Hierarchical Multi-robot Strategies Synthesis and Optimization under Individual and Collaborative Temporal Logic Specifications

Ruofei Bai[1], Ronghao Zheng[1,2*], Yang Xu[1], Meiqin Liu[2,3], Senlin Zhang[1,2]

Abstract

This paper presents a hierarchical framework to solve the multi-robot temporal task planning problem. We assume that each robot has its individual task specification and the robots have to jointly satisfy a global collaborative task specification, both described in linear temporal logic. Specifically, a central server firstly extracts and decomposes a collaborative task sequence from the automaton corresponding to the collaborative task specification, and allocates the subtasks in the sequence to robots. The robots can then synthesize their initial execution strategies based on locally constructed product automatons, combining the assigned collaborative tasks and their individual task specifications. Furthermore, we propose a distributed execution strategy adjusting mechanism to iteratively improve the time efficiency, by reducing wait time in collaborations caused by potential synchronization constraints. We prove the completeness of the proposed framework under assumptions, and analyze its time complexity and optimality. Extensive simulation results verify the scalability and optimization efficiency of the proposed method.

Keywords:
Multi-robot, task planning, linear temporal logic.

1. Introduction

Multi-robot task planning widely exists in many areas, such as smart logistics, autonomous inspection, intelligent manufacturing, etc. It remains a challenge to efficiently formalize and solve the task planning problems under complex task requirements. Formal methods based on model checking theories, such as linear temporal logic (LTL), have draw increasing attention in recent years, due to its user-friendly syntax and expressive power in describing temporally constrained task specifications. The temporal logic specification can be automatically transformed to correct-by-construction controller for robots, providing feedback and guarantees for robot behaviors [1]. Given a global LTL task specification, the execution strategies for robots can be synthesized by searching on a constructed automaton that combines all robots' environment models and the automaton corresponding to the LTL formula.

In this paper, we focus on a situation where each robot has its individual finite LTL task specification, and the robots have to jointly satisfy a global collaborative finite LTL task specification. Each robot's individual tasks can be satisfied by itself, while the completion of collaborative tasks may re-

quire several robots of different types. Prior studies of multi-robot task planning under locally given task specifications rely on either online reactive planning or centralized computation. Most online methods assume that the collaboration requirements are integrated into the local LTL specification, so that the assignments of collaborative tasks are completely or partially known in prior. In addition, the online methods may fail to optimize the performance from a global perspective. The methods based on centralized computation can be applied to more general situations, but usually afford exponential complexity and thus has poor scalability. While most existing temporal logic planning methods focus on the feasibility and the optimization of traveling distance, little attention has been paid to reduce the total time cost for robots. This problem is particularly prominent in temporal logic tasks with potential synchronization constraints, because robots may spend extra wait time in collaborations and thus become inefficient. The optimization of time cost considering wait time in collaborations has not been explicitly evaluated in previous works.

To mitigate the above issues, we propose a hierarchical multi-robot temporal task planning framework to synthesize feasible and optimized task execution strategies for robots. We do not assume the collaborative tasks are allocated in prior, and try to minimize the total time cost for robots.

First, a central server extracts a task sequence which satisfies the collaborative task specification from the related automaton, and allocates the subtasks in the sequence to robots by solving a constructed SMT (Satisfiability Modulo Theories) model. The SMT formulation includes collaboration requirements and synchronization constraints of tasks, and

---
*Corresponding author.

Email address: rzheng@zju.edu.cn (Ronghao Zheng[1,2])

[1]College of Electrical Engineering, Zhejiang University, Hangzhou, 310027, China

[2]State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou, 310027, China

[3]School of Electronics and Information Engineering, Xian Jiaotong University, Xian, 710049, China

also enables the robots to overcome unstable communication links between some task regions in real deployment. All feasible task assignments are iteratively generated and evaluated to continuously optimize the results, which can be easily implemented by the SMT formulation. Then, given the assigned collaborative tasks, the robots modify their individual task specifications by integrating the assigned collaborative tasks as well as the related temporal constraints to them, and then locally constructs its product automaton based on model checking theories. The initial execution strategy can then be obtained by searching the shortest on the automaton, which is optimal from individual robot's perspective. However, the locally obtained initial execution strategies may be time inefficient when collaborating with other robots because of their different arrival time to collaborative tasks. Finally, to optimize the total time cost, we propose a distributed execution strategy adjusting mechanism to iteratively improve the initial execution strategies based on inter-robot communication, which prevents the exponential complexity encountered in traditional methods based on product automaton. Specifically, a token is passed around the robots, and the robot who get the token has the authority to adjust its execution strategy to reduce the time cost. The adjustment is motivated by sequentially traversing all collaborative tasks and greedily reducing wait time in each collaboration. After a valid adjustment, the robot propagates its modified timeline to other robots, which includes its modified arrival time instances to the assigned collaborative tasks.

In our previous work [2], we have proposed a preliminary framework to solve the problem. We extend the work in [2] by introducing pruned local product automatons for robots and operating the execution strategy adjusting mechanism on them, which greatly improves the solving efficiency. Moreover, we design the distributed version of the proposed adjusting mechanism based on message exchanging, to distribute the computational burden among robots. We also formalize the optimization of execution strategies as a mixed integer linear programming (MILP) to compute the optimally execution strategies for robots given the assignments of collaborative tasks. This is used as a baseline method and further verifies the high efficiency and solution quality of the proposed adjusting mechanism. Finally, we prove the completeness of the proposed method under some assumptions.

The contributions of the paper are summarized as follows:

1. we propose a hierarchical multi-robot temporal task planning framework, which efficiently synthesizes execution strategies for robots satisfying both individual and global collaborative LTL specifications, without assuming that the collaborative tasks are pre-assigned to robots;

2. we propose a distributed execution strategy adjusting mechanism, in which the robots can iteratively improve the performance of their execution strategies via inter-agent communication, reducing the inefficient wait time in collaborations caused by potential synchronization constraints;

3. we prove the completeness of the proposed method under assumptions and analyze its time complexity. Extensive simulation experiments verify the scalability and efficiency of the proposed method.

1.1. Related Works

Existing studies in multi-robot task planning under temporal logic constraints fall into two categories: the top-down and the bottom-up patterns. On the one hand, studies in top-down pattern usually assume a globally given temporal logic specification for a team of robots. The task execution strategies of the robots can be obtained by searching for a path on a constructed product automaton that combines all robots' environment models and an automaton corresponding to the LTL formula. The construction of product automaton suffers from the exponential complexity. Also, the transformation from the LTL formula to the corresponding automaton has the exponential complexity w.r.t the length of the LTL formula. Several approaches have been proposed to improve the scalability of the traditional method, such as pruning redundant states of the environment models and the product automaton [3, 4]; sampling-based construction and searching approaches of the product automaton [5, 6, 7, 8]; and decomposition of the globally given task specification [9, 10, 11, 12]. Some abstraction-free temporal logic planning methods based on sampling strategy [13] or reinforcement learning [14] have also been investigated. Besides, some works convert the temporal logic planning problem into a mixed integer linear programming model, and then the off-the-shelf optimizer like Gurobi can be utilized to solve it [15, 16].

On the other hand, studies in bottom-up pattern typically distribute the task specification to individual robots, and the robots jointly satisfy some global task requirements [17]. M. Guo *et al*. [18] investigated the task coordination of loosely coupled multi-agent systems with dependent local tasks. The robots independently synthesize their off-line initial plans first, and then the collaborative actions in each robot's local tasks are performed with other robots' assistance, through online communication and computation. The above method is further modified in [19] to include heterogeneous capabilities of robots and online task swapping mechanism. J. Tumova *et al*. [20] considered a slightly different setting from [18], in which each robot have its local complex temporal logic specifications, including an independent motion specification and a dependent collaborative task specification. An two-phase automata-based method was proposed, where each robot's motion planning is synthesized based on a local automaton in a decentralized way. The local automaton is further pruned by removing states not related to the collaborative tasks. The method then constructs a centralized product automaton of the pruned local automaton, on which the collaboration strategies can be obtained. Despite that the sizes of local automatons have been greatly reduced by the pruning techniques, the method still has exponential complexity w.r.t. the robot number. Y. Kantaros

et al. [21] analyzed a similar situation considered in this paper, that individual robots have their independent individual task specifications, while they have to jointly satisfy a globally given collaborative task specification. Despite that, the collaborative task specification in [21] is restricted to intermittent communication tasks, that require several robots to intermittently gather at some locations to exchange information. The gathering locations are iteratively modified by an on-line optimization procedure to reduce the total travel distance. A common assumption of the aforementioned studies in bottom-up pattern is that the allocation of collaborative tasks are fully or partially known in prior.

Inspired by [21], we extend the global collaborative task specification into finite LTL; and aim to minimize the total time spent to finish all tasks considering inefficient wait time in collaborations, which has not been considered in [21]. We do not assume the allocation of collaborative tasks are explicitly given, and formulate an SMT model to flexibly address the task allocation problems and possible communication limitations in real execution.

The rest of this paper is organized as follows. In Sec. 2, we introduce the linear temporal logic, task specification, and formalize the problem we considered. Then in Sec. 3, we propose the hierarchical multi-robot temporal task planning framework to compute execution strategies for robots. An execution strategy adjusting mechanism is proposed in Sec. 4 to further improve the time efficiency of execution strategies for robots. The results of experiments are analysed in Sec. 5. Finally, we draw our conclusion in Sec. 6.

2. Preliminaries and Problem Formulation

2.1. Linear Temporal Logic

An LTL formula φ over a set of atom propositions $\mathcal{AP}$ are defined according to the following recursive grammar [22]:

$$\varphi ::= \text{true} \mid \pi \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \circ \varphi \mid \varphi_1 U \varphi_2,$$

where true is a predicate true and $\pi \in \mathcal{AP}$ is an atom proposition. The other two frequently used temporal operators $\diamond$ (eventually) and $\square$ (always) can be derived from operator U (until). In this paper, we only consider a kind of finite LTL [23] called LTL_f, which is a variant of original LTL that can be interpreted over finite sequences, and uses the same syntax as original LTL. Moreover, we exclude the $\bigcirc$ (next) operator from the syntax, since it is not meaningful in practical applications [24]. We refer the reader to [22] for the details of LTL semantics.

Given an LTL_f formula φ, a nondeterministic finite automaton (NFA) can be constructed which accepts exactly the sequences that make φ true.

Definition 1 (Nondeterministic Finite Automaton). A nondeterministic finite automaton (NFA) $\mathcal{F}$ is a tuple $\mathcal{F} := \langle Q_F, Q_F^0, \alpha, \delta, Q_F^F \rangle$, where Q_F is a finite set of states; $Q_F^0 \subseteq Q_F$ is a set of initial states; α is a set of Boolean formulas over $\pi \in \mathcal{AP}$; $\delta : Q_F \times Q_F \to \alpha$ is the transition condition of states in Q_F; Q_F^F is a set of accepting states.
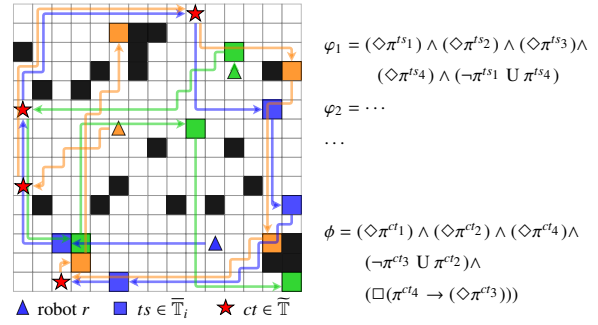

$$\varphi_1 = (\diamond\pi^{ts_1}) \wedge (\diamond\pi^{ts_2}) \wedge (\diamond\pi^{ts_3}) \wedge$$
$$(\diamond\pi^{ts_4}) \wedge (\neg\pi^{ts_1} U \pi^{ts_4})$$
$$\varphi_2 = \cdots$$
$$\cdots$$
$$\phi = (\diamond\pi^{ct_1}) \wedge (\diamond\pi^{ct_2}) \wedge (\diamond\pi^{ct_4}) \wedge$$
$$(\neg\pi^{ct_3} U \pi^{ct_2}) \wedge$$
$$(\square(\pi^{ct_4} \to (\diamond\pi^{ct_3})))$$

$\blacktriangle$ robot r $\quad$ $\blacksquare$ $ts \in \overline{\mathbb{T}}_i$ $\quad$ $\bigstar$ $ct \in \overline{\overline{\mathbb{T}}}$

Figure 1: The simulation results of am experiment with 3 robots.

A finite sequence of states $q_F \in Q_F$ is called a run ρ_F. A run ρ_F is called *accepting* if it starts from one initial state $q_F^0 \in Q_F^0$ and ends at an accepting state $q_F^F \in Q_F^F$. Given a finite sequence $\sigma = \sigma(1)\sigma(2)\ldots\sigma(L-1)\sigma(L)$, we say that σ describes a run ρ_F if $\sigma(i) \vDash \delta(\rho_F(i), \rho_F(i+1))$ for all $r \in [1 : L-1]$, i.e., $\sigma(i)$ is a set of atom propositions or negative propositions, which makes the Boolean formula $\delta(\rho_F(i), \rho_F(i+1))$ become true. Here $[1 : L-1]$ denotes a set of indexes increasing from 1 to $L-1$ by step 1. A finite sequence σ fulfills an LTL_f formula if at least one of its runs is accepting. The minimum requirements for a sequence to describe a run can be represented as an essential sequence, as in the following definition.

Definition 2 (Essential Sequence). Considering a sequence $\sigma = \sigma(1)\ldots\sigma(L)$, it is called *essential* if and only if it describes a run ρ_F in NFA $\mathcal{F}$ and $\sigma(i)\backslash\{\pi\} \nvDash \delta(\rho_F(i), \rho_F(i+1))$ for all $r \in [1 : L-1]$ and $\pi \in \mathcal{AP}$.

2.2. Task Requirements

Consider there is a set of robots $\mathcal{N} := \{1, ..., N\}$ operating in an environment, e.g., as in Fig. 1, which can be represented as a graph $\mathcal{W} = \langle Q, \mathcal{E} \rangle$. Here Q is the set of regions of interest and $\mathcal{E}$ contains the connectivity relations of regions in Q. Each robot $r \in \mathcal{N}$ has its specific capability $c_j \in Cap$ and thus can perform specific actions in the environment $\mathcal{W}$. The set $Cap := \{c_j\}_{j \in \{1,...,|Cap|\}}$ contains all capability types. We define $\mathcal{N}_j := \{r \mid \text{robot } r \text{ has the capability } c_j\}$.

There are several tasks to be completed by the robots, which are distributed in $\mathcal{W}$, as defined in Definition 3.

Definition 3 (Task Requirement). A task in $\mathcal{W}$ is a tuple $ts := \langle \pi^{ts}, \{(c_j, m_j^{ts})\}_{j \in I_{ts}}, q^{ts} \rangle$, where π^{ts} is the unique atom proposition of ts; $q^{ts} \in Q$ corresponds to the region associated with ts; and ts can be completed, i.e, π^{ts} becomes true, if at least m_j^{ts} robots with capability c_j are deployed simultaneously in the region q^{ts} and perform specific actions according to their capabilities, for all $j \in I_{ts}$. Here $I_{ts} \subseteq \{1, ..., |Cap|\}$ collects indexes of the capabilities required by ts.

In this paper, we consider the situation that: (1) each robot $r \in \mathcal{N}$ has its pre-assigned individual task specifications φ_r that can be satisfied by itself. Here φ_r is an LTL_f formula defined over $\overline{\mathbb{T}}_r$, which is robot r's individual task set. For each $ts \in \overline{\mathbb{T}}_r$, it holds that $|I_{ts}| = 1$, $m_j^{ts} = 1$, and $r \in \mathcal{N}_j$;
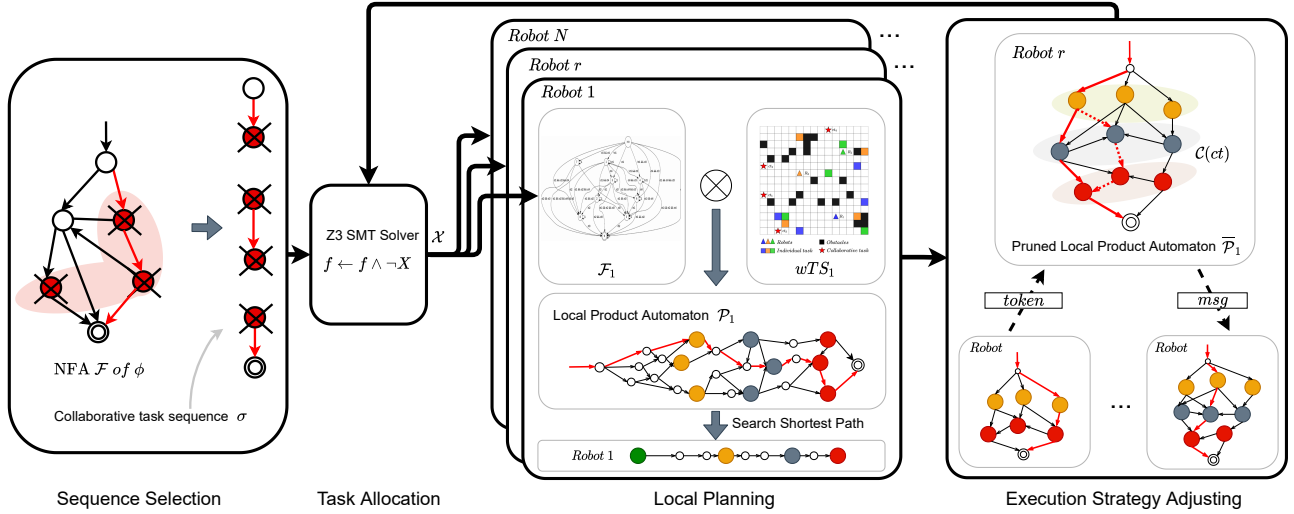
3

Figure 2: The proposed framework for multi-robot temporal logic task planning.

(2) the robots have to jointly satisfy a global LTL_f task specification ϕ defined over a set of tasks $\widetilde{\mathbb{T}}$, in which each task $ct \in \widetilde{\mathbb{T}}$ may has much heavier workloads and requires no less than one robot with several different capabilities. Note that we particularly use ct to represent a task in $\widetilde{\mathbb{T}}$, called *collaborative task*, to distinguish it from tasks in the set $\bigcup_{r \in \mathcal{N}} \overline{\mathbb{T}}_r$ in the following context.

We assume that the executions of individual task specification φ_r are independent of each other, i.e., do not influence the states of other robots. And they are also independent of the execution of collaborative task specification ϕ. More formally, we assume that: $\forall\, i, j \in \mathcal{N}$ and $i \neq j$, $\overline{\mathbb{T}}_i \cap \overline{\mathbb{T}}_j = \emptyset$; and $\forall\, r \in \mathcal{N}$, $\widetilde{\mathbb{T}} \cap \overline{\mathbb{T}}_r = \emptyset$. The assumption is reasonable because if some individual tasks and collaborative tasks are non-independent, these task requirements can be formulated into the global collaborative task specification ϕ.

The mobility and capability of each robot r can be formalized as a weighted transition system.

Definition 4 (Weighted Transition System)**.** A weighted transition system wTS of robot r is a tuple $wTS_r = \langle Q_r, q_r^0, \rightarrow_r, \omega_r, \mathcal{AP}_r, L_r \rangle$, where $Q_r \subseteq Q$ is a finite set of states corresponding to regions in $\mathcal{W}$; q_r^0 is the initial state; $\rightarrow_r \subseteq \mathcal{E}$ contains all pairs of adjacent states; $\omega_r : Q_r \times Q_r \rightarrow \mathbb{R}^+$ is a weight function measuring the cost for each transition; $\mathcal{AP}_r$ is the set of atom propositions related to the tasks in $\mathcal{W}$; $L_r : Q_r \rightarrow 2^{\mathcal{AP}_r}$ is a labeling function, and satisfies that (1) $\forall\, ts \in \overline{\mathbb{T}}_r$, $\pi^{ts} \in L_r(q^{ts})$; (2) $\forall\, ct \in \widetilde{\mathbb{T}}$, $\pi^{ct} \in L_r(q^{ct})$ iff $\exists j \in I_{ct}, r \in \mathcal{N}_j$.

A task execution strategy τ_r for each robot r is a *walk* in the wTS_r. The execution of τ_r will induce a sequence σ_i of sets of atom propositions, which follows the labeling function L_r.

2.3. Problem Formulation

Given task execution strategy τ_r for each robot r, the robots execute their plans asynchronously and synchronize with other assistant robots when performing the collaborative tasks. The total time cost to satisfy all task requirements is defined as $T^{\text{colla}} = \sum_{r \in \mathcal{N}} T_r^{\text{colla}}$, where T_r^{colla} denotes the amount of time robot r spends to execute τ_r, considering the wait time in each collaboration due to potential synchronization constraints. The problem to minimize the total time can be formalized as:

Problem 1. Consider a set $\mathcal{N}$ of robots operate in an environment $\mathcal{W}$, and suppose that the mobility and capability of each robot r is modeled as a weighted transition system wTS_r. Given individual LTL_f task specification φ_r for each robot r and a global collaborative task specification ϕ, find task execution strategy τ_r in wTS_r for each robot r which satisfies:

1. the execution of τ_r satisfies individual task specifications φ_r;
2. the joint behaviors of robots r satisfies collaborative task specification ϕ;
3. the total time cost T^{colla} is minimized.

3. Multi-Robot Temporal Task planning

In this section, we propose the multi-robot temporal task planning framework as shown in Fig. 2 to solve Problem 1. The proposed framework firstly finds a sequence of collaborative tasks which satisfies the collaborative task specification and allocates the tasks in the sequence to robots. Then each robot locally synthesizes its initial task execution strategy which satisfies the individual task specification and the assigned collaborative tasks requirements. Finally, a distributed execution strategy adjusting mechanism is proposed in Sec. 4 to optimize the initial task execution strategies. All feasible task assignments are iteratively generated and evaluated to continuously optimize the results. The overall multi-robot temporal task planning framework is shown in Alg. 1.

4

3.1. Selection and Decomposition of Execution Sequence

To find a collaborative task sequence satisfying the collaborative task specification ϕ, a three-step scheme is used:

1. prune the NFA $\mathcal{F}$ of ϕ by removing all impossible transitions that require assistance beyond what the robot set $\mathcal{N}$ can provide.
2. identify the decomposition states in the pruned $\mathcal{F}$ by utilizing the decomposition algorithm proposed in [9].
3. select an essential sequence which describes an accepting run in $\mathcal{F}$, and divide it into independent subsequences by the decomposition states along the run.

The way to prune $\mathcal{F}$ is straightforward by checking the atom propositions required to be true in each transition condition and the details are omitted here.

The essential sequence σ that describes an accepting run ρ_F in the pruned NFA $\mathcal{F}$ is actually a feasible collaborative task sequence that satisfies ϕ. Given an accepting run ρ_F of $\mathcal{F}$ and its corresponding essential sequence σ, the *decomposition states* along the run ρ_F can decompose σ into S (≥ 1) subsequences, i.e., $\sigma = \sigma^1; \sigma^2; \dots; \sigma^S$. Let $\mathcal{S} := [1 : S]$ denote the set of indexes. Such decomposition is proved to have two properties in [10]: (1) **Independence**. Execution of each subsequence σ^k will not violate another σ^j, $\forall k, j \in \mathcal{S}$ and $k \neq j$; (2) **Completeness**. The completion of all subsequences σ^k implies the completion of σ. The detailed definition of decomposition states can be found in Definition 9 and Theorem 2 of [9].

The m-th element $\sigma^k(m)$ in σ^k is defined as $\sigma^k(m) := \{ct_{(m,j)}^k\}_{j \in [1:|\sigma^k(m)|]}$, in which $ct_{(m,j)}^k \in \overline{\mathbb{T}}$ denotes the j-th task in the m-th element of σ^k. For notation simplicity, we use ct and π^{ct} interchangeably afterward. Note that we only consider the positive atom propositions in the essential sequence σ, and assume that the negative propositions are checked and guaranteed in online execution. Although it may impose extra implicit communications in real deployment, it can greatly simplify the task planning procedure. Similar assumption can also be found in [9, 25].

The execution of each sub-sequence σ^k is independent, while the execution of collaborative tasks within each σ^k must satisfy the following temporal constraints according to the accepting condition of $\mathcal{F}$.

- **Synchronization Constraints**. $\forall\ ct_{(m,j)}^k$, $ct_{(m,j')}^k \in \sigma^k(m)$, $j \neq j'$, $ct_{(m,j)}^k$ and $ct_{(m,j')}^k$ need to be executed synchronously;

- **Ordering Constraints**. $\forall\ ct_{(m,j)}^k$, $ct_{(m',j')}^k \in \sigma^k, m < m'$, $ct_{(m',j')}^k$ must be executed after $ct_{(m,j)}^k$.

The above decomposition of essential sequence relaxes the temporal constraints between the tasks in the essential sequence, which may help to reduce the unnecessary communication as well as the total time cost in real deployment, as stated in Remark 2 of Sec. 3.3.

The selection of the collaborative task sequence σ has implicit influences on the performance of the final results. It is time-consuming to investigate all feasible σ. Intuitively, the essential sequence related to an accepting run ρ_F in $\mathcal{F}$ which has less coupling in each transition, contains as many decomposition states as possible, and has the smallest possible length, may help to improve the execution performance. To reduce the complexity, in the sequel we just select the essential sequence σ that describes one of the shortest accepting run ρ_F in the pruned $\mathcal{F}$ to be the collaborative task sequence, and decompose it by the decomposition states along the run ρ_F. The simulation results illustrate that the proposed method can find solutions of high quality in practice under the selected σ.

3.2. SMT-Based Collaborative Task Allocation

After obtaining the decomposed collaborative task sequence $\sigma = \sigma^1; \sigma^2; \dots; \sigma^S$ and the related accepting run ρ_F of $\mathcal{F}$, we now consider the allocation of collaborative tasks in the sequence σ to satisfy the amount and types of assistance needed to complete each task, while following the temporal constraints implied by ϕ.

For notation simplicity, we give all $ct_{(m,j)}^k \in \sigma^k$ an index l, defined as $ct_l^k := ct_{(m,j)}^k$, where $l = \sum_{i=1}^{m-1} |\sigma^k(i)| + j$, $\forall\ m \in \left[1 : |\sigma^k| \right]$ and $k \in \mathcal{S}$. Then the l-th task in σ^k can be referred as ct_l^k. We define a set of Boolean variables $\mathcal{X}_i := \{x_r^{(k,l)} | ct_l^k \in \sigma^k, k \in \mathcal{S}\}$ for each robot r to indicate the task assignment results. A true $x_r^{(k,l)}$ implies that robot r is assigned to complete the task ct_l^k.

Now we construct the SMT model of the task allocation problem.

(1) **Collaboration Requirements.** A feasible task assignment must satisfy the amount and types of assistance needed to complete each collaborative task, as the following constraints: For each σ^k, $\forall\ ct_l^k \in \sigma^k$,

$$\sum_{r \in \mathcal{N}} \mathbf{1}\left(x_r^{(k,l)} \bigwedge r \in \mathcal{N}_j \right) \geq m_j^{ct}, \forall j \in I_{ct}$$

is satisfiable, where $\mathbf{1}(\cdot)$ is an indicator function defined as $\mathbf{1}(\text{true}) = 1$ and $\mathbf{1}(\text{false}) = 0$.

(2) **At Most One at a Time.** Each robot cannot participate in two distinct collaborative tasks which are executed synchronously. The constraints can be captured by: For each σ^k, $\forall\ ct_{l_1}^k, ct_{l_2}^k \in \sigma^k(m), l_1 \neq l_2$,

$$\neg x_r^{(k,l_1)} \bigvee \neg x_r^{(k,l_2)}, \forall\ r \in \mathcal{N}$$

is satisfiable.

(3) **Communication Reduction.** The intersection of two sets of robots that complete two consecutive collaborative tasks is not empty: For each σ^k,

$$\bigvee_{r \in \mathcal{N}} \left(\vee_{l \in \{l_{m-1}^k+1, \dots, l_m^k\}} x_r^{(k,l)} \bigwedge \vee_{l \in \{l_m^k+1, \dots, l_{m+1}^k\}} x_r^{(k,l)} \right)$$

is satisfiable, where $l_m^k = \sum_{j=1}^{m} |\sigma^k(j)|$, $\forall\ m \in [1 : |\sigma^k| - 1]$.

When the communication is limited in the environment, the above constraints (3) can be applied to guarantee that

each robot r only needs to communicate with its assistant robots $j \in \mathcal{R}(ct) \backslash \{r\}$ when executing the assigned collaborative task ct. The function $\mathcal{R} : ct_l^k \rightarrow 2^{\mathcal{N}}$ maps ct_l^k to a set of robots that perform the task. The constraint (3) requires that $\mathcal{R}(\sigma^k(m)) \bigcap \mathcal{R}(\sigma^k(m+1)) \neq \emptyset, \forall\, m \in \left[1 : |\sigma^k| - 1\right]$. The robot $r \in \mathcal{R}(\sigma^k(m)) \bigcap \mathcal{R}(\sigma^k(m+1))$ behaves like a coordinator to guarantee the execution order of tasks in $\sigma^k(m)$ and $\sigma^k(m+1)$. Here the function $\mathcal{R}$ is reloaded as $\mathcal{R}(\sigma^k(m)) = \bigcup_{ct \in \sigma^k(m)} \mathcal{R}(ct)$ according to SMT constraint (2) and the synchronization constraints in Sec. 3.1.

Remark 1. The SMT constraints (3) can be selectively applied to some pairs of consecutive collaborative tasks, whose locations may be far apart so that it is difficult to ensure the connectivity of the communication links between these regions.

The overall SMT formula f is a conjunction of all Boolean expressions of the above constraints (1)(2)(3). We define $X = \bigwedge_{r \in \mathcal{N}} \left(\bigwedge_{x_r^{(k,l)} \in \mathcal{X}_i} x_r^{(k,l)} \right)$ as the Boolean formula corresponds to a feasible task assignment $\{\mathcal{X}_r\}_{r \in \mathcal{N}}$ which satisfies f. By utilizing an SMT solver, we can iterate all valid assignments by adding the negation of current X into f, i.e., $f = f \wedge \neg X$. Therefore the solver will not return the same task assignment in the subsequent iterations, as shown in Line 15 of Alg. 1. Each feasible task assignment $\{\mathcal{X}_r\}_{r \in \mathcal{N}}$ will be passed into the subsequent local planning procedure to further investigate its performance.

Here we come up with a filtering strategy to fast filter the non-optimal task assignments. The efficiency of the filtering strategy will be shown in Sec. 5.

Filtering Strategy: If $\exists\, m < n$ and $m, n \in \mathbb{N}^+$, such that $\forall r \in \mathcal{N}, \mathcal{X}_r^n \geq \mathcal{X}_r^m$, then f can be directly modified as $f \wedge \neg X^n$ without proceeding the subsequent investigation procedures, where $\mathcal{X}_r^m, \mathcal{X}_r^n$ are the feasible task assignments for robot r in m-th and n-th iterations, respectively. Here $\mathcal{X}_r^n \geq \mathcal{X}_r^m$ means that $\forall\, ^n x_r^{(k,l)} \in \mathcal{X}_r^n$ and $^m x_r^{(k,l)} \in \mathcal{X}_r^m, \mathbf{1}(^n x_r^{(k,l)}) \geq \mathbf{1}(^m x_r^{(k,l)})$.

Given the obtained task assignment results, each robot will combine the assigned collaborative tasks with its local task specification to synthesize its initial execution strategy. The details will be discussed in the next subsection.

3.3. Local Plan Synthesis With Collaborative Tasks

To synthesize the local task execution strategy, the key problem is how to construct a local LTL_f formula that captures the local individual task specification φ_r, as well as the temporal constraints of the assigned collaboration tasks.

Given one feasible task assignments $\mathcal{X}$, we construct set $\widetilde{\mathbb{T}}_r^k := \left\{ ct_l^k \mid x_r^{(k,l)} \text{ is true} \right\}$ for all k and r. Let $\widetilde{\mathbb{T}}_r := \bigcup_{k \in \mathcal{S}} \widetilde{\mathbb{T}}_r^k$. We sort all tasks in $\widetilde{\mathbb{T}}_r$ in the increasing order of indexes k and l.

The modified local LTL_f formula $\widetilde{\varphi}_r$ for each robot r is the conjunction of φ_r and ϕ_r, i.e., $\widetilde{\varphi}_r = \varphi_r \wedge \phi_r$. Here ϕ_r is the LTL_f formula that captures the requirements of tasks $ct \in \widetilde{\mathbb{T}}_r$. There are two steps to construct ϕ_r and $\widetilde{\varphi}_r$:

Step 1: Formalize ordering constraints within each independent subsequence. We initialize the collaborative task specification ϕ_r^k corresponds to $\widetilde{\mathbb{T}}_r^k$ as $\phi_r^k = \Diamond(ct_{l_1}^k)$, where $ct_{l_1}^k$ is the first task in the sorted $\widetilde{\mathbb{T}}_r^k$. Then we use $ct_{l_m}^k \wedge \Diamond(ct_{l_{m+1}}^k)$ to iteratively substitute $ct_{l_m}^k$ in ϕ_r^k, where $ct_{l_m}^k$ and $ct_{l_{m+1}}^k$ are two consecutive tasks in sorted $\widetilde{\mathbb{T}}_r^k$, and $l_m < l_{m+1}$.

The asynchronous execution of several independent subsequences may stick into deadlock in real deployment, due to the different execution order in each robot's local execution strategy. We use Step 2 to prevent the deadlock.

Step 2: Determine the execution order of each independent subsequence. For each robot r, we sort ϕ_r^k in Step 1 in increasing order of index k, $\forall\, k \in \mathcal{S}$. Starting from $k = 1$ to $S - 1$, we iteratively replace $ct_{l_{max}}^k$ with $ct_{l_{max}}^k \wedge \Diamond(\phi_r^{k+1})$, where $ct_{l_{max}}^k$ is the last collaborative task in sorted $\widetilde{\mathbb{T}}_r^k$. Finally, $\phi_r = \phi_r^1$, and $\widetilde{\varphi}_r = \varphi_r \wedge \phi_r$.

Note that the Step 2 actually imposes ordering constraints among independent subsequences, which seems to make the decomposition of the task sequence σ in Sec. 3.1 meaningless. However, we will show it is not the case.

Remark 2. The decomposition of essential sequence in $\mathcal{F}$ reduces the amount of communication in real deployment. For example, assume that there exists an essential sequence $\sigma = \{ct_1^{k_1}\}\{ct_1^{k_2}\}$ in which $\sigma^{k_1} = \{ct_1^{k_1}\}$ and $\sigma^{k_2} = \{ct_1^{k_2}\}$ are two independent subsequences. If $\mathcal{R}(ct_1^{k_1}) \cap \mathcal{R}(ct_1^{k_2}) = \emptyset$, then the two tasks can be executed asynchronously. In contrast, without the decomposition of the essential sequence, all robots in $\mathcal{R}(ct_1^{k_2})$ have to wait the completion of $ct_1^{k_1}$, which increases the total time cost.

Given the wTS_r and $\mathcal{F}_r$ of $\widetilde{\varphi}_r$, each robot r can find its initial locally optimal run ρ_r on a product automaton $\mathcal{P}_r$ as defined in Definition 5.

Definition 5 (Product Automaton). The product automaton of wTS_r and $\mathcal{F}_r$ is a tuple $\mathcal{P}_r = wTS_r \otimes \mathcal{F}_r = \langle Q_P, Q_P^0, \rightarrow_P, \omega_P, Q_P^F \rangle$, where $Q_P = Q_r \times Q_F$; $Q_P^0 = q_i^0 \times Q_F^0$; $\rightarrow_P \subseteq Q_P \times Q_P$, which satisfies that $\forall\, (q_P, q_P') \in \rightarrow_P$, it holds that $(\Pi_r q_P, \Pi_r q_P') \in \rightarrow_i$ and $L_r(\Pi_r q_P) \vDash \delta(\Pi_F q_P, \Pi_F q_P')$. Here Π_r and Π_F represent the projections into the state spaces of wTS_r and $\mathcal{F}_r$, respectively; $\omega_P(q_P, q_P') = \omega_i(\Pi_r q_P, \Pi_r q_P')$; $Q_P^F \subseteq Q_r \times Q_F^F$.

To identify the states in $\mathcal{P}_r$ related to the execution of collaborative tasks $ct \in \widetilde{\mathbb{T}}_r$, we define the collaborative state in Def. 6.

Definition 6 (Collaborative State). For $\forall\, q_P \in Q_P$, $q_P \in C(ct)$ iff the following two conditions hold. Here $C(ct)$ is a collection of all collaborative states in $\mathcal{P}_r$ of collaborative task ct.

1. $ct \in L_r(\Pi_r q_P)$;
2. $\exists q_P' \in Q_P, q_P \in \delta(q_P', ct)$, and $\Pi_{\mathcal{F}} q_P' \neq \Pi_{\mathcal{F}} q_P$.

The second condition ensures that the robot plans to perform task ct at state q_P rather than just going through that state.

We utilize the Dijkstra algorithm to search for the shortest accepting run ρ_r in $\mathcal{P}_r$ for each robot $r \in \mathcal{N}$. The obtained optimal accepting run ρ_r satisfies both task specifications and motion and task performing capabilities of robot r. The initial task execution strategy τ_r of robot r can be obtained by projecting the run ρ_r into the state space of wTS_r, i.e., $\tau_r = \Pi_r \rho_r$.

Performing Execution Strategies: Given the task execution strategy τ_r for each robot $r \in \mathcal{N}$, robot r moves in the environment to perform tasks following the order in τ_r. (1) If robot r reaches a state $q_P \in Q_P$ that holds $L_r(q_P) \subseteq \overline{\mathbb{T}}_r$, it performs all tasks in $L_r(q_P)$ individually before going to the next region; (2) if robot r reaches a state $q_P \in Q_P$ and $\exists ct \in \widetilde{\mathbb{T}}_r, q_P \in C(ct)$, then robot r has to wait for the arrival of all other robots in $\mathcal{R}(ct)\backslash\{r\}$ and then perform the task together. Once robot r moves to the last state in τ_r and completes the potential tasks there, it completes its tasks.

When the robots execute the initially obtained task execution strategies, there may exist extra wait time in each collaboration, caused by different arrival times of robots to one collaborative task. This is because the robots synthesize their execution strategies independently without considering others. In the next section, the initial task execution strategies will be optimized.

Proposition 1. *(**Completeness**) If there exists a solution for Problem 1, then the proposed method can find task execution strategies $\{\tau_r\}_{r \in \mathcal{N}}$, when executed asynchronously and following the synchronization constraints in collaborations, can satisfy the individual task specification φ_r for each robot $r \in \mathcal{N}$ and the global collaborative task specification ϕ.*

Proof. If there exists a solution $\{\tau_r\}_{r \in \mathcal{N}}$ for Problem 1, it satisfies that: (1) the joint behaviors of the robots w.r.t the collaborative tasks must follow an essential sequence σ in the prune NFA $\mathcal{F}$ as in Sec. 3.1 which describes an accepting run; (2) the individual execution of each τ_r must induce an essential sequence of an accepting run ρ_r in $\mathcal{F}_r$ of $\widetilde{\varphi}_r$ in Sec. 3.3, since $\mathcal{F}_r$ contains all possible task sequences to satisfy φ_r, and the constraints about the assigned collaborative tasks. It is obvious that any existing solutions will also present in the searching space of the proposed method.

Then we prove that the proposed way for robots to perform their execution strategies will describe accepting runs on the NFAs of individual LTL$_f$ task specifications and also on the NFA of collaborative LTL$_f$ task specification. First, in the proposed framework, robots execute their execution strategies asynchronously and only synchronize with each other when performing collaborative tasks. In contrast, traditional method assumes synchronization of all robots in each step. According to Def. 3, robots only need to synchronize with each other when making the transition condition become true. During the robots moving between regions without performing actions, there is no enabled atom propositions. This is also valid because we exclude the next operator from finite LTL, so there always exist a feasible self loop at each state in $\mathcal{F}_r$ whose transition condition always being true.

Second, in the induced task sequence while robot perform-

Algorithm 1: Framework

Input : $\{\varphi_r\}_{r \in \mathcal{N}}, \phi, \mathcal{W}, \widetilde{\mathbb{T}}, \{\overline{\mathbb{T}}_r\}_{r \in \mathcal{N}}$
Output : $\{\tau_r\}_{r \in \mathcal{N}}$

1 Construct wTS_r for $r \in \mathcal{N}$.
2 Convert ϕ to its corresponding NFA $\mathcal{F}$.
3 Prune $\mathcal{F}$ and identify the decomposition states in $\mathcal{F}$. Select and decompose the collaborative task execution sequence $\sigma = \sigma^1; \sigma^2; \ldots; \sigma^S$ of an accepting run ρ_F in $\mathcal{F}$.
4 Construct the SMT-based task allocation model f.
5 **while** SAT(f) **do**
6     $\mathcal{X} \leftarrow$ SMT-solver(f)
7   Pass $\mathcal{X}$ to each robot.
8   **for** $r \in \mathcal{N}$ **do**
9   Construct LTL$_f$ formula $\phi_r \leftarrow$ that captures the assigned $ct \in \widetilde{\mathbb{T}}_r$ according to $\mathcal{X}$ and the temporal constraints
10        $\widetilde{\varphi}_r \leftarrow \varphi_r \bigwedge \phi_r$.
11        $\mathcal{P}_r \leftarrow wTS_r \bigotimes \mathcal{F}_r$ // $\mathcal{F}_r$ is the NFA of $\widetilde{\varphi}_r$
12        $\rho_r \leftarrow$ local optimal path of $\mathcal{P}_r$
13   Construct pruned local product automaton $\overline{\mathcal{P}}_r, \forall r \in \mathcal{N}$.
14   StrategyAdjustingMechanism$\left(\{\overline{\mathcal{P}}_r\}_{r \in \mathcal{N}}, \{\rho_r\}_{r \in \mathcal{N}}\right)$
15     $f \leftarrow f \bigwedge \neg X$
16 **return** $\tau_r \leftarrow \Pi_r \rho_r$ *for each robot* $r \in \mathcal{N}$.

---

ing their strategies, the individual tasks and collaborative tasks may be performed alternately. This is also valid because we have assumed that the execution of individual task specification are independent with each other and also with the collaborative task specification, so when robots execute individual tasks, the state of collaborative specification remains on the self-loop, with no proposition of collaborative tasks enabled. The satisfaction of individual task specification is also similar to the collaborative specification.

To conclude, if there exists a solution to Problem 1, it can be found by the proposed method. And performing the execution strategies by robots will satisfy both individual and collaborative task specifications. ∎

4. Optimization of Task Execution Strategies

In this section, we propose two methods to optimize the initial execution strategies obtained in Sec. 3.3, given the assignments of collaborative tasks. We first prune each robot's local product automaton, so that only the states related to the execution of collaborative tasks are preserved. Based on the pruned local product automaton, we then build a baseline MILP model to calculate the optimal execution strategies w.r.t the total time cost for robots. Finally, we extend the execution strategy adjusting mechanism proposed in [2] by operating it on the pruned product automatons, and develop a distributed version to better distribute the computation burden. Experimental results about these two methods will be shown in Sec. 5.

4.1. Pruning Local Product Automaton

To optimize time efficiency of the obtained execution strategies, we only care about the states related to the ex-

ecution of collaborative tasks. The reason is that the total time cost of robots depends on: (1) the time each robot spends to perform its execution strategies without considering wait time in collaborations; (2) the extra wait time each robot spends in each collaboration.

The pruned local product automaton of $\mathcal{P}_r$, denoted by $\overline{\mathcal{P}}_r$ only maintains three types of states: 1) initial states $q_P^0 \in Q_P^0$; 2) accept states $q_P^F \in Q_P^F$; 3) collaborative states $q \in C(ct)$ for all $ct \in \widetilde{\mathbb{T}}_r$. We define the pruned local product automaton as follows:

Definition 7. A pruned local product automaton $\mathcal{P}_r$ of robot r is a tuple $\overline{\mathcal{P}}_r = \langle Q_{\overline{P}}, Q_{\overline{P}}^0, \to_{\overline{P}}, \omega_{\overline{P}}, Q_{\overline{P}}^F \rangle$, where:

1. $Q_{\overline{P}} \in \left(\bigcup_{ct \in \widetilde{\mathbb{T}}_r} \{C(ct)\} \right) \bigcup Q_{\overline{P}}^0 \bigcup Q_{\overline{P}}^F$;
2. $Q_{\overline{P}}^0 = Q_P^0$;
3. $\to_{\overline{P}} \subseteq Q_{\overline{P}} \times Q_{\overline{P}}$, and $\forall (q_{\overline{P}}, q'_{\overline{P}}) \in \to_{\overline{P}}$, there exists a path ρ from $q_{\overline{P}}$ to $q'_{\overline{P}}$ in $\mathcal{P}_i$. There are three types of edges in $\to_{\overline{P}}$. $\forall (q_{\overline{P}}, q'_{\overline{P}}) \in \to_{\overline{P}}$:
   (a) if $q_{\overline{P}} \in C(ct), q'_{\overline{P}} \in C(ct'), ct, ct' \in \widetilde{\mathbb{T}}_r$, then ct must be the previous elements of ct' in $\widetilde{\mathbb{T}}_r$;
   (b) else, if $q_{\overline{P}} \in Q_{\overline{P}}^0$ and $q'_{\overline{P}} \in C(ct)$, then ct must be the first element in $\widetilde{\mathbb{T}}_r$;
   (c) else, if $q_{\overline{P}} \in C(ct)$ and $q'_{\overline{P}} \in Q_{\overline{P}}^F$, then ct must be the last element in $\widetilde{\mathbb{T}}_r$.
4. $\omega_{\overline{P}}(q_{\overline{P}}, q'_{\overline{P}}) = \sum_{k=1}^{|\rho|-1} \omega_P(\rho(k), \rho(k-1))$, here ρ is the shortest directed path connecting $q_{\overline{P}}$ and $q'_{\overline{P}}$ in $\mathcal{P}_r$;
5. $Q_{\overline{P}}^F = Q_P^F$.

The pruned product automaton $\overline{\mathcal{P}}_r$ has a hierarchy structure, in which each level contains all collaborative states corresponds to one specific collaborative tasks in $\widetilde{\mathbb{T}}_r$ (except the first and the last levels), and the order of each level follows the order of corresponding tasks in $\widetilde{\mathbb{T}}_r$.

Proposition 2. *The pruned local product automaton $\overline{\mathcal{P}}_r$ can be constructed in $O(\Delta^2 \cdot |\widetilde{\mathbb{T}}_r| \cdot (E \cdot lgE))$, where Δ is the magnitude of collaborative states in $\mathcal{P}_r$ for each collaborative task $ct \in \widetilde{\mathbb{T}}_r$, and $E \cdot lgE$ is the time complexity of using Dijkstra algorithm on $\overline{\mathcal{P}}_r$ to search for the shortest path from $q_{\overline{P}}$ to $q'_{\overline{P}}$ for an edge $(q_{\overline{P}}, q'_{\overline{P}})$ in Def. 7, and E is the number of edges in $\mathcal{P}_r$.*

4.2. MILP-based Baseline Method

In this section, we aim to find the optimal execution strategies for robots to satisfy all task requirements while minimizing the total time cost. We formalize the optimization problem as an MILP model. The solution to the MILP model is a set of accepting paths on robots' pruned local product automatons, from which the execution strategies for robots can then be extracted. The MILP model can be solved by off-the-shell solving tools like Gurobi [26].

The MILP model is based on the pruned local product automaton. First, we define a set of Boolean variable $\mathcal{Y}_r = \{y_r^{(i,j)} | (q_i, q_j) \in \to_{\overline{P}}\}$ for each robot $r \in \mathcal{N}$. The variable $y_r^{(i,j)}$

is true indicating that the corresponding edge (q_i, q_j) is selected in the accepting run of robot r. The final execution strategy of robot r can be generated according to the value of $\mathcal{Y}_r$. The values of variables in $\mathcal{Y}_r$ are guaranteed to generate a feasible accepting run on $\overline{\mathcal{P}}_r$ by applying following constraints:

$$y_r^{(i,j)} = 1 \ or \ 0, \forall \ (q_i, q_j) \in \to_{\overline{P}}, r \in \mathcal{N}; \tag{1}$$

$$\sum_{q_k \in Pre(i)} y_r^{(k,i)} = \sum_{q_j \in Sub(i)} y_r^{(i,j)} \le 1, \forall \ q_i \in Q_{\overline{P}} \backslash (Q_{\overline{P}}^0 \cap Q_{\overline{P}}^F); \tag{2}$$

$$\sum_{q_i \in Q_{\overline{P}}^0} \sum_{q_j \in Sub(i)} y_r^{(i,j)} = 1; \tag{3}$$

$$\sum_{q_i \in Q_{\overline{P}}^F} \sum_{q_k \in Pre(i)} y_r^{(k,i)} = 1. \tag{4}$$

Here $Pre(i)$ and $Sub(i)$ denote the set of predecessors and successors of state q_i in $\overline{\mathcal{P}}_r$ respectively.

By applying the above constraints (1) − (4), the variables $y_r^{(i,j)} \in \mathcal{Y}_r$ that take true can generate an accepting path on the prune local product automaton $\overline{\mathcal{P}}_r$.

Then we formalize the wait time of robots in collaborations. For each $ct_l^k \in \widetilde{\mathbb{T}}_r$, we define two types of auxiliary variables $t_r^{(k,l)}$ and $delay_r^{(k,l)}$ respectively. The variable $delay_r^{(k,l)}$ equals to the total wait time for robot r after the execution of ct_l^k. Initially, all these variables are assigned 0. We have

$$t_r^{(k,l)} = \sum_{i \in Pre(j)} y_r^{(i,j)} \cdot (t_r^{(k',l')} + \omega_{\overline{P}}(q_i, q_j)), \tag{5}$$

where $ct_{l'}^{k'}$ is the previous task of ct_l^k in $\widetilde{\mathbb{T}}_r$, and $q_i \in C(ct_{l'}^{k'})$, $q_j \in C(ct_l^k)$. According to the Def. 7, there exists an edge $(q_i, q_j) \in \to_{\overline{P}}$, and $\omega_{\overline{P}}(q_i, q_j)$ is the transition time between the two states. The variable $t_r^{(k,l)}$ actually represents robot r' ideal arrival time to the region of task ct_l^k. Note that for the first element in $\widetilde{\mathbb{T}}_r$, it has no previous element, so it only takes the second term $\omega_{\overline{P}}(q_i, q_j)$ into account.

We define a set of variables $\mathcal{Z} = \{z_l^k | \exists ct_l^k \in \widetilde{\mathbb{T}}, z_l^k = \max_{r \in \mathcal{N}}(t_r^{(k,l)} + delay_r^{(k',l')})\}$ to represent the latest arrival time for each collaborative task. Here $delay_r^{(k',l')}$ is the total wait time for robot r after the execution of $ct_{l'}^{k'}$. We have the constraints that:

$$delay_r^{(k,l)} = z_l^k - t_r^{(k,l)}, \forall ct_l^k \in \mathbb{T}_r. \tag{6}$$

Finally, the objective function is defined as:

$$J = \sum_{r \in \mathcal{N}} t_r^{(k_e, l_e)} + delay_r^{(k_e, l_e)}, \tag{7}$$

where k_e, l_e corresponds to $ct_{l_e}^{k_e}$, the last element in $\widetilde{\mathbb{T}}_r$.

After solving the above MILP model, we can get each robot's execution strategy according to the value of variables in $\mathcal{X}_r$.

Time Complexity: the number of variables in the MILP model is $O(N \cdot \Delta^2 \cdot |\widetilde{\mathbb{T}}_r| + |\widetilde{\mathbb{T}}|)$. For each robot $r \in \mathcal{N}$, the hierarchy structured pruned local product automaton $\overline{\mathcal{P}}_r$ has
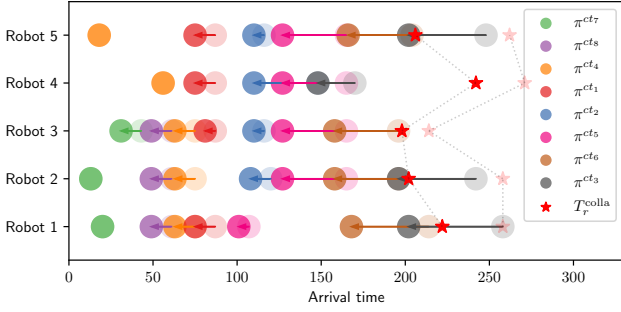
Figure 3: Robots' arrival time to the assigned collaborative tasks before (transparent discs) and after (solid discs) applying the execution strategy adjusting mechanism in an experiment with 5 robots. The red stars represent each robot's time cost before (transparent) and after (solid) the adjusting while accounting the potential wait time in collaborations. The collaborative LTL$_f$ formula is $\phi = (\Diamond(\pi^{ct_1} \wedge (\Diamond\pi^{ct_2}))) \wedge (\Diamond\pi^{ct_3}) \wedge (\Diamond\pi^{ct_4}) \wedge (\neg\pi^{ct_1} \mathbf{U} (\Diamond\pi^{ct_6})) \wedge (\Diamond\pi^{ct_7}) \wedge (\Diamond\pi^{ct_8}) \wedge (\neg\pi^{ct_5} \mathbf{U} \pi^{ct_8})$. The figure illustrates the effect of the proposed adjusting mechanism in reducing total time cost.

$O(|\widetilde{\mathbb{T}}_r|)$ levels. Each level contains collaborative states for one ct in $\widetilde{\mathbb{T}}_r$ of magnitude Δ in $\mathcal{P}_r$. The number of edges in $\overline{\mathcal{P}}_r$ is $O(\Delta^2|\widetilde{\mathbb{T}}_r|)$, and so is the number of variables in $\mathcal{Y}_r$. Additionally, the number of variables $t_r^{(k,l)}$ and $delay_r^{(k,l)}$ equals to the number of $\widetilde{\mathbb{T}}_r$, and the number of variable $\mathcal{Z}$ equals to $\widetilde{\mathbb{T}}$.

4.3. Iterative Execution Strategy Adjusting Mechanism

In this subsection, we propose an distributed execution strategy adjusting mechanism to iteratively optimize the total time cost, in which the computational burden is distributed among robots. A preliminary mechanism has been introduced in [2], and in this paper, we further improve the method by operating on the pruned local product automaton, which can greatly improve its solving efficiency, as shown in Sec. 5.

Intuitively, the adjusting mechanism investigates whether the total time cost can be reduced by adjusting robots' arrival time to regions of collaboration. The adjusting is inspired by reducing inefficient wait time in each collaborations. Fig. 3 shows robots' arrival time to the assigned collaborative tasks before and after applying the proposed adjusting mechanism. The proposed execution strategy adjusting mechanism operates in a distributed manner, as described in Alg. 2. We assume that the communication network between the robots are connected, so that messages can spread over all robots. We will explain the details of the adjusting mechanism in the following context.

4.3.1. Initialization

In this procedure, each robot computes a timeline and propagate it to all other robots. Let tl_r denotes the timeline of robot r, which can be constructed according to Alg. 3. The timeline tl_r records robot r's ideal arrival time to the region of the assigned collaborative tasks according to its current execution strategy ρ_r, without considering the potential wait time. It also includes the time robot r spends to finish all its tasks, also without considering wait time (Line 5−6, Alg. 3). After robot r receiving $tl_{r'}$ from all other robots $r' \in \mathcal{N}\backslash\{r\}$,

---

Algorithm 2: StrategyAdjustingMechanism

1 Initialize:
2 $count \leftarrow 0$, $ct \leftarrow$ the first task in $\widetilde{\mathbb{T}}^{sort}$.
3 $tl_r \leftarrow$ ComputeTimeline(r), for $r \in \mathcal{N}$.
4 Each robot $r \in \mathcal{N}$ propagates tl_r to other robots.
5 $msg \leftarrow \langle False, \_, \_, \_, ct, count\rangle$.
6 Propagate msg to all robots.
7
8 when robot r received(msg):
9 $\langle success, r', received, tl_{r'}, ct, count\rangle \leftarrow msg$.
10 if $success$ then
11   Modify robot r's $tl_{r'}$ to the received $tl_{r'}$.
12 if $r \notin received \wedge r \in \mathcal{R}(ct) \wedge r =$ FindLatest(ct) then
13   Add robot r into $received$.
14 $can\_opt \leftarrow$ AdjustStrategy(ct, $isLatest =$ true).
15 $ct^{next} \leftarrow$ the next task of ct in $\widetilde{\mathbb{T}}^{sort}$.
16   if can_opt then
17 $tl_r \leftarrow$ ComputeTimeline(r).
18 $msg \leftarrow \langle canOpt, r, \{\}, tl_r, ct^{next}, count + 1\rangle$.
19   propagate msg to all other robots.
20   else
21 $r' \leftarrow$ FindEarliest(ct).
22 $token \leftarrow \langle r, r', msg\rangle$.
23   Propagate $token$ to r', the earliest robot.
24 else
25   send msg to other robots not in $received$.
26
27 when robot r received(token):
28 $\langle r, r', msg\rangle \leftarrow token$.
29 if $r = r'$ then
30   can_opt$\leftarrow$ AdjustStrategy(ct, $isLatest=$false).
31 $ct^{next} \leftarrow$ the next task of ct in $\widetilde{\mathbb{T}}^{sort}$.
32   if can_opt then
33 $count \leftarrow count + 1$.
34 $tl_r \leftarrow$ ComputeTimeline(r).
35   else
36   if ct is the last task then
37   if $count = 0$ then
38   Send ternimation to other robots.
39   **Ternimate**.
40   else
41 $count \leftarrow 0$, $ct^{next} \leftarrow$ the first task in $\widetilde{\mathbb{T}}^{sort}$.
42 $msg \leftarrow \langle canOpt, r, \{\}, tl_r, ct^{next}, count\rangle$.
43   propagate msg to all other robots.
44 else
45   received.add(r).
46   Propagate $token$ to r'.
47
48 **Function** FindLatest(ct):
49 $ct^{pre} \leftarrow$ the previous task of ct in $\widetilde{\mathbb{T}}_r$.
50 $r \leftarrow \arg\max_{r\in\mathcal{R}(ct)} \{t(ct^{pre}) - t_r(ct^{pre}) + t_r(ct)\}$.
51   **return** r
52
53 **Function** FindEarliest(ct):
54 $ct^{pre} \leftarrow$ the previous task of ct in $\widetilde{\mathbb{T}}_r$.
55 $r \leftarrow \arg\min_{r\in\mathcal{R}(ct)} \{t(ct^{pre}) - t_r(ct^{pre}) + t_r(ct)\}$.
56   **return** r

9

Algorithm 3: ComputeTimeline(r)

1   $tl_r = [\ ]$.
2   **for** $ct \in \widetilde{\mathbb{T}}_r$ **do**
3     $t_r(ct) = \sum_{j=1}^{j^{ct}-1} \omega_r(\rho_r(j), \rho_r(j+1))$.
4   tl_r.append($t_r(ct)$).
5   $T_r = \sum_{j=1}^{|\rho_r|-1} \omega_r(\rho_r(j), \rho_r(j+1))$.
6   tl_r.append(T_r).
7   **return** tl_r

Algorithm 4: ComputeTimeCost

1   $delay_r \leftarrow 0$, for $r \in \mathcal{N}$.
2   **for** $ct \in \widetilde{\mathbb{T}}^{sort}$ **do**
3     $\hat{t}_r(ct) = t_r(ct) + delay_r$, for $r \in \mathcal{R}(ct)$.
4     $t(ct) = \max_{r \in \mathcal{R}(ct)} \hat{t}_r(ct)$.
5     $delay_r = t(ct) - t_r(ct)$, for $r \in \mathcal{R}(ct)$.
6   $T_r^{\text{colla}} \leftarrow T_r + delay_r$, for $r \in \mathcal{N}$.
7   $T^{\text{colla}} = \sum_{r \in \mathcal{N}} T_r^{\text{colla}}$.
8   **return** T^{colla}

---

it can locally compute the actual execution time of the assigned collaborative tasks and the total time cost, considering the potential synchronization constraints with other robots, as shown in Alg. 4. In Alg. 4, each robot can obtain the exact arrival time of each robot to one collaborative task in $\widetilde{\mathbb{T}}^{sort}$, i.e., $\hat{t}_r(ct)$ denotes the time when robot r arrives at the region of ct, including previous time $delay_r$ due to the synchronization constraints in collaborations. Note that $delay_r$ is iteratively modified as the execution of collaborative tasks. Moreover, the Alg. 4 can also calculate the total time cost for all robots, denoted as T^{colla} (Line $6-7$), which will be used in the adjusting procedure to check whether a modified execution strategy can result in a better performance.

Additionally, the central server will generate a message msg and propagate it to all robots, which will start the adjusting procedure. More specifically, this msg will make the last robot arriving to the first collaborative task automatically get the token. The formal definition of the message msg will be given in next subsection.

4.3.2. *Adjusting Procedure*

In this procedure, the robots traverse all collaborative tasks to investigate whether the wait time caused by synchronization constraints and the total time cost can be reduced by adjusting their execution strategy locally. The procedure depends on the propagation of two types of messages defined as follows.

- *msg*, the broadcast message. A *msg* is a tuple $\langle canOpt, r, received, tl_r, ct, count \rangle$, where $canOpt$ is a Boolean variable indicating whether the previous robot r optimizes the total time cost; tl_r is the modified timeline for robot r; ct is the current collaborative task to be investigated and $count$ counts the times of successful optimization of the total time cost within one cycle.

- *token*, the directional message. A *token* is a tuple

Algorithm 5: AdjustStrategy(ct, $isLatest$)

1   $ct^{pre} \leftarrow$ the previous task of ct in $\widetilde{\mathbb{T}}_r$.
2   **for** $q \in C(ct) \backslash \{\rho_r(j^{ct})\}$ **do**
3     $\rho'_r \leftarrow \rho_r[..j^{pre}-1] + \text{Dijkstra}(\mathcal{P}_r, \rho_r(j^{pre}), q)$
4       $+ \text{Dijkstra}(\mathcal{P}_r, q, Q_P^F)$
5   **if**
     $(isLatest \land t(ct^{pre}) - t_r(ct^{pre}) + t'_r(ct) < t_r(ct))$
     or $((\neg isLatest) \land$
     $t_r(ct) < t(ct^{pre}) - t_r(ct^{pre}) + t'_r(ct) \leq t(ct))$
     then
6       $T_{\text{cand}}^{\text{colla}} \leftarrow \text{ComputeTimeCost}(\rho'_r)$
7   **if** $T_{\text{cand}}^{\text{colla}} < T^{\text{colla}}$ **then**
8         $\rho_r \leftarrow \rho'_r$
9   **return** true

10   **return** false

$\langle r, r', msg \rangle$, where r and r' are source and target robot respectively.

When a robot r receives an msg from a robot r' (as the case after the initialization procedure), it checks whether it is the latest robot participating into the collaboration of task ct by calling function `FindLatest()` (Line 12, Alg. 2). If it is not the case, robot r adds itself into the set *received* of the msg, and sends the msg to all other robots not in *received* through communication network. If it is the case, then robot r starts to adjust its execution strategy, as described in Alg. 5. The detailed will be explained later. The Alg. 5 returns a Boolean value $canOpt$, indicating whether robot r can find another execution strategy so that the total time cost can be optimized. If $canOpt$ is true, then a new msg is constructed, which includes the modified tl_r according to robot r's new execution strategy (Line $16-19$, Alg. 2).

Otherwise, if $canOpt$ is false, robot r finds the earliest robot arriving to the current collaborative task ct by calling function `FindEarliest()`. Then robot r construct a message $token = \langle r, r', msg \rangle$ and send it to robot r' through the communication network. If in a special case $r = r'$, then a message msg is directly constructed and is sent to all other robots. Note that although current robot r can find out which robot should get the token for task ct^{next}, the msg still need to be propagated to all other robots to update their local tl_r.

When a robot r receives a $token$, i.e, it is the earliest robot arriving to current collaborative task, it has the authority to adjust its execution strategy, trying to reduce its wait time in current collaboration and the total time cost (Line $29-30$, Alg. 2. The adjusting process is similar to what the latest robot does. The difference is, if the total time cost cannot be optimized, the token will be directly passed for the next collaborative task in $\widetilde{\mathbb{T}}^{sort}$.

Furthermore, the detailed adjusting strategy is explained in Alg. 5. Assume that robot r takes the token to adjust its execution strategy for one task $ct \in \widetilde{\mathbb{T}}_r$. The set $C(ct)$ of $\mathcal{P}_r$ actually provides all candidates states corresponds to the execution of task ct, namely, all possible arrival time instances to ct. The robot r randomly traverses all state $q \in C(ct) \backslash \{\rho_r(j^{ct})\}$ until finding one candidate state q which can optimize T^{colla}. For each candidate state q, robot r searches a candidate run ρ'_r in $\mathcal{P}_r$: first maintain the origi-

nal ρ_r unchanged until $\rho_r(j^{pre} - 1)$; then concatenate a new shortest path starting from $\rho_r(j^{prev})$ to the accepting states of $\mathcal{P}_r$, while forcing the path to pass through q (Line $3 - 4$). Here $\rho_r(j^{prev})$ is the collaborative state of task ct^{prev} in ρ_r and ct^{prev} is the previous collaborative task of ct in $\widetilde{\mathbb{T}}_r$ (Line 2). The candidate run ρ'_r is selected to be the new ρ_r if the following two conditions hold (Lines $5 - 8$):

(a) if robot r is the latest robot, it holds that $t(ct^{prev}) - t_r(ct^{prev}) + t'_r(ct) < t_r(ct)$;

   if robot r is the earliest robot, it holds that $t_i(ct) < t(ct^{prev}) - t_r(ct^{prev}) + t'_r(ct) \leq t(ct)$.

(b) $T^{\text{colla}}_{\text{cand}} < T^{\text{colla}}$, here $T^{\text{colla}}_{\text{cand}}$ is the total time cost after replacing ρ_r by ρ'_r.

The condition (a) ensures that in the candidate run ρ'_r, the time robot r, the latest robot arriving to ct, is advanced (or delayed, for the earliest robot), such that the wait time for other robots in the collaboration of task ct is reduced. Here $t'_i(ct)$ is calculated from the candidate run ρ'_r. The condition (b) further guarantees the above local greedily adjustment procedure will also contribute to the total time cost T^{colla}. If robot r cannot find a qualified candidate run ρ'_r that satisfies the above two conditions, then the algorithm will return false.

4.3.3. Termination

If in one iteration no optimization happens, i.e., $count = 0$, then the adjusting procedure will terminate. Otherwise, $count > 0$ means the modified execution strategy results in better total time cost T^{colla}, and the adjusting procedure will go on to the next iteration to further optimize current strategies.

The above condition will be checked by the earliest robot arriving to the last task in $\widetilde{\mathbb{T}}^{sort}$ if it cannot optimize the total time cost after adjusting its execution strategy. If the termination condition holds, then the robot will send the termination signal to all other robots to stop the adjusting procedure (Line $36 - 39$, Alg. 2).

Remark 3. The proposed adjusting mechanism generates non-trivial execution strategies (trivial means the robot may pass through some unnecessary regions to deliberately postpone the arrival time in the collaboration), because we search the shortest path passing through the collaborative state which corresponds to the actual execution of tasks as in Def. 6.

4.4. Completeness, Optimality and Complexity

Proposition 3. *The proposed framework with execution strategy adjusting mechanism is complete.*

Proof. The proof follows Prop. 1. Note that in the adjusting procedure, robots may select another accepting runs on their pruned local product automatons, which also has a corresponding run on the original local product automaton. Such change does not violate Prop. 1, so the proposed framework with execution strategy adjusting mechanism is also complete under the same assumptions as in Prop. 1. ∎

Proposition 4. *The total time cost decreases monotonically when applying the execution strategy adjusting mechanism and the adjusting procedure terminates within finite iterations.*

Proof. Given the collaborative task assignments, T^{colla} decreases monotonically from $T^{\text{colla}}_{\text{init}}$ until convergence, where $T^{\text{colla}}_{\text{init}}$ is the total time cost of initial execution strategies before adjusting, according to Lines $5-8$ of Alg. 5. In addition, T^{colla} will not be smaller than T^{indiv}, which is the ideal time cost of robots' initial execution strategies without accounting wait time. Considering the resolution of discretization of the environment is limited, we conclude that Algorithm 5 will terminate within finite iterations. ∎

Proposition 5. *The worst case time complexity of Alg. 5 is $O\left((T^{\text{colla}}_{\text{init}} - T^{\text{indiv}}) \cdot |\overline{\mathcal{P}}_r| \cdot (E \cdot \lg E + |\sigma| \cdot N)\right)$, where N is the number of robots, $|\overline{\mathcal{P}}_r|$ is the maximum number of states of the local product automaton and E is the maximum number of edges in $\overline{\mathcal{P}}_r$. $|\sigma|$ is the number of all distinct collaborative tasks in the selected accepting run σ in Sec. 3.1.*

Proof. In Alg. 5, the robot with token will traverse all candidate collaborative states in $\overline{\mathcal{P}}_r$ in each adjustment in the worst case (Line 2). The number of adjustments is limited by $O(T^{\text{colla}}_{\text{init}} - T^{\text{indiv}})$ as in Prop. 4. For each candidate collaborative state, the robot with token utilizes Dijkstra algorithm to search a run ρ'_r with complexity $O(E \cdot \lg E)$, and also calls Alg. 4 to validate the candidate run with time complexity $|\sigma| \cdot N$. To summarize, the time complexity of Alg. 5 is $O\left((T^{\text{colla}}_{\text{init}} - T^{\text{indiv}}) \cdot |\overline{\mathcal{P}}_r| \cdot (E \cdot \lg E + |\sigma| \cdot N)\right)$. ∎

Optimality: The proposed framework does not guarantee to find the optimal results, because in the adjusting mechanism, the robots greedily adjust their task execution strategies, rather than exhaustively investigating the combinations of all possible candidates. The MILP formulation can obtain the optimal execution strategies given the assignment of collaborative tasks, which scales poorly as the number of robots increases (see experimental results in Sec. 5). Moreover, the selection of the collaborative task sequence σ in Sec. 3.1 also influences the performance of the final results.

5. Experiment

In this section, we evaluate the scalability, solution quality and efficiency of the proposed method. All the experiments are performed on a Ubuntu 16.04 server with CPU Intel Xeon E5-2660 at 2.00GHz and 128 GB of RAM. We use Z3 [27] SMT solver to solve the SMT formulation of collaborative task allocation problem. The baseline MILP method is solved by Gurobi v9 [26]. All the programs are written in Python 3, and we use the package *igraph*[4] to support our operations on graphs, which is implemented in C language.

We assume a grid map environment where some local tasks and collaborative tasks are randomly distributed. Each

---

[4]https://igraph.org/python/

Table 1: Baseline Method (B.S.) Vs The proposed Method (Proposed)

N	$|\widetilde{\mathbb{T}}|$	$|\ddot{\mathcal{P}}|$	B.S.(sec/sec)	Proposed(sec/sec) first	Proposed(sec/sec) best
2	4	34848	266/12.4	252/11.5	252/47.5
2	6	161376	338/97.0	352/28.2	352/57.6
3	4	3241792	333/41962.6	342/19.8	330/195.1
3	6	1.7×10^7	-/-	390/66.2	326/1800
5	6	1.4×10^{11}	-/-	819/52.9	802/1800
50	6	5.2×10^{97}	-/-	13702/700.4	13702/1800
100	6	2.7×10^{195}	-/-	26514/3548.7	26514/3548.7

The items in column B.S. and Proposed represent: $T^{\text{colla}}/t^{\text{cal}}$, where T^{colla} is the total time cost and t^{cal} is the solving time. The "-" items indicate memory overflow.

robot r has a local task specification φ_r, e.g., $\varphi_r = (\Diamond \pi^{ts_1}) \wedge (\Diamond \pi^{ts_2}) \wedge (\Diamond \pi^{ts_3}) \wedge (\Diamond \pi^{ts_4}) \wedge (\neg \pi^{ts_1} \text{ U } \pi^{ts_4})$. Additionally, the team of robots needs to collaborate to satisfy collaborative task specification ϕ, e.g., $\phi = (\Diamond \pi^{ct_1}) \wedge (\Diamond \pi^{ct_2}) \wedge (\Diamond \pi^{ct_4}) \wedge (\neg \pi^{ct_3} \text{ U } \pi^{ct_2}) \wedge (\Diamond (\pi^{ct_4} \wedge (\Diamond \pi^{ct_3})))$, which has $|\widetilde{\mathbb{T}}| = 4$. In our simulations, we assume that the completion of each ct in $\widetilde{\mathbb{T}}$ requires collaboration of several robots having different capabilities from a set $\{c_1, c_2, c_3\}$. The type and amount of robots needed for each collaborative task are randomly generated in the simulation.

5.1. Scalability of the Framework

We compare the proposed method with the traditional method based on the product automaton with state pruning techniques, which is similar to [20]. More specifically, each robot r constructs a local product automaton $\dot{\mathcal{P}}_r = wTS_r \otimes \dot{\mathcal{F}}_i$, where the insignificant states in wTS_r are pruned to maintain only the states corresponding to the task regions. $\dot{\mathcal{F}}_i$ is the NFA of φ_r. The we build a global product automaton $\ddot{\mathcal{P}} = \dot{\mathcal{P}} \otimes \mathcal{F}$, where $\dot{\mathcal{P}} = \dot{\mathcal{P}}_1 \otimes \cdots \otimes \dot{\mathcal{P}}_N$ and $\mathcal{F}$ corresponds to ϕ. Here the $\dot{\mathcal{F}}$, $\dot{\mathcal{P}}$ and $\ddot{\mathcal{P}}$ are used to distinguish from the notations used in our proposed method. The task execution strategy can be obtained by searching the shortest accepting run in $\ddot{\mathcal{P}}$. Note that the shortest accepting run may not be optimal w.r.t the total time cost, because the searching process doesn't take the potential wait time into account. It remains a problem about how to efficiently find a path on above automaton $\ddot{\mathcal{P}}$ while also considering the potential wait time in collaborations, and here we just take the shortest path. This only has little impact on our evaluation of the scalability of the proposed method.

We investigate the performance of the two methods in different task requirements and robot numbers, and keep the environment size as 30×30 in all cases, as shown in Tab. 1. For the proposed framework, we record the time when obtaining the first feasible solutions and the best solutions within 30 minutes. The baseline method based on global product automaton quickly becomes intractable as the number of robots N and collaborative tasks $|\widetilde{\mathbb{T}}|$ increases. Roughly speaking, the computational time of our proposed framework grows proportional to the number of robots. This is mainly because we accumulate the time spent by all robots to construct
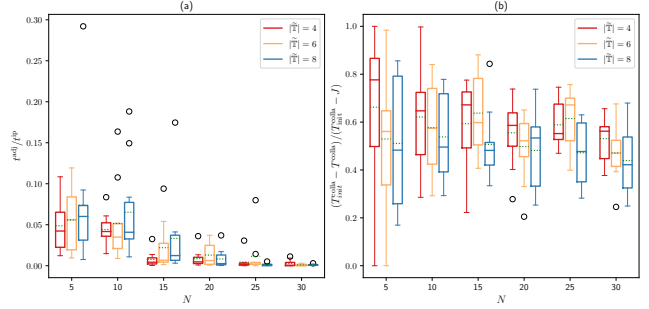

Figure 4: The comparison of the adjusting method and the MILP method under different number of collaborative tasks: (a) $t^{\text{adj}}/t^{\text{ip}}$; (b) $(T^{\text{colla}}_{\text{init}} - T^{\text{colla}})/(T^{\text{colla}}_{\text{init}} - J)$. Here t^{adj} denotes the time token by the execution strategy adjusting mechanism, t^{ip} denotes the time token by Gurobi to solve the MILP problem. Each entry shows the average (green dotted line), the median, 25th percentile, 75th percentile, the max, the minimum and outliers out of 10 experiments.

their local product automatons. We have to point out that our method runs much quicker when the local planning procedure is distributed to individual robots, and scales well with large number of robots. The results illustrate that the proposed method can quickly generate feasible execution strategies for the robots, and has high scalability compared with the baseline method.

5.2. Optimization Efficiency of the Adjusting Mechanism

We conduct extensive simulations to investigate the performance of the execution strategy adjusting mechanism versus the baseline MILP method. The number of robots varies from $\{5, 10, 15, 20, 25, 30\}$ and the number of collaborative tasks changes from 4, 6 to 8. We also investigate the performance under three types of environment sizes. In each case, we run both the proposed framework with adjusting mechanism for 10 times, where all tasks are randomly distributed in the grid map environment for 10 times under the same task specifications. For the proposed framework with adjusting mechanism, we stop the simulation once the running time exceeds 30 minutes, and select the best result obtained up to the end time. We then compare the running time and the solution quality of the proposed framework with adjusting mechanism versus with MILP method under the same assignment of collaborative tasks.

In Fig. 4(a), we compare the solving time of the proposed adjusting mechanism with the MILP method under different number of robots and different types of collaborative tasks. It is obvious that in all cases, the adjusting mechanism runs much faster than the MILP method, and the advantage of the adjusting mechanism becomes significant as robot number increases. Surprisingly, the adjusting mechanism stops within one second in all cases, while the MILP method takes more than 10 hours to solve the case with 30 robots. The detailed comparison is shown in Tab. 2. Furthermore, in Fig. 4(b), we evaluate how much the total time cost can be optimized by the proposed adjusting mechanism. This is evaluated by $(T^{\text{colla}}_{\text{init}} - T^{\text{colla}})/(T^{\text{colla}}_{\text{init}} - J)$, where $T^{\text{colla}}_{\text{init}}$ corresponds to the total time cost of robots' initial execution strategies before adjustment, T^{colla} is the total time cost after

Table 2: Comparison of Computational Time for Pruning product automatons, Adjusting Mechanism and MILP Method

	$|\widetilde{\mathbb{T}}| = 4$			$|\widetilde{\mathbb{T}}| = 6$			$|\widetilde{\mathbb{T}}| = 8$		
N	$\bar{t}^{\text{prune}}$ (sec)	t^{adj} (sec)	t^{ip} (sec)	$\bar{t}^{\text{prune}}$ (sec)	t^{adj} (sec)	t^{ip} (sec)	$\bar{t}^{\text{prune}}$ (sec)	t^{adj} (sec)	t^{ip} (sec)
5	8.14 ± 3.93	0.07 ± 0.04	2.05 ± 1.53	30.49 ± 5.47	0.17 ± 0.12	3.78 ± 1.89	48.59 ± 14.38	0.28 ± 0.16	6.43 ± 4.20
10	13.7 ± 3.12	0.27 ± 0.08	7.21 ± 3.37	22.16 ± 7.16	0.26 ± 0.16	8.31 ± 6.87	38.53 ± 13.04	0.36 ± 0.19	8.24 ± 5.33
15	12.87 ± 2.74	0.35 ± 0.15	181.18 ± 229.33	25.74 ± 5.74	0.56 ± 0.28	91.24 ± 100.05	38.43 ± 13.08	0.63 ± 0.26	77.86 ± 80.47
20	10.90 ± 2.99	0.41 ± 0.21	146.72 ± 148.56	25.00 ± 7.36	0.66 ± 0.37	261.13 ± 358.53	29.25 ± 12.54	0.45 ± 0.29	516.50 ± 1186.96
25	8.32 ± 3.65	0.33 ± 0.18	1741.57 ± 4160.64	22.79 ± 9.21	0.6 ± 0.31	808.92 ± 1454.72	28.71 ± 15.82	0.86 ± 0.51	2816.32 ± 2164.77
30	5.17 ± 2.05	**0.38 ± 0.21**	**4129.2 ± 6706.97**	17.31 ± 6.97	**0.6 ± 0.28**	**4941.11 ± 5694.07**	30.05 ± 10.86	**0.87 ± 0.42**	**4271.79 ± 8339.85**

applying the adjusting mechanism and J is the result of the MILP method. As in Fig. 4(b), the optimization effect of the adjusting mechanism varies when the problem is small, however, it tends to converge within around 40% to 60% as the number of robots increases, which verifies that the effect of proposed adjusting mechanism to reduce the total time cost.

Specifically, the detailed time spent for solving each case are shown in Tab. 2. The table compares the average pruning time ($\bar{t}^{\text{prune}}$) for robots to prune their local product automaton as in Def. 7, the time spent by the adjusting mechanism (t^{adj}) and the time spent by the MILP method (t^{ip}) under different numbers of collaborative tasks and numbers of robots. Note that pruning robots' local product automatons is the common preliminary step before applying the adjusting mechanism or the MILP method. The average time spent by robots to prune their local product automatons, $\bar{t}^{\text{prune}}$, grows slightly as the number of collaborative tasks increases from 4 to 8. This is because on average, the robots are assigned more collaborative tasks, so that the size of their pruned product automatons also creases as in Prop. 2. The same trend also exists in t^{adj} and t^{ip}, despite some exceptions when the robot population is small.

Thanks to the preliminary procedure of pruning local product automatons, the MILP method can get the results within several seconds when the robot number is small. However, t^{ip} grows significantly as the number of robots increases. Note that in a case with $N = 30$ and $|\widetilde{\mathbb{T}}| = 8$, the MILP method takes more than 7 hours to get the solution, while the adjusting mechanism only takes 1.93 seconds to converge. Surprisingly, the proposed execution strategy adjusting mechanism solving almost all cases within a second (maximum value is 2.24s).

Moreover, we also compare the above metrics of the two methods under different environment sizes, varying from 20×20, 30×30 to 40×40 grid map. The results are shown in Fig. 5. The performance of the proposed method is consistent with it in Fig. 4. The performance of the proposed methods shows no obvious difference under different environment size in Fig. 5, because the environment size doesn't influence the sizes of the pruned local product automatons according to Prop. 2. The environment size may change the time robot spend to move between different regions, and we can see from Fig. 5(b) that such change doesn't obviously affect the optimization effect.
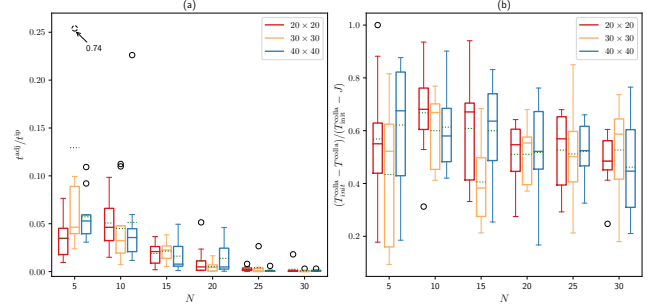

Figure 5: The comparison of the adjusting method and the MILP method under different environment sizes: (a) $t^{\text{adj}}/t^{\text{ip}}$; (b) $(T^{\text{colla}}_{\text{init}} - T^{\text{colla}})/(T^{\text{colla}}_{\text{init}} - J)$. Each entry shows the average (green dotted line), the median, 25th percentile, 75th percentile, the max, the minimum and outliers out of 10 experiments.
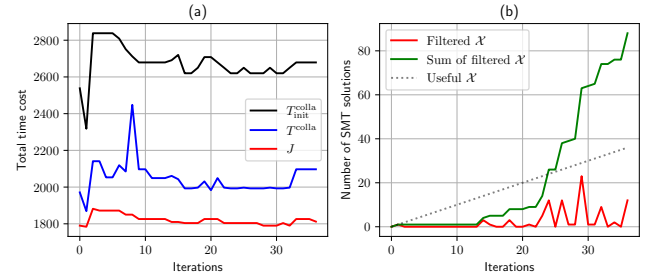

Figure 6: (a) The optimization process of the total time cost under different assignment of collaborative tasks and (b) the filtered SMT solutions in each iteration by utilizing the filtering strategy in an experiment with 15 robots.

5.3. Iterative Optimization and Filtering of SMT Solutions

The proposed framework iteratively evaluates each feasible assignments of collaborative tasks, which is generated by solving an SMT model (Sec. 3.2). And we also propose a filtering strategy to screen out non-optimal assignments in advance. Fig. 6 (a) plots the optimization process of the total time cost under each possible assignment of collaborative tasks generated by the SMT solver, in a case with 15 robots, $|\widetilde{\mathbb{T}}| = 6$ and environment size is 30×30. In each iteration (each kinds of feasible task assignments), the total time cost is reduced from $T^{\text{colla}}_{\text{init}}$ to T^{colla} by the proposed adjusting mechanism. Although the MILP method can provide better solutions in each iteration, as shown by the red line, it takes much more time than the adjusting mechanism to get the results. In addition, as shown in Fig. 6 (b), the filtering strategy proposed in in Sec. 3.2 can efficiently filter non-optimal solu-

tions to the collaborative task assignment problem according to the history solutions, avoiding unnecessary computation. The strategy only needs to maintain a set of solutions whose size is propositional to the number of iterations. Note that the filtering efficiency depends on the inner computation mechanism of the SMT solver and the specific task requirements.

6. Conclusions and Future Work

In this paper, we propose a hierarchical task planning framework that can efficiently coordinate multiple robots under individual and collaborative temporal logic specifications. A central server first extracts a task sequence satisfying the collaborative task specification, decomposes it, and then allocates subtasks to the robots. All feasible task assignments are iteratively generated by an SMT solver and evaluated by the robots. Then the robots synthesize their execution strategies based on locally constructed product automatons. To further optimize the total time cost, we propose an distributed execution strategy adjusting mechanism that allows robots adjust their strategies to minimize inefficient wait time in collaborations. We prove the completeness of the proposed method under our assumptions and extensive simulation results verify the scalability and efficiency of the proposed method. Future work is to improve the efficiency of the task allocation procedure and expand the current framework to complete LTL to enable infinite tasks.

References

[1] H. Kress-Gazit, M. Lahijanian, V. Raman, Synthesis for robots: Guarantees and feedback for robot behavior, Annual Review of Control, Robotics, and Autonomous Systems 1 (1) (2018) 211–236. doi:10.1146/annurev-control-060117-104838.

[2] R. Bai, R. Zheng, M. Liu, S. Zhang, Multi-robot task planning under individual and collaborative temporal logic specifications (2021). arXiv:2108.11597.

[3] D. Khalidi, D. Gujarathi, I. Saha, T* : A heuristic search based path planning algorithm for temporal logic specifications, in: Proceedings of the 2020 IEEE International Conference on Robotics and Automation, 2020, pp. 8476–8482. doi:10.1109/ICRA40945.2020.9196928.

[4] Y. Kantaros, M. M. Zavlanos, Intermittent connectivity control in mobile robot networks, in: Proceedings of the 2015 49th Asilomar Conference on Signals, Systems and Computers, 2015, pp. 1125–1129. doi:10.1109/ACSSC.2015.7421315.

[5] C. Ioan Vasile, C. Belta, Sampling-Based Temporal Logic Path Planning (2013). arXiv:1307.7263.

[6] Y. Kantaros, M. M. Zavlanos, Sampling-based control synthesis for multi-robot systems under global temporal specifications, in: Proceedings of 2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems, 2017, pp. 3–14. doi:10.1145/3055004.3055027.

[7] Y. Kantaros, M. M. Zavlanos, Distributed optimal control synthesis for multi-robot systems under global temporal tasks, in: Proceedings of the 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems, 2018, pp. 162–173. doi:10.1109/ICCPS.2018.00024.

[8] Y. Kantaros, M. M. Zavlanos, Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems, The International Journal of Robotics Research 39 (7) (2020) 812–836. doi:10.1177/0278364920913922.

[9] P. Schillinger, M. Bürger, D. V. Dimarogonas, Decomposition of Finite LTL Specifications for Efficient Multi-agent Planning, Springer

International Publishing, Cham, 2018, pp. 253–267. doi:10.1007/978-3-319-73008-0_18.

[10] P. Schillinger, M. Bürger, D. V. Dimarogonas, Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems, The International Journal of Robotics Research 37 (7) (2018) 818–838. doi:10.1177/0278364918774135.

[11] F. Faruq, D. Parker, B. Laccrda, N. Hawes, Simultaneous task allocation and planning under uncertainty, in: Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2018, pp. 3559–3564. doi:10.1109/IROS.2018.8594404.

[12] C. Banks, S. Wilson, S. Coogan, M. Egerstedt, Multi-agent task allocation using cross-entropy temporal logic optimization, in: Proceedings of the 2020 IEEE International Conference on Robotics and Automation, 2020, pp. 7712–7718. doi:10.1109/ICRA40945.2020.9197066.

[13] X. Luo, Y. Kantaros, M. M. Zavlanos, An abstraction-free method for multirobot temporal logic optimal control synthesis, IEEE Transactions on Robotics (2021) 1–21doi:10.1109/TRO.2021.3061983.

[14] P. Schillinger, M. Bürger, D. V. Dimarogonas, Hierarchical ltl-task mdps for multi-agent coordination through auctioning and learning, The international journal of robotics research (2019).

[15] Y. E. Sahin, P. Nilsson, N. Ozay, Provably-correct coordination of large collections of agents with counting temporal logic constraints, in: Proceedings of the 2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems, 2017, pp. 249–258. doi:10.1145/3055004.3055021.

[16] Y. E. Sahin, P. Nilsson, N. Ozay, Multirobot coordination with counting temporal logics, IEEE Transactions on Robotics 36 (4) (2020) 1189–1206. doi:10.1109/TRO.2019.2957669.

[17] S. Moarref, H. Kress-Gazit, Decentralized control of robotic swarms from high-level temporal logic specifications, in: 2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS), 2017, pp. 17–23. doi:10.1109/MRS.2017.8250926.

[18] M. Guo, D. V. Dimarogonas, Bottom-up motion and task coordination for loosely-coupled multi-agent systems with dependent local tasks, in: Proceedings of the 2015 IEEE International Conference on Automation Science and Engineering, 2015, pp. 348–355. doi:10.1109/CoASE.2015.7294103.

[19] M. Guo, D. V. Dimarogonas, Task and motion coordination for heterogeneous multiagent systems with loosely coupled local tasks, IEEE Transactions on Automation Science and Engineering 14 (2) (2017) 797–808. doi:10.1109/TASE.2016.2628389.

[20] J. Tumova, D. V. Dimarogonas, Decomposition of multi-agent planning under distributed motion and task ltl specifications, in: Proceedings of the 2015 54th IEEE Conference on Decision and Control, 2015, pp. 7448–7453. doi:10.1109/CDC.2015.7403396.

[21] Y. Kantaros, M. Guo, M. M. Zavlanos, Temporal logic task planning and intermittent connectivity control of mobile robot networks, IEEE Transactions on Automatic Control 64 (10) (2019) 4105–4120. doi:10.1109/TAC.2019.2893161.

[22] C. Baier, J.-P. Katoen, Principles of Model Checking (Representation and Mind Series), The MIT Press, 2008.

[23] G. De Giacomo, R. De Masellis, M. Montali, Reasoning on ltl on finite traces: Insensitivity to infiniteness, in: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI Press, 2014, p. 1027–1033. doi:10.5555/2893873.2894033.

[24] M. Kloetzer, C. Belta, A Fully Automated Framework for Control of Linear Systems from Temporal Logic Specifications 53 (1) 287–297. doi:10.1109/TAC.2007.914952.

[25] X. Luo, M. M. Zavlanos, Temporal logic task allocation in heterogeneous multi-robot systems (2021). arXiv:arXiv:2101.05694.

[26] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual (2021).

[27] L. De Moura, N. Bjørner, Z3: An efficient smt solver, in: Proceedings of the Theory and Practice of Software, Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer-Verlag, Berlin, Heidelberg, 2008, p. 337–340. doi:10.1007/978-3-540-78800-3_24.