varphi \cap \neq_\varphi$, if $\omega_1 \in \Omega_\nu$ and $\omega_2 \in \Omega^-_\nu$, then the starting time of ω_2 is larger than the finishing time of ω_1. Similar arguments hold for $\preceq_\varphi$ and $\neq_\varphi$ separately. □

5.4. Overall Algorithm

The complete algorithm can be obtained by combining Alg. 1 to compute R-posets and Alg. 2 to assign sub tasks in the R-posets. More specifically, as summarized in Alg. 3, the NBA associated with the given task φ is derived and pruned as described in Sec. 1. Afterwards, within the allowed time budget t_0, once the set of R-posets $\mathcal{P}_\varphi$ derived from Alg. 1 is nonempty, any R-poset $P_\varphi \in \mathcal{P}_\varphi$ is fed to the task assignment Alg. 2 to compute the current best assignment J and its

Algorithm 3: Complete algorithm for time minimization under collaborative temporal tasks

Input : Task formula φ, time budget t_0.
Output: Assignment $J^\star$, makespan $T^\star$

1 Compute $\mathcal{B}_\varphi$ given φ
2 Compute $\mathcal{B}_\varphi^-$ by pruning $\mathcal{B}_\varphi$; `// Sec. 5.1`
3 Initialize $\mathcal{J} = \varnothing$;
4 **while** *time* $< t_0$ **do**
5 $\quad P_\varphi \leftarrow$ `compute_poset`$(\mathcal{B}_\varphi^-)$; `// Alg. 1`
6 $\quad (J, T, T_\Omega) \leftarrow$ `BnB`(P_φ) ; `// Alg. 2`
7 $\quad$ Store (J, T, T_Ω) in $\mathcal{J}$;
8 Select $J^\star, T_\Omega^\star$ with minimum $T^\star$ among $\mathcal{J}$;
9 Get time list T_Ω for each task;
10 **Return** $J^\star, T^\star, T_\Omega^\star$;

makespan T, which is stored in a solution set $\mathcal{J}$. This procedure is repeated until the computation time elapsed. By then, the optimal assignment $J^\star$ and its makespan $T^\star$ are returned as the optimal solution.

Remark 7. It is worth noting that even though Alg. 1 and Alg. 2 are presented sequentially in Lines 5 - 6. They can be implemented and run in parallel, i.e., more R-posets are generated and stored in Line 5, while other R-posets are used for task assignment in Line 6. Moreover, it should be emphasized that Alg. 3 is an *anytime* algorithm meaning that it can run for any given time budget and generate the current best solution. As more time is allowed, either better solutions are found or confirmations are given that no better solutions exist. ■

Finally, once the optimal plan $J^\star$ is computed with the format defined in (8), i.e., the action sequence τ_n and is assigned to agent n with the associated time stamps $t_{\omega_1} t_{\omega_2} \cdots t_{\omega_n}$ in $T_\Omega^\star$. In other words, agent n can simply execute this sequence of subtasks at the designated time, namely ω_k at time T_{ω_k}. Then, it is ensured that all task can be fulfilled in minimum time. However, such way of execution can be prone to uncertainties in system model such as fluctuations in action duration and failures during execution, which will be discussed in the next section.

6. Online Adaptation

Since there are often uncertainties in practice, e.g., the agents may transit faster or slower due to disturbances, an action might be finished earlier or latter, or failures may occur during missions, the optimal plan derived above might be invalid during online execution. Thus, in this section, we first analyze these uncertainties in the execution time and agent failures, for which online adaptation methods are proposed.

6.1. Online Synchronization under Uncertain Execution Time

Uncertainty in the execution time can cause delayed or early termination of subtasks. Without proper synchronization, the consequences can be disastrous. For instance, one collaborative action is started without waiting for one delayed collaborator, or one subtask ω_2 is started before another subtask ω_1 is finished, which violates the ordering constraints $\omega_1 \neq_\varphi \omega_2$. To overcome these drawbacks, we propose an adaptation algorithm that relies on online synchronization and distributed communication.

More specifically, consider the optimal assignment $J^\star$ and the local sequence of subtasks for agent n: $\tau_n = \omega_n^1 \omega_n^2 \cdots \omega_n^{K_n}$. Without loss of generality, agent n just finished executing ω_n^{k-1} and is during the transition to perform subtask ω_n^k at the designated region. Additionally, the agent with the largest ID is chosen as the temporary leader of the sub-group performing the subtask ω_n^k, and this leadership lasts until the end of this subtask. No matter the transition is delayed or accelerated, the following synchronization procedure can be enforced to ensure a correct execution of the derived plan even under uncertainties:

(i) *Before* execution. In order to start executing ω_n^k, if agent n is the leader, a "`start`" synchronization message is sent by agent n to another leader agent m, for each subtask ω_m^ℓ satisfying $(\omega_n^k, \omega_m^\ell) \in \leq_\varphi$. This message indicates that the execution of ω_n^k is started thus ω_m^ℓ can be started. On the other hand, for each subtask ω_m^ℓ satisfying $(\omega_m^\ell, \omega_n^k) \in \leq_\varphi$, leader agent n waits for the "`start`" synchronization message from agent m. This message indicates that the execution of ω_m^ℓ is started thus ω_n^k can be started. Last, for each subtask ω_h^ℓ satisfying $\{\omega_n^k, \omega_h^\ell, \dots\} \subseteq \neq_\varphi$, leader agent n checks the "`start`" or "`stop`" message from agent h to ensure that not all subtasks in $\{\omega_n^k, \omega_h^\ell, \dots\}$ are executing. Moreover, the labels of self loop associated with the subsequent subtasks must be forbidden to satisfy the R-poset. Namely, each subset of agents that are executing $\omega_{\ell'}$ publishes the labels of self loop σ_ℓ^s associated with the subsequent subtask ω_ℓ. Thus, other agents would satisfy the requirements in σ_ℓ^s before $\omega_{\ell'}$ is finished.

(ii) *During* execution. If ω_n^k is a collaborative behavior, then agent n sends another synchronization message to the leader agent of this behavior to start executing this action. The leader coordinates with all collaborators to start simultaneously after relevant constraints within the R-posets are satisfied and the synchronization messages of all collaborators are received. Otherwise, if ω_n^k is a local action, then agent n starts the execution directly.

(iii) *After* execution. After the execution of ω_n^k is finished, for each subtask ω_h^ℓ satisfying $\{\omega_n^k, \omega_h^\ell, \dots\} \subseteq \neq_\varphi$, a "`stop`" synchronization message is sent by agent n to agent h, This message indicates that the execution of ω_n^k is finished thus ω_h^ℓ can be started. The above procedure is summarized in Fig. 8.

Remark 8. The synchronization protocol above is event-based, i.e., only the subtasks within the partial relations are required to synchronize, which are much less than within the whole subtasks. In comparison, the product-based solution (Baier & Katoen, 2008) and the sampling-based solution (Kantaros & Zavlanos, 2020) require full synchronization during execution, meaning that the movement of all agents should be synchronized for each transition in the global plan. Moreover, the Mixed Integer Linear Program (MILP)-based solution in (Luo & Zavlanos, 2022; Jones et al., 2019) requires no synchronization as each agent simply executes the subtasks according to the optimal time plan. The planning algorithm in (Schillinger et al.,

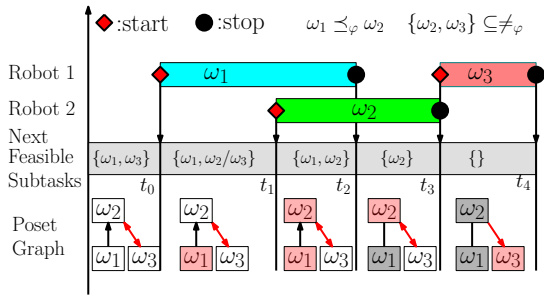

Figure 8: Illustration of the online synchronization process in Sec. 6.1. Consider the ordering $\omega_1 \preceq_\varphi \omega_2$ and $\{\omega_1, \omega_3\} \subseteq \neq_\varphi$. The "`Start`" and "`Stop`" messages are marked by red diamonds and black circles.

2018) also requires no synchronization as the local subtasks of each agent are designed to be independent, however losing optimality. As validated in the numerical experiments, the synchronization protocol offers great flexibility and robustness against fluctuation in task durations during execution, while ensuring correctness. ■

6.2. Plan Adaptation under Agent Failures

Whenever an agent cannot continue executing its remaining subtasks due to certain failure, a more complex adaptation method is required as the unfinished subtasks should be re-assigned to other agents.

Assume that agent N_d has the optimal plan $\tau_{N_d} = \omega_{N_d}^1 \omega_{N_d}^2 \cdots \omega_{N_d}^{K_{N_d}}$. It fails at time $t = T_d$ after the execution of subtask $\omega_{N_d}^{k_d-1}$ and during the transition to subtask $\omega_{N_d}^{k_d}$. Consequently, the set of unfinished tasks is given by

$$\widehat{\Omega}_{N_d} = \{\omega_{N_d}^{k'},\, k' = k_d, k_d+1, \cdots, K_{N_d}\}, \tag{13}$$

which is communicated to other agents before agent N_d failed. Note that if an agent fails during the execution of a subtask, this subtask has to be re-scheduled and thus re-executed.

Given this set of subtasks, the easiest recovery is to recruit another new agent N_d' with the same capabilities as agent N_d and takes over all tasks in $\widehat{\Omega}_{N_d}$. However, this is not always feasible, meaning that $\widehat{\Omega}_{N_d}$ needs to be assigned to other existing agents within the team. Then, the BnB algorithm in Alg. 2 is modified as follows. First, the initial root node now consists of the subtasks that are already accomplished, i.e.,

$$\nu_0' = (\tau_1', \cdots, \tau_{N_d-1}', \tau_{N_d+1}', \cdots, \tau_N'), \tag{14}$$

where $\tau_n' = \omega_n^1 \omega_n^2 \cdots \omega_n^{K_n}$ and K_n is the last subtask accomplished at time $t = T_d$, for each agent $n = 1, \cdots, N_d-1, N_d+1, \cdots, N$. Namely, agent N_d is excluded from the node definition. Second, the node expansion now re-assigns all unfinished tasks: $\widehat{\Omega}_d = \bigcup_{n\in\mathcal{N}} \widehat{\Omega}_n$, where $\widehat{\Omega}_n \subset \Omega_\varphi$ is the set of unfinished tasks for agent $n \in \mathcal{N}$, defined similarly as in (13). Namely, each remaining task is selected according to the *same* partial ordering constraints P_φ as before agent failure, for node expansion. Afterwards, the same branching rules and the methods for calculating lower and upper bounds are followed. Consequently, an adapted plan $\widehat{J}^\star$ can be obtained from the same anytime BnB algorithm. It is worth noting that the above adaptation algorithm shares the same completeness and optimality property as Alg. 3.

Last but not least, when there are multiple failed agents, the above procedure can be applied with minor modifications, e.g., the node definition excludes all failed agents.

7. Algorithmic Summary

To summarize, the proposed planning algorithm in Alg. 3 can be used offline to synthesize the complete plan that minimizes the time for accomplishing the specified collaborative temporal tasks. During execution, the proposed online synchronization scheme can be applied to overcome uncertainties in the duration of certain transition or actions. Moreover, whenever one or several agents have failures, the proposed adaptation algorithm can be followed to re-assign the remaining unfinished tasks. In the rest of this section, we present the analysis of completeness, optimality and computational complexity for Alg. 3.

Theorem 5 (Completeness)**.** *Given enough time, Alg. 3 can return the optimal assignment $J^\star$ with minimum makespan $T^\star$.*

Proof. To begin with, Lemma 2 shows that any R-poset obtained by Alg. 1 is accepting. As proven in Lemma 3, the underlying DFS search scheme finds all accepting runs of $\mathcal{B}_\varphi^-$ via exhaustive search. Thus, the complete R-posets $\mathcal{P}_\varphi$ returned by Alg. 1 after termination is ensured to cover all accepting words of the original task. Moreover, Lemma 4 shows that any assignment $J^\star$ from the BnB Alg. 2 satisfies the input R-poset from Alg. 1. Combining these two lemmas, it follows that any assignment $J^\star$ from Alg. 3 satisfies the original task formula. Second, since both Algs. 1 and 2 are exhaustive, the complete set of R-posets and the complete search tree of all possible assignments under each R-poset are searched for the optimal solution. As a result, once both sets are enumerated, the derived assignment $J^\star$ is optimal over all possible solutions. □

The computational complexity of Alg. 3 is analyzed as follows. To generate one valid R-poset in Alg. 1, the worst case time complexity is $\mathcal{O}(M^2)$, where M is the maximum number of subtasks within the given task thus bounded by the number of edges in the pruned NBA $\mathcal{B}_\varphi^-$. However, as mentioned in Sec. 3.2, the size of $\mathcal{B}$ is double exponential to the size of $|\varphi|$. The number of R-posets is upper bounded by the number of accepting runs within $\mathcal{B}_\varphi^-$, thus worst-case combinatorial to the number of nodes within $\mathcal{B}_\varphi^-$. Furthermore, regarding the BnB search algorithm, the search space in the worst case is $\mathcal{O}(M! \cdot N^M)$ as the possible sequence of all subtasks is combinatorial and the possible assignment is exponential to the number of agents. However, the worst-time complexity to compute the upper bound via Alg. 2 remains $\mathcal{O}(M \cdot N)$ as it greedily assigns the remaining subtasks, while the complexity to compute the lower bound is $\mathcal{O}(M^2)$ as it relies on a BFS over the R-poset graph $\mathcal{G}_{P_\varphi}$. How to decompose the overall formula while ensuring the satisfaction of each subformula, thus overcoming the bottleneck in the size of $\mathcal{B}_\varphi$, remains our ongoing work.

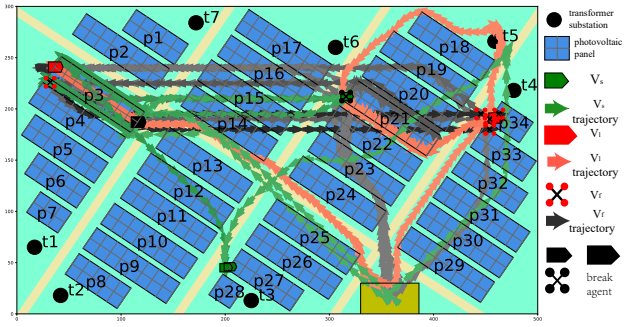

Figure 9: PV power station in the numerical simulation study, which consists of PV panels $\mathtt{p}_i$, roads, inverters/transformers $\mathtt{t}_i$ and base stations b. The arrow trajectories are the paths of the agents executing the LTL formula φ_1. The arrow direction is the motion direction and the arrow density correspond to the velocity of different agents.

Remark 9. The exponential complexity above is expected due to the NP-hardness of the considered problem. However, as emphasized earlier, the main contribution of the proposed algorithm is the anytime property. Namely, it can return the best solution within the given time budget, which is particularly useful for real-time applications where computation time is limited. ■

8. Simulations and Experiment

This section contains the numerical validation over large-scale multi-agent systems, both in simulation and actual hardware. The proposed approach is implemented in Python3 on top of Robot Operating System (ROS) to enable communication across planning, control and perception modules. All benchmarks are run on a workstation with 12-core Intel Conroe CPU. More detailed descriptions can be found in (Liu et al., 2022).

8.1. Simulations and Experiment

The numerical study simulates a team of multiple UGVs and UAVs that are responsible for maintaining a remote photovoltaic (PV) power station. We first describe the scenario and three types of tasks, followed by the results obtained via the proposed method. Then, we introduce various changes in the environment and agent failures, in order to validate the proposed online adaptation algorithm. Third, we perform scalability analysis of our method by increasing the system size and the task complexity. Lastly, we compare our methods against several strong baselines, in terms of optimality, computation time and adaptation efficiency.

8.1.1. Workspace Description

Consider a group of UAVs and UGVs working in a PV power station for long-term daily maintenance. As shown in Fig. 9, the station consists of mainly three parts: PV panels $\mathtt{p}_1, \cdots, \mathtt{p}_{34}$, roads, inverter/transformer substations $\mathtt{t}_1, \cdots, \mathtt{t}_7$ and the robot base station $\mathtt{b}$. Furthermore, there are one type of UAVs and two types of UGVs. The UAVs V_f are quadcopters which can move freely between all interested places. The larger type of UGVs, denoted by V_l, has the limitation of not going to PV panels or transformers; the smaller ones V_s can travel more freely, e.g., under the PV panels but not under the transformers. As a result, different types of robots have different motion models $\mathcal{G}_n$ as described in Sec. 4.1 and distinctive action models. The traveling time among the regions of interest is estimated by the route distance and their respective speed. Detailed descriptions of the workspace and robot model are omitted here due to limited space. Interested readers please refer to (Liu et al., 2022). Note that some actions can be performed alone while some require direct collaboration of several agents, e.g., one V_s can $sweep$ debris under the PV panel while one V_l and two V_s are required to $repair$ a broken PV panel.

8.1.2. Task Description

For the nominal scenario, we consider a system of moderate size, including 12 agents: 6 V_f, 3 V_l and 3 V_s. Scalability analyses to larger systems are performed later in Sec. 8.1.5. Moreover, we consider a complex task and test it with agent failure. This task can be specified as the following LTL formulas, requires a series of limited actions to maintain the photovoltaic power station:

$$\begin{aligned}\varphi_1 =& \Diamond(\mathtt{repair}_{\mathtt{p}_3} \wedge \neg \mathtt{scan}_{\mathtt{p}_3} \wedge \Diamond \mathtt{scan}_{\mathtt{p}_3}) \wedge \Diamond(\mathtt{wash}_{\mathtt{p}_{21}} \wedge \\ & \Diamond \mathtt{mow}_{\mathtt{p}_{21}} \wedge \Diamond \mathtt{scan}_{\mathtt{p}_{21}}) \wedge \Diamond(\mathtt{sweep}_{\mathtt{p}_{21}} \wedge \neg \mathtt{wash}_{\mathtt{p}_{21}} \wedge \\ & \Diamond \mathtt{mow}_{\mathtt{p}_{21}}) \wedge \Diamond(\mathtt{fix}_{\mathtt{t}_5} \wedge \neg \mathtt{p}_{18}) \wedge \neg \mathtt{p}_{24}\, U\, \mathtt{sweep}_{\mathtt{p}_{27}} \\ & \wedge \Diamond(\mathtt{wash}_{\mathtt{p}_{34}} \wedge \bigcirc \mathtt{scan}_{\mathtt{p}_{34}}),\end{aligned} \tag{15}$$

where the locations of these subtasks are chosen across the workspace. Thus, a strategy to minimize the completion time is crucial.

8.1.3. Results

In this section, we present the results of the proposed method, including the computation of R-posets, task assignment via the BnB search algorithm, and the task execution.

Partial analysis: The NBA $\mathcal{B}_{\varphi_1}$ associated with task φ_1 in (15) contains 707 states and 16044 edges. And the pruning step reduces 84.9% edges within 30.43 second. Then, the Alg. 1 explores 4 accepting runs in $0.14s$ to find the first R-poset and gets the best R-poset in $22.40s$. Finally, as showed in Fig. 11, we choose the best R-poset P_φ^p with 10 subtasks, whose language $L(P_\varphi^p)$ has 525 words. In P_φ^p, there are multiple subtasks that can be executed in parallel such as ω_2 with ω_4, ω_1 with ω_7. However, these subtasks are still ordered as no subtask set can be executed independent with the left subtasks. This means that we cannot divide the word into a series of independent parts using the method in (Schillinger et al., 2018). It is worth noting that this R-poset has only one subtask ω_7 with $\mathtt{mow}_{\mathtt{p}_{21}}$, which is required twice in φ_1. It follows additional partial orders as $\omega_4 \preceq_\varphi \omega_8, \omega_6 \preceq_\varphi \omega_8$. This means that our method can find a more efficient R-poset with relations not explicitly written in the formula. There are two $\neq_\varphi$ relations $(\omega_2, \omega_3), (\omega_4, \omega_6)$, due to the constrains $\mathtt{repair}_{\mathtt{p}_3} \wedge \neg \mathtt{scan}_{\mathtt{p}_3}$ and $\mathtt{sweep}_{\mathtt{p}_{21}} \wedge \neg \mathtt{wash}_{\mathtt{p}_{21}}$ in formula φ_1. Until execute ω_8, all subtasks have the labels of self loop $\neg p_{24}$ due to $\neg \mathtt{p}_{24} U \mathtt{sweep}_{\mathtt{p}_{27}}$.

Task assignment: Then, during the task assignment step, the first valid solution is found in $0.131s$. Afterwards, at $t =$

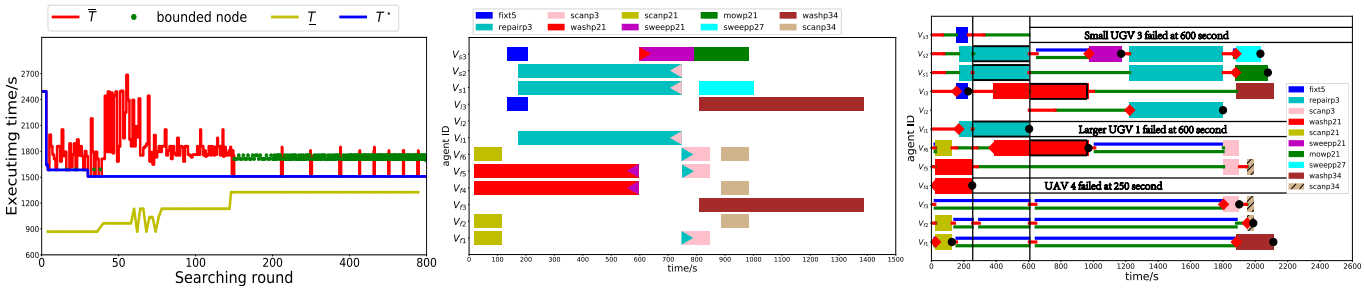

Figure 10: **Left**: Illustration of the upper and lower bounds $\overline{T}_\nu$, $\underline{T}_\nu$, and the optimal value $T^\star$, along with the BnB search process. **Middle**: Gantt graph of offline planning. **Right**: Gantt graph of the plan execution under agent failures and fluctuated subtask duration during the execution. Additional lines and labels are added to highlight the online synchronization process. Red segments denote that the agents are during transition among regions, green seqments denote "communicate with collaborate leader", and blue segments denote "leader communicate with other leader for `start` and `stop` messages". The "`start`" message is marked by the red diamond and the "`stop`" message by the black dot, both of which are published by the respective leader. Three agent failures are highlighted in black.

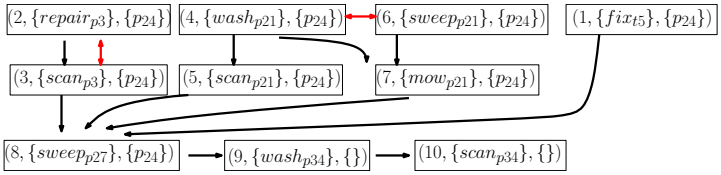

Figure 11: R-Poset graph of task φ_1, in which the negative labels of σ_ℓ are omitted here for simplicity. The relations $\preceq_\varphi$, $\neq_\varphi$ are marked by black and red arrows.

Table 2: Scalability analyses of the proposed method[1]

System (V_f, V_s, V_l)	t_{φ_1} $[s]$	t_{φ_2} $[s]$	t_{φ_3} $[s]$
$(8, 4, 4)$	$0.13, 5.9$	$0.14, 1.08$	$0.23, 4.81$
$(12, 6, 6)$	$0.15, 4.6$	$0.08, 1.85$	$0.22, 5.23$
$(16, 8, 8)$	$0.13, 5.0$	$0.10, 1.56$	$0.21, 4.33$
$(20, 10, 10)$	$0.53, 8.4$	$0.09, 2.11$	$0.58, 6.20$
R-Poset analysis	$64.4, 71.1$	$8.4, 37.9$	$136.4, 552.5$

[1] System size is given by the number of different types of agents. The associated solution time is measured by two time stamps: 1) when the first solution or poset is returned; 2) when the optimal solution or the best poset with largest language is returned.

$1.75s$, a node is reached and its estimated lower bound is larger than the current upper bound, and thus cut off from the search tree. Overall, around 84.2% of visited nodes are cut off, which clearly shows the benefits of the "bounding" mechanism. Then, the estimated upper bound rapidly converges to the optimal $T^\star = 1388.5s$ in $3.34s$ by exploring 30 nodes. This is due to the branching efficiency during the BnB search, using the estimated lower bounds as heuristics. Lastly, the whole search tree is exhausted after more than 10 hours, due to the complexity of the problem.

As shown in Fig. 10, in the optimal task assignment, for the same type of task `wash`, different types of agent V_{f3}, V_{l3} are employed for $\texttt{wash}_{\texttt{p}_{34}}$ and V_{f4}, V_{f5} for $\texttt{wash}_{\texttt{p}_{21}}$. All the constraints of $\preceq_\varphi, \neq_\varphi$ are satisfied, e.g., $\texttt{mow}_{\texttt{p}_{21}}$ should be executed after $\texttt{wash}_{\texttt{p}_{21}}$, $\texttt{sweep}_{\texttt{p}_{21}}$ and $\texttt{wash}_{\texttt{p}_{21}}$ should not be executed at the same time. These relations are denoted by triangles of the corresponding color. Moreover, most subtasks without these relations are executed in parallel, such as ω_1, ω_2. These parallelisms dramatically reduce the makespan.

8.1.4. Online Adaptation

In this subsection, we simulate two practical scenarios to validate the proposed online adaptation algorithm: (i) fluctuations in the execution time of subtasks; (ii) agent failures during online execution,

First of all, we artificially change the executing time of certain subtasks. For instance, the executing time of the maintenance tasks for smaller panels is reduced in comparison to the large panels, e.g., the execution time of $\texttt{wash}_{\texttt{p}_{34}}$ are reduced to $141s$ from $565s$, as the size of $\texttt{p}_{34}$ is 25% of $\texttt{p}_{10}$. The transfer time between different regions will be disturbed. The proposed online synchronization method in Sec. 6.1 is applied during execution to dynamically accommodate these fluctuations. As shown in Fig. 10, V_{s_1}, V_{s_2} arrive p_3 first and then begin "communicate with collaborate leader" until leader V_{l_1} arrives p_3 and returns the message "synchronization begins". After finish task $\texttt{scan}_{\texttt{p}_{21}}$, agent V_{f_2} go to the proper place quickly but cannot start $\texttt{scan}_{\texttt{p}_{34}}$ as its cooperators are not arrived and the related partial orders are also not satisfied. Thus, it turns to the states of "communicate with other leader for `start` and `stop` messages" and "communicate with collaborate leader" .

Secondly, more severe scenarios are simulated where agents break down during task execution and thus are removed from the team. More specifically, vehicle V_{f_4} breaks down at $250s$, V_{l_1}, V_{s_3} break down at $600s$ during the execution of φ_1. Consequently, as shown in Fig. 10, $\texttt{wash}_{\texttt{p}_{21}}$ is re-assigned to other agent as one of its cooperaters V_{f_4} is failed. Subtask $\texttt{repair}_{\texttt{p}_3}$ is continuing as its cooperation situation and partial relations are still satisfied. As described in Sec. 6.2, the set of unfinished tasks is re-assigned to the remaining agents by re-identifying the current node in the BnB search tree and continue the planning process. It can be seen that no subtasks are assigned to V_{f_4} anymore in the updated assignment. Then, as V_{l_1}, V_{s_3} break down at $600s$, the execution of subtask $\texttt{repair}_{\texttt{p}_3}$ is interrupted. It is executed again by vehicles V_{l_2} and V_{s_1}, V_{s_3} at $1222s$. In the same way, the unfinished subtasks are re-assigned to the remaining agents. It is worth noting that the partial ordering constraints are respected at all time during the adaptation. For instance, V_{f_1} cannot execute the subtask $\texttt{wash}_{\texttt{p}_{34}}$ before $\texttt{mow}_{\texttt{p}_{21}}$ is started, as $\texttt{mow}_{\texttt{p}_{21}} \preceq_\varphi \texttt{wash}_{\texttt{p}_{34}}$ holds. All

subtasks are fulfilled at $2109s$, despite of the above contingencies. The trajectories of the agents are showed in Fig. 9.

8.1.5. Scalability Analysis

To further validate the scalability of the proposed methods, the following tests are performed: (i) the same task with increased team sizes, e.g., 16, 24, 32 and 40; (ii) more LTL formulas with different structures.

As summarized in Table 2, as the system size is increased from 8 to 40, the computation time to obtain the *first* solution for task φ_1 remains almost unchanged, while the time taken to compute the optimal value increases slightly. This result verifies that the proposed anytime algorithm is beneficial especially for large-scale systems, as it can returns a high-quality solution fast, and close-to-optimal solutions can be returned as time permits. Secondly, more tasks φ_2, φ_3 are considered as follows:

$$\begin{aligned}\varphi_2 =& \Diamond(\mathtt{wash}_{\mathtt{p}_{11}} \wedge \neg \mathtt{scan}_{\mathtt{p}_i} \wedge \Diamond \mathtt{scan}_{\mathtt{p}_{11}} \wedge \Diamond((\mathtt{mow}_{\mathtt{p}_{11}} \\ & \wedge \neg \mathtt{wash}_{\mathtt{p}_{11}}) \wedge \Diamond(\mathtt{sweep}_{\mathtt{p}_{11}} \wedge \neg \mathtt{mow}_{\mathtt{p}_{11}}))) \wedge \\ & \Diamond(\mathtt{temp}_{\mathtt{p}_{25}} \wedge \Diamond \mathtt{repair}_{\mathtt{p}_{25}} \wedge \Diamond((\mathtt{scan}_{\mathtt{p}_{25}} \wedge \\ & \neg \mathtt{wash}_{\mathtt{p}_{25}}) \wedge \Diamond(\mathtt{sweep}_{\mathtt{p}_{25}} \wedge \neg \mathtt{p}_{26}))) \wedge \Diamond \mathtt{temp}_{\mathtt{t}_4},\end{aligned} \tag{16}$$

$$\begin{aligned}\varphi_3 =& \Diamond(\mathtt{temp}_{\mathtt{p}_{25}} \wedge \Diamond \mathtt{repair}_{\mathtt{p}_{25}} \wedge \Diamond((\mathtt{scan}_{\mathtt{p}_{25}} \wedge \neg \mathtt{wash}_{\mathtt{p}_{25}}) \\ & \wedge \Diamond(\mathtt{sweep}_{\mathtt{p}_{25}} \wedge \neg \mathtt{p}_{26}))) \wedge \Diamond \mathtt{temp}_{\mathtt{t}_4} \wedge \Diamond(\mathtt{sweep}_{\mathtt{p}_8} \\ & \wedge \Diamond \mathtt{wash}_{\mathtt{p}_8}) \wedge \Diamond \mathtt{repair}_{\mathtt{p}_4} \bigcirc \neg \mathtt{p}_5 \wedge \Diamond(\mathtt{sweep}_{\mathtt{p}_8} \\ & \wedge \neg \mathtt{wash}_{\mathtt{p}_8} \wedge \Diamond \mathtt{scan}_{\mathtt{p}_8}) \wedge \neg \mathtt{temp}_{\mathtt{t}_4} \, U \, \mathtt{fix}_{\mathtt{t}_4},\end{aligned} \tag{17}$$

where $\mathcal{B}_{\varphi_2}$ contains 216 states and $\mathcal{B}_{\varphi_3}$ contains 970 states. As summarized in Table 2 , the computation time of both R-posets and tasks assignment are increased significantly, as the task becomes more complex. However, the time when the first solution is obtained in tasks assignment does not monotonically increase due to the polynomial complexity of upper bound method.

8.1.6. Comparison

The proposed method is compared against several state-of-the-art methods in the literature. More specifically, four methods below are compared:

Prod: the standard solution (Baier & Katoen, 2008) that first computes the Cartesian products of all agent models, then computes the product Büchi automaton, and searches for the accepting run within. As the brute-force method, it is well-known to suffer from complexity explosion.

Milp: the optimization-based solution that formulates the assignment problem of R-posets as a MILP, the compute optimal assignment similar to (Luo & Zavlanos, 2022; Jones et al., 2019), i.e., instead of the search method. The partial relations are formulated as constraints in the program. An open source solver GLPK (Makhorin, 2008) is used.

Samp: the sampling based method proposed in (Kantaros & Zavlanos, 2020). Compared with the product-based methods, it does not pre-compute the complete system model. Instead, it relies on a sampling strategy to explore only relevant search space. However, since it does not support collaborative actions natively, we modify the definition of transitions there slightly.

Decomp: the task assignment strategy proposed in (Schillinger et al., 2018). As discussed earlier in Sec. 1, the proposed task decomposition strategy only allows completely independent subtasks. Furthermore, since it does not support collaborative actions, collaborative subtasks are decomposed manually.

Table 3: Comparisons to other methods[1]

Method	$t_{\mathtt{fir}}\,[s]$	$t_{\mathtt{opt}}\,[s]$	$t_{\mathtt{fin}}\,[s]$	$T_{\mathtt{obj}}\,[s]$	$N_{\mathtt{sync}}$
Prod	∞	∞	∞	–	–
	∞	∞	∞	–	–
Milp	2069.27	2069.27	2069.27	1058.47	–
	∞	∞	∞	–	–
Samp	328.59	1838.96	∞	1968.03	24
	3280.68	16294.30	∞	1968.03	24
Decomp	580.16	580.16	4581.3	1266.99	0
	1151.24	1151.24	5082.07	1267.00	0
Ours	24.81	25.26	∞	1058.47	8
	28.12	37.40	∞	1058.47	8

[1] $t_{\mathtt{fir}}$ is the time to get the first solution; $t_{\mathtt{opt}}$ the time to get the optimal solution; $t_{\mathtt{fin}}$ the time to complete the search; $T_{\mathtt{obj}}$ the makespan; and $N_{\mathtt{sync}}$ the number of synchronization during execution.

Comparisons are performed under different system sizes, namely, 12 and 24 agents, to compare not only efficiency but also scalability. To begin with, the nominal system of 12 agents under task φ_2 is considered. The above four methods are used to solve the same planning problem. The results are summarized in Table 3. Since the methods **Prod**, **Milp** and **Decomp** are not anytime, the time to obtain the first solution equals to the time when the optimal solution is obtained. It can be seen that **Prod** fails to generate any solution within $11h$ as the system-wide product automata for both cases has more than 10^{19} states. The **Milp** method is only applicable for small-scale systems, which returns the optimal solution in $0.5h$ but fails to return any solution within $16h$ for the large-scale case. The **Samp** method has the anytime property but it takes ten times more time to generate the first solution, compared with our method. In addition, since the subtasks are executed in sequence, the actual time of task completion is significantly longer. The **Decomp** method can solve both problems but the overall time for task completion is longer than our results, which matches our analyses in Remark 5. In comparison, our method returns the first solution for both cases in less than $30s$ and the optimal solution within another $10s$. It can be seen that the task completion time remains the same for both cases.

Lastly, the last column in Table 3 compares the number of synchronizations required during execution. Although the same solution is obtained, the **Milp** method requires more synchronization than our method. This is because our method requires only synchronization for relations within the R-posets, rather than all consecutive subtasks. The **Prod** and **Samp** methods require more synchronization due to their fully sequential execution, while the **Decomp** method requires no synchronization as the local subtasks of each agent are independent.

8.2. Hardware Experiment

For further validation with hardware, a similar setup is built as shown in Fig. 12. In total 4 UAVs and 2 UGVs are deployed in the workspace of of $4 \times 5\,m^2$. Each robot communicates wirelessly to the control PC via ROS, of which the state is monitored by the OptiTrack system. Different tasks and scenarios

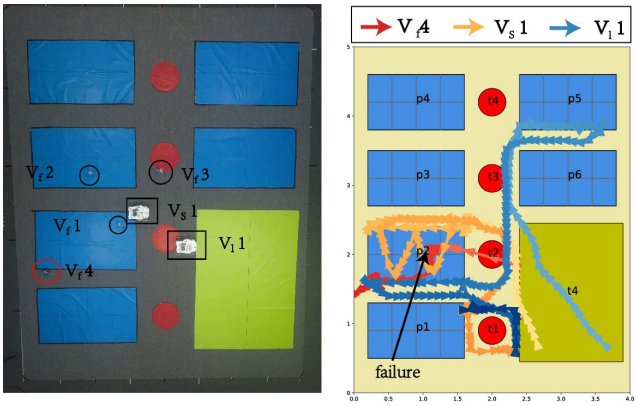

Figure 12: **Left**: Layout of the experiment setup. **Right**: Actual trajectories of each agent when UAV 4 is manually taken down at $75s$.

are designed to show how the proposed methods perform on actual hardware.

8.2.1. Workspace and Task Description

The workspace mimics the PV farm described in the numerical simulation. As showed in Fig. 12, there are 6 PV panels ($\mathtt{p}_1$-$\mathtt{p}_6$, marked in blue), 4 transformer substations ($\mathtt{t}_1$-$\mathtt{t}_4$, marked in red) and 1 base station (b_1 marked in yellow). Moreover, 4 UAVs and 2 UGVs are deployed to maintain the PV farm, where the $UAVs$ are Crazyflie mini-drones, denoted by V_f, and the $UGVs$ are cars with four mecanum wheels (denoted by V_s, V_l). Existing mature navigation controllers are used and omitted here for brevity. The routine maintenance task can be specified with the following LTL formulas:

$$\begin{aligned}\varphi_4 =& \Diamond(\mathtt{repair}_{\mathtt{p}_2} \wedge \neg \mathtt{scan}_{\mathtt{p}_2} \wedge \Diamond \mathtt{scan}_{\mathtt{p}_2} \wedge \Diamond(\mathtt{sweep}_{\mathtt{p}_2} \\ & \wedge \neg \mathtt{repair}_{\mathtt{p}_2})) \wedge \Diamond \mathtt{fix}_{\mathtt{t}_1} \wedge \Diamond \mathtt{scan}_{\mathtt{p}_3} \wedge \Diamond \mathtt{wash}_{\mathtt{p}_5},\end{aligned} \tag{18}$$

which can be understood in a similar way to φ_1 in (15).

8.2.2. Results

First, we describe the nominal scenario. Following the procedure described in Alg. 3, the LTL formula is converted to its NBA $\mathcal{B}$ with 62 nodes and 521 edges. And the pruned NBA $\mathcal{B}^-$ has 62 nodes and 377 edges. Only one R-poset is found with Alg. 1, which contains 6 subtasks whose language $L(P)$ equals to the full language $L(\mathcal{B}^-)$. Furthermore, Alg. 2 finds the optimal task assignment within $3.7s$, which has the estimated makespan of $124s$, after exploring 59 nodes. During execution, it is worth noting that due to collision avoidance and communication delays, the fluctuations in the time of navigation and task execution are *significant*. Consequently, the proposed online synchronization protocol in Sec. 6.1 plays an important role to ensure that the partial constraints are respected during execution, instead of simply following the optimal schedule. The execution of the complete task lasts $170s$ and the resulting trajectories are shown in Fig. 12. Moreover, to test the online adaptation procedure as described in Sec. 6.2, one UAV V_{f_4} is stopped manually to mimic a motor failure during execution at $75s$. During adaptation, a new node is located the BnB search tree given the set of unfinished tasks and the search is continued until a new plan is found within $0.8s$. As a result, UAV V_{f_1} takes over the subtask $\mathtt{scan}_{\mathtt{p}_2}$ to continue the overall mission. The resulting trajectory is shown in Fig. 12, where the trajectory of V_{f_4} before failure is shown in red, and the trajectory in blue is that of another UAV taking over the subtasks. In the end, the complete task is accomplished in $178s$.

9. Conclusion

In this work, a novel anytime planning algorithm has been proposed for the minimum-time task planning of multi-agent systems under complex and collaborative temporal tasks. Furthermore, an online adaptation algorithm has been proposed to tackle fluctuations in the task duration and agent failures during online execution. Its efficiency, optimality and adaptability have been validated extensively via simulations and experiments. Future work includes the distributed methods for computing the products between local and global R-posets. This can potentially alleviate the complexity bottleneck when translating a long LTL formula into its associated NBA.

References

Arai, T., Pagello, E., Parker, L. E. et al. (2002). Advances in multi-robot systems. *IEEE Transactions on Robotics and Automation*, *18*, 655–661.

Astrachan, O. (2003). Bubble sort: an archaeological algorithmic analysis. *ACM Sigcse Bulletin*, *35*, 1–5.

Baier, C., & Katoen, J.-P. (2008). *Principles of Model Checking*. MIT press.

Belta, C., Yordanov, B., & Gol, E. A. (2017). *Formal Methods for Discrete-time Dynamical Systems*. Springer.

Chen, Y., Ding, X. C., Stefanescu, A., & Belta, C. (2011). Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics*, *28*, 158–171.

Cliff, O. M., Fitch, R., Sukkarieh, S., Saunders, D. L., & Heinsohn, R. (2015). Online localization of radio-tagged wildlife with an autonomous aerial robot system. In *Robotics: Science and Systems*.

Fink, J., Hsieh, M. A., & Kumar, V. (2008). Multi-robot manipulation via caging in environments with obstacles. In *2008 IEEE International Conference on Robotics and Automation* (pp. 1471–1476).

Gastin, P., & Oddoux, D. (2001). Fast LTL to Büchi automata translation. In *Computer Aided Verification* (pp. 53–65). Springer.

Gini, M. (2017). Multi-robot allocation of tasks with temporal and ordering constraints. In *AAAI Conference on Artificial Intelligence*.

Guo, M., & Dimarogonas, D. V. (2015). Multi-agent plan reconfiguration under local LTL specifications. *The International Journal of Robotics Research*, *34*, 218–235.

Guo, M., & Dimarogonas, D. V. (2016). Task and motion coordination for heterogeneous multiagent systems with loosely coupled local tasks. *IEEE Transactions on Automation Science and Engineering*, *14*, 797–808.

Guo, M., & Zavlanos, M. M. (2018). Multirobot data gathering under buffer constraints and intermittent communication. *IEEE Transactions on Robotics*, *34*, 1082–1097.

Hochba, D. S. (1997). Approximation algorithms for NP-hard problems. *ACM Sigact News*, *28*, 40–52.

Hoos, H. H., & Stützle, T. (2004). *Stochastic Local Search: Foundations and Applications*. Elsevier.

Jones, A. M., Leahy, K., Vasile, C., Sadraddini, S., Serlin, Z., Tron, R., & Belta, C. (2019). ScRATCHS: Scalable and robust algorithms for task-based coordination from high-level specifications. In *International Symposium of Robotics Research* (pp. 1–16).

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

Khamis, A., Hussein, A., & Elmogy, A. (2015). Multi-robot task allocation: A review of the state-of-the-art. *Cooperative Robots and Sensor Networks 2015*, (pp. 31–51).

Lahijanian, M., Andersson, S. B., & Belta, C. (2011). Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Transactions on Robotics*, *28*, 396–409.

Lavaei, A., Soudjani, S., Abate, A., & Zamani, M. (2022). Automated verification and synthesis of stochastic hybrid systems: A survey. *Automatica*, *146*, 110617.

Liu, Z., Guo, M., & Li, Z. (2022). Time minimization and online synchronization for multi-agent systems under collaborative temporal logic tasks (extended version). *arXiv preprint*, (p. arXiv:2208.07756).

Luo, L., Chakraborty, N., & Sycara, K. (2015). Distributed algorithms for multirobot task assignment with task deadline constraints. *IEEE Transactions on Automation Science and Engineering*, *12*, 876–888.

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

Luo, X., & Zavlanos, M. M. (2022). Temporal logic task allocation in heterogeneous multi-robot systems. *IEEE Transactions on Robotics*, *38*, 3602–3621.

Makhorin, A. (2008). Glpk (gnu linear programming kit). *http://www. gnu. org/s/glpk/glpk*.

Morrison, D. R., Jacobson, S. H., Sauppe, J. J., & Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, *19*, 79–102.

Nunes, E., & Gini, M. (2015). Multi-robot auctions for allocation of tasks with temporal constraints. In *AAAI Conference on Artificial Intelligence*. volume 29.

Sahin, Y. E., Nilsson, P., & Ozay, N. (2019). Multirobot coordination with counting temporal logics. *IEEE Transactions on Robotics*, *36*, 1189–1206.

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

Schillinger, P., Bürger, M., & Dimarogonas, D. V. (2016). Decomposition of finite LTL specifications for efficient multi-agent planning. In *International Symposium on Distributed Autonomous Robotic Systems*.

Sedgewick, R. (2001). *Algorithms in C, part 5: Graph Algorithms*. Pearson Education.

Simovici, D. A., & Djeraba, C. (2008). *Mathematical Tools for Data Mining*. Springer.

Torreño, A., Onaindia, E., Komenda, A., & Štolba, M. (2017). Cooperative multi-agent planning: A survey. *ACM Computing Surveys*, *50*, 1–32.

Toth, P., & Vigo, D. (2002). An overview of vehicle routing problems. *The Vehicle Routing Problem*, (pp. 1–26). SIAM.

Tumova, J., & Dimarogonas, D. V. (2016). Multi-agent planning under local ltl specifications and event-based synchronization. *Automatica*, *70*, 239–248.

Ulusoy, A., Smith, S. L., Ding, X. C., Belta, C., & Rus, D. (2013). Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research*, *32*, 889–911.

Varava, A., Hang, K., Kragic, D., & Pokorny, F. T. (2017). Herding by caging: a topological approach towards guiding moving agents via mobile robots. In *Robotics: Science and Systems* (pp. 696–700).