Time Minimization and Online Synchronization for Multi-agent Systems under Collaborative Temporal 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 extremely 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 sub-tasks 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 Linear Temporal Logic (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. Analyses of its soundness, completeness and optimality as the minimal completion time are provided. It is also shown 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, an adaptation algorithm is proposed to synchronize execution status and re-assign unfinished subtasks dynamically to maintain correctness and optimality. Both algorithms are validated rigorously over large-scale systems via numerical simulations and hardware experiments, against several strong 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. And they are usually deployed for complex tasks difficult for single robot as delivery vehicles [1], poachers detector [2] , cooperation transporter [3], herdsman swarm [4]. Furthermore, to specify these complex tasks, many recent work propose to use formal languages such as Linear Temporal Logic (LTL) formulas [5], as an intuitive yet powerful way to describe both spatial and temporal requirements on the team behavior, see [6, 7, 8, 9] for examples.

Multi-agent task planning under LTL specification can be classified in the following four aspects: (i) *collaborative tasks*. In a bottom-up fashion, [10, 11, 12] assume local LTL tasks and dynamic environment, where collaborative tasks are allowed in [12]. In a top-down fashion, [7, 8, 13, 14, 15] consider team-wise tasks, but no direct collaboration among the agents; (ii) the *optimization criteria*. Most aforementioned work [7, 12, 13, 14, 15] optimizes the summed cost of all agents, while [8] evaluates a weighted balance between this cost and the task completion time. (iii) the *synchronization requirement*. Synchronization happens when two or more agents communicate regarding the starting time of next the next sub-task. The work in [7, 16, 14] requires full synchronization before each sub-task due to the product-based solution, while [8] imposes no synchronization by allowing only independent sub-tasks and thus limiting efficiency. And lastly (iv) the *solution basis*. Solutions based on solving a MILP as in [13, 14, 15] often can not guarantee a feasible solution within a time budget.

TODOLIU: add some reference in these paragraph Although many works have designed the sound and optimal solutions for Multi-agent task planning under LTL specification [17], there are still series challenges for more practical applications: (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 underlying 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 sub-

[1]This work was supported by the National Natural Science Foundation of China under grants 61973006, T2121002; and by Beijing Natural Science Foundation under grant JQ20025.

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

tasks 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. The team-wise task is given 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. TODOLIU: add referenceDue 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 itself. 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. The overall algorithm is proven to be complete and sound for the considered objective, and also shown empirically that a feasible and near-optimal solution is quickly reached. Besides, an online synchronization protocol is proposed 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, an adaptation algorithm is proposed to dynamically reassign unfinished subtasks online to maintain 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: The formal problem description is given in Sec. 3. Main components of the proposed framework are presented in Sec. 4-6. Experiment studies are shown in Sec. 7, followed by conclusions and future work in Sec. 8.

2. Preliminary

This section contains the preliminaries essential for this work, namely, Linear Temporal Logic, Büchi Automaton, and the general definition of partially ordered set,

2.1. 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 [5] 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 $\square$ (*always*), $\Diamond$ (*eventually*), $\Rightarrow$ (*implication*). The full semantics and syntax of LTL are omitted here for brevity, see e.g., [5].

An infinite word w over the alphabet 2^{AP} is defined as an infinite sequence $\omega = \sigma_1\sigma_2\cdots, \sigma_i \in 2^{AP}$. The language of φ is defined as the set of words that satisfy φ, namely, $L_\varphi = Words(\varphi) = \{\boldsymbol{\omega} \mid \omega \models \varphi\}$ and $\models$ is the satisfaction relation. However, there is a special class of LTL formula called *co-safe* formulas, 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.

2.2. Nondeterministic Büchi Automaton

Given an LTL formula φ mentioned above, the associated Nondeterministic Büchi Automaton (NBA) 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 $w = \sigma_1\sigma_2\cdots$, the resulting *run* [5] within $\mathcal{B}$ 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 $i \geqslant 0$. A run is called *accepting* if it holds that $\inf(\rho) \bigcap Q_F \neq \varnothing$, where $\inf(\rho)$ is the set of states that appear in ρ infinitely often. In general, an accepting run can be written in the prefix-suffix structure, where the prefix starts from an initial state and ends in an accepting state; and the suffix is a cyclic path that contains the same accepting state.

2.3. Partially Ordered Set

A partial order [18] 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 $x \parallel y$.

3. Problem Formulation

3.1. Collaborative Multi-agent Systems

Consider a team of N agents operating in a workspace as $W \subset \mathbb{R}^3$. 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

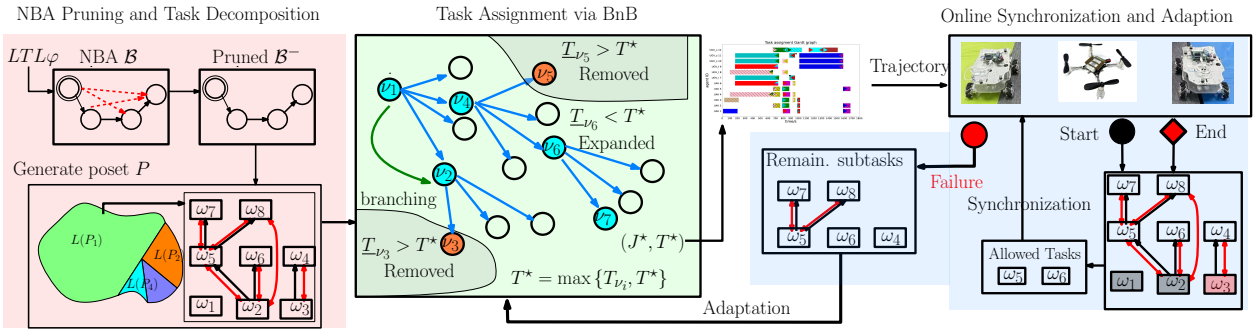

Figure 1: Overall structure of the proposed framework, which consists of three main parts: the computation of the posets, BnB search, and online execution.

agent $n \in \mathcal{N} = \{1, \cdots, N\}$ can navigate within these regions following its own transition graph, i.e., $\mathcal{G}_n = (\mathcal{W}, \rightarrow_n, d_n)$, 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 our previous work [12], 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; $a_\ell \in \mathcal{A}^c$, $\forall \ell = 1, \cdots, \ell_k$. Thus, each collaborative action has a fixed duration pre-defined by $d : \mathcal{C} \rightarrow \mathbb{R}_+$.

3.2. Task Specification

First, the following three types of atomic propositions can be introduced:

- p_m is *true* when *any* agent $n \in \mathcal{N}$ is at region $W_m \in \mathcal{W}$; Let $\mathbf{p} \triangleq \{p_m, \forall W_m \in \mathcal{W}\}$.
- 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 $\mathbf{a} \triangleq \{a_k^m, \forall W_m \in \mathcal{W}, a_k \in \mathcal{A}^l\}$;
- 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. 2.1. 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.

3.3. 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. ■

In the following sections, we present the main solution of this work. As shown in Fig. 1, the optimal plan synthesis which is performed offline is first described in Sec. 4, and then the online adaptation strategy to handle dynamic changes in Sec. 5. The overall solution is summarized in Sec. 6 with analyses for completeness, optimality and complexity.

4. Optimal Plan Synthesis

The optimal plan synthesis aims to solve Problem 1 offline. In particular, it consists of three main components: (i) the preprocessing 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.

4.1. Büchi Automaton Pruning

As the first step, the NBA $\mathcal{B}_\varphi$ associated with the task φ is derived, e.g., via translation tools [19]. Note that $\mathcal{B}_\varphi$ has the structure as defined in Def. 1, which however can be overly redundant. For instance, some transitions can be decomposed equivalently into other transitions. And removing such transitions, the size of the underlying NBA can be greatly reduced. More specifically, pruning of $\mathcal{B}_\varphi$ consists of the following 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 the accessibility capability of agents. If infeasible, such transition is removed in the pruned automaton.

(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.

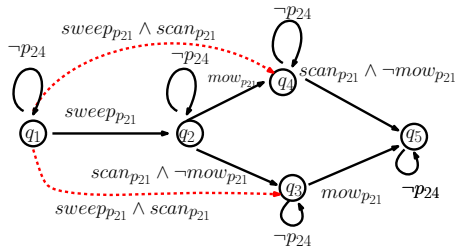

Figure 2: Example of decomposable transitions. Any transition in red dashed line are all decomposable satisfied by Def. 2.

(iii) Remove decomposable transitions. TODOLIU: Decomposable transitions are formally defined in Def. 2 below. A decomposable transition can always be satisfied as requiring *simultaneous* execute the alphabets of two sub-transitions. Thus the remove of decomposable transition will not reduce the information of NBA if we relax all the left edges can be *simultaneous* executed. As illustrated in Fig. 2, 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. ■

An example of decomposable transitions is shown in Fig. 2.. 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 ω that is accepted by $\mathcal{B}$, then an equivalent run ω' can be found that is accepted by $\mathcal{B}^-$.*

Proof. Consider an accepting word $\omega = \cdots \{\sigma_1 \wedge \sigma_2\} \cdots$ and the associated run in $\mathcal{B}$ is given by $\rho = \cdots q_1 q_3 \cdots$ of $\mathcal{B}$. Thus there exists state q_2 satisfying that $q_2 = \delta(\sigma_1, q_1)$, $q_3 = \delta(\sigma_2, q_2)$, $q_3 = \delta(\sigma_1 \wedge \sigma_2, q_1)$. Moreover, as the labels are not instantaneous, an equivalent *word* $\omega' = \cdots \{\sigma_1 \wedge \sigma_2\}\{\sigma_1 \wedge \sigma_2\} \cdots$ can be created by adding an additional time step. The associated run in $\mathcal{B}^-$ is given by $\rho' = \cdots q_1 q_2 q_3 \cdots$ where $q_2 = \delta(\sigma_1 \wedge \sigma_2, q_1)$, $q_3 = \delta(\sigma_1 \wedge \sigma_2, q_2)$. This completes the proof. □

Example 1. The NBA $\mathcal{B}_\varphi$ associated with the task formula in (13) has 707 states and 16044 edges with more than 1.29×10^7 accepting *word*, translated via [19]. After the pruning process described above, the pruned automaton $\mathcal{B}_\varphi^-$ has 707 states and 2423 edges with 174469 accepting *word*. The NBA pruning step reduces 84.9% edges. ■

4.2. Task Decomposition

TODOLIU: Task decomposition is used to find the suitable subtasks of the team-wise task. Different from simple reachability task, temporal tasks can impose strict constraints on the ordering of sub-tasks. For instance, the task $\Diamond(\omega_1 \wedge \Diamond(\omega_2 \wedge \Diamond\omega_3))$ specifies that $\omega_1, \omega_2, \omega_3$ should be satisfied in sequence, while $\Diamond\omega_1 \wedge \Diamond\omega_2 \wedge \Diamond\omega_3$ 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:

$$\Omega_\varphi = \{(\ell, \sigma_\ell, \sigma_\ell^s), \forall \ell = 1, \cdots, L\}, \tag{4}$$

where ℓ is the index of subtask σ_ℓ, and $\sigma_\ell \subseteq \Sigma$ satisfies two conditions: (i) $q_{\ell+1} \in \delta(q_{\ell-1}, \sigma_\ell)$; and (ii) $q_{\ell+1} \notin \delta(q_\ell, \sigma_\ell^-)$, where $\sigma_\ell^- = \sigma_\ell \backslash \{s\}$, $\forall s \in \sigma_\ell$, and $\sigma_\ell^s \subseteq \Sigma$ also satisfies two conditions: (i) $q_{\ell-1} \in \delta(q_{\ell-1}, \sigma_\ell^s)$; and (ii) $q_{\ell-1} \notin \delta(q_{\ell-1}, \sigma_\ell^{s-})$, where $\sigma_\ell^{s-} = \sigma_\ell^{s-} \backslash \{s\}$, $\forall s \in \sigma_\ell^{s-}$. ■

Example 2. As shown in Fig. 2, the subtasks associated with $q_1 q_2$ $q_4 q_5$ are given by $\Omega = \{(1, \texttt{sweep}_{p_{21}}, \neg p_{24}), (1, \texttt{mow}_{p_{21}}, \neg p_{24}), (1, \texttt{scan}_{p_{21}} \wedge \neg \texttt{mow}_{p_{21}}, \neg p_{24})\}$

In other words, a subtask $(\ell, \sigma_\ell, \sigma_\ell^s)$ consists of its index ,its set of position or action propositions and self-loop requirement. The index should *not* be neglected as the same set of propositions, namely sub-tasks, can appear multiple times in the run. And the self-loop should be satisfied before executing. Moreover, it also requires that each subtask σ_ℓ satisfies the segment of the task from q_ℓ to $q_{\ell+1}$. Note that the decomposition Ω_φ imposes directly a strict and complete ordering of the subtasks within, namely it requires that the subtasks 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 1. Note that most related work [7, 10, 11, 16, 13, 14, 15] treats the fulfillment of robot actions as *instantaneous*, i.e., the associated proposition becomes True once the action is finished. Thus, the probability of two actions are fulfilled at the exact same time instant is of measure zero. The above two relations can be simplified into one "less than" relation as [13]. On the contrary, for the action model described in Sec. 3.1, each action has a duration that the associated proposition is True during the *whole* period. ■

The above definition is illustrated in Fig. 3. 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 (Poset of Subtasks)**.** One partially ordered set (*poset*) over the decomposition Ω_φ is given by

$$P_\varphi = (\Omega_\varphi, \preceq_\varphi, \neq_\varphi), \tag{5}$$

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

Similar to the original notion of poset in [18], 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 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 poset that has few partial relations.

Remark 2. The above two relations, i.e., $\preceq_\varphi$ and $\neq_\varphi$, are chosen in the definition of posets due to following observations: as illustrated in Fig. 3 and explained in Remark 1, 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. In particular, given any two transitions in an accepting run, their temporal constraint can be expressed via the $\preceq_\varphi$. Secondly, within each transition, there are often subtasks in the "negated" set, meaning that they can not be executed concurrently, which can be expressed via the $\neq_\varphi$ relation. ■

Remark 3. It is worth noting that the proposed notion of posets contains the "decomposable states" proposed in [8] 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 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 poset, the language of the underlying system is much more restricted. In particular, the language of a poset is defined as follows.

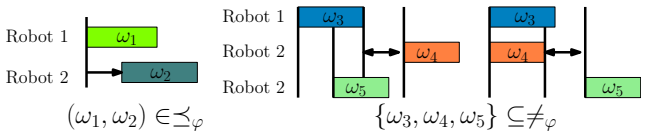

Figure 3: Illustration of the two partial relations contained in the poset. **Left**: $\omega_1 \preceq_\varphi \omega_2$ requires that task ω_2 is started after task ω_1. **Middle&Right**: $\{\omega_3, \omega_4, \omega_5\} \subset \neq_\varphi$ requires that there is no time when $\omega_3, \omega_4, \omega_5$ are all executing.

Definition 6 (Language of Poset)**.** Given a 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 $L_\varphi = \{W_\varphi\}$, where W_φ is a finite *word* constructed with the set of subtasks in P_φ, i.e.,

$$W_\varphi = (t_1, \omega_1)(t_2, \omega_2) \cdots (t_L, \omega_L), \tag{6}$$

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) $\forall\{\omega_i\} \subseteq \neq_\varphi, \exists \omega_\ell, \omega_{\ell'} \in \{\omega_i\}, t_\ell + d_\ell < t_{\ell'}$ holds, where d_ℓ and $d_{\ell'}$ are the durations of subtasks $\omega_\ell, \omega_{\ell'} \in \Omega_\varphi$. With a slight abuse of notation, W_φ can also denote the simple sequence of alphabets $\omega_1 \omega_2 \cdots \omega_L$. ■

Given the above definition, a poset P_φ is called *accepting* if its language satisfies the original task specification, i.e., $\mathcal{L}(P_\varphi) \subseteq \mathcal{L}(\varphi)$. In other words, instead of directly searching for the accepting word of φ, we can focus on finding the accepting 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 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 at least one valid poset within any given time budget, while incrementally adding more 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 [20]. 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 poset P is initialized as $P = (\Omega, \mathbb{F}, 2^\Omega)$ in Line 8, namely, a fully-ordered 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. If so, it means the relative ordering of this two adjacent subtasks can potentially be relaxed or removed from $\preceq_\varphi$ as in Line 20. 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 resulting poset is a new and valid poset that have less partial ordering

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

Input : Pruned NBA $\mathcal{B}_\varphi^-$, time budget t_0.
Output: 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 $P = (\Omega, \preceq_\varphi, \neq_\varphi)$, and $\preceq_\varphi = \mathbb{F}$;
8 Set $L(P) = Que = \{W\}$ and $I_1 = I_2 = \varnothing$;
9 **while** $|Que| > 0$ **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 if not in Que;
17 Add W' to $L(P)$ if not in $L(P)$;
18 **else**
19 Add (ω_1, ω_2) to I_2;
20 Remove $\{I_1 \setminus I_2\}$ from $\preceq_\varphi$;
21 **for** $\{\omega_i\} \subseteq \neq_\varphi$ **do**
22 $W' \leftarrow$ Replace $\omega_i \in \{\omega_i\}$ in W by $\bigcup \omega_i$;
23 **if** W' *is accepting* **then**
24 Remove $\{\omega_i\}$ from $\neq_\varphi$;
25 **for** W *in* $L(P)$ **do**
26 Get path ρ' by σ_ℓ of W';
27 **for** $i = 0, 1, \cdots, |\rho'|$ **do**
28 $\sigma^p_{\ell_1} = \sigma^p_{\ell_1} \cap \delta^{-1}(\rho'[i-1], \rho'[i-1])$
29 Add P to $\mathcal{P}_\varphi$, add $L(P)$ to $\mathcal{L}_\varphi$;
30 Continue the modified DFS, and update ρ;
31 **return** $\mathcal{P}_\varphi$, $\mathcal{L}_\varphi$;

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. After that, self-loop σ_ℓ^s is calculated by checking all feasible word in line 27. If this new word is accepting, it means that this subtasks set do not belong to the relation $\neq_\varphi$ in Line 24. Note that the resulting P is only one of the posets and the associated language is given 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 posets following the same procedure.

Example 3. Continuing from example 2, we can get a poset with $\leqslant_\varphi = \{(1,2),(1,3)\}, \neq_\varphi = \{(2,3)\}$.

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

Proof. Due to the definition of accepting poset, it suffices to show that the language $L(P)$ derived above for reach P is accepting. To begin with, as shown in Line 17, any word W added to $L(P)$ is accepting. Second, assume that there exists a word $W \in L_\varphi$ 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 9-19, any accepting word W that satisfies the ordering constraints will be added to $L(P)$. In other words, starting from the initial word W_0 associated with the run ρ, there always exists a sequence of switching operation in Line 13 that results in the new word W. With respect to each ordering relation within $\neq_\varphi$, it is even 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 $W \in L_\varphi$, it will be first added to Que in Line 9-19 and then verified in Line 22, thus $W \in L(P)$. This completes the proof. □

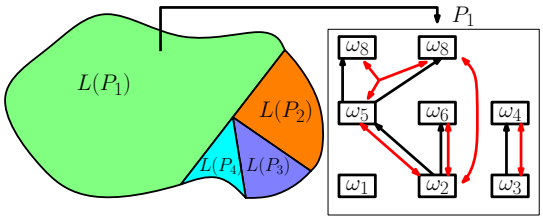

Figure 4: **Left**: an illustration of the relation between the accepting language of different posets $L(P_i)$ and the accepting language of the task $L(\varphi)$. **Right**: an example of the poset graph $\mathcal{G}_{P_\varphi}$, where the relations $\preceq_\varphi$ and $\neq_\varphi$ are marked by black and red arrows, respectively.

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

Last but not least, similar to the Hasse diagram [18], the following graph can be constructed given one poset P_φ.

Definition 7 (Poset Graph)**.** The poset graph of $P_\varphi = (\Omega, \preceq_\varphi, \neq_\varphi, \Omega_0)$ 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, $\Omega_0 \subseteq \Omega$ is set of root nodes that do not have incoming edges. An undirected 'edge' $(\omega_1, \omega_2, \dots) \in R$, if $\{\omega_1, \omega_2, \dots\} \in \neq_\varphi$. ■

The 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. Note that $\mathcal{G}_{P_\varphi}$ can be disconnected with multiple root nodes. An example of a poset graph is shown in Fig. 4.

4.3. Task Assignment

Given the set of 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:

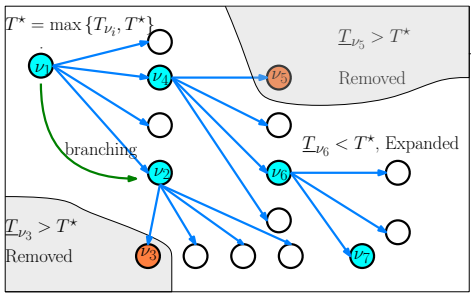

Figure 5: Illustration of the main components in the BnB search, i.e., the node expansion and branching to generate explore new nodes (in green arrow); and the lower and upper bounding to avoid undesired branches (in orange).

Problem 2. Given any 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, the above problem is an NP-hard as it includes the multi-vehicle routing problem [21, 22], and the job-shop scheduling problem [23] as special instances which are well-known to be NP-hard. Here we proposed an anytime assignment algorithm based on the Branch and Bound (BnB) search method [24, 25]. 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. 5, 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.

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

$$\nu = (\tau_1, \tau_2, \cdots, \tau_N), \tag{7}$$

where τ_n is the ordered sequence of tasks assigned to agent $n \in \mathcal{N}$. To avoid producing infeasible nodes, we assign only the next task satisfies the partial order. Let ν be the current node, Ω_ν be the assigned task and Ω_ν^- be the unassigned task. The *next* subtask ω' to be assigned is chosen from the poset $\mathcal{G}_{P_\varphi}$ defined in Def. 7:

$$\omega' \in \Omega_\nu, \ \forall \omega' \in \text{Pre}(\omega), \tag{8}$$

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

Branching. Given the set of nodes to be expanded, the branching method determines the order in which these child nodes are visited. We propose to use $A^\star$ search here as the heuristic function matches well with the lower bounds introduced in the sequel.

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. And 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

Algorithm 2: `upper_bound`$(\cdot)$: Compute the upper bound of solutions rooted from a node

Input : Poset P_φ, node ν.
Output: Assignment $\overline{J}_\nu$, upper bound $\overline{T}_\nu$.
1 **while** $\Omega_\nu^- \neq \varnothing$ **do** // (**??**)
2 **forall** $\omega \in \Omega_\nu^-$ *and* $\{n\} \subset \mathcal{N}$ *satisfy* ω **do**
3 Compute and save $\nu_{n,\omega}$ by assigning ω to agent set $\{n\}$
4 Select $\nu_\star^+ = \textbf{argmax}_{\nu \in \{\nu_{n,\omega}^+\}} \{\eta_\nu\}$; // (10)
5 $\nu \leftarrow \nu_\star^+$;
6 Compute and makespan T_ν and subtask start time T_Ω of assignment J_ν
7 **Return** $\overline{J}_\nu = J_\nu$, $\overline{T}_\nu = T_\nu$,$\overline{T}_\Omega = T_\Omega$;

of all solutions rooted from this node is estimated via a greedy task assignment policy, as summarized in Alg. 2:

$$\overline{J}_\nu, \overline{T}_\nu, \overline{T}_\Omega = \texttt{upper\_bound}(\nu, P_\varphi), \tag{9}$$

where $\overline{T}_\nu$ is 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 $\{n\} \subset \mathcal{N}$ in Line 2, 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:

$$T_\nu = \max_{n \in \mathcal{N}} \{T_{\tau_n}\}, \ T_\nu^{\text{s}} = \sum_{\omega \in \Omega_\nu} D_\omega N_\omega, \ \eta_\nu = \frac{T_\nu^{\text{s}}}{T_\nu}, \tag{10}$$

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 calculate by (10); and T_ν^{s} is the total execution time of all subtasks ω given its duration D_ω and the number of participants N_ω. Thus, the child node with the highest η_ν is chosen as the next node to expand in Line 4-5. 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 of subtasks is obtained by solving a simple linear program as in Line 6.

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. The details of optimization functions for the upper(lower) bound can be found in the supplementary material.

TODOLIU: Given the above components, the complete BnB algorithm can be stated as in Alg. 3. 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 to store un-visited nodes Q 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 $T^\star, J^\star, T_\Omega^\star$ 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 only

Algorithm 3: `BnB`$(\cdot)$: Anytime BnB algorithm for task assignment

Input : Agents $\mathcal{N}$, 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 = {(ν_0, 0)};
2  Set T* = ∞ and J* = ();
3  while (Q not empty) and (time < t_0) do
4      Take node ν off Q;
5      J̄_ν, T̄_ν, T̄_Ω = upper_bound(ν, P_φ) ;        // Alg. 2
6      if T* > T̄_ν then
7          Set T* = T̄_ν and J* = J̄_ν, T*_Ω = T̄_Ω;
8      Expand child nodes {ν+} from ν ;
9      forall ν+ ∈ {ν+} do
10         T_ν = lower_bound(ν+, P_φ)
11         if T_ν ⩽ T* then
12             Add (ν+, T_ν) into Q.
13 Return J*, T*, T*_Ω;
```

Algorithm 4: 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 B_φ given φ ;                          // [19]
2  Compute B_φ^- by pruning B_φ ;                 // Sec. 4.1
3  Initialize J = ∅;
4  while time < t_0 do
5      P_φ ← compute_poset(B_φ^-) ;               // Alg. 1
6      (J, T, T_Ω) ← BnB(P_φ) ;                   // Alg. 3
7      Store (J, T, T_Ω) in J;
8  Select J*, T*_Ω with minimum T* among J;
9  Get time list T_Ω for each task;
10 Return J*, T*, T*_Ω;
```

store the node with potential to get a better solution into Q in line 11,12. This process repeat itself until time elapsed or the whole search tree is exhausted.

4.4. Overall Algorithm

The complete algorithm can be obtained by combining Alg. 1 to compute posets and Alg. 3 to assign sub tasks in the posets. More specifically, as summarized in Alg. 4, 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 posets $\mathcal{P}_\varphi$ derived from Alg. 1 is nonempty, any poset $P_\varphi \in \mathcal{P}_\varphi$ is fed to the task assignment Alg. 3 to compute the current best assignment J and its 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 4. It is worth noting that even though Alg. 1 and Alg. 3 are presented sequentially in Line 5 - 6. They can be implemented and run in parallel, i.e., more posets are generated and stored in Line 5, while other posets are used for task assignment in Line 6. Moreover, it should be emphasized that Alg. 4 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 (7), 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 $T^\star_\Omega$. 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 the overall task can be fulfilled in minimum time. However, such a way of execution can be prone to uncertainties in the system model such as fluctuations in action duration and failures during execution, which will be discussed in the next section.

5. Online Adaptation

Since there are often uncertainties in the model, e.g., the agents may transit faster or slower due to disturbances due to the additional constrains of self-loop σ^s_ℓ, an action might be finished earlier or latter, or failures may occur during mission, 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.

5.1. Online Synchronization under Uncertain Execution Time

Uncertainty in the execution time can cause delay or early termination of subtasks. Without proper synchronization, the consequences can be disastrous such as one collaborative action is started without waiting for one delayed collaborator. To overcome these potential 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. No matter how much 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, a "`start`" synchronization message is sent by agent n to agent m, for each subtask ω_m^ℓ satisfying $(\omega_n^k, \omega_m^\ell) \in \preceq_\varphi$. 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 $(\omega_m^\ell, \omega_n^k) \in \preceq_\varphi$, 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 $\{\omega_n^k, \omega_h^\ell, \dots\} \subseteq \neq_\varphi$, agent n checks the "`start`" or "`stop`" synchronization message from agent h to ensure that not all subtasks in $\{\omega_n^k, \omega_h^\ell, \dots\}$ is executing. Moreover, the self-loop constrain associated with the subsequent subtasks must be followed to satisfy the poset. Namely, each subset of agents that are executing $\omega_{\ell'}$ publishes the self-loop requirement σ^s_ℓ associated with the subsequent subtask ω_ℓ. It means

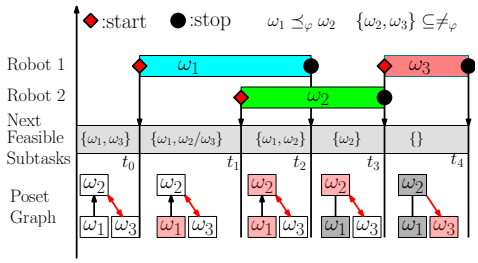

Figure 6: Illustration of the online synchronization process described in Sec. 5.1. Consider the constraints $\omega_1 \preceq_\varphi \omega_2$ and $\{\omega_1, \omega_3\} \subseteq \neq_\varphi$. The "`Start`" and "`Stop`" messages are marked by red diamonds and black circles, respectively. The agent can only execute the subtask if all relevant partial relation is satisfied.

that all 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 action, then agent n sends another synchronization message to each collaborative agent to start executing this action. 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. 6. The synchronization for collaborative tasks is easier as a collaborative subtask can only start once all participants are present at the location.

Remark 5. 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 the complete set of subtasks. ■

5.2. Plan Adaptation under Agent Failures

Whenever an agent is experiencing motor failures that it can not continue executing its remaining subtasks, 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:

$$\widehat{\Omega}_{N_d} = \{\omega_{N_d}^{k'},\, k' = k_d, k_d+1, \cdots, K_{N_d}\}, \tag{11}$$

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. 3 is modified as follows. First, the initial root node now consists of the subtasks that are already accomplished by each functional agent, i.e.,

$$\nu_0' = (\tau_1', \cdots, \tau_{N_d-1}', \tau_{N_d+1}', \cdots, \tau_N'), \tag{12}$$

where $\tau_n' = \omega_n^1 \omega_n^2 \cdots \omega_n^{K_n}$ and K_n is last subtask be 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 (11). 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 more importantly, 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. 4.

6. Algorithmic Summary

To summarize, the proposed planning algorithm in Alg. 4 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 first present the analysis of completeness, optimality and computational complexity for Alg. 4.

Theorem 3 (Completeness)**.** *Given enough time, Alg. 4 can return the optimal assignment $J^\star$ with minimum makespan $T^\star$.*

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

Second, the computational complexity of Alg. 4 is analyzed as follows. To generate one valid 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. 2.2, the size of $\mathcal{B}$ is double exponential to the size of $|\varphi|$. The number of 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 is in the worst case $\mathcal{O}(M! \cdot N^M)$ 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 poset graph $\mathcal{G}_{P_\varphi}$. How to decompose the length of an overall formula while ensuring the satisfaction of each subformula, thus overcoming the bottleneck in the size of $\mathcal{B}_\varphi$, remains a part of our ongoing work.

Remark 6. The exponential complexity above is expected due to the NP-hardness of the considered problem. However, as emphasized previously, the main contribution of the proposed algorithm is the anytime property. In other words, it can return the best solution within the given time budget, which is particularly useful for real-time applications where computation time is limited. ■

7. Simulations and Experiment

This section contains the numerical validation over large-scale multi-agent systems, both in simulation and on 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 and experiment videos can be found in the supplementary file.

7.1. 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 task, 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.

7.1.1. Workspace Description

Consider a group of UAVs and UGVs that work within a PV power station for long-term daily maintenance. As shown in Fig. 7, 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 V_F and two types of UGVs V_s, V_l as table 1 showed. The UAVs V_f are quadcopters which can move freely between

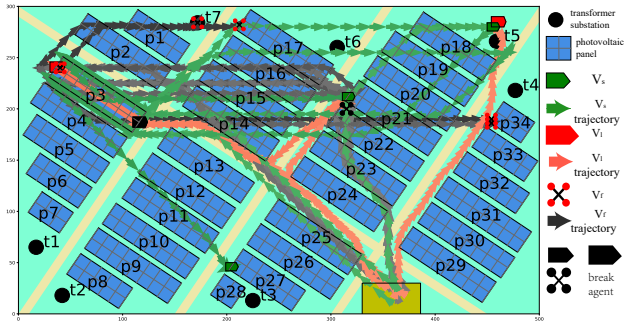

Figure 7: Simulated PV power station in the numerical study, which consists of PV panels $\mathtt{p}_i$, roads, inverters/transformers $\mathtt{t}_i$ and base stations b. The arrow trajectories are the path of Agent swarm executing the LTL formula φ_1. And the arrow direction is the motion direction and the arrow density correspond to the velocity of different agents.

Table 1: definition of agent with function

Agent type	**label**	**Capable action**	**Speed**(m/s)
Quadcopter	V_f	$temp, scan, wash$	10
Larger UGV	V_l	$wash, repair_l, fix$	4
Smaller UGV	V_s	$sweep, mow, repair_s, fix$	4

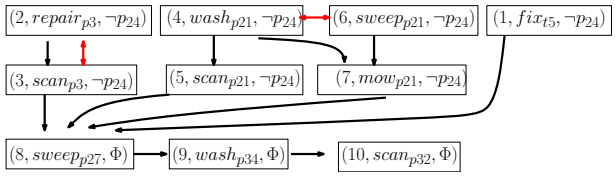

Figure 8: Poset graph of task φ_1. The relations $\preceq_\varphi$, $\neq_\varphi$ are marked by black and red arrows, respectively.

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. The actions required and the Descriptions of these regions of interest and robot actions are summarized in Table. 2.

7.1.2. 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 analysis to larger systems are performed later in Sec. 7.1.5. 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:

$$\begin{aligned}\varphi_1 = &\Diamond(\mathtt{repair}_{\mathtt{p}_3} \wedge \neg \mathtt{scan}_{\mathtt{p}_3} \wedge \Diamond \mathtt{scan}_{\mathtt{p}_3}) \wedge \Diamond(\mathtt{wash}_{\mathtt{p}_{21}} \wedge \Diamond \mathtt{mow}_{\mathtt{p}_{21}} \wedge \\ &\Diamond \mathtt{scan}_{\mathtt{p}_{21}}) \wedge \Diamond(\mathtt{sweep}_{\mathtt{p}_{21}} \wedge \neg \mathtt{wash}_{\mathtt{p}_{21}} \wedge \Diamond \mathtt{mow}_{\mathtt{p}_{21}}) \wedge \Diamond(\mathtt{fix}_{\mathtt{t}_5} \wedge \\ &\neg \mathtt{p}_{18}) \wedge \neg \mathtt{p}_{24}\, U\, \mathtt{sweep}_{\mathtt{p}_{27}} \wedge \Diamond(\mathtt{wash}_{\mathtt{p}_{34}} \wedge \bigcirc \mathtt{scan}_{\mathtt{p}_{34}})\end{aligned} \tag{13}$$

The location of these subtasks are chosen across the workspace thus a coordination strategy to minimize completion time is crucial for the task.

Table 2: Description of related regions and agent actions.

Proposition	Description	Duration [s]
$\mathtt{p}_1, \cdots, \mathtt{p}_{34}$	34 PV panels.	\
b	Base stations for all agents to park and charge.	\
$\mathtt{t}_1, \cdots, \mathtt{t}_7$	7 transformers.	\
$\mathtt{temp}_{\mathtt{p}_i,\mathtt{t}_i}$	Measure temperature of panel $\mathtt{p}_i$ and transformer $\mathtt{t}_i$. Requires 1 *temp* action.	10
$\mathtt{sweep}_{\mathtt{p}_i}$	Sweep debris around any panel $\mathtt{p}_i$. Requires 1 *sweep* action.	190
$\mathtt{mow}_{\mathtt{p}_i,\mathtt{t}_i}$	Mow the grass under panel $\mathtt{p}_i$ or transformer $\mathtt{t}_i$. Requires 1 *mow* action.	190
$\mathtt{fix}_{\mathtt{t}_i}$	Fix malfunctional transformer $\mathtt{t}_i$. Requires 2 *fix* action collaborations	72
$\mathtt{repair}_{\mathtt{p}_i}$	Repair broken panel $\mathtt{p}_i$. Requires 2 $repair_s$,1 $repair_l$ action collaborations .	576
$\mathtt{wash}_{\mathtt{p}_i}$	Wash the dirt off panel $\mathtt{p}_i$. Requires 2 *wash* actions collaborations.	565
$\mathtt{scan}_{\mathtt{p}_i,\mathtt{t}_i}$	Build 3D models of panel $\mathtt{p}_i$ or transformer $\mathtt{t}_i$ for inspection. Requires 3 *scan* action.	95

7.1.3. Results

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

Partial analysis: The NBA $\mathcal{B}_{\varphi_1}$ associated with task one in (13) contains 707 states and 16044 edges. And the pruning step reduced 84.9% edges within 30.43 second. Then, the Alg. 1 explores 4 accepting runs in $0.14s$ to find the first poset and get the one of the best poset in $22.40s$. Finally showed in 8, we choose the best poset P_φ^p with 10 subtasks, whose language $L(P_\varphi^p)$ has 525 *Words*. In P_φ^p, there are multiple subtasks 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. That means we cannot divide the word into series independent parts with the method in [8]. It's worth noting this poset has only one subtask ω_7 with $\mathtt{mow}_{\mathtt{p}_{21}}$, which is required in twice in φ_1. And it follows additions partial orders as $\omega_4 \leqslant_\varphi \omega_6, \omega_2 \leqslant_\varphi \omega_6$. That means our method found a more efficient poset with relations not explicitly written in the formula. There are two $\neq_\varphi$ relations as $(\omega_1, \omega_3), (\omega_2, \omega_4)$, due to the constrains $\square(\mathtt{repair}_{\mathtt{p}_3} \rightarrow \neg\mathtt{scan}_{\mathtt{p}_3})$ and $\mathtt{sweep}_{\mathtt{p}_{21}} \wedge \neg\mathtt{wash}_{\mathtt{p}_{21}}$ in formula φ_1. Before execute ω_8, all subtasks has the self-loop constrains $\neg p_{24}$ due to $\neg \mathtt{p} U \mathtt{sweep}_{\mathtt{p}_{27}}$.

Task assignment Then, during the task assignment step, the first valid solution is found in $0.131s$. Afterwards, at $t = 1.75s$, a node is reached and its estimated lower bound is larger than current upper bound, and thus cut off 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 converged to the optimal $T^\star = 1388.5s$ in $3.34s$ with exploring 30 nodes. This is due to the branching efficiency during the BnB search, by 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.

In the optimal task assignment, for the same type of task wash, different type of agent are employed as V_{f3}, V_{l9} for $\mathtt{wash}_{\mathtt{p}_{21}}$ and V_{f4}, V_{f5} for $\mathtt{wash}_{\mathtt{p}_{34}}$. And all the constrains of $\leqslant_\varphi, \neq_\varphi$ are satisfied such as $\mathtt{mow}_{\mathtt{p}_{21}}$ should be executed after $\mathtt{repair}_{\mathtt{p}_3}$, $\mathtt{sweep}_{\mathtt{p}_2 1}$ should not be execute at same time which denote by triangles of the corresponding color. What's more, most subtasks without these relations are executed parallel as ω_1, ω_2. These parallelisms dramatically reduce the make span. In the trajectory graph 7, the self-loop constrains of each subtasks are all satisfied as before executing ω_8, no agent is permitted enter the PV panels P_{24}, while V_f, V_s are allowed crossing other PV panels.

7.1.4. Online Adaptation

Here we simulate the scenario with fluctuations in the execution time of subtasks while several agents break down during the online execution.

Firstly, the executing time of the maintenance tasks for smaller panels is reduced in comparison to the large panels, e.g., the execution time of $\mathtt{wash}_{\mathtt{p}_{34}}$ are reduced to $141s$ from $565s$, as the area of $\mathtt{p}_{34}$ is 25% of $\mathtt{p}_{10}$. And the transfer time between different regions will be disturbed. The proposed online synchronization method in Sec. 5.1 is applied during execution to dynamically accommodate these fluctuations. As the fig 9 showed, V_{s_1}, V_{s_2} arrived p_3 first and then began "waiting for collaborators" until $V_l 1$ arrive p_3 and start to executing $\mathtt{repair}_{\mathtt{p}_3}$. And after finished task $\mathtt{scan}_{\mathtt{p}_{21}}$, agent V_{f_2}, V_{f_6} go to proper place quickly but cannot start $\mathtt{scan}_{\mathtt{p}_3}$ until $\mathtt{repair}_{\mathtt{p}_3}$ finished. Thus, they turn to the state of "waiting for preceding subtasks in partial order".

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_3} breaks down at $200s$, V_{l_1}, V_{s_3} break down at $600s$ during the execution of φ_1. Consequently, as shown in Fig. 9, $\mathtt{wash}_{\mathtt{p}_{21}}$ is re-assigned to other agent as one of its cooperaters V_{f_3} is failed. And subtask $\mathtt{repair}_{\mathtt{p}_3}$ is continuing as its cooperate situation and partial relations are still satisfied. As described in Sec. 5.2, 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_3} anymore in the updated assignment. Then, as V_{l_1}, V_{s_3} break down at $600s$, the execution of subtask $\mathtt{repair}_{\mathtt{p}_3}$ is interrupted. And it is executed again by vehicles V_{l_2} and V_{s_1}, V_{s_3} at $1222s$.

7.1.5. 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) test more LTL formula list with different structure.

As summarized in Table 3, as the system size is increased from 8 to 40, the computation time to obtain the *first* solution for task φ_2 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

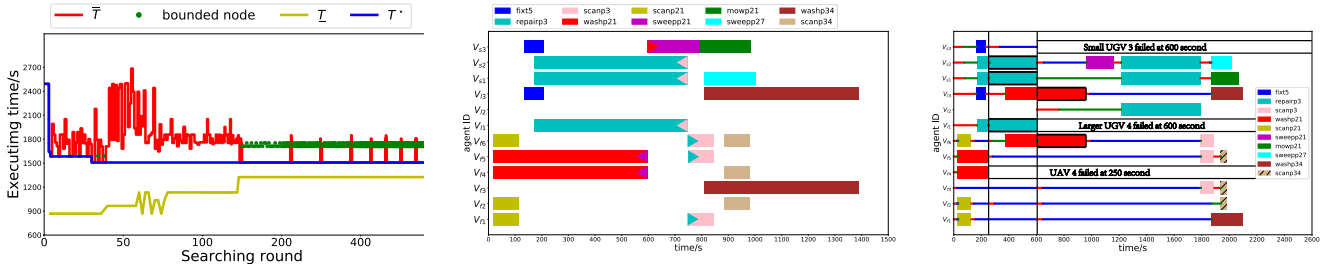

Figure 9: **Left:**Illustration of the upper and lower bounds $\overline{T}_\nu$, $\underline{T}_\nu$, and the optimal value $T^\star$, along with the BnB search process. **Middle:**Gantt graph of prefix situation. **Right:** Gantt graph of the plan execution under agent failures and fluctuated subtask duration during the execution. Additional lines are added to highlight the online synchronization process. Red seqments indicate that the agents are during transition among regions, green seqments for "waiting for collaborators", and blue segments for "waiting for preceding subtask in partial order". What's more, three failures agent are highlighted in black. Interrupted subtasks are repeated in the new-assignment.

Table 3: Scalability analyses of the proposed method

[1] System (V_f, V_s, V_l)	[3]t_{φ_1} $[s]$	t_{φ_2} $[s]$	[3]t_{φ_3} $[s]$
$(8,4,4)$	$0.13, 5.9$	$0.14, 1.08$	$0.23, 4.81$
$(12,6,6)$	$0.15, 4.6$	$0.08, 1.85$	$0.22, 5.23$
$(16,8,8)$	$0.13, 5.0$	$0.10, 1.56$	$0.21, 4.33$
$(20,10,10)$	$0.53, 8.4$	$0.09, 2.11$	$0.58, 6.20$
[2]Poset analysis	$64.4, 71.1$	$8.4, 37.9$	$136.4, 552.5$

[1] The number of different types of agents.
[2] The associated solution time, when the first poset is returned and when the best poset with largest language is returned.
[3] The associated solution time, when the first solution is returned, and when the optimal solution is returned.

Table 4: Comparison to other methods.

[1]Method	[2]$t_{\texttt{fir}}$ $[s]$	[2]$t_{\texttt{opt}}$ $[s]$	[2]$t_{\texttt{fin}}$ $[s]$	$T_{\texttt{obj}}$ $[s]$	[2]$N_{\texttt{sync}}$
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] For each method, the first row measures the time for the system of 12 agents while the second row for 24 agents.
[2] The time to derive the first solution, the time to derive the optimal solution, the termination time, the objective as the task completion time, and the number of synchronizations required.

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 as φ_2, φ_3 are considered as follows:

$$\begin{aligned}\varphi_2 =&\Diamond(\texttt{wash}_{\texttt{p}_{11}} \wedge \neg\texttt{scan}_{\texttt{p}_i} \wedge \Diamond\texttt{scan}_{\texttt{p}_{11}} \wedge \Diamond((\texttt{mow}_{\texttt{p}_{11}} \wedge \neg\texttt{wash}_{\texttt{p}_{11}}) \wedge \\ &\Diamond(\texttt{sweep}_{\texttt{p}_{11}} \wedge \neg\texttt{mow}_{\texttt{p}_{11}}))) \wedge \Diamond(\texttt{temp}_{\texttt{p}_{25}} \wedge \Diamond\texttt{repair}_{\texttt{p}_{25}} \wedge \Diamond((\texttt{scan}_{\texttt{p}_{25}} \\ &\wedge \neg\texttt{wash}_{\texttt{p}_{25}}) \wedge \Diamond(\texttt{sweep}_{\texttt{p}_{25}} \wedge \neg\texttt{p}_{26}))) \wedge \Diamond\texttt{temp}_{\texttt{t}_4},\end{aligned} \tag{14}$$

$$\begin{aligned}\varphi_3 =&\Diamond(\texttt{temp}_{\texttt{p}_{25}} \wedge \Diamond\texttt{repair}_{\texttt{p}_{25}} \wedge \Diamond((\texttt{scan}_{\texttt{p}_{25}} \wedge \neg\texttt{wash}_{\texttt{p}_{25}}) \wedge \Diamond(\texttt{sweep}_{\texttt{p}_{25}} \wedge \\ &\neg\texttt{p}_{26}))) \wedge \Diamond\texttt{temp}_{\texttt{t}_4} \wedge \Diamond(\texttt{sweep}_{\texttt{p}_8} \wedge \Diamond\texttt{wash}_{\texttt{p}_8}) \wedge \Diamond\texttt{repair}_{\texttt{p}_4} \bigcirc \neg\texttt{p}_4 \\ &\wedge \Diamond(\texttt{sweep}_{\texttt{p}_8} \wedge \neg\texttt{wash}_{\texttt{p}_8} \wedge \Diamond\texttt{scan}_{\texttt{p}_8}) \wedge \neg\texttt{temp}_{\texttt{t}_4} \, U \, \texttt{fix}_{\texttt{t}_4},\end{aligned} \tag{15}$$

where $\mathcal{B}_{\varphi_2}$ contains 216 state and $\mathcal{B}_{\varphi_3}$ contains 970 state. As summarized in Table 3 , the computation time of both 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.

7.1.6. 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 [5] that first computes the Cartesian products of all agent models, then computes the product Büchi automaton, and searches for the accepting run within. **Milp**: the optimization-based solution that formulates the assignment problem of posets as a MILP the compute optimal assignment similar to [13, 15]. An open source solver GLPK [26] is used. **Samp**: the sampling based method proposed in [7]. It relies on a sampling strategy to explore only relevant search space instead compute complete system model. **Decomp**: the task assignment strategy proposed in [8]. As discussed earlier in Sec. 1, the proposed task decomposition strategy only allows completely independent subtasks.

Two identical comparisons are performed under different system sizes, namely 12 and 24 agents, to compare not only efficiency but also scalability under task φ_2. As summarized in Table. 4, the results are compared in the following four aspects: the time to derive the first solution, the time to derive the optimal solution, the termination time, the optimal solution, and finally the number of online synchronizations required. 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 the **Prod** failed to generate any solution within $11h$ as the system-wide product automaton for both cases has more than 10^{19} states. The **Milp** method is only applicable for small problems, which returns the optimal solution in $0.5h$ but fails to return any solution within $16h$ for the large problem. The **Samp** method has the anytime property but it takes ten times the time to generate the first feasible solution, compared with our method. The **Decomp** method can solve both problems but the overall time

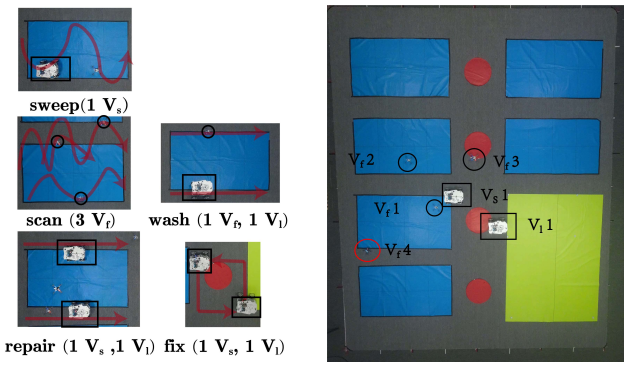

Figure 10: Snapshots of the task execution. **Left**: Trajectory and collaborators of subtasks. **Right**: UAV 4 (marked in red) is manually stopped to mimic motor failure, while the rest of the team adapts and continues the task execution.

for task completion is longer than our results, which matches our analyses in Remark 3. 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, which is consistent with the **Milp** method.

Last but not least, the last column in Table 4 compares the number of synchronizations required during execution. And our method require less synchronization.

7.2. Hardware Experiment

For further validation with hardware, a similar set-up as the simulation is built as shown in Fig. 10. 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 are designed to show how the proposed methods perform on actual hardware. Experiment videos can be found in the supplementary file.

The workspace mimics the PV farm described in the numerical simulation. As showed in Fig. 10, 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 Crazyfly mini-drones (denoted by V_f) and the UGVs are four-wheel driven cars with mecanum wheels (denoted by V_s, V_l). Existing mature navigation controllers are used and omitted here for brevity. The routine maintenance task considered can be specified with the following LTL formulas:

$$\begin{aligned}\varphi_4 =& \Diamond(\mathtt{repair}_{\mathtt{p}_2} \wedge \neg \mathtt{scan}_{\mathtt{p}_2} \wedge \Diamond \mathtt{scan}_{\mathtt{p}_2} \wedge \Diamond(\mathtt{sweep}_{\mathtt{p}_2} \wedge \mathtt{repair}_{\mathtt{p}_2})) \\ & \wedge \Diamond \mathtt{fix}_{\mathtt{t}_1} \wedge \Diamond \mathtt{scan}_{\mathtt{p}_3} \wedge \Diamond \mathtt{wash}_{\mathtt{p}_5}\end{aligned} \tag{16}$$

The proposed algorithm are successfully executed, following the procedure described in Alg. 4, 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 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. Moreover, to test the online adaptation procedure as described in Sec. 5.2, one UAV V_{f_4} is stopped manually to mimic a motor failure during execution at $75s$ and 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.

8. 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 variant.

References

[1] P. Toth, D. Vigo, An overview of vehicle routing problems, The vehicle routing problem (2002) 1–26.

[2] O. M. Cliff, R. Fitch, S. Sukkarieh, D. L. Saunders, R. Heinsohn, Online localization of radio-tagged wildlife with an autonomous aerial robot system, in: Robotics: Science and Systems, 2015.

[3] J. Fink, M. A. Hsieh, V. Kumar, Multi-robot manipulation via caging in environments with obstacles, in: 2008 IEEE International Conference on Robotics and Automation, IEEE, 2008, pp. 1471–1476.

[4] A. Varava, K. Hang, D. Kragic, F. T. Pokorny, Herding by caging: a topological approach towards guiding moving agents via mobile robots., in: Robotics: Science and Systems, 2017, pp. 696–700.

[5] C. Baier, J.-P. Katoen, Principles of model checking, MIT press, 2008.

[6] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, D. Rus, Optimality and robustness in multi-robot path planning with temporal logic constraints, The International Journal of Robotics Research 32 (8) (2013) 889–911.

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

[8] P. Schillinger, M. Bürger, D. V. Dimarogonas, Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems, The international journal of robotics research 37 (7) (2018) 818–838.

[9] M. Guo, M. M. Zavlanos, Multirobot data gathering under buffer constraints and intermittent communication, IEEE transactions on robotics 34 (4) (2018) 1082–1097.

[10] M. Guo, D. V. Dimarogonas, Multi-agent plan reconfiguration under local ltl specifications, The International Journal of Robotics Research 34 (2) (2015) 218–235.

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

[12] M. Guo, D. V. Dimarogonas, Task and motion coordination for heterogeneous multiagent systems with loosely coupled local tasks, IEEE Transactions on Automation Science and Engineering 14 (2) (2016) 797–808.

[13] X. Luo, M. M. Zavlanos, Temporal logic task allocation in heterogeneous multi-robot systems, IEEE Transactions on Robotics 38 (6) (2022) 3602–3621.

[14] Y. E. Sahin, P. Nilsson, N. Ozay, Multirobot coordination with counting temporal logics, IEEE Transactions on Robotics 36 (4) (2019) 1189–1206.

[15] A. M. Jones, K. Leahy, C. Vasile, S. Sadraddini, Z. Serlin, R. Tron, C. Belta, Scratchs: Scalable and robust algorithms for task-based coordination from high-level specifications, in: Proc. Int. Symp. Robot. Res., 2019, pp. 1–16.

[16] X. Luo, Y. Kantaros, M. M. Zavlanos, An abstraction-free method for multirobot temporal logic optimal control synthesis, IEEE Transactions on Robotics (2021).

[17] A. Lavaei, S. Soudjani, A. Abate, M. Zamani, Automated verification and synthesis of stochastic hybrid systems: A survey, Automatica 146 (2022) 110617.
[18] D. A. Simovici, C. Djeraba, Mathematical tools for data mining, SpringerVerlag, London (2008).
[19] P. Gastin, D. Oddoux, Fast LTL to büchi automata translation, in: Computer Aided Verification, Springer, 2001, pp. 53–65.
[20] R. Sedgewick, Algorithms in C, part 5: graph algorithms, Pearson Education, 2001.
[21] M. Gini, Multi-robot allocation of tasks with temporal and ordering constraints, in: AAAI Conference on Artificial Intelligence, 2017.
[22] A. Khamis, A. Hussein, A. Elmogy, Multi-robot task allocation: A review of the state-of-the-art, Cooperative Robots and Sensor Networks 2015 (2015) 31–51.
[23] P. Brucker, B. Jurisch, B. Sievers, A branch and bound algorithm for the job-shop scheduling problem, Discrete applied mathematics 49 (1-3) (1994) 107–127.
[24] E. L. Lawler, D. E. Wood, Branch-and-bound methods: A survey, Operations research 14 (4) (1966) 699–719.
[25] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, E. C. Sewell, Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning, Discrete Optimization 19 (2016) 79–102.
[26] A. Makhorin, Glpk (gnu linear programming kit), http://www. gnu. org/s/glpk/glpk. html (2008).