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

Zesen Liu, Meng Guo and Zhongkui Li

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.**

Index Terms—**Networked Robots, Linear Temporal Logic, Task Coordination, Anytime Search.**

I. 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 [1]. 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 [2]; and a team of drones can surveil a large terrain and detect poachers [3]. 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 [4]; and a team of mobile vehicles can collaboratively herd moving targets via formation [5]. Furthermore, to specify these complex tasks, many recent work propose to use formal languages such as Linear Temporal Logic (LTL) formulas [6], as an intuitive yet powerful way to describe both spatial and temporal requirements on the team behavior, see [7], [8], [9], [10] for examples.

The authors are with the State Key Laboratory for Turbulence and Complex Systems, Department of Mechanics and Engineering Science, College of Engineering, Peking University, Beijing 100871, China. E-mail: `1901111653, meng.guo, zhongkli@pku.edu.cn`. 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.

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 [2]. It is particularly so when certain metric should be minimized, such as the completion time or the summed cost of all robots. Considerable results are obtained in many recent work, e.g., via optimal planning algorithms such as mixed integer linear programming (MILP) in [15], [16], [17], and search algorithms over state or solution space as in [8], [9], [14]. 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 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 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. 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. 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,

TABLE I
COMPARISON OF RELATED WORK AS DISCUSSED IN SEC. II-B, REGARDING THE PROBLEM FORMULATION AND KEY FEATURES.

Ref.	Syntax	Collaboration	Objective	Solution	Anytime	Synchronization	Adaptation
[11], [12]	Local-LTL	No	Summed Cost	Dijkstra	No	Event-based	Yes
[13]	Local-LTL	Yes	Summed Cost	Dijkstra	No	Event-based	Yes
[8], [14]	Global-LTL	No	Summed Cost	Sampling	Yes	All-time	No
[9]	Global-LTL	No	Balanced	Martins' Alg.	No	None	No
[15]	Global-LTL	Yes	Summed Cost	MILP	No	All-time	No
[16], [17]	Global-cLTL	No	Summed Cost	MILP	No	Partial	No
Ours	Global-LTL	Yes	Min Time	BnB	Yes	Event-based	Yes

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 10 robots and 30 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. II reviews related work. The formal problem description is given in Sec. IV. Main components of the proposed framework are presented in Sec. V-VII. Experiment studies are shown in Sec. VIII, followed by conclusions and future work in Sec. IX.

II. RELATED WORK

A. Multi-agent Task and Motion 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 sub-tasks and then assigning them to the team, see [18], [19], [20] for comprehensive surveys. Such tasks can have additional constraints, such as time windows [21], robot capacities [22], and ordering constraints [23], [24]. 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 [19], [21], [22], and MinMAX that minimizes the maximum cost of a robot over all robots [24], similar to the *makespan* of all tasks. Typical solutions can be categorized into centralized methods such as Mixed Integer Linear Programming (MILP) [18] and search-based methods [22]; and decentralized methods such as market-based methods [21] and distributed constraint optimization (DCOP) [23]. However, since many task planning problems are in general NP-hard or even NP-complete [19], meta-heuristic approaches are used to gain computational efficiency, e.g., local search [25] and genetic algorithms [20]. One type of problem is particularly of relevance to this work, namely the Multi-Vehicle Routing Problem (MVRP) in operation research [19], [20], 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 [23], [24]. Moreover, collaborative tasks and synchronization during online execution are often neglected in the aforementioned work, where planning and execution are mostly decoupled.

On the other hand, motion planning problem for multi-agent systems aims at designing cooperative control policies such that a team-wise control objective is reached, e.g., collision-free navigation [26], formation [27], consensus [28] and coverage [29]. Such objectives are self-sustained but lacks a high-level purpose for task completion. In this work, we extend these results by incorporating them into the team-wise task as collaborative sub-tasks.

B. Temporal Logic Tasks

Temporal logic formulas can be used to specify complex robotic tasks, such as Probabilistic Computation Tree Logic (PCTL) in [30], Linear Temporal Logics (LTL) in [8], [9], [11], [31], and counting LTL (cLTL) in [16]. As summarized in Table IV, the most related work can be compared in the

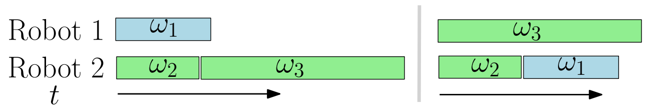


Fig. 1. Comparison of the planning results based on decompositional states in [9] (**left**) and the partial ordering proposed in this work (**right**). Note that the sub-task ω_3 has to be completed after ω_2, while ω_1 is independent.

following four aspects: (i) *collaborative tasks*. In a bottom-up fashion, [11], [12], [13] assume local LTL tasks and dynamic environment, where collaborative tasks are allowed in [13]. In a top-down fashion, [8], [9], [15], [16], [17] consider team-wise tasks, but no direct collaboration among the agents; (ii) the *optimization criteria*. Most aforementioned work [8], [13], [15], [16], [17] optimizes the summed cost of all agents, while [9] 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 sub-task. The work in [8], [14], [16] requires full synchronization before each sub-task due to the product-based solution, while [9] imposes no synchronization by allowing only independent sub-tasks and thus limiting efficiency. This work however proposes an online synchronization strategy for sub-tasks 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 [15], [16], [17] 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.

C. Branch and Bound

Branch and bound (BnB) is a search paradigm to solve discrete and combinatorial optimization problems exactly [32], [33]. All candidate solutions are represented by a rooted tree within the solution space. Via branching and bounding intelligently, an efficient search strategy can be derived by pruning early branches that are provably suboptimal. It has been successfully applied to various NP-hard problems, e.g., the traveling salesman problem [34], multi-vehicle routing problem [19], [20], and the job-shop scheduling problem [35]. Different from the commonly-seen BnB algorithms that are applied to MILP directly, we propose in this work a novel BnB search strategy *specifically* designed for the planning problem of multi-agent systems under complex temporal tasks. It takes into account both the partial ordering constraints imposed by the temporal task, and the synchronization constraints due to inter-agent collaboration.

III. Preliminary

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

A. 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 [6] 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., [6].

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, $L_\varphi = Words(\varphi) = \{W \mid W \models \varphi\}$ 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.

B. 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* [6] 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. It is easy to show that such a run is accepting if the prefix is repeated once, while the suffix is repeated infinitely often. In general, the size of $\mathcal{B}$ is double exponential to the size of $|\varphi|$.

C. Partially Ordered Set

A partial order [36] 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 $x \parallel y$.

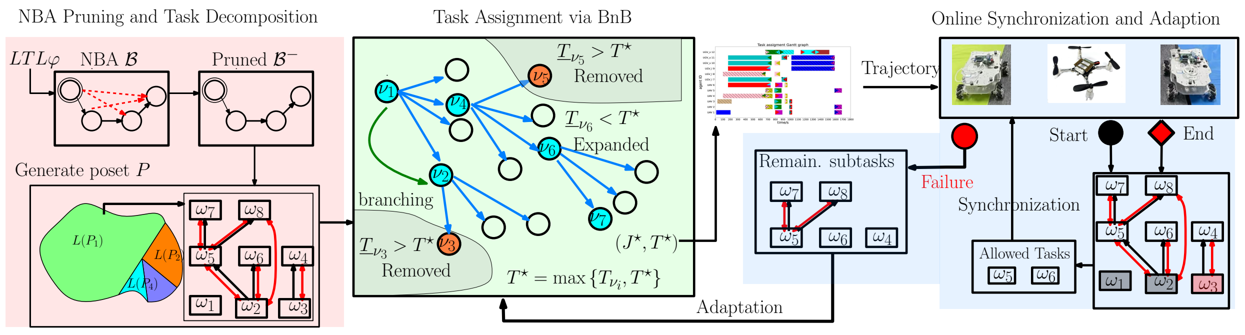


Fig. 2. Overall structure of the proposed framework, which consists of three main parts: the computation of the posets, BnB search, and online execution.

IV. Problem Formulation

A. 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 agent $n \in \mathcal{N} = \{1, \cdots, N\}$ can navigate within these regions following its own transition graph, i.e., $\mathcal{G}_n = (\mathcal{W}, \to_n, d_n)$, where $\to_n \subseteq \mathcal{W} \times \mathcal{W}$ is the allowed transition for agent n; and $d_n : \to_n \to \mathbb{R}_+$ maps each transition to its time duration.

Moreover, similar to the action model used in our previous work [13], each robot $n \in \mathcal{N}$ is capable of performing a set of actions: $\mathcal{A}_n \triangleq \mathcal{A}_n^1 \cup \mathcal{A}_n^c$, where $\mathcal{A}_n^1$ is a set of *local* actions that can be independently performed by agent n itself; $\mathcal{A}_n^c$ is a set of *collaborative* actions that can only be performed in collaboration with other agents. 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 strictly *ordered* sequence of collaborative actions, namely:

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

where ℓ_k is the number of actions required; each collaborative action $a_\ell \in \mathcal{A}_n^c$ for some agent n, $\forall a_\ell \in C_k$. A collaborative behavior is done when each collaborative action it requires is done. Thus, each collaborative action has a fixed duration pre-defined by $d : \mathcal{C} \to \mathbb{R}_+$. For instances, formation [27], herding [37], consensus [28] and coverage [29] are some common collaborative behaviors of multi-agent systems.

Remark 1. Different from [15], [16], [17], 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 agent that is capable can be recruited for the collaborative behavior. ■

B. 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 $a_k \in \mathcal{A}_n^1$. Let $\mathbf{a} \triangleq \bigcup_{n\in\mathcal{N}} \mathcal{A}_n^1$, $\hat{\mathbf{a}} \triangleq \{a_k^m, \forall W_m, a_k \in \mathcal{W}, \mathbf{a}\}$;
- c_k^m is *true* when the collaborative behavior C_k in (1) is performed at region W_m. Let $\mathbf{c} \triangleq \{c_k, \forall C_k \in \mathcal{C}\}$, $\hat{\mathbf{c}} \triangleq \{c_k^m, \forall C_k, W_m \in \mathcal{C}, \mathcal{W}\}$.

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

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

where the syntax of LTL is introduced in Sec. III-A. Moreover, in order to provide a clear optimization criteria, we consider task formulas of the following structure: $\varphi = \Box\Diamond\varphi'$, where φ' is a sc-LTL formula that can be satisfied in finite time, see Sec. III-A. Denote by $\mathbf{t}_\varphi = t_0 t_1 \cdots t_\ell \cdots$ be the infinite sequence of time instances when φ' is satisfied. Then the frequency of φ above being satisfied is defined as

$$T_\varphi = \max_{\ell \geqslant 0}\{t_{\ell+1} - t_\ell\}, \tag{3}$$

where $T_\varphi > 0$ if task φ is satisfied by the team.

Remark 2. As described previously in Sec. I, the minimum time cost in (3) is significantly different from the summed time cost of all agents, as in [8], [9], [13], [15]. 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 maintenance within a remote Photovoltaic power plant. One collaborative task considered in Sec. VIII is given by:

$$\begin{aligned}\varphi_1 = &\Diamond \texttt{repair-scan}_{\texttt{p31}} \wedge \Diamond \texttt{fix-scan}_{\texttt{t6}} \\ &\wedge \Diamond \texttt{wash}_{\texttt{p15}} \wedge \Diamond \texttt{mow}_{\texttt{t8}},\end{aligned} \tag{4}$$

namely, to repair and scan certain PV panels in a given order, fix and scan the transformers, wash the surface of other panels, and mow the grass beneath. ■

C. 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 trivial to show that this problem belongs to the class of NP-hard problems [38], [39], as its core coincides with the makespan minimization problem of flow-shop scheduling problems. Various approximate algorithms have been proposed in [20], [25]. 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. V, and then the online adaptation strategy to handle dynamic changes in Sec. VI. The overall solution is summarized in Sec. VII with analyses for completeness, optimality and complexity.

V. Optimal Plan Synthesis

The optimal plan synthesis aims to solve Problem 1 offline. As mentioned previously in Sec. II-B, most related work requires the synchronized product of all agents' model, 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.

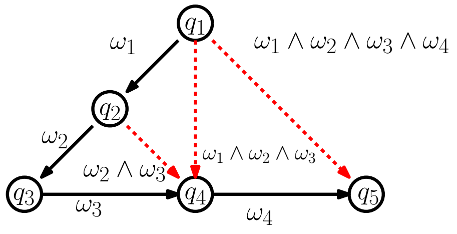
A. Büchi Automaton Pruning

As the first step, the NBA $\mathcal{B}_\varphi$ associated with the task φ is derived, e.g., via translation tools [40]. 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 are redundant as they 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 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 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, 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.

(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.



ω_1
$\omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4$
q_1
q_2
ω_2
$\omega_1 \wedge \omega_2 \wedge \omega_3$
$\omega_2 \wedge \omega_3$
q_3
q_4
q_5
ω_3
ω_4

Fig. 3. Example of decomposable transitions. The transitions within $(q_1, q_4), (q_2, q_4), (q_1, q_5)$ are all decomposable by Def. 2.

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. 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.

Example 2. The NBA $\mathcal{B}_\varphi$ associated with the task formula in (4) has 54 states and 337 edges with 1838 accepting *word*, translated via [40]. After the pruning process described above, the pruned automaton $\mathcal{B}_\varphi^-$ has 53 states and 185 edges with 240 accepting *word*. The NBA pruning step reduces 51% edges and 86.9% accepting *word* without losing any information. ■

B. 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 $(\Diamond(\omega_1 \wedge \Diamond(\omega_2 \wedge \Diamond\omega_3)))$ 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 2-tuples:

$$\Omega_\varphi = \{(\ell,\, \sigma_\ell),\, \forall \ell = 0, 1, \cdots, L\}\,, \tag{5}$$

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

In other words, a subtask $(\ell,\, \sigma_\ell)$ consists of its index and its set of position or action propositions. 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. Moreover, the two conditions in the above definition require that each subtask σ_ℓ satisfies the segment of the task from q_ℓ to $q_{\ell+1}$, and σ_ℓ only contains the essential propositions for that segment. Thus, every element inside σ_ℓ needs to be fulfilled for the subtask to be fulfilled. 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 $(h,\, \sigma_h), (\ell,\, \sigma_\ell) \in \Omega_\varphi$, the following two types of relations are defined:

(I) "less equal": $\preceq_\varphi \subseteq \Omega_\varphi \times \Omega_\varphi$. If $((h,\, \sigma_h), (\ell,\, \sigma_\ell)) \in \preceq_\varphi$ or equivalently $(h,\, \sigma_h) \preceq_\varphi (\ell,\, \sigma_\ell)$, then $(h,\, \sigma_h)$ has to be *started* before $(\ell,\, \sigma_\ell)$ is started.

(II) "opposed": $\neq_\varphi \subseteq \Omega_\varphi \times \Omega_\varphi$. If $((h,\, \sigma_h), (\ell,\, \sigma_\ell)) \in \neq_\varphi$ or equivalently $(h,\, \sigma_h) \neq_\varphi (\ell,\, \sigma_\ell)$, then $(h,\, \sigma_h)$ has to be *finished* before $(\ell,\, \sigma_\ell)$ is *started*. ■

Remark 3. Note that most related work [8], [11], [12], [14], [16], [17] 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. On the contrary, for the action model described in Sec. IV-A, each action has a duration that the associated proposition is True during the *whole* period. Consequently, it is essential to distinguish these two relations defined above, namely, whether one sub-task should be started or finished before another sub-task. ■

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 (Poset of Subtasks)**.** One partially ordered set (*poset*) over the decomposition Ω_φ is given by

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

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

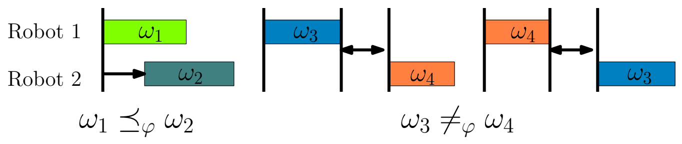


Fig. 4. Illustration of the two partial relations contained in the poset. **Left**: $\omega_1 \preceq_\varphi \omega_2$ requires that task ω_2 is started after task ω_1. **Middle&Right**: $\omega_3 \neq_\varphi \omega_4$ includes two cases: task ω_3 is finished before task ω_4 is started, or vice versa.

Similar to the original notion of poset in [36], 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, $\omega_1 \neq_\varphi \omega_2$ and $\omega_2 \neq_\varphi \omega_3$ can not imply $\omega_1 \neq_\varphi \omega_3$. 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., $(h,\, \sigma_h) \preceq_\varphi (\ell,\, \sigma_\ell)$ and $(h,\, \sigma_h) \neq_\varphi (\ell,\, \sigma_\ell)$, 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 $\neq_\varphi \triangleq \mathbb{F}$ 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 4. 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. 4 and explained in Remark 3, these two relations can describe any possible temporal relation between two 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 5. It is worth noting that the proposed notion of posets contains the "decomposable states" proposed in [9] 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.

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 $\rho$;
4  while $time < t_0$ do
       /* Subtask decomp. by Def. 3      */
5      Compute $\Omega$ and word $W$ given $\rho$;
6      if $W \notin \mathcal{L}_\varphi$ then
7          Set $P = (\Omega, \preceq_\varphi, \neq_\varphi)$, and $\preceq_\varphi = \neq_\varphi = \mathbb{F}$;
8          Set $L(P) = Que = \{W\}$ and $I_1 = I_2 = \varnothing$;
           /* Reduce partial relations    */
9          while $|Que| > 0$ do
10             $W \leftarrow Que.pop()$;
11             for $i = 1, 2, \cdots, |\rho|$ do
12                 $\omega_1 = W[i]$, $\omega_2 = W[i+1]$;
13                 $W' \leftarrow$ Switching $\omega_1$ and $\omega_2$ within $W$;
14                 if $W'$ is accepting then
15                     Add $(\omega_1, \omega_2)$ 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 $(\omega_1, \omega_2)$ to $I_2$;
20         Remove $\{I_1 \setminus I_2\}$ from $\preceq_\varphi$;
21         for $(\omega_1, \omega_2) \in \neq_\varphi$ do
22             $W' \leftarrow$ Replace $\omega_1, \omega_2$ in $W$ by $\omega_1 \cup \omega_2$;
23             if $W'$ is accepting then
24                 Remove $(\omega_1, \omega_2)$ from $\neq_\varphi$;
25         Add $P$ to $\mathcal{P}_\varphi$, add $L(P)$ to $\mathcal{L}_\varphi$;
26     Continue the modified DFS, and update $\rho$;
27 return $\mathcal{P}_\varphi$, $\mathcal{L}_\varphi$;
```

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.,

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

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) $t_\ell + d_\ell < t_{\ell'}$ or $t_{\ell'} + d_{\ell'} < t_\ell$ holds, $\forall (\omega_\ell, \omega_{\ell'}) \in \neq_\varphi$, 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 [41]. 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 $P = (\Omega, \mathbb{F}, \mathbb{F})$ 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 constraints. Furthermore, for any two subtasks that belong to the "opposed" relation, a new word is generated by allowing both subtasks to be fulfilled simultaneously in Line 22. If this new word is accepting, it means that these two subtasks 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.

Lemma 1. *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 both 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. □

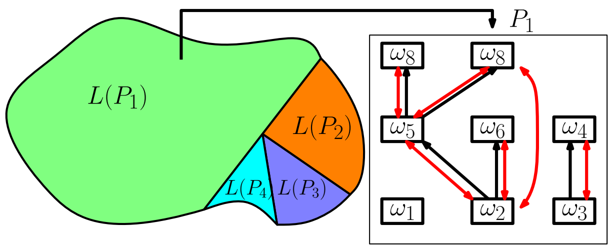


Fig. 5. **Left**: an illustration of the relation between the accepting language of different posets $L(P_i)$ and the accepting language of the task $L(\varphi)$. **Right**: an example of the poset graph $\mathcal{G}_{P_\varphi}$, where the relations $\preceq_\varphi$ and $\neq_\varphi$ are marked by 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.

Lemma 2. *The outputs of Alg. 1 satisfy that $L(P_i) \cap L(P_j) = \varnothing$, $\forall i \neq j$, and $L(P_i) \subset \mathcal{L}_\varphi \subset L_\varphi$, $\forall P_i \in \mathcal{P}_\varphi$. Moreover, given enough time $t_0 \to \infty$, the complete set of posets can be returned, i.e., $\mathcal{L}_\varphi = \cup_{P_i \in \mathcal{P}_\varphi} L(P_i) = L_\varphi$.*

Proof. The first part can be proven by contradiction. Assume that there exists two posets P_1, $P_2 \in \mathcal{P}_\varphi$ and one accepting word $W \in L_\varphi$ such that $w \in L(P_1) \cap \in L(P_2)$ holds. Since the set of subtasks within the poset is simply the union of all subtasks within each word, it implies that $\Omega_1 = \Omega_2$. Then, as discussed in Lemma 1, $W \in L(P_1)$ 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 $\mathcal{B}_\varphi^-$. In other words, the complete set of posets $\mathcal{P}_\varphi$ returned by the algorithm after full termination is ensured to cover all accepting words of the underlying NBA. As discussed earlier, the pruning procedure does not effect the complete set of accepting words, given the model of the multi-agent system. Thus, it can be concluded that the returned language set $\mathcal{L}_\varphi$ is equivalent to the original task specification. □

Last but not least, similar to the Hasse diagram [36], 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; $R \subset \Omega \times \Omega$ is the set of undirected edges. A edge $(\omega_1, \omega_2) \in E$ if two conditions hold: (i) $(\omega_1, \omega_2) \in \preceq_\varphi$; and (2) 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. A edge $(\omega_1, \omega_2) \in R$, if $(\omega_1, \omega_2) \in \neq_\varphi$. ■

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. 5.

C. 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:

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. ■

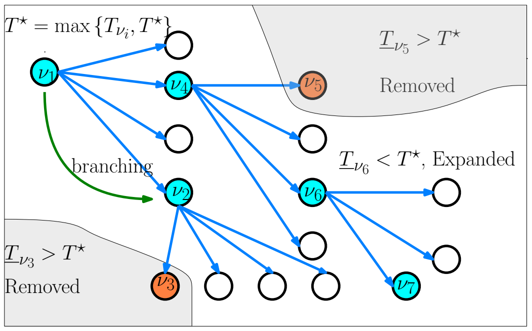
To begin with, even without the requirements of partial ordering and collaborative actions, the above problem includes the multi-vehicle routing problem [19], [20], and the job-shop scheduling problem [35] as special instances. Both problems are well-known to be NP-hard. Thus, the above problem is also NP-hard and most likely no exact solutions with polynomial complexity exist. The most common and straightforward solution is to formulate a Mixed Integer Linear Program (MILP) with $N{\cdot}L{\cdot}(L{+}1)$ Boolean variables, where N is the number of agents and L is the number of subtasks. Each Boolean variable stands for whether the ℓ_1 position in the plan of agent n takes the ℓ_2 subtask or does nothing. There are two major drawbacks of this approach: (i) the computation complexity and time grow exponentially with the problem size; (ii) there is no intermediate solution before the optimal solution is generated, often via a MILP solver, e.g., CPLEX [42]. Both drawbacks hinder the usage of this approach in large-scale real-time applications, where a timely good solution is far 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 [32], [33]. 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. 6, 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.,

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

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, $\nu =$



$T^\star = \max\{T_{\nu_i}, T^\star\}$
$\underline{T}_{\nu_5} > T^\star$
Removed
$\underline{T}_{\nu_6} < T^\star$, Expanded
branching
$\underline{T}_{\nu_3} > T^\star$
Removed
ν_1
ν_2
ν_3
ν_4
ν_5
ν_6
ν_7

Fig. 6. 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).

$((\omega_1, \omega_2), (), ())$ means that two subtasks ω_1, ω_2 are assigned to agent 1, whereas no tasks to agents 2 and 3. With slight abuse of notation, denote by

$$\Omega_\nu = \{\omega \in \tau_n,\ \forall n \in \mathcal{N}\},\ \Omega_\nu^- = \Omega \backslash \Omega_\nu, \tag{9}$$

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. Starting from the root node with zero assignment, the tree is expanded by generating child nodes where subtasks are partially assigned to the agents, until all the subtasks are assigned. More specifically, let ν be the current node of the search tree. The *next* subtask ω to be assigned is chosen from the poset $\mathcal{G}_{P_\varphi}$ defined in Def. 7 if *all* of its parent subtasks are already assigned, i.e.,

$$\omega' \in \Omega_\nu,\ \forall \omega' \in \text{Pre}(\omega), \tag{10}$$

where Ω_ν is the set of assigned subtasks from (9); Pre(ω) is the set of preceding or parenting subtasks of ω in $\mathcal{G}_{P_\varphi}$. Once this subtask ω is chosen, the succeeding or child node ν^+ of ν in the search tree is created by assigning ω to any agent that is capable of executing ω, i.e.,

$$(p_n,\ p_n^+) \in \rightarrow_n, \text{ and } a^+ \in \mathcal{A}_n^1, \text{ or } a_n^+ \in \mathcal{A}_n^{\text{c}}, \tag{11}$$

where p_n is the current position of agent n at node ν, while p_n^+ is the desired position of agent n at node ν^+; a^+ is the desired local action to be performed at node ν^+, while a_n^+ is the desired collaborative action to be performed at node ν^+ projected to agent n. In other words, agent n can transit to p_n^+ and perform a^+ or a_n^+. Thus, given node $\nu = (\tau_1, \cdots, \tau_N)$, the child node $\nu^+ = (\tau_1^+, \cdots, \tau_N^+)$ is given by

$$\tau_n^+ = (\tau_n,\ \omega), \text{ and } \tau_{n'}^+ = \tau_{n'},\ \forall n' \neq n. \tag{12}$$

If there are multiple agents satisfying the condition in (11), then multiple child nodes are expanded, one for each agent.

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 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.

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

```
Input  : Poset $P_\varphi$, node $\nu$.
Output: Assignment $\overline{J}_\nu$, upper bound $\overline{T}_\nu$.
1 while $\Omega_\nu^- \neq \varnothing$ do                                    // (9)
2     forall $\omega \in \Omega_\nu^-$ and $n \in \mathcal{N}$ do
3         Assign $\omega$ to agent $n$;
4         Compute and save child node $\nu_{n,\omega}^+$;              // (12)
5     Select $\nu_\star^+ = \textbf{argmax}_{\nu \in \{\nu_{n,\omega}^+\}} \{\eta_\nu\}$ ;              // (15)
6     $\nu \leftarrow \nu_\star^+$;
7 Compute assignment $J_\nu$ and makespan $T_\nu$ ;      // (14)
8 Return $\overline{J}_\nu = J_\nu$, $\overline{T}_\nu = T_\nu$;
```

Lower and upper bounding. For each new node in the search tree, it is checked against the estimated lower and upper bounds of the optimal solution. If this node can not produce a better solution, it is discarded from the search tree and no new nodes will be expanded from it. Therefore, the accuracy of these bounds effects greatly the efficiency of a BnB algorithm, which can degenerate to an exhaustive search, if no such bounds are available. More specifically, given a node ν, the upper bound of all solutions rooted from this node is estimated via a greedy task assignment policy, as summarized in Alg. 2:

$$\overline{J}_\nu,\ \overline{T}_\nu = \texttt{upper\_bound}(\nu,\ P_\varphi), \tag{13}$$

where $\overline{T}_\nu$ is upper bound, and $\overline{J}_\nu$ is the associated complete assignment *with time stamps*, i.e.,

$$J = (J_1,\ J_2, \cdots, J_N), \tag{14}$$

where $J_n = ((t_1,\ \omega_1), (t_2,\ \omega_2), \cdots, (t_{K_n}, \omega_{K_n}))$ is the assignment for agent $n \in \mathcal{N}$, and t_k is the starting time for subtask ω_k, $\forall (t_k,\ \omega_k) \in J_n$. From node ν, any task $\omega \in \Omega_\nu^-$ is assigned to any allowed agent $n \in \mathcal{N}$ 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:

$$T_\nu = \max_{n \in \mathcal{N}} \{T_{\tau_n}\},\ T_\nu^{\text{s}} = \sum_{\omega \in \Omega_\nu} T_\omega N_\omega,\ \eta_\nu = \frac{T_\nu^{\text{s}}}{T_\nu}, \tag{15}$$

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 current makespan calculate by (15); and T_ν^{s} is the total execution time of all subtasks ω given its duration T_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 5-6. This procedure is repeated until no subtasks remain unassigned. Afterwards, a complete assignment J_ν with time stamps is obtained by solving a simple linear program given the sequence of subtasks the last node ν. Afterwards, the upper bound $\overline{T}_\nu$ is given the makespan of the complete assignment, as in Line 7.

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. More specifically, let the current node be ν, the set of unfinished subtasks is given by Ω_ν^- in (9). The

Algorithm 3: `BnB(·)`: Anytime BnB algorithm for task assignment

```
   Input  : Agents $\mathcal{N}$, poset $P_\varphi$, time budget $t_0$.
   Output: Best assignment $J^\star$ and makespan $T^\star$.
 1 Initialize root node $\nu_0$ and queue $Q = \{\nu_0\}$;
 2 Set $T^\star = \infty$ and $J^\star = ()$;
 3 while (Q not empty) or (time < $t_0$) do
 4     Take node $\nu$ off $Q$;
 5     $\overline{J}_\nu, \overline{T}_\nu$ = upper_bound($\nu$, $P_\varphi$) ;  // Alg. 2
 6     $\underline{T}_\nu$ = lower_bound($\nu$, $P_\varphi$) ;        // (18)
 7     Compute bounds $\overline{T}_\nu$ and $\underline{T}_\nu$ by (13) and (18);
 8     if $T^\star > \overline{T}_\nu$ then
 9         Set $T^\star = \overline{T}_\nu$ and $J^\star = \overline{J}_\nu$;
10     if $\underline{T}_\nu \geqslant T^\star$ then
11         Continue ;          // discard branch
12     else
13         Branch on $\nu$ by (11) and (12);
14         Choose new node $\nu^+$ by estimated $\underline{T}_{\nu^+}$;
15         Store $\nu^+$ to $Q$;
16 Return $J^\star, T^\star$;
```

first lower bound $\underline{T}_{\nu,1}$ on the makespan to accomplish Ω_ν^- is computed via analyzing the poset graph $\mathcal{G}_{P_\varphi} = (\Omega,\, E,\, R)$ in (7). In particularly, consider the directed graph $\mathcal{G}'_{P_\varphi} = (\Omega'_\nu,\, E \cap R)$, where Ω'_ν is the set of nodes associated with the unfinished subtasks, and the set of edges contains the edge $(\omega',\, \omega)$ that belongs to both $\preceq_\varphi$ and $\neq_\varphi$. Then, starting from the set of root nodes that does not have parent nodes, a BFS procedure is used to traverse all nodes in the graph $\mathcal{G}'_{P_\varphi}$. Denote by $\Gamma_\nu^- = \{\tau_{\omega_f,\omega_g}\}$ the set of paths from any root node ω_f to any other node ω_g in the graph. Then, the first lower bound $\underline{T}_{\nu,1}$ on the makespan is given by

$$\underline{T}_{\nu,1}(\nu,\, P_\varphi) = \mathbf{max}_{\tau\in\Gamma_\nu^-}\{T_\tau\}, \tag{16}$$

where T_τ is the accumulated duration of path $\tau = \omega_0\omega_1\cdots\omega_P$, i.e., $T_\tau = \sum_{p=1}^{P} T_{\omega_p}$. In other words, the lower bound $\underline{T}_{\nu,1}$ assumes that there are as many agents as needed for the sub-tasks, while the partial ordering constraints for the subtasks are ensured. Secondly, another lower-bound $\underline{T}_{\nu,2}$ is estimated by ignoring the partial ordering constraints:

$$\underline{T}_{\nu,2}(\nu,\, P_\varphi) = \frac{\sum_{\omega\in\Omega_\nu^-} T_\omega N_\omega}{N}, \tag{17}$$

where T_ω and N_ω are the duration and number of agents required for subtask $\omega \in \Omega_\nu^-$ as defined in (15).

Consequently, the lower bound on the makespan of all solutions rooted from ν is given as the minimum of these two lower bounds above:

$$\underline{T}_\nu = \texttt{lower\_bound}(\nu,\, P_\varphi) = \min\{\underline{T}_{\nu,1},\, \underline{T}_{\nu,2}\}, \tag{18}$$

which can be computed efficiently. It is worth noting that the task assignments associated with $\underline{T}_\nu$ above is infeasible as it either violates the partial ordering constraints or the current agent capacities.

Algorithm 4: Complete algorithm for time minimization under collaborative temporal tasks

```
   Input  : Task formula $\varphi$, time budget $t_0$.
   Output: Assignment $J^\star$, makespan $T^\star$
 1 Compute $\mathcal{B}_\varphi$ given $\varphi$ ;             // [40]
 2 Compute $\mathcal{B}_\varphi^-$ by pruning $\mathcal{B}_\varphi$ ;      // Sec. V-A
 3 Initialize $\mathcal{J} = \varnothing$;
 4 while time < $t_0$ do
 5     $P_\varphi \leftarrow$ compute_poset($\mathcal{B}_\varphi^-$) ;    // Alg. 1
 6     $(J, T) \leftarrow$ BnB($P_\varphi$) ;               // Alg. 3
 7     Store $(J, T)$ in $\mathcal{J}$;
 8 Select $J^\star$ with minimum $T^\star$ among $\mathcal{J}$;
 9 Return $J^\star$, $T^\star$;
```

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 solvers. ■

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 contains only ν_0. Then, within the time budget, a node ν is taken from Q for expansion. The upper and lower bound associated with ν is computed in Lines 5 and 6. If the upper bound $\overline{T}_\nu$ is less than the current best-known value $T^\star$, then $T^\star$ is replaced by $\overline{T}_\nu$ and the associated plan $J^\star$ is saved. On the other hand, if the lower bound $\underline{T}_\nu$ is larger than the current $T^\star$, it means no partial solution rooted from this node can produced a better solution, thus discarded for expansion in Line 11. Otherwise, the child node ν^+ of ν is generated and expanded according to their estimated lower bounds in Line 14. This process repeat itself until time elapsed or the whole search tree is exhausted.

Lemma 3. *Any task assignment $J^\star$ obtained from Alg. 3 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 Alg. 2 at certain node in the search tree, it suffices to show that any solution of Alg. 2 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 poset graph as defined in (10). 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. □

D. 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 7. 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 (14), i.e., an optimal sequence of subtasks $\omega_1\omega_2\cdots\omega_{K_n}$ is assigned to agent n with the associated time stamps $t_1t_2\cdots t_{K_n}$. In other words, agent n can simply execute this sequence of subtasks at the designated time, namely ω_k at time t_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.

VI. Online Adaptation

Since there are often uncertainties in the model, 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 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.

A. 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. 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 partial ordering constraints $\omega_1 \neq_\varphi \omega_2$. These cases would all lead to a failed task execution. 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:

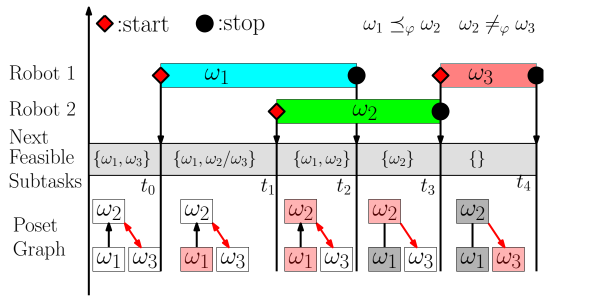


Fig. 7. Illustration of the online synchronization process described in Sec. VI-A. Consider the constraints $\omega_1 \preceq_\varphi \omega_2$ and $\omega_2 \neq_\varphi \omega_3$. The "`Start`" and "`Stop`" messages are marked by red diamonds and black circles, respectively. The agent can only execute the subtask if all relevant partial relation is satisfied.

(i) *Before* execution. To start executing ω_n^k, for each subtask ω_m^ℓ satisfying $(\omega_n^k, \omega_m^\ell) \in \preceq_\varphi$, a "`start`" synchronization message is sent by agent n to agent m. 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 $(\omega_h^\ell, \omega_n^k) \in \neq_\varphi$, agent n waits from the "`stop`" synchronization message from agent h. This message indicates that the execution of ω_h^ℓ is finished thus ω_n^k can be started; (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 $(\omega_n^k, \omega_h^\ell) \in \neq_\varphi$, 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. 7. The synchronization for collaborative tasks is easier as a collaborative subtask can only start once all participants are present at the location.

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 the complete set of subtasks. In comparison, the product-based solution [6] and the sampling-based solution [8] require full synchronization during task 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 [15], [17] requires no synchronization as each agent simply executes the subtasks according to the optimal time plan. The planning algorithm in [9] 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. ■

B. 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:

$$\widehat{\Omega}_{N_d} = \{\omega_{N_d}^{k'},\, k' = k_d, k_d+1, \cdots, K_{N_d}\}, \tag{19}$$

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.,

$$\nu_0' = (\tau_1', \cdots, \tau_{N_d-1}', \tau_{N_d+1}', \cdots, \tau_N'), \tag{20}$$

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 (19). 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.

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.

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

Proof. To begin with, Lemma 1 shows that any poset obtained by Alg. 1 is accepting. As proven in Lemma 2, 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 3 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. III-B, 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 in (18) 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 9. 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. ■

VIII. 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.

A. Simulations and Experiment

The numerical study simulates a team of multiple UGVs and UAVs that are responsible for maintaining a remote

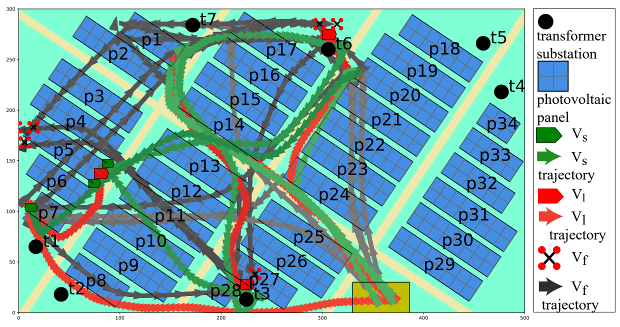


Fig. 8. Simulated PV power station in the numerical study, which consists of PV panels $\mathtt{p}_i$, roads, inverters/transformers $\mathtt{t}_i$ and base stations b. The arrow trajectories are the path of Agent swarm executing the LTL formula φ_3. And the arrow direction is the motion direction and the arrow density correspond to the velocity of different agents.

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.

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. 8, the station consists of mainly three parts: PV panels, roads, inverter/transformer substations and the robot base station. These areas of interest are highlighted in Fig. 8: different areas of PV panels are indicated by $\mathtt{p}_1, \cdots, \mathtt{p}_{34}$; the base station for all agents is denoted by $\mathtt{b}$; the transformer substations are indicated by $\mathtt{t}_1, \cdots, \mathtt{t}_7$.

Furthermore, there are one type of UAVs and two types of UGVs. The UAVs are quadcopters that are capable of fast movement to inspect the operation condition of PV panels and transformers, and supervise the maintenance, denoted by V_f. The UGVs have two sizes: the larger ones can collaborate with other UGVs or UAVs to clean and repair the PV panels and transformers, named V_l; the smaller ones can travel more freely, e.g., under the PV panels and between the transformers, in order to mow grass or sweep debris within these narrow areas, named V_s. As a result, different types of robots have different motion model $\mathcal{G}_n$ as described in Sec. IV-A and distinctive action models. The traveling time among the regions of interest is estimated by the route distance and their respective speed. Descriptions of these regions of interest and robot actions are summarized in Table. II. 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 one V_f are required to wash dirt off the surface of PV panels.

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. VIII-A5. Moreover, we consider the following three

TABLE II
DESCRIPTION OF RELATED REGIONS AND AGENT ACTIONS.

Proposition	**Description**	**Duration** [s]
$\mathtt{p}_1, \cdots, \mathtt{p}_{34}$	34 PV panels.	\
$\mathtt{b}$	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 one V_f.	10
$\mathtt{sweep}_{\mathtt{p}_i}$	Sweep debris around any panel $\mathtt{p}_i$. Requires one V_s.	190
$\mathtt{mow}_{\mathtt{p}_i, \mathtt{t}_i}$	Mow the grass under panel $\mathtt{p}_i$ or transformer $\mathtt{t}_i$. Requires one V_s.	190
$\mathtt{fix}_{\mathtt{t}_i}$	Fix malfunctional transformer $\mathtt{t}_i$. Requires one V_l and one V_s	72
$\mathtt{repair}_{\mathtt{p}_i}$	Repair broken panel $\mathtt{p}_i$. Requires one V_s to repair and two V_f to guide.	576
$\mathtt{wash}_{\mathtt{p}_i}$	Wash the dirt off panel $\mathtt{p}_i$. Requires one V_l to wash and one V_f to monitor the progress.	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 three V_f.	95

tasks with increasing complexity.

Task One: A simple and regular maintenance task, which includes to repair the panel $\mathtt{p}_{31}$, fix the transformer $\mathtt{t}_6$, wash the panel $\mathtt{p}_{15}$, and mow grass around the transformer $\mathtt{t}_8$. This task can be specified as the following LTL formulas:

$$\begin{aligned}\varphi_1 = & \mathtt{repair\text{-}scan}_{\mathtt{p}_{31}} \wedge \mathtt{fix\text{-}scan}_{\mathtt{t}_6} \\ & \wedge \Diamond \mathtt{wash}_{\mathtt{p}_{15}} \wedge \Diamond \mathtt{mow}_{\mathtt{p}_8},\end{aligned} \tag{21}$$

where the "repair and scan" and "fix and scan" routines are defined as follows:

$$\begin{aligned}\mathtt{repair\text{-}scan}_{\mathtt{p}_i} = & \Diamond \left(\mathtt{repair}_{\mathtt{p}_i} \wedge \Diamond \mathtt{scan}_{\mathtt{p}_i}\right) \\ & \wedge \Box \left(\mathtt{repair}_{\mathtt{p}_i} \to \neg \mathtt{scan}_{\mathtt{p}_i}\right); \\ \mathtt{fix\text{-}scan}_{\mathtt{t}_i} = & \Diamond (\mathtt{fix}_{\mathtt{t}_i} \wedge \Diamond \mathtt{scan}_{\mathtt{t}_i}) \\ & \wedge \Box (\mathtt{fix}_{\mathtt{t}_i} \to \neg \mathtt{scan}_{\mathtt{t}_i}),\end{aligned}$$

which specifies that a scanning is required *after* a panel is repaired or a transformer is fixed. This task can be clearly divided into several parallel sub-tasks that can be executed simultaneously and independently. The location of these sub-tasks are chosen across the workspace thus a coordination strategy to minimize completion time is crucial for the task.

Task Two: A complex maintenance task that involves several sequential routines. It includes to deep clean several panels and monitor the temperature of several transformers. It can be expressed as the following LTL formulas:

$$\varphi_2 = \Big(\bigwedge_{i=11,20} \mathtt{deep\text{-}clean}_{\mathtt{p}_i}\Big) \wedge \Diamond \mathtt{temp}_{\mathtt{t}_4}, \tag{22}$$

where the "deep cleaning" routine consists of three steps: wash the panel, mow the grass, and finally sweep the debris:

$$\begin{aligned}\mathtt{deep\text{-}clean}_{\mathtt{p}_i} = & \Diamond (\mathtt{wash}_{\mathtt{p}_i} \wedge \Diamond \mathtt{scan}_{\mathtt{p}_i} \\ & \wedge \Diamond \left(\mathtt{mow}_{\mathtt{p}_i} \wedge \Diamond \mathtt{sweep}_{\mathtt{p}_i}\right)) \\ & \wedge \Box \left((\mathtt{wash}_{\mathtt{p}_i} \vee \mathtt{mow}_{\mathtt{p}_i}) \to \neg \mathtt{sweep}_{\mathtt{p}_i}\right) \\ & \wedge \Box \left(\mathtt{wash}_{\mathtt{p}_i} \to \neg \mathtt{mow}_{\mathtt{p}_i}\right) \\ & \wedge \Box \left(\mathtt{wash}_{\mathtt{p}_i} \to \neg \mathtt{scan}_{\mathtt{p}_i}\right),\end{aligned}$$

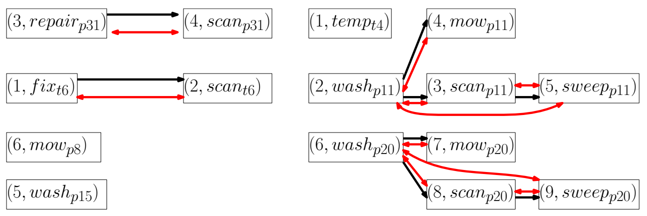


Fig. 9. **Left**: Poset graph of task φ_1; **Right**: Poset graph of task φ_2. The relations $\preceq_\varphi$, $\neq_\varphi$ are marked by black and red arrows, respectively.

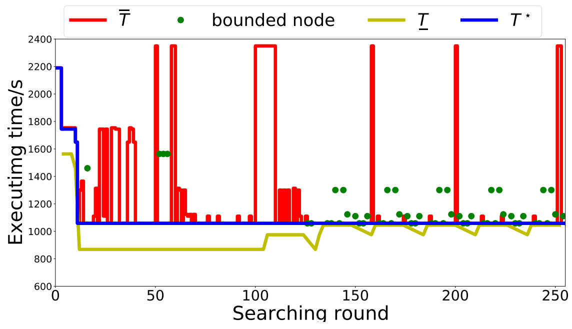


Fig. 10. Illustration of the upper and lower bounds $\overline{T}_\nu$, $\underline{T}_\nu$, and the optimal value $T^\star$, along with the BnB search process.

which indicates that for safety reasons the deep cleaning should be performed in sequence of washing, mowing and sweeping. Clearly, compared with Task One, Task Two contains more constraints on the sequential order of subtasks, thus requiring more synchronization during execution. However, since there are many panels that require deep cleaning and the temperature of many transformers should be measured, these processes can be performed in parallel. Thus most agents especially the UAVs are responsible for more than one subtasks, which is quite different from Task One.

Task Three: As the most complex task, Task Three combines the previous two tasks. Namely, it consists of all possible maintenance tasks, including the repair and fixation of panels and transformers, the deep cleaning of panels, and the monitoring of panels and transformers, as the following formulas:

$$\begin{aligned}\varphi_3 = &\big(\bigwedge_{i=1,5} \texttt{repair-scan}_{\texttt{p}_i}\big) \wedge \big(\bigwedge_{i=17,24} \Diamond \texttt{temp}_{\texttt{p}_i}\big)\\ &\wedge \big(\bigwedge_{i=3,6} \texttt{fix-scan}_{\texttt{t}_i}\big) \wedge \big(\bigwedge_{i=7,20} \texttt{deep-clean}_{\texttt{p}_i}\big)\end{aligned} \tag{23}$$

of which the routines are defined in (21) and (22). Clearly, for a team of 12 agents, Task Three with 18 sub-tasks is significantly much more complex than the previous two tasks, and thus much harder to coordinate the whole system.

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.

Task One: The NBA $\mathcal{B}_\varphi$ associated with task one in (21) contains 54 states and 430 edges. And the pruning step reduced 1.9% states and 56% edges within 0.25 second. The Alg. 1 explores 180 accepting runs in 0.04 second and finds only one poset P_φ with 6 subtasks that can describe all the accepting runs in L_φ. That means we can use P_φ to completely replace the $\mathcal{B}_\varphi$ without losing any message.

This poset P_φ constructed of 6 subtasks, and the partial relations $\preceq_\varphi$ has 2 elements, and $\neq_\varphi$ has 2 elements. Note that $\mathcal{L}(P_\varphi)$ is equal to $\mathcal{L}(\mathcal{B}^-_\varphi)$ so that this one poset provides a compact representation of all the allowed words of formula φ_1. The corresponding diagram of poset $\mathcal{G}_{P_\varphi}$ is shown in Fig. 9, where the subtasks are separated into four parallel branches and connected by their partial relations. Clear correspondences can be found from the LTL formula to the partial relations: $\Diamond(\texttt{fix}_{\texttt{t}_6} \wedge \Diamond\texttt{scan}_{\texttt{t}_6})$ maps to $\texttt{fix}_{\texttt{t}_6} \preceq_\varphi \texttt{scan}_{\texttt{t}_6}$; $\Diamond(\texttt{repair}_{\texttt{p}_{31}} \wedge \Diamond\texttt{scan}_{\texttt{p}_{31}})$ to $\texttt{repair}_{\texttt{p}_{31}} \preceq_\varphi \texttt{scan}_{\texttt{p}_{31}}$; $\Box(\texttt{fix}_{\texttt{p}_i} \rightarrow \neg\texttt{scan}_{\texttt{p}_i})$ to $\texttt{fix}_{\texttt{t}_6} \neq_\varphi \texttt{scan}_{\texttt{t}_6}$; and $\Box(\texttt{repair}_{\texttt{p}_i} \rightarrow \neg\texttt{scan}_{\texttt{p}_i})$ to $\texttt{repair}_{\texttt{p}_{31}} \neq_\varphi \texttt{scan}_{\texttt{p}_{31}}$. Visualization of the final assignment is omitted here due to limited space, and given in the supplementary file. Then, during the task assignment, the first valid solution is found in $0.24s$. Afterwards, at $t = 0.55s$, a node is reached and its estimated lower bound is larger than current upper bound, and thus cut off the search tree. Overall, around 33% 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 solution in $3.12s$ with exploring 21 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 question. In the optimal task assignment, all 6 V_f are deployed. All agents are assigned to only one task except V_{s12}, V_{f6} which have two subtasks. It can be seen that tasks without partial constraints such as $\texttt{repair}_{\texttt{p}_{31}}$, $\texttt{repair}_{\texttt{p}_{15}}$ are distributed to different agents. The tasks with partial constrains are always satisfied, as shown by the marked triangles. The makespan is determined by the execution of $\texttt{repair}_{\texttt{p}_{31}}$, $\texttt{scan}_{\texttt{p}_{31}}$ as $731s$.

Task Two: Compared with Task One, task two contains more ordering constraints, e.g., the sub-task $\texttt{deep-clean}_{\texttt{p}_i}$ requires that 5 different subtasks are executed in certain sequences. The associated NBA $\mathcal{B}_{\varphi_2}$ is much larger with 216 states and 2255 edges, After pruning, it is reduced to 203 states and 1524 edges in $4.3s$. Via Alg. 1, 8 posets are found after exploring 6870 accepting runs in $31.6s$, and the first poset is obtained in $4.6s$. As shown in Fig. 9, the poset with the largest language $L(P_\varphi) = 5670$ is chosen, which means that this poset contains most (around 82.5%) of the accepting runs in $\mathcal{B}^-_{\varphi_2}$. The poset graph consists of 3 independent branches and there are no relation $\preceq_\varphi$ among any two subtasks in different branches. Moreover, there exist no direct paths from any subtask in $\texttt{scan}_{\texttt{p}_{11}}$ to any subtask in $\texttt{mow}_{\texttt{p}_{11}}$. That means we can execute these subtasks in parallel which can significantly reduce the makespan of the whole task.

During the task assignment, the first solution is obtained in $0.35s$ with a makespan $2189s$. As shown in Fig. 10, 11 nodes are explored by the proposed BnB method and the optimal solution is found in $5.3s$ with the optimal makespan $1058s$. Since the posets have more ordering constraints, which reduces the size of solution space and increases the accuracy of

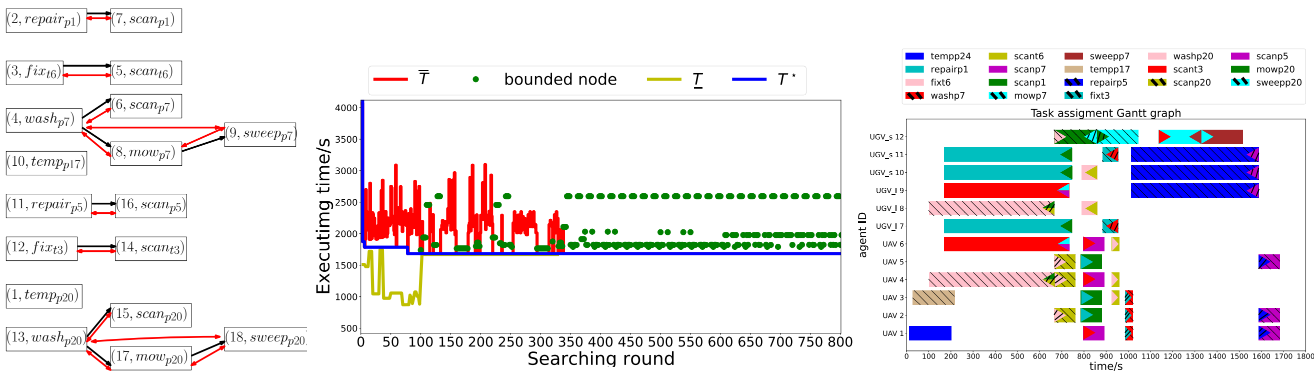


Fig. 11. **Left**: Poset graph associated with task φ_3; **Middle**: Evolution of the upper and lower bounds $\overline{T}_\nu$, $\underline{T}_\nu$, and the optimal value $T^\star$ during the BnB search process; **Right**: Gantt graph of the optimal task assignment, where the tasks with partial constrains are marked by triangles.

the estimation of the upper and lower bounds, it actually takes less explorations to find the optimal plan. For the same reason, the estimated lower and upper bounds are quite close. Around 20% of the explored nodes are removed from the search tree as their estimated lower bounds are larger than the known upper bound, which are shown as green dots in Fig. 10. It is also interesting to see that for some nodes the estimated upper bound can be quite large, which however does not effect the best value $T^\star$ as monotonically decreasing. Visualization of the final assignment is omitted here and given in the supplementary file. The subtasks $\texttt{wash}_{\texttt{p}_{11}}$ and $\texttt{wash}_{\texttt{p}_{20}}$ are executed in parallel by agents V_{f2}, V_{l2} and V_{f6}, V_{l3}, respectively. The slight difference in their starting time is due to the difference in transition time between regions. Moreover, it is worth noting that the subtasks $\texttt{scan}_{\texttt{p}_{11}}$ and $\texttt{mow}_{\texttt{p}_{11}}$ both begin at the moment when $\texttt{wash}_{\texttt{p}_{11}}$ is finished. It shows that our algorithm can not only handle parallel routines but also independent subtasks within these routines. Notice that all UAVs are deployed first to monitor the progress of $\texttt{wash}$ routines, then to collaboratively scan the panels.

Task Three: As the most complex task, the associated NBA is already too big to compute for the given computer memory after $12h$. Thus, the routines are treated separately for computing their Nabs and then composed together. This is allowed as all routines are related to different actions at different regions. Afterwards, their posets are derived separately via Alg. 1, and then composed together while maintaining consistency. As shown in Fig. 11, the poset contains 18 subtasks and 22 partial constrains. The subtasks are divided into 8 independent branches, which matches the intuition as each routine corresponds to an independent branch.

During the BnB search, the first valid solution is recorded in $0.52s$ with a makespan $4330s$. In total, 800 nodes are explored and the optimal solution is obtained after exploring 78 nodes at $37.24s$. And 70% of the explored nodes are removed from the search tree. The overall computation time is increased significantly due to two aspects: first, the computation time of both upper and lower bounds increases linearly with the number of subtasks; second, more nodes are explored due to the increased complexity of the posets. But we can still get a suitable solution quickly after exploring 6 nodes, of which the upper bound is within 6.5% above the optimal value.

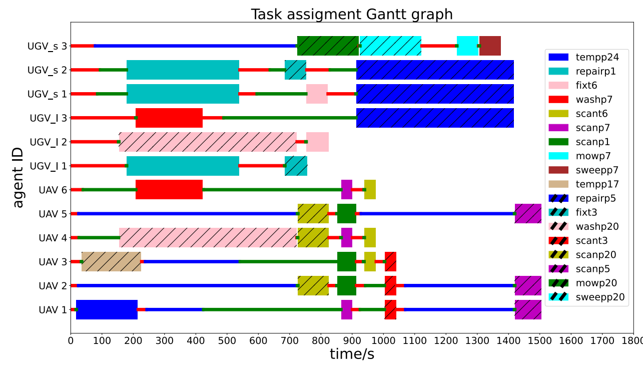


Fig. 12. Gantt graph of the plan execution under fluctuated subtask duration for task Three. Additional lines are added to highlight the online synchronization process. Red segments indicate that the agents are during transition among regions, green segments for “waiting for collaborators”, and blue segments for “waiting for preceding subtask in partial order”.

The optimal assignment is shown in Fig. 11, of which the makespan is 1683 seconds. It is noticed that in this case there might be multiple optimal solutions with the same makespan. As the makespan is mainly determined by the final subtask $\texttt{scan}_{\texttt{P}_5}$. The UAVs need to wait for a long time until the previous subtask $\texttt{repair}_{\texttt{P}_5}$ is finished. Consequently, the relative ordering of other subtasks such as $\texttt{scan}_{\texttt{P}_7}$, $\texttt{scan}_{\texttt{T}_6}$, $\texttt{scan}_{\texttt{T}_3}$ does not affect the makespan. The proposed algorithm always returns the same optimal solution due to the branching strategy. Lastly, it can be seen that the makespan of the whole task would be reduced further if more V_s, V_l are deployed, so that the subtask $\texttt{fix} - \texttt{scan}$ can be executed earlier. The final assignment is shown in Fig 11.

Notably, the associated MILP for the same problem consists of 38880 Boolean variables, which can not be solved within a reasonable time or memory. In contrast, the proposed algorithm returns the first valid solution in $3.4s$, and within 5% margin of the optimal value in $6.74s$.

4) Online Adaptation: As an important part of the contribution, we now simulate the following two practical scenarios to validate the proposed online adaptation algorithm.: (i) fluctuations in the execution time of subtasks; (ii) several agents break down during the online execution,

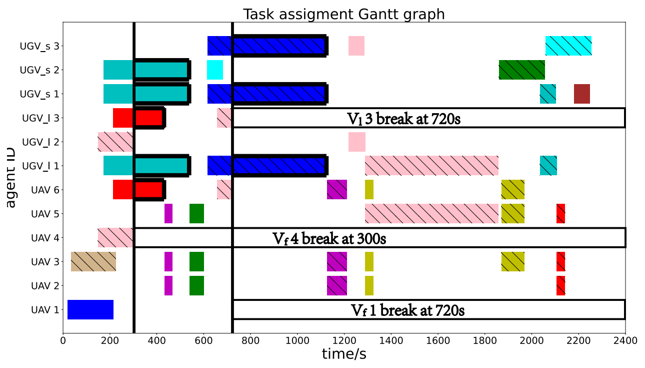


Fig. 13. Gantt graph of the plan execution under agent failures during the execution of task Three. Three failures are highlighted in black. Interrupted subtasks are repeated in the new-assignment. The same legend for subtasks is shared with Fig. 12.

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 $\texttt{wash}_{\texttt{p}_7}$ are reduced to $212s$ from $565s$, as the size of $\texttt{p}_1, \texttt{p}_3, \texttt{p}_6, \texttt{p}_7$ is 37.5% of $\texttt{p}_{10}$. Consider the most complex task Three. The proposed online synchronization method in Sec. VI-A is applied during execution to dynamically accommodate these fluctuations. Compared with the nominal case, the execution result in Fig. 12 shows that the relative ordering constraints are kept and the collaborative tasks are executed successfully. Instead of following strictly the schedule in $J^\star$, the agents rely on the synchronization protocol. For instance, since the task $\texttt{wash}_{\texttt{p}_7}$ is finished earlier, the subsequent task $\texttt{mow}_{\texttt{p}_7}$ can be started immediately, given the relation $\texttt{wash}_{\texttt{p}_7} \leqslant \texttt{mow}_{\texttt{p}_7}$ and $\texttt{wash}_{\texttt{p}_7} \neq \texttt{mow}_{\texttt{p}_7}$. To give another example, vehicles $V_{l_1}, V_{s_1}, V_{s_2}$ are required to execute task $\texttt{repair}_{\texttt{p}_1}$. Its exact schedule in $J^\star$ is $[173s, 746s]$, while actually it is started at $170s$ and finished at $533s$, which is much earlier than scheduled. Afterwards, all vehicles start immediately executing the subsequent subtasks without waiting until $746s$. Similarly, vehicle V_{s_3} arrives at $\texttt{p}_{20}$ at around $72s$, but waits until the subtask $\texttt{wash}_{\texttt{p}_{20}}$ is finished at $718s$, in order to start executing the subtask $\texttt{mow}_{\texttt{p}_{20}}$.

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 $300s$, V_{f_1}, V_{l_3} break down at $720s$ during the execution of task Three. Consequently, as shown in Fig. 13, the execution of subtask $\texttt{wash}_{\texttt{p}_{20}}$ is interrupted at $300s$. As described in Sec. VI-B, 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. Moreover, the subtask $\texttt{wash}_{\texttt{p}_{20}}$ is executed again by vehicles V_{l_3} and V_{f_6} at $720s$. Afterwards, since V_{f_1}, V_{l_3} break down at $720s$, the execution of subtask $\texttt{wash}_{\texttt{p}_{20}}$ is interrupted *again* at $720s$. In the same way, the unfinished subtasks are re-assigned to the remaining agents. Consequently, the subtask $\texttt{wash}_{\texttt{p}_{20}}$ is executed again by vehicles V_{l_1} and V_{f_5} at $720s$ at $1300s$. It is worth noting that the partial ordering constraints are respected at all time during the adaptation. For instance, V_{f_6} cannot execute the subtask $\texttt{scan}_{\texttt{p}_5}$ before $\texttt{repair}_{\texttt{p}_5}$ is finished, as $\texttt{scan}_{\texttt{p}_5} \neq_\varphi \texttt{repair}_{\texttt{p}_5}$ holds. All subtasks are fulfilled at $2141s$, despite of the above contingencies.

TABLE III
SCALABILITY ANALYSES OF THE PROPOSED METHOD

[1] System (V_f, V_s, V_l)	[3] $t_{\texttt{sol}}$ $[s]$	[2] Task $(N_{\texttt{t}}, N_{\texttt{p}})$	[3] $t_{\texttt{sol}}$ $[s]$
$(4,2,2)$	$(0.50, 1.57, 2.84)$	$(1,1)$	$(0.03, 0.17, 0.21)$
$(8,4,4)$	$(0.47, 1.73, 3.91)$	$(2,2)$	$(0.09, 0.53, 1.38)$
$(12,6,6)$	$(0.44, 2.03, 5.53)$	$(3,3)$	$(0.20, 1.21, 4.65)$
$(16,8,8)$	$(0.50, 1.18, 4.08)$	$(4,4)$	$(0.35, 1.09, 13.29)$
$(20,10,10)$	$(0.53, 1.39, 3.67)$	$(5,5)$	$(0.58, 3.26, 36.15)$

[1] The number of different types of agents as system size increases.
[2] The number of transformers and panels that should be serviced for maintenance in the task.
[3] The associated solution time, measured by three time stamps when: the first solution is returned, the solution within 10% margin of the optimal solution is returned, and the optimal solution is returned.

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., 8, 16, 24, 32 and 40; (ii) increased complexity of considered tasks, e.g., by adding more maintenance tasks at different regions.

As summarized in Table III, as the system size is increased from 8 to 40, the computation time to obtain the *first* solution for task Three 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, the subtasks $\texttt{fix-scan}_{\texttt{t}_j}$ and $\texttt{repair-scan}_{\texttt{p}_i}$ within task Two is extended by adding more transformers and panels to maintain. As summarized in Table III right, the computation time of both posets and tasks assignment are increased significantly, as the task becomes more complex. However, the time for the 10% margin of the optimal solution does not monotonically increase. It is worth noting that as the number of tasks increases, the size of NBA explodes such that the NBA pruning steps become infeasible. Instead, the resulting poset of each subtask is composed algorithmically into the complete set of posets. Since the algorithm to compute the upper and lower bounds during the BnB search has a complexity linear to the task complexity, the assignment algorithm can still return the optimal value in proper time given the computed posets. For example, for the extreme case of 45 subtasks when $N_{\texttt{t}} = N_{\texttt{p}} = 5$, the first solution is found within $0.58s$ while the close-to-optimal solution is returned in $3.26s$.

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 [6] 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.

TABLE IV
COMPARISON TO OTHER METHODS.

[1]Method	[2]$t_{\texttt{first}}$ $[s]$	[2]$t_{\texttt{opt}}$ $[s]$	[2]$t_{\texttt{final}}$ $[s]$	[2]$T_{\texttt{obj}}$ $[s]$	[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.

Milp: the optimization-based solution that formulates the assignment problem of posets as a MILP the compute optimal assignment similar to [15], [17], i.e., instead of the search method. The partial relations are formulated as constraints in the program. An open source solver GLPK [43] is used.

Samp: the sampling based method proposed in [8]. 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 [9]. As discussed earlier in Sec. I, the proposed task decomposition strategy only allows completely independent subtasks. Furthermore, since it does not support collaborative actions, collaborative subtasks are decomposed manually.

Two identical 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. As summarized in Table. IV, 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. In addition, since the subtasks are executed in sequence by the solution, 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, which is consistent with the **Milp** method.

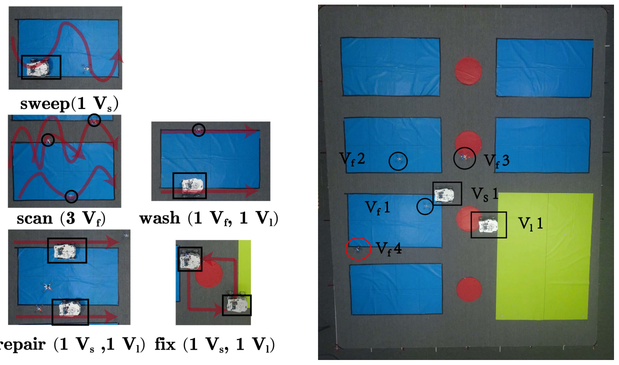


Fig. 14. Snapshots of the task execution. **Left**: Trajectory and collaborators of subtasks. **Right**: UAV 4 (marked in red) is manually stopped to mimic motor failure, while the rest of the team adapts and continues the task execution.

Last but not least, the last column in Table IV compares the number of synchronizations required during execution. Although the same solution is obtained, the **Milp** method requires more synchronization during execution than our method. This is because our method requires only synchronization for relations within the posets, rather than all consecutive subtasks. The **Prod** and **Samp** methods require more synchronization due to its fully sequential execution, while the **Decomp** requires no synchronization as the local subtasks of each agent are independent.

B. Hardware Experiment

For further validation with hardware, a similar set-up as the simulation is built as shown in Fig. 14. 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.

1) Workspace and Task Description: The workspace mimics the PV farm described in the numerical simulation. As showed in Fig. 14, there are 6 PV panels ($\texttt{p}_1$-$\texttt{p}_6$, marked in blue), 4 transformer substations ($\texttt{t}_1$-$\texttt{t}_4$, 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:

$$\begin{aligned}\varphi_4 =& \Diamond(\texttt{repair}_{\texttt{p}_2} \wedge \Diamond\texttt{scan}_{\texttt{p}_2} \wedge \Diamond\texttt{sweep}_{\texttt{p}_2}) \wedge \\ & \Box(\texttt{repair}_{\texttt{p}_2} \rightarrow \neg\texttt{scan}_{\texttt{p}_2}) \wedge \\ & \Box(\texttt{repair}_{\texttt{p}_2} \rightarrow \neg\texttt{sweep}_{\texttt{p}_2}) \\ & \wedge \Diamond\texttt{fix}_{\texttt{t}_1} \wedge \Diamond\texttt{scan}_{\texttt{p}_3} \wedge \Diamond\texttt{wash}_{\texttt{p}_5},\end{aligned} \tag{24}$$

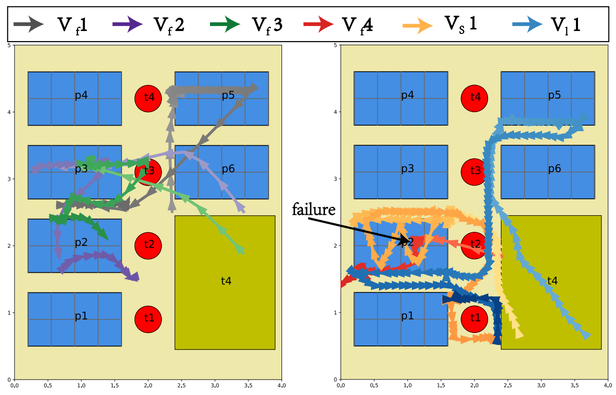


Fig. 15. Agent trajectories during the task execution. **Left**: The normal scenario. **Right**: UAV 4 is manually taken down at $75s$.

of which the description is similar to Task Two in (22).

2) Results: First, we describe the nominal scenario. 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. During execution, it is worth noting that due to collision avoidance and communication delay, the fluctuation in the time of navigation and task execution is *significant*. Consequently, the proposed online synchronization protocol in Sec. VI-A 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. 15. Moreover, to test the online adaptation procedure as described in Sec. VI-B, one UAV V_{f_4} is stopped manually to mimic a motor failure during execution at $75s$, as shown in Fig. 14. 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 $\texttt{scan}_{\texttt{P}_2}$ to continue the overall mission. The resulting trajectory is shown in Fig. 15, where the trajectory of V_{f_4} before failure is shown in red, and the trajectory in blue is another UAV taking over the subtasks. In the end, the complete task is accomplished in $178s$.

IX. 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] T. Arai, E. Pagello, L. E. Parker *et al.*, "Advances in multi-robot systems," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 5, pp. 655–661, 2002.

[2] P. Toth and D. Vigo, "An overview of vehicle routing problems," *The Vehicle Routing Problem*, pp. 1–26, 2002.

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

[4] J. Fink, M. A. Hsieh, and 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.

[5] A. Varava, K. Hang, D. Kragic, and 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.

[6] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT press, 2008.

[7] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.

[8] Y. Kantaros and M. M. Zavlanos, "Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 812–836, 2020.

[9] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems," *The International Journal of Robotics Research*, vol. 37, no. 7, pp. 818–838, 2018.

[10] M. Guo and M. M. Zavlanos, "Multirobot data gathering under buffer constraints and intermittent communication," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1082–1097, 2018.

[11] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local ltl specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.

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

[13] M. Guo and D. V. Dimarogonas, "Task and motion coordination for heterogeneous multiagent systems with loosely coupled local tasks," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 797–808, 2016.

[14] X. Luo, Y. Kantaros, and M. M. Zavlanos, "An abstraction-free method for multirobot temporal logic optimal control synthesis," *IEEE Transactions on Robotics*, 2021.

[15] X. Luo and M. M. Zavlanos, "Temporal logic task allocation in heterogeneous multi-robot systems," *arXiv preprint arXiv:2101.05694*, 2021.

[16] Y. E. Sahin, P. Nilsson, and N. Ozay, "Multirobot coordination with counting temporal logics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1189–1206, 2019.

[17] A. M. Jones, K. Leahy, C. Vasile, S. Sadraddini, Z. Serlin, R. Tron, and 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.

[18] A. Torreño, E. Onaindia, A. Komenda, and M. Štolba, "Cooperative multi-agent planning: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–32, 2017.

[19] M. Gini, "Multi-robot allocation of tasks with temporal and ordering constraints," in *AAAI Conference on Artificial Intelligence*, 2017.

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

[21] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithms for multirobot task assignment with task deadline constraints," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 3, pp. 876–888, 2015.

[22] R. Fukasawa, H. Longo, J. Lysgaard, M. P. De Aragão, M. Reis, E. Uchoa, and R. F. Werneck, "Robust branch-and-cut-and-price for the capacitated vehicle routing problem," *Mathematical Programming*, vol. 106, no. 3, pp. 491–511, 2006.

[23] J. C. Boerkoel Jr, L. R. Planken, R. J. Wilcox, and J. A. Shah, "Distributed algorithms for incrementally maintaining multiagent simple temporal networks," in *International Conference on Automated Planning and Scheduling*, 2013.

[24] E. Nunes and M. Gini, “Multi-robot auctions for allocation of tasks with temporal constraints,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[25] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*. Elsevier, 2004.

[26] S. M. LaValle, *Planning Algorithms*. Cambridge univ. press, 2006.

[27] Y. Q. Chen and Z. Wang, “Formation control: a review and a new consideration,” in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 3181–3186.

[28] Z. Li, Z. Duan, G. Chen, and L. Huang, “Consensus of multiagent systems and synchronization of complex networks: A unified viewpoint,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 1, pp. 213–224, 2009.

[29] M. Mesbahi and M. Egerstedt, *Graph Theoretic Methods in Multiagent Networks*. Princeton University Press, 2010, vol. 33.

[30] M. Lahijanian, S. B. Andersson, and C. Belta, “Temporal logic motion planning and control with probabilistic satisfaction guarantees,” *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 396–409, 2011.

[31] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta, “Formal approach to the deployment of distributed robotic teams,” *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 158–171, 2011.

[32] E. L. Lawler and D. E. Wood, “Branch-and-bound methods: A survey,” *Operations Research*, vol. 14, no. 4, pp. 699–719, 1966.

[33] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, “Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning,” *Discrete Optimization*, vol. 19, pp. 79–102, 2016.

[34] M. Jünger, G. Reinelt, and G. Rinaldi, “The traveling salesman problem,” *Handbooks in Operations Research and Management Science*, vol. 7, pp. 225–330, 1995.

[35] P. Brucker, B. Jurisch, and B. Sievers, “A branch and bound algorithm for the job-shop scheduling problem,” *Discrete Applied Mathematics*, vol. 49, no. 1-3, pp. 107–127, 1994.

[36] D. A. Simovici and C. Djeraba, “Mathematical tools for data mining,” *SpringerVerlag, London*, 2008.

[37] X. Pan, C. S. Han, K. Dauber, and K. H. Law, “A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations,” *Ai & Society*, vol. 22, no. 2, pp. 113–132, 2007.

[38] D. S. Hochba, “Approximation algorithms for np-hard problems,” *ACM Sigact News*, vol. 28, no. 2, pp. 40–52, 1997.

[39] D. P. Bovet, P. Crescenzi, and D. Bovet, *Introduction to the Theory of Complexity*. Prentice Hall London, 1994, vol. 7.

[40] P. Gastin and D. Oddoux, “Fast LTL to büchi automata translation,” in *Computer Aided Verification*. Springer, 2001, pp. 53–65.

[41] R. Sedgewick, *Algorithms in C, part 5: graph algorithms*. Pearson Education, 2001.

[42] R. Lima and E. Seminar, “Ibm ilog cplex-what is inside of the box,” in *Proc. 2010 EWO Seminar*, 2010, pp. 1–72.

[43] A. Makhorin, “Glpk (gnu linear programming kit),” *http://www. gnu. org/s/glpk/glpk. html*, 2008.