Temporal Logic Task Allocation in Heterogeneous Multi-Robot Systems

Xusheng Luo and Michael M. Zavlanos, *Senior Member, IEEE*

***Abstract*—We consider the problem of optimally allocating tasks, expressed as global Linear Temporal Logic (LTL) specifications, to teams of heterogeneous mobile robots of different types. Each task may require robots of multiple types. To obtain a scalable solution, we propose a hierarchical approach that first allocates specific robots to tasks using the information about the tasks contained in the Nondeterministic Büchi Automaton (NBA) that captures the LTL specification, and then designs low-level paths for robots that respect the high-level assignment. Specifically, motivated by "lazy collision checking" methods in robotics, we first prune and relax the NBA by removing all negative atomic propositions, which simplifies the planning problem by checking constraint satisfaction only when needed. Then, we extract sequences of subtasks from the relaxed NBA along with their temporal orders, and formulate a Mixed Integer Linear Program (MILP) to allocate these subtasks to robots. Finally, we define generalized multi-robot path planning problems to obtain low-level paths that satisfy both the high-level task allocation and the constraints captured by the negative atomic propositions in the original NBA. We show that our method is complete for a subclass of LTL that covers a broad range of tasks and present numerical simulations demonstrating that it can generate paths with lower cost, considerably faster than existing methods.**

***Index Terms*—Formal methods in robotics and automation, task planning, path planning for multiple mobile robots or agents, motion and path planning.**

I. Introduction

Robot motion planning traditionally consists of generating trajectories between a start and a goal region, while avoiding obstacles [1]. More recently, new planning methods have been proposed that can handle a richer class of tasks than standard point-to-point navigation that also include temporal goals subject to time constraints. Such tasks can be captured using formal languages, such as Linear Temporal Logic (LTL) [2], and include sequencing or coverage [3], data gathering [4], intermittent communication [5], and persistent surveillance [6], to name a few. A survey on formal specifications and synthesis techniques for robotic systems can be found in [7].

In this paper, we consider LTL tasks that require robots of different types to collaborate to satisfy the specification. Each task may require robots of multiple types to accomplish. The specific robots assigned to each task are immaterial, as long as they are of the desired type. An example of such an LTL task is: *At least two robots of type 1 pick up the mail by visiting houses in a given order. Next, visit a delivery site. Never leave the delivery site until one ground robot of type 2 is present to pick up the mail. Repeat this process infinitely often.* In this task, several robots are required to work cooperatively and meet simultaneously at the same place. The specific robots to participate are not specified by the LTL formula. Instead, it is only required that no less than five robots of type 1 and one robot of type 2 collaborate to accomplish this task. We refer to this problem as the Multi-Robot Task Allocation (MRTA) problem for LTL tasks, in short, LTL-MRTA. Existing control synthesis methods under temporal logic specifications, such as the ones proposed in [8–10] build a product automaton composed of the Nondeterministic Büchi Automaton (NBA) that captures the LTL specification and the discrete transition systems describing the motion of robots. Then, these methods employ graph search techniques to find the optimal plan that satisfies the LTL specification. However, as the number of robots, the size of the environment, and the complexity of the LTL task grows, the size of this product graph grows exponentially large and, therefore, graph search methods become intractable. This is more so the case for LTL-MRTA problems as possible assignments of robots to tasks increases the complexity of the LTL specification dramatically.

To mitigate the computational complexity of the LTL-MRTA problem, we propose a novel hierarchical approach that first allocates robots to tasks using the information about tasks provided by the NBA, and then designs low-level robot paths that respect the high-level assignment. Specifically, we first prune and relax the NBA by removing all negative atomic propositions. This step is motivated by "lazy collision checking" methods in robotics [11] and allows to simplify the planning problem by checking constraint satisfaction only when needed. Then, we extract sequences of subtasks from the relaxed NBA along with their temporal orders, and formulate a Mixed Integer Linear Program (MILP), inspired by the vehicle routing problem [12], to allocate these subtasks to robots, while respecting the temporal order constraints. The solution to this MILP generates a time-stamped task allocation plan for each robot, that is a sequence of essential waypoints that the robot needs to visit. Finally, given this time-stamped task allocation plan for each robot, we formulate a sequence of generalized multi-robot path planning (GMRPP) problems, one for each subtask, to obtain executable paths that also respect the negative atomic propositions that were relaxed from the original NBA. We show through extensive simulations that our method can handle LTL-MRTA problems with up to 10^{90} states in the product graph, considerably outperforming existing methods. Moreover, we provide theoretical guarantees on the completeness and soundness of our proposed framework, under mild assumptions on the structure of the NBA that were satisfied by all meaningful LTL specifications we considered in practice. While not theoretically optimal, our method is

Xusheng Luo and Michael M. Zavlanos are with the Department of Mechanical Engineering and Materials Science, Duke University, Durham, NC 27708, USA. {xusheng.luo, michael.zavlanos}@duke.edu. This work is supported in part by ONR under agreement #N00014-18-1-2374 and by AFOSR under the award #FA9550-19-1-0169.

still able to improve on the cost of the returned plans, unlike existing methods in the literature that only focus on feasibility.

A. Related work

In existing literature on optimal control synthesis methods from LTL specifications, LTL tasks are either assigned locally to the robots in a multi-robot team, as in [10, 13] or a global LTL specification is assigned to the team that captures the collective behavior of all robots. In the latter case, the global LTL specification can explicitly assign tasks to the individual robots, as in [14–16], or it may not explicitly assign tasks to the robots as in [17–20], and our current work in this paper.

Global temporal logic specifications that do not explicitly allocate tasks to robots typically need to be decomposed in order to obtain the required allocation. For example, [21, 22] decompose a global specification directly into local specifications and assign them to individual robots. Similarly, [23–25] decompose a global specification into multiple subtasks by exploiting the structure of the finite automata. Particularly, [24] converts temporal planning problems to standard planning problems by defining actions based on transitions in the NBA, while [23] defines subtasks associated with transitions in the NBA and synthesize plans for these subtasks that can be reused to efficiently synthesize plans for new LTL formulas. [25] also defines subtasks associated with transitions in the automaton, and use reinforcement learning to learn plans that execute these subtasks under uncertainty. However here, we do not assume that these subtasks are preassigned to the robots.

Temporal logic control synthesis without an explicit assignment of robots to tasks has been considered in [26] that combine the vehicle routing problem with metric temporal logic specifications and leverage MILP to solve this problem for heterogeneous robots. However, this approach can only handle finite horizon tasks and does not design the low-level executable paths as we do here. An alternative approach is proposed in [27] that decomposes a global automaton into individual automata that are assigned to the heterogeneous robots and then builds a synchronous product of these automata to synthesize parallel plans. However, the size of the synchronous product automaton grows exponentially large with the number of robots. Also, the requirement that parallel plans exist does not allow application of this method to tasks that lack such parallel executions. In relevant literature, teams of homogeneous robots have also been modeled using Petri Nets as in [20, 28]. Specifically, [20] proposes a job shop problem under safe temporal logic specifications, but do not consider the "eventually" operator so that liveness in terms of good future outcomes can not be guaranteed. Additionally, this approach only focuses on robot coordination at the task level without considering execution. To the contrary, [28] selects multiple shortest accepting runs in the NBA and for each accepting run, determine whether an executable plan exists. Finally, [29–31] automatically decompose the automaton representation of the LTL formula into independent subtasks that can be fulfilled by different robots. However, they only consider LTL formulas that can be satisfied by finite robot trajectories. Also, subtasks subject to precedence relations can only be executed by a single robot.

Common in the above approaches is that they do not consider cooperative tasks where robots need to meet at a common location to complete a task, Such tasks require strong synchronization between robots. In our recent work [32, 33], we have proposed a sampling-based planning method named STyLuS* that incrementally builds trees to approximate the product of the NBA and the model of the team. Using the powerful biased sampling method proposed in [33], STyLuS* can synthesize plans for product automata with up to 10^{800} states without considering collision avoidance. However, STyLuS* requires global LTL specifications that explicitly assign tasks to robots. Although a subset of specifications we consider here can be converted into explicit LTL formulas by enumerating all possible task assignments and connecting them with "OR" operators, this would result in exponentially long LTL formulas. Furthermore, the biased sampling strategy in STyLuS* needs a fixed assignment of robots to tasks and biases search towards finding a plan for this fixed assignment. If the assignment is not given, biased STyLuS* will need to be run combinatorially many times, one for each possible assignment. With unbiased sampling, [32] shows that STyLuS* can only solve problems with product automata that have 10^{10} states. Instead, our proposed method can synthesize plans for problems with 10^{90} states while considering collision avoidance. On the other hand, model-checkers like NuSMV [34], focus on finding feasible paths and are incapable of optimizing cost. As stated in [33], NuSMV can only handle problems with 10^{30} states, and can not easily process exponentially long LTL formulas generated by explicitly expressing task assignments.

Among other methods that focus on cooperative tasks, [19] focuses on specifications capturing behaviors of homogeneous robotic swarms at the swarm and individual levels, but they can only impose universal or existential constraints, that is, all robots or some robots visit a certain region. This limitation is addressed in [35, 36] that relies on counting linear temporal logic (cLTL+/cLTL) to capture constraints on the number of robots that must be present in different regions. The authors formulate an Integer Linear Program (ILP) inspired by Bounded Model Checking techniques [37], but can only guarantee feasibility of the resulting paths. Instead our hierarchical method also takes into consideration the quality of the solution at each level. A sequential planning approach is proposed in [38] that augments the LTL specification by introducing time and, unlike our proposed approach, plans low-level plans for the robots, one at a time, while treating the other robots as obstacles. Common in the methods in [35, 36, 38] is that the size of the workspace has a significant effect on the computation time. To mitigate the complexity due to the size of the workspace, [39] proposes a hierarchical framework that abstracts the workspace by aggregating states with the same observations. As we show in Section VII, our proposed method scales better than the method in [39], and provides lower cost solutions with less runtime. Also, unlike our method, the completeness of solutions is not guaranteed in [39].

B. Contributions

We propose a new hierarchical approach to the LTL-MRTA problem that first assigns robots to tasks and then plans robot

paths that satisfy the high level assignment. Our approach differs from common methods that rely on the product automaton [8–10] or on the Bounded Model Checking [37] in that it directly operates on the NBA. Under mild assumptions on the NBA that are satisfied by a subclass of LTL formulas that cover a broad class of tasks in practice, we showed that our method is complete and sound. While not theoretically optimal, our method still incorporates optimization steps to improve on the cost of the returned plans. To the best of our knowledge, this is the first LTL-MRTA method that is both complete for a subclass of LTL and includes operations to optimize the synthesized plans. One unique aspect of our approach is a clever pruning and relaxation of the NBA that removes all negative atomic propositions, and is motivated by "lazy collision checking" methods in robotics. This step significantly simplifies the planning problem by allowing to check constraint satisfaction only when needed and, as a result, contributes to significantly increasing scalability of our method. To the best of our knowledge, this is the first time that "lazy collision checking" methods that are common in point-to-point navigation are used for high-level robot planning. Another unique aspect of our method is its ability to infer the temporal order of tasks from the automaton, which can capture the parallel execution of subtasks. Compared to existing methods, our approach returns lower cost plans in significantly less time.

The rest of the paper is organized as follows. In Sections II and III we present preliminaries and define the problem formulation, respectively. We describe the high-level task assignment component in Sections IV and V. Specifically, in Section IV we prune and relax the NBA, identify subtasks from the NBA and infer temporal orders between them. Then, in Section V we formulate a MILP to obtain the high-level plans. In Section VI, we examine the completeness and soundness of these plans, while in Section VII we present simulation results. Finally, Section VIII concludes the paper. For completeness, the low-level component of our method to obtain executable paths, which is based on existing multi-robot path planning techniques, is presented in Appendix B in the full version [40].

II. Preliminaries

A. Linear temporal logic

Linear Temporal Logic (LTL) is composed of a set of atomic propositions $\mathcal{AP}$, the boolean operators, conjunction $\wedge$ and negation $\neg$, and temporal operators, next $\bigcirc$ and until $\mathcal{U}$ [2]. LTL formulas over $\mathcal{AP}$ follow the grammar

$$\phi := \top \mid \pi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc \phi \mid \phi_1 \,\mathcal{U}\, \phi_2,$$

where $\top$ is unconditionally true and π is the boolean-valued atomic proposition. Other temporal operators can be derived from $\mathcal{U}$ such as $\Diamond\phi$ means ϕ will be eventually true sometime in the future and $\Box\phi$ means ϕ is always true from now on.

An infinite *word* w over the alphabet $2^{\mathcal{AP}}$, the power set of the set of atomic propositions, is defined as an infinite sequence $w = \sigma_0\sigma_1\ldots \in (2^{\mathcal{AP}})^\omega$, where ω denotes an infinite repetition and $\sigma_k \in 2^{\mathcal{AP}}$, $\forall k \in \mathbb{N}$. The language $\texttt{Words}(\phi) = \{w | w \models \phi\}$ is defined as the set of words that satisfy the LTL formula ϕ, where $\models \subseteq (2^{\mathcal{AP}})^\omega \times \phi$ is the satisfaction relation. An LTL formula ϕ can be translated into a Nondeterministic Büchi Automaton (NBA) [41]:

Definition 2.1: (NBA) A *Nondeterministic Büchi Automaton* B is a tuple $B = (\mathcal{Q}, \mathcal{Q}_0, \Sigma, \rightarrow_B, \mathcal{Q}_F)$, where $\mathcal{Q}$ is the set of states; $\mathcal{Q}_0 \subseteq \mathcal{Q}$ is a set of initial states; $\Sigma = 2^{\mathcal{AP}}$ is an alphabet; $\rightarrow_B \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is the transition relation; and $\mathcal{Q}_F \subseteq \mathcal{Q}$ is a set of accepting states.

An *infinite run* ρ_B of B over an infinite word $w = \sigma_0\sigma_1\sigma_2\ldots$, $\sigma_k \in \Sigma$, $\forall k \in \mathbb{N}$, is a sequence $\rho_B = q_0 q_1 q_2 \ldots$ such that $q_0 \in \mathcal{Q}_0$ and $(q_k, \sigma_k, q_{k+1}) \in \rightarrow$, $\forall k \in \mathbb{N}$. An infinite run ρ_B is called *accepting* if $\texttt{Inf}(\rho_B) \cap \mathcal{Q}_F \neq \varnothing$, where $\texttt{Inf}(\rho_B)$ represents the set of states that appear in ρ_B infinitely often. If an LTL formula is satisfiable, then there exists an accepting run that can be written in the prefix-suffix structure such that the prefix part, connecting an initial state to an accepting state, is traversed only once and the suffix part, a cycle around the accepting state, is traversed infinitely often. The words σ that induce an accepting run of B constitute the accepted language of B, denoted by $\mathcal{L}_B$. It is shown in [2] that for any given LTL formula ϕ over a set of atomic propositions $\mathcal{AP}$, there exists a NBA B_ϕ over alphabet $\Sigma = 2^{\mathcal{AP}}$ such that $\mathcal{L}_{B_\phi} = \texttt{Words}(\phi)$, where $\texttt{Words}(\phi)$ is the set of words accepted by ϕ.

B. Partially ordered set

A finite partially ordered set or poset $P = (X, <_P)$ is a pair consisting of a finite base set X and a binary relation $<_P \subseteq X \times X$ that is reflexive, antisymmetric, and transitive. Let $x, y \in X$ be two distinct elements. We write $x <_P y$ if $(x, y) \in <_P$, and $x \|_P y$ if x and y are incomparable. Moreover, we say x is covered by y or y covers x, denoted by $x \prec_P y$, if $x <_P y$ and there is no distinct $z \in X$ such that $x <_P z <_P y$. An antichain is a subset of a poset in which any two distinct elements are incomparable. The width of a poset is the cardinality of a maximal antichain. Finally, a chain is a subset of a poset in which any two distinct elements are comparable. The height of a poset is the cardinality of a maximal chain.

A linear order $L_X = (X, <_L)$ is a poset such that $x <_L y$, $x = y$ or $y <_L x$ holds for any pair of $x, y \in X$. A linear extension $L_P = (X, <_L)$ of a poset P is a linear order such that $x <_L y$ if $x <_P y$, i.e., a linear order that preserves the partial order. We define Ξ_P as the set of all linear extensions of a poset P. Note that a poset and its linear extensions share the same base set X_P. Given a collection of linear orders Ξ, the poset cover problem focuses on reconstructing a single poset P or a set of posets $\{P_1, \ldots, P_k\}$ such that $\Xi_P = \Xi$ or $\cup_{i=1}^{k} \Xi_{P_i} = \Xi$. As shown in [42], the poset cover problem is NP-complete. Moreover, the partial cover problem focuses on finding a single poset P such that Ξ_P contains the maximum number of linear orders in Ξ, i.e., $\Xi_P \subseteq \Xi$ and $\nexists P'$ s.t. $\Xi_{P'} \subseteq \Xi$ and $|\Xi_{P'}| > |\Xi_P|$. It is shown in [42] that the partial cover problem can be solved in polynomial time.

III. Problem Definition

A. Transition system

Consider a discrete workspace containing $l \in \mathbb{N}^+$ labeled regions, where each region can span multiple cells, and denote by $\mathcal{L} = \{\ell_k\}_{k\in[l]}$ the set of regions, where $[l]$ is the shorthand

notation for $\{1,\dots,l\}$. We also assume that the workspace contains obstacles which do not overlap with regions. A cell is *region-free* if it does not belong to any region, and a path connecting two regions is *label-free* if it only passes through region-free cells. We represent the workspace by a graph $E = (S, \rightarrow_E)$ where S is the finite set of free cells and $\rightarrow_E \subseteq S \times S$ captures the adjacency relation.

Given the workspace E, we consider a team of n heterogeneous robots. We assume that these robots are of m different types and every robot belongs to exactly one type. Let $\mathcal{K}_j, j \in [m]$, denote the set that collects all robots of type j, so that $\sum_{j\in[m]} |\mathcal{K}_j| = n$ and $\mathcal{K}_j \cap \mathcal{K}_{j'} = \emptyset$ if $j \neq j'$, where $|\cdot|$ is the cardinality of a set. We collect all n robots in the set $\mathcal{R}$, i.e., $\mathcal{R} = \{\mathcal{K}_j\}_{j\in[m]}$. Finally, we use $[r, j]$ to represent robot r of type j, where $r \in \mathcal{K}_j, j \in [m]$. To model the motion of robot $[r, j]$ in the workspace, we define a transition system (TS) for this robot as follows.

Definition 3.1: (TS) A transition system for robot $[r, j]$ is a tuple $\text{TS}_{r,j} = (S, s^0_{r,j}, \rightarrow_{r,j}, \Pi_{r,j}, L_{r,j})$ where: (a) S is the set of free cells; (b) $s^0_{r,j}$ is the initial location of robot $[r, j]$; (c) $\rightarrow_{r,j} \subseteq \rightarrow_E \bigcup \cup_{s_{r,j}\in S}\{(s_{r,j}, s_{r,j})\}$ is the transition relation that allows the robots to remain idle or move between cells; (d) $\Pi_{r,j} = \cup_{k\in[l]}\{p^k_{r,j}\} \cup \{\epsilon\}$ where the atomic proposition $p^k_{r,j}$ is true if robot $[r, j]$ is at region ℓ_k and ϵ denotes the empty label; and (e) $L_{r,j} : S \rightarrow \Pi_{r,j}$ is the labeling function that returns the atomic proposition satisfied at location $s_{r,j}$.

Next we define the product transition system (PTS), which captures all possible combinations of robot behaviors.

Definition 3.2: (PTS) Given n transition systems $\text{TS}_{r,j} = (S, s^0_{r,j}, \rightarrow_{r,j}, \Pi_{r,j}, L_{r,j})$, the product transition system is a tuple $\text{PTS} = (S^n, s^0, \rightarrow, \Pi, L)$ where: (a) $S^n = S \times \cdots \times S$ is the finite set of collective robot locations; (b) s^0 are the initial locations of the robots; (c) $\rightarrow \subseteq S^n \times S^n$ is the transition relation so that $(s, s') \in \rightarrow$ if $s_{r,j} \rightarrow_{r,j} s'_{r,j}$ for all $r \in \mathcal{K}_j, \forall\, j \in [m]$; (d) $\Pi = \cup_{i\in[|\mathcal{K}_j|],j\in[m],k\in[l]}\{\pi^k_{i,j}\} \cup \{\epsilon\}$, where the atomic proposition $\pi^k_{i,j}$ is true if there exist at least i robots of type j, denoted by $\langle i, j\rangle$, at region ℓ_k at the same time, i.e., $\pi^k_{i,j} \Leftrightarrow |\{r \in \mathcal{K}_j : L_{r,j}(s_{r,j}) = p^k_{r,j}\}| \geq i$; (e) and $L : S^n \rightarrow 2^\Pi$ is the labeling function.

B. Task specification

In this paper, we consider MRTA problems where the task can require the same fleet of robots to visit different regions in sequence, e.g., to deliver objects between different regions. To capture such tasks, we define *induced atomic propositions* over the set Π defined in Definition 3.2 as follows.

Definition 3.3: (Induced atomic propositions) For each basic atomic proposition $\pi^k_{i,j} \in \Pi$, we define an infinite set of induced atomic propositions $\{\pi^{k,\chi}_{i,j}\}_{\chi\in\mathbb{N}}$, where χ is a connector that binds the truth of atomic propositions with identical i, j and χ. Specifically, when $\chi = 0$, $\pi^{k,\chi}_{i,j}$ is equivalent to $\pi^k_{i,j}$ whose truth is state-dependent. When $\chi \neq 0$, the truth of $\pi^{k,\chi}_{i,j}$ is state-and-path-dependent, meaning that it additionally depends on other induced atomic propositions that share the same i, j and χ. That is, both $\pi^{k,\chi}_{i,j}$ and $\pi^{k',\chi}_{i,j}$ with $\chi \neq 0$ are true if the same i robots of type j visit regions ℓ_k and $\ell_{k'}$. Furthermore, the negative atomic proposition $\neg\pi^{k,\chi}_{i,j}$ is equivalent to its basic counterpart $\neg\pi^k_{i,j}$, i.e., less than $\langle i, j\rangle$ robots are at region ℓ_k.

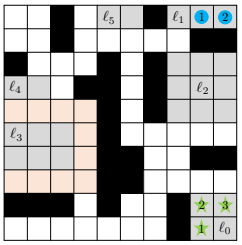


Fig. 1. Illustration of the workspace for Example 1.

Let $\mathcal{AP}$ collect all basic and induced atomic propositions and denote by $\Sigma = 2^{\mathcal{AP}}$, its power set. In what follows, we omit the superscript χ when $\chi = 0$. We denote by LTL^χ, the set of formulas defined over the set of basic and induced atomic propositions and by LTL^0, the set of formulas defined only over basic atomic propositions, respectively. Clearly, $\text{LTL}^\chi \supset \text{LTL}^0$, which means that LTL^χ can capture a broader class of tasks. While there exists literature on feasible control synthesis over LTL^0 [35, 36], to the best of our knowledge there is no work on optimal control synthesis over LTL^χ formulas. Next, we introduce the notion of *valid* temporal logic tasks.

Definition 3.4: (Valid temporal logic task) A temporal logic task specified by a LTL^χ formula defined over $\mathcal{AP}$ is *valid* if atomic propositions with the same nonzero connector χ involve the same number of robots of the same type.

Example 1: (Valid temporal logic tasks) Consider a mail delivery task amidst the COVID-19 pandemic (shown in Fig. 1) where three robots of type 1 (green stars) and two robots of type 2 (blue circles) are located at region ℓ_0 and ℓ_1, respectively, ℓ_2 is an office building that the robots visit to pick up the mail, ℓ_3 and ℓ_5 are delivery sites, and ℓ_4 is a control room from where robots are guided to the orange area to get disinfected and then drop off the mail at the delivery site ℓ_3. Consider two delivery tasks: (i) Two robots of type 1 visit building ℓ_2 to collaboratively pick up the mail and deliver it to the delivery site ℓ_3, and one robot of type 2 must visit the control room ℓ_4 to disinfect robots of type 1 before they get to the delivery site ℓ_3. *(ii)* One robot of type 1 travels between building ℓ_2 and the delivery site ℓ_3 back and forth to transport equipment, assuming that the disinfection area operates automatically after task (i). Observe that in Fig. 1, the atomic propositions satisfied by initial robot locations are $\pi^0_{3,1}$ and $\pi^1_{2,2}$. Tasks (i) and (ii) can be captured by the valid formulas $\phi_1 = \Diamond((\pi^{2,1}_{2,1} \wedge \neg\pi^3_{2,1}) \wedge \Diamond\pi^{3,1}_{2,1}) \wedge \Diamond\pi^4_{1,2} \wedge \neg\pi^3_{2,1}\, \mathcal{U}\, \pi^4_{1,2}$ and $\phi_2 = \Box\Diamond(\pi^{2,1}_{1,1} \wedge \Diamond\pi^{3,1}_{1,1})$. Note that when binding two atomic propositions, the value of χ is immaterial as long as it is the same non-zero number. Therefore, ϕ_2 can also be written as $\Box\Diamond(\pi^{2,2}_{1,1} \wedge \Diamond\pi^{3,2}_{1,1})$. However, both formulas $\Diamond(\pi^{2,1}_{1,1} \wedge \Diamond\pi^{3,1}_{2,1})$ and $\Diamond(\pi^{2,1}_{2,2} \wedge \Diamond\pi^{3,1}_{2,1})$ are invalid as they connect different numbers of robots i and robot types j, respectively.

Let s^t be the collective state at time t. A path of length h is defined as $\tau = s^0 \dots s^h$ and it captures the collective behavior of the team such that $s^{t-1} \rightarrow s^t, \forall t \in [h]$. Given a valid LTL^χ formula ϕ, a path $\tau = \tau^{\text{pre}}[\tau^{\text{suf}}]^\omega$ in a prefix-suffix structure that satisfies ϕ exists since there exists an accepting run in

prefix-suffix form, where the prefix part $\tau^{\text{pre}} = s^0 \dots s^{h_1}$ is executed once followed by the indefinite execution of the suffix part $\tau^{\text{suf}} = s^{h_1} \dots s^{h_1+h_2} s^{h_1+h_2+1}$, where $s^{h_1+h_2+1} = s^{h_1}$ [2]. We say that a path τ satisfies ϕ if (a) the trace, defined as $\texttt{trace}(\tau) := L(s^0) \dots L(s^{h_1})[L(s^{h_1}) \dots L(s^{h_1+h_2+1})]^\omega$, belongs to $\texttt{Words}(\phi^0)$, where ϕ^0 is obtained by replacing all induced atomic propositions in ϕ by their counterparts with the zero connector and (b) it is the same $\langle i, j \rangle$ that satisfy the induced atomic propositions $\pi_{i,j}^{k,\chi}$ in ϕ sharing the same nonzero connector χ.

C. Problem definition

Given a path $\tau_{r,j} = s_{r,j}^0, s_{r,j}^1, \dots, s_{r,j}^h$ of length h for robot $[r, j]$, we define the cost of $\tau_{r,j}$ as $J(\tau_{r,j}) = \sum_{t=0}^{h-1} d(s_{r,j}^t, s_{r,j}^{t+1})$, where $d : S \times S \to \mathbb{R}^+ \cup \{0\}$ is a cost function that maps a pair of free cells to a non-negative value, for instance, travel distance or time. The cost of path τ that combines all robot paths $\tau_{r,j}$ of length h is given by

$$J(\tau) = \sum_{r \in \mathcal{K}_j, j \in [m]} J(\tau_{r,j}). \tag{1}$$

For plans written in prefix-suffix form, we get

$$J(\tau) = \beta J(\tau^{\text{pre}}) + (1 - \beta) J(\tau^{\text{suf}}), \tag{2}$$

where $\beta \in [0, 1]$ is a user-specified parameter. Then, the problem addressed in this paper can be formulated as follows.

Problem 1: Consider a discrete workspace with labeled regions and obstacles, a team of n robots of m types, and a valid formula $\phi \in$ *LTL*$^\chi$. Plan a path for each robot such that the specification ϕ is satisfied and the cost in (2) is minimized.

We refer to Problem 1 as the Multi-Robot Task Allocation problem under LTL specifications or LTL-MRTA. This is a single-task robot and multi-robot task (ST-MR) problem, where a robot is capable of one task and a task may require multiple robots. Since the ST-MR problem is NP-hard [43], so is the LTL-MRTA problem. Consequently, existing approaches to this problem become intractable for large-scale applications [35]. In this work, we propose a new hierarchical framework to solve LTL-MRTA problems efficiently.

D. Assumptions

In this section, we discuss assumptions on the workspace and the NBA translated from the LTL specifications that are necessary to ensure completeness of our proposed method.

1) Workspace: The following assumption ensures that regions are well-defined and mutually exclusive.

Assumption 3.5: (Workspace) Regions are disjoint, and each region spans consecutive cells. There exists a label-free path between any two regions, between any two label-free cells, and between any label-free cells and any regions.

If regions are overlapping or span multiple clusters of cells, we can define additional atomic propositions to satisfy Assumption 3.5. Assumption 3.5 implies that there are no "holes" inside regions that generate different labels, label-free cells are connected, and each region is adjacent to a label-free cell.

2) Nondeterministic Büchi Automaton (NBA): Given a team of n robots and an LTL$^\chi$ formula ϕ, we can find a path τ that satisfies ϕ by operating on the corresponding NBA $\mathcal{A}_\phi = (\mathcal{V}, \mathcal{E})$, which can be constructed using existing tools, such as LTL2BA developed by [44]; see also Fig. 2 for the NBA of tasks (i) and (ii). Note that the NBA is essentially a graph. Thus, in the remainder of this paper, we refer to the NBA by the graph $\mathcal{A}_\phi$. Before discussing assumptions on the NBA $\mathcal{A}_\phi$, we present a list of pre-processing steps to obtain an "equivalent" NBA that does not lose any feasible paths that satisfy the specification ϕ. The goal is to remove infeasible and redundant transitions in the NBA to reduce its size.

Let the propositional formula γ associated with a transition $v_1 \xrightarrow{\gamma} v_2$ in the NBA $\mathcal{A}_\phi$ be in *disjunctive normal form* (DNF), i.e, $\gamma = \bigvee_{p \in \mathcal{P}} \bigwedge_{q \in \mathcal{Q}_p} (\neg) \pi_{i,j}^{k,\chi}$, where the negation operator can only precede the atomic propositions and $\mathcal{P}$ and $\mathcal{Q}_p$ are proper index sets. Note that any propositional formula has an equivalent DNF [2]. We call $\mathcal{C}_p^\gamma = \bigwedge_{q \in \mathcal{Q}_p} (\neg) \pi_{i,j}^{k,\chi}$ the p-th *clause* of γ that includes a set $\mathcal{Q}_p$ of positive and negative *literals* and each positive literal is an atomic proposition $\pi_{i,j}^{k,\chi} \in \mathcal{AP}$. Let $\mathsf{cls}(\gamma)$ denote the set of clauses in γ. Let $\mathsf{lits}^+(\mathcal{C}_p^\gamma)$ and $\mathsf{lits}^-(\mathcal{C}_p^\gamma)$ be the *positive subformula* and *negative subformula*, consisting of all positive and negative literals in the clause $\mathcal{C}_p^\gamma$. Those subformulas are $\top$ (constant true) if the corresponding literals do not exist. In what follows, we do not consider self-loops when we refer to edges in $\mathcal{A}_\phi$, since self-loops can be captured by vertices. We call the propositional formula γ a *vertex label* if $v_1 = v_2$, otherwise, an *edge label*. With a slight abuse of notation, let $\gamma : \mathcal{V} \to \Sigma$ and $\gamma : \mathcal{V} \times \mathcal{V} \to \Sigma$ be the functions that map a vertex and edge to its vertex label and edge label, respectively. Given an edge (v_1, v_2), we call labels $\gamma(v_1)$ and $\gamma(v_2)$ the *starting* and *end* vertex labels, respectively. Next, we pre-process the NBA $\mathcal{A}_\phi$ by removing infeasible clauses and merging redundant literals. Consider a vertex or edge label γ in $\mathcal{A}_\phi$,

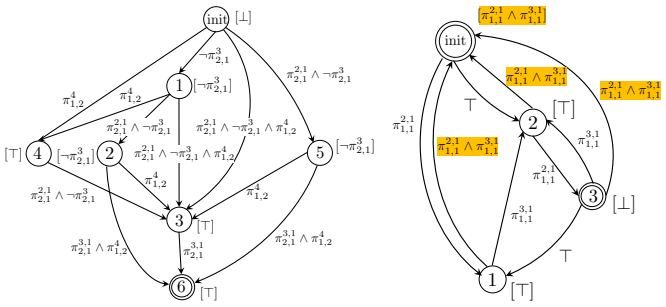
(1) Absorption in $\mathsf{lits}^+(\mathcal{C}_p^\gamma)$*:* For each clause $\mathcal{C}_p^\gamma \in \mathsf{cls}(\gamma)$, we delete the positive literal $\pi_{i,j}^k \in \mathsf{lits}^+(\mathcal{C}_p^\gamma)$, replacing it with $\top$, if another $\pi_{i',j}^{k,\chi'} \in \mathsf{lits}^+(\mathcal{C}_p^\gamma)$ exists such that $i \leq i'$. This is because if $\langle i', j \rangle$ are at region ℓ_k, i.e., $\pi_{i',j}^{k,\chi'}$ is true, so is $\pi_{i,j}^k$. Similarly, we replace $\pi_{i,j}^k$ by $\pi_{i-i',j}^k$ if $i > i'$, since $i - i'$ additional robots are needed to make $\pi_{i,j}^k$ true if $\pi_{i',j}^{k,\chi'}$ is true.

(2) Absorption in $\mathsf{lits}^-(\mathcal{C}_p^\gamma)$*:* We delete the negative literal $\neg\pi_{i,j}^k \in \mathsf{lits}^-(\mathcal{C}_p^\gamma)$, if another $\neg\pi_{i',j}^k \in \mathsf{lits}^-(\mathcal{C}_p^\gamma)$ exists such that $i' < i$. This is because if $\neg\pi_{i',j}^k$ is true, so is $\neg\pi_{i,j}^k$.

(3) Mutual exclusion in $\mathsf{lits}^+(\mathcal{C}_p^\gamma)$*:* We delete the clause $\mathcal{C}_p^\gamma \in \mathsf{cls}(\gamma)$, replacing it with constant false $\bot$, if there exist two positive literals $\pi_{i,j}^{k,\chi}, \pi_{i,j}^{k',\chi} \in \mathsf{lits}^+(\mathcal{C}_p^\gamma)$ such that $k \neq k'$ and $\chi \neq 0$. The reason is that the same i robots of type j cannot be at different regions at the same time.

(4) Mutual exclusion in $\mathsf{lits}^+(\mathcal{C}_p^\gamma)$ *and* $\mathsf{lits}^-(\mathcal{C}_p^\gamma)$*:* We delete the clause $\mathcal{C}_p^\gamma \in \mathsf{cls}(\gamma)$ if there exists a positive literal $\pi_{i,j}^{k,\chi} \in \mathsf{lits}^+(\mathcal{C}_p^\gamma)$ and a negative literal $\neg\pi_{i',j}^k \in \mathsf{lits}^-(\mathcal{C}_p^\gamma)$ such that $i' \leq i$. This is because these literals are mutually exclusive.

(5) Violation of team size: For each clause $\mathcal{C}_p^\gamma \in \mathsf{cls}(\gamma)$, let $\mathsf{lits}^+(j')$ denote literals in $\mathsf{lits}^+(\mathcal{C}_p^\gamma)$ that involve robots of type j', i.e., $\mathsf{lits}^+(j') = \{\pi_{i,j}^{k,\chi} \in \mathsf{lits}^+(\mathcal{C}_p^\gamma) | j = j'\}$. We delete the clause $\mathcal{C}_p^\gamma$ if the total required number of robots of type j exceeds the size $|\mathcal{K}_j|$, i.e., if there exists $j \in [m]$ such that $\sum_{\pi_{i,j}^{k,\chi} \in \mathsf{lits}^+(j)} i > |\mathcal{K}_j|$.



(a) NBA $\mathcal{A}_\phi$ for the task (i) (b) NBA $\mathcal{A}_\phi$ for the task (ii)

Fig. 2. NBA $\mathcal{A}_\phi$ for tasks (i) and (ii), where self-loops are omitted and the corresponding vertex labels are placed in square brackets.

Note that these pre-processing steps do not compromise any accepting words in $\mathcal{L}(\mathcal{A}_\phi)$ that can be generated by a feasible path. With a slight abuse of notation, we continue to use $\mathcal{A}_\phi$ to refer to the NBA obtained after these pre-processing steps.

Consider an edge $e = (v_1, v_2)$ and its starting vertex v_1 in $\mathcal{A}_\phi$ and assume that the current state of $\mathcal{A}_\phi$ is vertex v_1. To transition to v_2, certain robots need to simultaneously reach certain regions or avoid certain regions in order to make $\gamma(v_1, v_2)$ true, while maintaining $\gamma(v_1)$ true en route. We assume that the transition to v_2 occurs immediately once $\gamma(v_1, v_2)$ becomes true. Therefore, we can define by a *subtask* the set of actions that need to be taken by robots in order to activate a transition in the NBA.

Definition 3.6: (Subtask) Given an edge (v_1, v_2) in the NBA $\mathcal{A}_\phi$, a subtask is defined by the associated edge label $\gamma(v_1, v_2)$ and starting vertex label $\gamma(v_1)$.

Subtasks can be viewed as generalized reach-avoid tasks where robots visit or avoid certain regions (the "reach" part) while satisfying the starting vertex labels along the way (the "avoid" part, which here is more general than avoiding obstacles). Observe that every accepting run in the NBA consists of a sequence of subtasks. However, not all sequences of subtasks are feasible. In what follows, we restrict the accepting runs to those that can complete the task by first providing insights using the following example.

Example 1: continued (Subtasks) The NBAs for tasks (i) and (ii) before pre-processing are shown in Fig. 2. After pre-processing, the NBA $\mathcal{A}_\phi$ for task (i) does not change whereas some labels (orange color) in the NBA $\mathcal{A}_\phi$ for task (ii) become $\bot$ due to step *(3)*. In Fig. 2(a), v_{init} is the initial vertex and v_6 is the accepting vertex. Observe that all vertices have self-loops except for the initial vertex v_{init}. In each accepting run, e.g., $v_{\text{init}}, v_1, v_2, v_3, v_6, v_6^\omega$, the satisfaction of an edge label leads to the satisfaction of its end vertex label, assuming this end vertex label is not $\bot$. For instance, label $\pi^{2,1}_{2,1} \wedge \neg\pi^3_{2,1}$ of edge (v_1, v_2) implies label $\neg\pi^3_{2,1}$ of vertex v_2, and label $\neg\pi^3_{2,1}$ of edge (v_{init}, v_1) implies label $\neg\pi^3_{2,1}$ of its end vertex v_1. Intuitively, the completion of a subtask indicated by the satisfaction of its edge label, automatically activates the subtasks that immediately follow it indicated by the satisfaction of their starting vertex labels. This is because once the edge is enabled, its end vertex label should be satisfied at the next time instant; otherwise, progress in the NBA $\mathcal{A}_\phi$ will get stuck. The same observation also applies

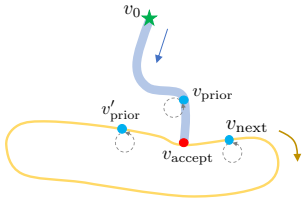


Fig. 3. Graphical depiction of the accepting run in the prefix-suffix structure when v_{accept} does not have a self-loop. The shaded blue line and the orange loop represent the prefix and suffix part, respectively. The arrow indicates the progression direction and the gray circles indicate the self-loops.

to the NBA in Fig. 2(b) where the vertex v_{init} is both an initial and accepting vertex and v_3 is another accepting vertex. The accepting run $v_{\text{init}}, v_2, v_3, (v_1, v_2, v_3)^\omega$ includes one pair of initial and accepting vertices, v_{init} and v_3, and the accepting run $v_{\text{init}}, v_2, v_1, v_{\text{init}}^\omega$ (although infeasible) includes one pair of initial and accepting vertices, v_{init} and v_{init}. Note that we view the two v_{init} vertices differently, one as the initial vertex and the other as the accepting vertex. Furthermore, label $\pi^{3,1}_{1,1}$ of edge (v_1, v_2) implies label $\top$ of its end vertex v_2; the same holds for the edge (v_2, v_{init}) and its end vertex v_{init} (although infeasible). It is noteworthy that even though the accepting vertex v_3 does not have a self-loop, the satisfaction of the label $\pi^{2,1}_{1,1}$ of its incoming edge (v_2, v_3) leads to the satisfaction of the label $\top$ of its outgoing edge (v_3, v_1). If the satisfaction of the incoming edge label does not imply satisfaction of the outgoing edge label, then progress in the NBA will get stuck at v_3 since the label $\pi^{2,1}_{1,1} \wedge \pi^{3,1}_{1,1}$ of edge (v_3, v_{init}) is infeasible and the transition between regions ℓ_2 and ℓ_3 requires more than one time steps; see Fig. 1, which makes the label $\pi^{3,1}_{1,1}$ of edge (v_3, v_2) unsatisfiable at the next time instant.

Motivated by the observations in Example 1, we introduce the notions of *implication* and *strong implication* between two propositional formulas. Then, we define a *restricted accepting run* in the NBA $\mathcal{A}_\phi$ in a prefix-suffix structure. The completeness of our method relies on the assumption that the set of restricted accepting runs in the NBA $\mathcal{A}_\phi$ is nonempty.

Definition 3.7: (Implication and strong implication) Given two propositional formulas γ and γ' over $\mathcal{AP}$, we say that formula γ implies γ', denoted by $\gamma \Longrightarrow \gamma'$, if for each clause $\mathcal{C}^\gamma_p \in \mathsf{cls}(\gamma)$, there exists a clause $\mathcal{C}^{\gamma'}_{p'} \in \mathsf{cls}(\gamma')$ such that $\mathcal{C}^{\gamma'}_{p'}$ is a subformula of $\mathcal{C}^\gamma_p$, i.e., all literals in $\mathcal{C}^{\gamma'}_{p'}$ also appear in $\mathcal{C}^\gamma_p$. By default, $\top$ is a subformula of any clause. In addition, formula γ strongly implies γ', denoted by $\gamma \Longrightarrow_s \gamma'$, if $\gamma \Longrightarrow \gamma'$, and for each clause $\mathcal{C}^{\gamma'}_{p'} \in \mathsf{cls}(\gamma')$, there exists a clause $\mathcal{C}^\gamma_p \in \mathsf{cls}(\gamma)$ such that $\mathcal{C}^{\gamma'}_{p'}$ is a subformula of $\mathcal{C}^\gamma_p$.

Intuitively, if $\gamma \Longrightarrow \gamma'$ or $\gamma \Longrightarrow_s \gamma'$, robot locations that satisfy γ also satisfy γ'.

Definition 3.8: (Restricted accepting run) Given the pre-processed NBA $\mathcal{A}_\phi$ corresponding to an LTL$^\chi$ formula, we call any accepting run in a prefix-suffix structure $\rho = \rho^{\text{pre}}[\rho^{\text{suf}}]^\omega = v_0, \ldots, v_{\text{prior}}, v_{\text{accept}}[v_{\text{next}}, \ldots, v'_{\text{prior}}, v_{\text{accept}}]^\omega$ (see Fig. 3), a restricted accepting run, if it satisfies conditions:

(a) If a vertex is both an initial vertex v_0 and an accepting vertex v_{accept}, we treat it as two different vertices, namely an initial vertex and an accepting vertex. The accepting vertex v_{accept} appears only once at the end in both the prefix and suffix parts. In the prefix part $v_0, \ldots, v_{\text{prior}}, v_{\text{accept}}$, if a vertex appears

multiple times, all repetitive occurrences are consecutive. The same holds for the suffix part $v_{\text{next}}, \ldots, v'_{\text{prior}}, v_{\text{accept}}$;
(b) There only exist one initial vertex v_0 and one accepting vertex v_{accept} in the accepting run (they can appear multiple times in a row). Different accepting runs can have different pairs of initial and accepting vertices;
(c) In the prefix part, only initial and accepting vertices, v_0 and v_{accept}, are allowed not to have self-loops, i.e., their vertex labels can be $\perp$. In the suffix part, only the accepting vertex v_{accept} is allowed not to have a self-loop;
(d) For any two consecutive vertices v_1, v_2 in the accepting run ρ, if $v_1 \neq v_2$, $v_2 \neq v_{\text{accept}}$ and v_2 has a self-loop, then the edge label $\gamma(v_1, v_2)$ strongly implies the end vertex label $\gamma(v_2)$, i.e., $\gamma(v_1, v_2) \Longrightarrow_s \gamma(v_2)$;
(e) In the suffix part ρ^{suf}, if $v_{\text{accept}} = v_{\text{next}}$ (this happens when v_{accept} has a self-loop), then ρ^{suf} only contains the vertex v_{accept}. Meanwhile, the label of the edge $(v_{\text{prior}}, v_{\text{accept}})$ implies the label of the vertex v_{accept}, i.e., $\gamma(v_{\text{prior}}, v_{\text{accept}}) \Longrightarrow \gamma(v_{\text{accept}})$;
(f) In the suffix part, if $v_{\text{accept}} \neq v_{\text{next}}$ (this can happen when v_{accept} does not have a self-loop), then the label of the edge $(v_{\text{prior}}, v_{\text{accept}})$ implies the label of the edge $(v_{\text{accept}}, v_{\text{next}})$, i.e., $\gamma(v_{\text{prior}}, v_{\text{accept}}) \Longrightarrow \gamma(v_{\text{accept}}, v_{\text{next}})$. Also, the label $\gamma(v_{\text{prior}}, v_{\text{accept}})$ implies the label of the edge $(v'_{\text{prior}}, v_{\text{accept}})$, i.e., $\gamma(v_{\text{prior}}, v_{\text{accept}}) \Longrightarrow \gamma(v'_{\text{prior}}, v_{\text{accept}})$. Note that v_{prior} and v'_{prior} can be different.

In what follows, we discuss the conditions in Definition 3.8 in more detail. Specifically, conditions (a) and (b) require that a restricted accepting run is "simple". Specifically, condition (a) states that vertices v_0 and v_{accept} can be treated differently since they mark different progress towards accomplishing a task. The prefix and suffix parts of a restricted accepting run end once v_{accept} is reached, as in [8]. By aggregating consecutive identical vertices in the prefix part of a restricted accepting run into one single vertex, there are no identical vertices in the "compressed" prefix part. That is, it contains no cycles. The presence of a cycle is redundant since it implies negative progress towards accomplishing the task. The same applies to the suffix part. On the other hand, condition (b) states that a restricted accepting run is basically an accepting run defined in Section II-A that is further defined over a pair of initial and accepting vertices. In Section IV, we extract smaller sub-NBAs from the NBA $\mathcal{A}_\phi$ for each pair of initial and accepting vertices, which helps reduce complexity of the problem.

Conditions (c)-(f) require that the completion of a subtask in a restricted accepting run automatically activates the subtasks that immediately follow it; see Example 1. This ensures that robots are given adequate time to undertake subsequent subtasks after completing the current subtask. Accepting runs that do not satisfy conditions (c)-(d) are disregarded. In fact, in Section IV-A we prune vertices and edges in the NBA that violate these conditions, further reducing the size of the NBA. Finally, the implication $\gamma(v_{\text{prior}}, v_{\text{accept}}) \Longrightarrow \gamma(v_{\text{accept}}, v_{\text{next}})$ in condition (f) requires that the robot locations enabling the last edge in the prefix part of a restricted accepting run also enable the first edge in the suffix part. As a result, we can find the prefix and suffix parts of a restricted accepting run separately. Otherwise, the progress in the NBA $\mathcal{A}_\phi$ may get stuck since these two edge labels need to be satisfied at two consecutive time instants, similar to conditions (d) and (e). Also, as the suffix part of a restricted accepting run is a loop, robots need to return to their initial locations in the suffix part after executing the suffix part once. The relation $\gamma(v_{\text{prior}}, v_{\text{accept}}) \Longrightarrow \gamma(v'_{\text{prior}}, v_{\text{accept}})$ requires that the initial locations in the suffix part of a restricted accepting run enable the edge $(v'_{\text{prior}}, v_{\text{accept}})$, which ensures that the robots can travel back to the initial locations in the suffix part and, as a result, activate the transition in $\mathcal{A}_\phi$ back to the vertex v_{accept} that allows to repeat the suffix part ρ^{suf}. Finally, we make the following assumption on the structure of the NBA $\mathcal{A}_\phi$.

Assumption 3.9: (Existence of restricted accepting runs) The set of restricted accepting runs is non-empty.

We note that the sets of restricted accepting runs for tasks (i) and (ii) satisfy Assumption 3.9. Common robotic tasks, such as sequencing and coverage, have NBAs that contain restricted accepting runs. However, there is also a small subclass of LTL where the "next" operator directly precedes an atomic proposition that violates this assumption. For instance, $\Diamond(\pi^{2,1}_{1,1} \wedge \bigcirc \pi^{3,1}_{1,2})$ requires a second robot to visit region ℓ_3 immediately after the first robot reaches ℓ_2, which does not allow for any physical time between the completion of the two consecutive subtasks. On the other hand, the LTL formula $\Diamond(\pi^{2,1}_{1,1} \wedge \bigcirc(\pi^{2,1}_{1,1} \mathcal{U} \pi^{3}_{2,1}))$ satisfies the assumption.

3) Robot paths: The definition of restricted accepting runs is based entirely on the structure of the NBA and logical implication relations. However, Definition 3.8 does not describe how to characterize robot paths that induce restricted accepting runs. Next, we discuss conditions under which robot paths satisfy restricted accepting runs. We call such paths *satisfying paths* and we assume such satisfying paths exist.

Definition 3.10: (Satisfying paths of restricted accepting runs) Given a team of n robots and a valid specification $\phi \in$ *LTL*^{X}, a robot path τ is a satisfying path that induces a restricted accepting run, if the following conditions hold:
(a) If a vertex label is satisfied by the path τ, it is always satisfied by the same clause that is always satisfied by the same fleet of robots;
(b) If a clause in an edge label is satisfied by the path τ, then a clause in the end vertex label is also satisfied. Moreover, the fleet of robots satisfying the positive subformula of the clause in the end vertex label is the same as the fleet of robots satisfying the positive subformula of the clause in the corresponding edge label;
(c) Robot locations enabling the edges $(v_{\text{accept}}, v_{\text{next}})$ and $(v'_{\text{prior}}, v_{\text{accept}})$ in the suffix part of a restricted accepting run are identical to robot locations enabling the edge $(v_{\text{prior}}, v_{\text{accept}})$ in the prefix part.

Definition 3.10 is closely related to the definition of a restricted accepting run. Specifically, condition (a) in Definition 3.10 requires that once a fleet of robots satisfies a vertex label in a restricted accepting run, then these robots remain idle during the next time instant so that the same clause in this vertex label is still satisfied. This satisfies condition (a) in Definition 3.8. Furthermore, condition (b) in Definition 3.10 requires that once a fleet of robots satisfies an edge label in a restricted accepting run, then these robots remain idle during

the next time instant so that the clause in the end vertex label that is implied by the clause that is satisfied in the edge label is also satisfied. This satisfies condition (d) in Definition 3.8.

Finally, condition (c) in Definition 3.10 requires that the robot locations enabling the edge $(v_{\text{prior}}, v_{\text{accept}})$ in the prefix part of a restricted accepting run coincide with the initial locations of the robots that enable the edge $(v'_{\text{prior}}, v_{\text{accept}})$ in the suffix part of the restricted accepting run, as per condition (f) in Definition 3.8. Therefore, condition (c) in Definition 3.10 requires that the robots travel along a loop so that the suffix part of the restricted accepting run is executed indefinitely. In what follows, we make the following assumption.

Assumption 3.11: (Existence of paths) Robot paths exist that satisfy the restricted accepting runs in the NBA $\mathcal{A}_\phi$.

E. Outline of the proposed method

An overview of our proposed method is shown in Alg. 1, which first finds prefix paths and then suffix paths. The process of finding prefix or suffix paths consists of relaxation and correction stages. Specifically, during the relaxation stage, we ignore the negative literals in the NBA $\mathcal{A}_\phi$ and formulate a MILP to allocate subtasks to robots and determine time-stamped robot waypoints that satisfy the task assignment. To this end, we first prune the NBA $\mathcal{A}_\phi$ by deleting infeasible transitions and then relax it by removing negative subformulas so that transitions in the relaxed NBA are solely satisfied by robots that meet at certain regions [line 1]; see Section IV-A. The idea to temporarily remove negative literals from the NBA is motivated by "lazy collision checking" methods in robotics and allows to simplify the planning problem as the constraints are not considered during planning and are only checked during execution, when needed. Then, since by condition (b) in Definition 3.8, restricted accepting runs contain only one initial vertex and one accepting vertex, for every sorted pair of initial and accepting vertices by length in the relaxed NBA, we extract a sub-NBA of smaller size [line 2]; see Section IV-B, where $\mathcal{A}_{\text{subtask}}(v_{\text{init}}, v_{\text{accept}})$ is the sub-NBA including only initial vertex v_{init}, accept vertex v_{accept} and other intermediate vertices. The sub-NBAs are used to extract subtasks and temporal orders between them captured by a set of posets ([lines 3-4], see Section IV-C), and construct routing graphs, one for each poset, that capture the regions that the robots need to visit and the temporal order of the visits so that the subtasks extracted from the sub-NBAs are satisfied; see Section V-A. Finally, given the routing graph corresponding to each poset we formulate a MILP inspired by the vehicle routing problem to obtain a high-level task allocation plan along with time-stamped waypoints that the robots need to visit to satisfy the task assignment [lines 6-7]; see Section V-B. During the correction stage, we introduce the negative literals back into the NBA and formulate a collection of generalized multi-robot path planning problems, one for each poset, to design low-level executable robot paths that satisfy the original specification ([line 8], see Section V-C). Viewing the final states of the prefix paths as the initial states, a similar process is conducted for the sub-NBA $\mathcal{A}_{\text{subtask}}(v_{\text{accept}}, v_{\text{accept}})$ to find the suffix paths. Alg. 1 can terminate after a specific number of paths is found or all possible alternatives are explored. Under the mild assumptions discussed in Section III-D, completeness of our proposed method is shown in Theorem 6.8 in Section VI.

Algorithm 1: Algorithm for LTL-MRTA

```
1  Prune and relax the NBA ;
2  foreach sorted sub-NBA A_subtask(v_init, v_accept) do
       ;                                          ▷ Compute the prefix path
3      Prune the sub-NBA A_subtask(v_init, v_accept);
4      Infer the set of posets {P_pre};
5      foreach sorted poset P_pre do
6          Build the routing graph ;
7          Formulate MILP to get prefix high-level plans ;
8          Formulate generalized multi-robot path planning to get prefix
             low-level paths ;
           ;                                      ▷ Compute the suffix path
9          Prune the sub-NBA A_subtask(v_accept, v_accept);
10         Infer the set of posets {P_suf};
11         foreach sorted poset P_suf do
12             Build the routing graph;
13             Formulate MILP to get suffix high-level plans;
14             Formulate generalized multi-robot path planning to get suffix
                 low-level paths;
```

Remark 3.12: We note that Assumption 3.11 on the existence of satisfying paths is only a sufficient condition that needs to be satisfied to show completeness of our proposed method, as shown in the theoretical analysis of Section VI. The robot path returned by our method may not be a satisfying path, although it still satisfies the specification ϕ.

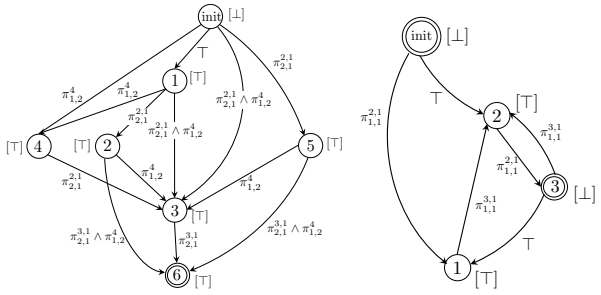
IV. Extraction of Subtasks from the NBA and Inferring their Temporal Order

In this section, we first prune and relax the NBA $\mathcal{A}_\phi$ by removing infeasible transitions and negative literals. As discussed before, this step is motivated by "lazy collision checking" methods in robotics and allows to simplify the planning problem by checking constraint satisfaction during the execution of the plans rather than their synthesis. Then, we extract sub-NBAs from the relaxed NBA and use these sub-NBAs to obtain sequences of subtasks and a temporal order between them that need to be satisfied so that the global specification is satisfied.

A. Pruning and relaxation of the NBA

To prune infeasible transitions from the NBA $\mathcal{A}_\phi$, we first delete all edges labeled with $\bot$, as they cannot be enabled. We also delete vertices and edges in $\mathcal{A}_\phi$ that do not belong to restricted accepting runs, as defined in Definition 3.8. Specifically, we delete all vertices without self-loops except for the initial and accepting vertices, as per condition (c) in Definition 3.8. Furthermore, for every vertex other than the accepting vertex, we delete all its incoming edges with edge labels that do not strongly imply its vertex label, as per condition (d) in Definition 3.8. Finally, we delete every vertex, except for the initial vertex, that cannot be reached by other vertices. We note that these pruning steps do not compromise any feasible solution to Problem 1 that induces a restricted accepting run in $\mathcal{A}_\phi$, as shown in Lemma 6.2 in Section VI.

We denote by $\mathcal{A}_\phi^-$ the resulting pruned NBA. Given the pruned NBA $\mathcal{A}_\phi^-$, we further relax it by replacing each negative literal in vertex or edge labels with $\top$. Let $\mathcal{A}_{\text{relax}}$ denote the relaxed NBA. Note that, when the specification ϕ does not involve negative atomic propositions, we have $\mathcal{A}_{\text{relax}} = \mathcal{A}_\phi^-$. Furthermore, Lemma 6.3 states that the language accepted by $\mathcal{A}_\phi^-$ is included in the language accepted by $\mathcal{A}_{\text{relax}}$, so this relaxation step does not remove feasible solutions to



(a) $\mathcal{A}_{\text{relax}}$ for task (i) (b) $\mathcal{A}_{\text{relax}}$ for task (ii)

Fig. 4. The relaxed NBA $\mathcal{A}_{\text{relax}}$ for tasks (i) and (ii).

Problem 1. In other words, $\mathcal{A}_{\text{relax}}$ is an over-approximation of $\mathcal{A}_\phi^-$. However, a solution to Problem 1 based on $\mathcal{A}_{\text{relax}}$ may not satisfy the specification ϕ. Therefore, the correction stage is necessary in order to introduce the negative literals back in the NBA and modify the solution obtained from $\mathcal{A}_{\text{relax}}$ in order to satisfy ϕ. Note that $\mathcal{A}_\phi^-$ and $\mathcal{A}_{\text{relax}}$ are sub-NBAs of $\mathcal{A}_\phi$ in terms of vertices and edges. Thus, labels and runs in $\mathcal{A}_\phi^-$ and $\mathcal{A}_{\text{relax}}$ can be mapped to labels and runs in $\mathcal{A}_\phi$. For instance, for an edge label γ in $\mathcal{A}_{\text{relax}}$, we denote by γ_ϕ the corresponding label in $\mathcal{A}_\phi$ (including negative literals).

Example 1: continued (Pruning and relaxation of the NBA $\mathcal{A}_\phi$) The pruned NBA $\mathcal{A}_\phi^-$ for the task (i) is the same as the original NBA in Fig. 2(a). The relaxed NBA $\mathcal{A}_{\text{relax}}$ is shown in Fig. 4(a). The pruned NBA $\mathcal{A}_\phi^-$ for the task (ii) is the same as the relaxed NBA $\mathcal{A}_{\text{relax}}$ in Fig. 4(b). Particularly, $\mathcal{A}_\phi^-$ is obtained from $\mathcal{A}_\phi$ in Fig. 2(b) by removing edges $(v_1, v_{\text{init}}), (v_2, v_{\text{init}}), (v_3, v_{\text{init}})$ and replacing $\gamma(v_{\text{init}})$ with $\bot$.

B. Extraction of sub-NBA $\mathcal{A}_{\text{subtask}}$ from $\mathcal{A}_{\text{relax}}$

In this section, we extract multiple sub-NBAs from the relaxed NBA $\mathcal{A}_{\text{relax}}$, one for every pair of initial and accepting vertices in $\mathcal{A}_{\text{relax}}$. Then, in Section IV-C, we determine the temporal order among subtasks in every sub-NBA.

1) Sorting the pairs of initial and accepting vertices by path length: As required by condition (b) in Definition 3.8, every restricted accepting run in $\mathcal{A}_{\text{relax}}$ contains one pair of initial and accepting vertices. In what follows, we sort all pairs of initial and accepting vertices in $\mathcal{A}_{\text{relax}}$ in an ascending order so that the pair of initial and accepting vertices connected by a restricted accepting run with the shortest length appears first. Then in Section IV-B2, we extract a sub-NBA from $\mathcal{A}_{\text{relax}}$ for each pair in this ascending order. Intuitively, the sub-NBAs corresponding to restricted accepting runs of shorter length generally will contain fewer subtasks to be completed.

a) Computation of the shortest simple prefix path: Given a pair of an initial vertex v_0 and an accepting vertex v_{accept} in $\mathcal{A}_{\text{relax}}$, we first compute the shortest simple path from v_0 to v_{accept} in terms of the number of edges/subtasks, where a simple path does not contain any repeating vertices, as per condition (a) in Definition 3.8 that excludes cycles from restricted accepting runs. This step corresponds to the prefix part of a restricted accepting run. To this end, we first remove all other initial vertices and accepting vertices from $\mathcal{A}_{\text{relax}}$. This will not affect the restricted accepting runs in $\mathcal{A}_{\text{relax}}$ associated with the pair v_0 and v_{accept} due to condition (b) in Definition 3.8. Then, depending on whether the initial vertex v_0 has a self-loop, we proceed as follows.

(1) If v_0 does not have a self-loop, i.e., $\gamma(v_0) = \bot$: We remove all outgoing edges v_0 in $\mathcal{A}_{\text{relax}}$ with label γ, if the initial robot locations do not satisfy the corresponding edge label γ_ϕ (including the negative literals) in $\mathcal{A}_\phi$. We emphasize that we need to check satisfaction of γ_ϕ in the NBA $\mathcal{A}_\phi$ instead of satisfaction of γ in the relaxed NBA $\mathcal{A}_{\text{relax}}$, since if initial robot locations cannot enable an edge starting from v_0 in $\mathcal{A}_\phi$, there is no reason to consider this edge in any NBA.

(2) If v_0 has a self-loop, i.e., $\gamma(v_0) \neq \bot$: We check whether the initial robot locations satisfy $\gamma_\phi(v_0)$ in the NBA $\mathcal{A}_\phi$. If yes, we do nothing; otherwise, we proceed as in case (1) in this part and remove the self-loop of v_0 as well as all its outgoing edges in $\mathcal{A}_{\text{relax}}$ if the initial robot locations do not satisfy the corresponding edge label γ_ϕ in $\mathcal{A}_\phi$.

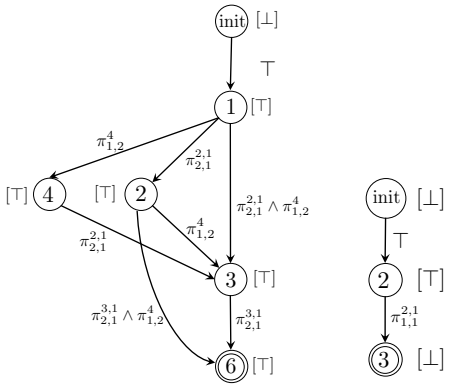
Next, the shortest simple path connecting v_0 and v_{accept} can be found using Dijkstra's algorithm. Note that if a vertex is both an initial and accepting vertex, we treat it once as the initial vertex and once as the accepting vertex, although it appears twice in the shortest simple path.

b) Computation of the shortest simple suffix cycle: Next, we compute the shortest simple cycle around v_{accept} in $\mathcal{A}_{\text{relax}}$, where repeating vertices only appear at the beginning and at the end of the simple cycle. This step corresponds to the suffix part of a restricted accepting run, which is conducted in the original NBA $\mathcal{A}_{\text{relax}}$. If v_{accept} in $\mathcal{A}_{\text{relax}}$ has a self-loop, then the length of the shortest simple cycle is 0. Otherwise, similar to steps in Section a) used to find the shortest simple prefix path, we first remove all other accepting vertices from $\mathcal{A}_{\text{relax}}$ and then remove all initial vertices (including v_0) if they do no have self-loops. In this way, the only vertex that does not have a self-loop is the accepting vertex v_{accept}. This will not affect those restricted accepting runs that are related to v_0 and v_{accept} due to conditions (b) and (c) in Definition 3.8.

Finally, the length associated with the pair v_0 and v_{accept} is equal to the total length of the shortest simple prefix path and the shortest simple suffix cycle connecting these vertices in $\mathcal{A}_{\text{relax}}$. By default, if no simple path or cycle exists for the pair v_0 and v_{accept}, the length is infinite, which means there is no restricted accepting run for this pair. We repeat this process for all pairs of initial and accepting vertices in $\mathcal{A}_{\text{relax}}$ and sort them in ascending order in terms of the total length. As discussed before, we plan first for pairs with shorter length since they contain fewer subtasks to be completed.

2) Extraction of the sub-NBA $\mathcal{A}_{\text{subtask}}$: For every pair of vertices v_0 and v_{accept} in $\mathcal{A}_{\text{relax}}$ connected by a simple path of finite total length in the above ascending order, our goal is to determine time-stamped task allocation plans for all robots that induce the simple prefix path and simple suffix cycle in $\mathcal{A}_{\text{relax}}$ connecting v_0 and v_{accept}. To do this, we extract one sub-NBA from the NBA $\mathcal{A}_{\text{relax}}$ that we can use to construct the prefix part of the plan and one that we can use to construct the suffix part of the plan, respectively. Here, we discuss the sub-NBA for the prefix part. The sub-NBA for the suffix part is similar and is discussed in Appendix A-C in [40].

Given the pair of vertices v_0 and v_{accept}, we construct a prefix sub-NBA $\mathcal{A}_{\text{subtask}}$ by the following three steps. First, we follow exactly the same steps in Section a) that computes



(a) $\mathcal{A}_{\text{subtask}}$ for task (i) (b) $\mathcal{A}_{\text{subtask}}$ for task (ii)

Fig. 5. Sub-NBA $\mathcal{A}_{\text{subtask}}$ for the prefix part of tasks (i) and (ii) in Example 1, obtained from the NBA $\mathcal{A}_{\text{relax}}$ in Fig. 4.

the shortest simple prefix path to prune the NBA $\mathcal{A}_{\text{relax}}$. Next, we remove all outgoing edges from v_{accept} if $v_{\text{accept}} \neq v_0$, because we focus on the prefix part. Finally, let $\mathcal{V}_{\text{s}}$ denote the set that contains all remaining vertices in $\mathcal{A}_{\text{relax}}$ that belong to some path connecting v_0 and v_{accept}. Then, we construct a sub-NBA $\mathcal{A}_{\text{subtask}} = (\mathcal{V}_{\text{s}}, \mathcal{E}_{\text{s}})$ from $\mathcal{A}_{\text{relax}}$ that includes all edges that connect the vertices in $\mathcal{V}_{\text{s}}$. The sub-NBA $\mathcal{A}_{\text{subtask}}$ contains prefix parts of all restricted accepting runs associated with the pair v_0 and v_{accept}, as shown in Lemma 6.4.

Example 1: continued (Sub-NBA $\mathcal{A}_{\text{subtask}}$) The sub-NBA $\mathcal{A}_{\text{subtask}}$ for the prefix parts of plans associated with tasks (i) and (ii) are shown in Fig. 5. For task (i), given the pair v_{init} and v_6 in the relaxed NBA $\mathcal{A}_{\text{relax}}$ in Fig. 4(a), the total length is $3 + 0 = 3$ (edges $(v_{\text{init}}, v_3), (v_{\text{init}}, v_4), (v_{\text{init}}, v_5)$ were removed since v_{init} does not have a self-loop and all robots initially located inside regions ℓ_0 and ℓ_1 do not satisfy their labels; see Fig. 1). The NBA $\mathcal{A}_{\text{subtask}}$, shown in Fig. 5(a) is obtained by removing edges $(v_{\text{init}}, v_3), (v_{\text{init}}, v_4), (v_{\text{init}}, v_5), (v_5, v_3), (v_5, v_6)$ and vertex v_5 from $\mathcal{A}_{\text{relax}}$. For task (ii), given the pair v_{init} and v_{init}, there is no cycle leading back to v_{init}, so the total length is infinite and there is no corresponding sub-NBA $\mathcal{A}_{\text{relax}}$. The total length for the pair v_{init} and v_3 is $2 + 2 = 4$. The NBA $\mathcal{A}_{\text{subtask}}$ is shown in Fig. 5(b), where edges (v_{init}, v_1) and (v_1, v_2) are removed since v_{init} does not have a self-loop and initial robot locations do not satisfy their labels.

Example 1: continued (Subtasks in $\mathcal{A}_{\text{subtask}}$) The sub-NBA $\mathcal{A}_{\text{subtask}}$ is composed of subtasks that need to be satisfied in specific orders to reach the accepting vertex. For instance, the path $v_{\text{init}}, v_1, v_4, v_3, v_6$ in Fig. 4(a) requires that first $\langle 1, 2\rangle$ visits the control room ℓ_4, then $\langle 2, 1\rangle$ visit the office building ℓ_2 and finally the same two robots of type 1 drop off the mail at the delivery site ℓ_3. By definition of task (i), the temporal order between these subtasks specifies that the time when $\langle 1, 2\rangle$ visits the control room ℓ_4 is independent from the time when $\langle 2, 1\rangle$ pick up the mail at the building ℓ_2, and that $\langle 1, 2\rangle$ visiting ℓ_4 and $\langle 2, 1\rangle$ visiting ℓ_2 should occur prior to $\langle 2, 1\rangle$ visiting the delivery site ℓ_3.

3) Pruning the sub-NBA $\mathcal{A}_{\text{subtask}}$: Observe that the sub-NBA $\mathcal{A}_{\text{subtask}}$ in Fig. 5(a) still constitutes a large portion of $\mathcal{A}_{\text{relax}}$ in Fig. 4(a), which is common in practice, since there are typically many more edges than vertices in $\mathcal{A}_{\text{relax}}$. However, some edges/subtasks are "redundant" in that they can be decomposed into more elementary edges/subtasks. Therefore, in what follows, we further prune the NBA $\mathcal{A}_{\text{subtask}}$ by removing such redundant edges.

Recall Definition 3.6 where subtasks are defined by their edge labels and starting vertex labels. Next we define the notion of equivalent subtasks.

Definition 4.1: (Equivalent subtasks) Subtasks (v_1, v_2) and (v_1', v_2') in an NBA $\mathcal{A}$ are equivalent, denoted by $(v_1, v_2) \sim (v_1', v_2')$, if $\gamma(v_1) = \gamma(v_1')$, $\gamma(v_1, v_2) = \gamma(v_1', v_2')$ and they are not in the same path that connects the same pair of initial and accepting vertices.

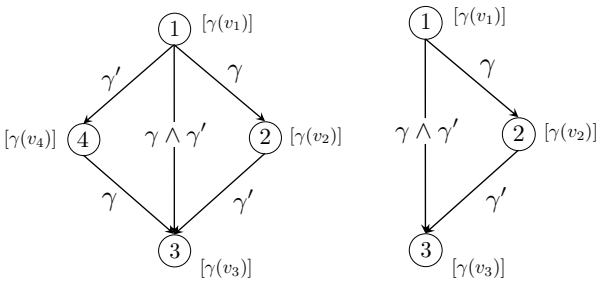
The last condition in Definition 4.1 is necessary since two subtasks in the same path mark different progress towards completing a task, even if they have identical labels. Recall that in task (i) in Example 1, certain regions can be visited in parallel. To capture the parallel visits, we define the following two properties over vertices in $\mathcal{A}_{\text{subtask}}$, namely, the independent diamond (ID) property adapted from [45] and the sequential triangle (ST) property over vertices; see also Fig. 6.

Definition 4.2: (Independent diamond property) Given four different vertices v_1, v_2, v_3, v_4 in the NBA $\mathcal{A}_{\text{subtask}}$, we say that these four vertices satisfy the ID property if (a) $\gamma(v_1) = \gamma(v_2) = \gamma(v_4)$; (b) $v_1 \xrightarrow{\gamma}_B v_2 \xrightarrow{\gamma'}_B v_3$; (c) $v_1 \xrightarrow{\gamma'}_B v_4 \xrightarrow{\gamma}_B v_3$; (d) $v_1 \xrightarrow{\gamma\wedge\gamma'}_B v_3$; (e) $\gamma_\phi(v_3) = \top$ if $v_3 = v_{\text{accept}}$.

Intuitively, if vertices v_1, v_2, v_3, and v_4 in $\mathcal{A}_{\text{subtask}}$ satisfy the ID property (see Fig. 6(a)), then conditions (a)-(c) in Definition 4.2 imply that the subtasks $(v_1, v_2) \sim (v_4, v_3)$ and $(v_1, v_4) \sim (v_2, v_3)$ in $\mathcal{A}_{\text{subtask}}$ are equivalent, while conditions (b)-(d) in Definition 4.2 state that their order is arbitrary, i.e., one can proceed the other or they can occur simultaneously. We refer to (v_1, v_3) as the *composite* subtask and (v_1, v_2), (v_1, v_4) as the *elementary* subtasks. Although both can lead to vertex v_3, composite subtasks are "redundant", since elementary subtasks can be executed independently and, therefore, their labels are easier to satisfy, compared to composite tasks that need to be executed simultaneously and, therefore, more conditions need to hold so that their labels are satisfied. Note that we conduct the $\top$-check in condition (e) in Definition 4.2 on the NBA $\mathcal{A}_\phi$ so that condition (f) in Definition 3.8 is satisfied which means that the set of restricted accepting runs is not affected if the edge (v_1, v_3) is removed. This result is formally shown in Lemma 6.5. In words, if $\gamma_\phi(v_3) \neq \top$ with $v_3 = v_{\text{accept}}$, and if a restricted accepting run traverses edges (v_1, v_{accept}) and $(v_{\text{accept}}, v_{\text{next}})$ where $v_{\text{accept}} \neq v_{\text{next}}$, then condition (f) in Definition 3.8 states that $\gamma_\phi(v_1, v_{\text{accept}}) \implies \gamma_\phi(v_{\text{accept}}, v_{\text{next}})$. However $\gamma_\phi(v_2, v_{\text{accept}})$ and $\gamma_\phi(v_4, v_{\text{accept}})$ may not imply $\gamma_\phi(v_{\text{accept}}, v_{\text{next}})$ since they are subformulas of $\gamma_\phi(v_1, v_{\text{accept}})$. Therefore, removing the composite edge (v_1, v_{accept}) risks emptying the set of restricted accepting runs.

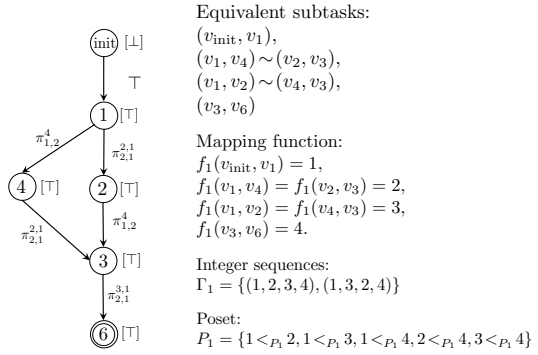
Definition 4.3: (Sequential triangle property) Given three different vertices v_1, v_2, v_3 in the NBA $\mathcal{A}_{\text{subtask}}$, we say that these three vertices v_1, v_2, v_3 satisfy the ST property if (a)$v_1 \xrightarrow{\gamma}_B v_2 \xrightarrow{\gamma'}_B v_3$; (b) $v_1 \xrightarrow{\gamma\wedge\gamma'}_B v_3$; (c) $\gamma_\phi(v_3) = \top$ if $v_3 = v_{\text{accept}}$.

If vertices v_1, v_2, and v_3 in $\mathcal{A}_{\text{subtask}}$ satisfy the ST property (see



(a) ID property (b) ST property

Fig. 6. Independent diamond and sequential triangle properties.



(a) $\mathcal{A}^-_{\text{subtask}}$ for task (i) (b) Subtasks and poset

Fig. 7. The NBA $\mathcal{A}^-_{\text{subtask}}$ and corresponding subtasks.

Fig. 6(b)), then conditions (a) and (b) in Definition 4.3 state that subtask (v_1, v_2) should be satisfied no later than (v_2, v_3). Note that if vertices v_1, v_2, v_3, v_4 satisfy the ID property, then v_1, v_2, v_3 and v_1, v_4, v_3 satisfy the ST property. Using these two properties, we remove all edges from $\mathcal{A}_{\text{subtask}}$ associated with composite subtasks and denote by $\mathcal{A}^-_{\text{subtask}}$ the resulting pruned $\mathcal{A}_{\text{subtask}}$. A composite subtask can be an elementary subtask of another composite subtask at a higher layer. Thus, removing composite subtasks is vital for reducing the size of $\mathcal{A}_{\text{subtask}}$. Similar to pruning $\mathcal{A}_\phi$ to get $\mathcal{A}^-_\phi$, the feasibility of Problem 1 is not compromised by pruning composite subtasks from the NBA $\mathcal{A}_{\text{subtask}}$, as shown in Lemma 6.5.

Example 1: continued (ID and ST properties and the resulting NBA $\mathcal{A}^-_{\text{subtask}}$) In the NBA $\mathcal{A}_{\text{subtask}}$ for task (i), shown in Fig. 5(a), the vertices v_1, v_2, v_3, v_4 satisfy the ID property and the vertices v_2, v_3, v_6 ($\gamma_\phi(v_6) = \top$ in Fig. 2(a)) satisfy the ST property. Thus we delete (v_1, v_3) and (v_2, v_6). The resulting $\mathcal{A}^-_{\text{subtask}}$ is shown in Fig. 7(a). The NBA $\mathcal{A}^-_{\text{subtask}}$ of task (ii) is the same as $\mathcal{A}_{\text{subtask}}$ since there are no composite subtasks.

C. Inferring the temporal order between subtasks in $\mathcal{A}^-_{\text{subtask}}$

In this section, we infer the temporal relation between subtasks in the pruned NBA $\mathcal{A}^-_{\text{subtask}} = (\mathcal{V}_s, \mathcal{E}_s)$. For this, we rely on partially ordered sets introduced in Section II-B. Specifically, let Θ denote the set that collects all simple paths connecting v_0 and v_{accept} in $\mathcal{A}^-_{\text{subtask}}$. We focus on simple paths since condition (a) in Definition 3.8 excludes cycles. Given a simple path $\theta \in \Theta$, let $\mathcal{T}(\theta)$ denote the set of subtasks in θ. We say that two simple paths θ_1 and θ_2 have the same set of subtasks if $\mathcal{T}(\theta_1) = \mathcal{T}(\theta_2)$. Then we partition Θ into subsets of simple paths that contain the same set of subtasks, that is, $\Theta = \cup_e \Theta_e$ where $\mathcal{T}(\theta_1) = \mathcal{T}(\theta_2)$ for all $\theta_1, \theta_2 \in \Theta_e$ and $\theta_1 \neq \theta_2$. The reason for this partition is that we want to map simple paths in $\mathcal{A}^-_{\text{subtask}}$ to posets, and the set of linear extensions generated by a poset has the same set of elements.

Given a subset Θ_e of simple paths in the partition, with a slight abuse of notation, let $\mathcal{T}(\Theta_e)$ denote the set of corresponding subtasks. Let the function $f_e : \mathcal{T}(\Theta_e) \to [|\mathcal{T}(\Theta_e)|]$ map each subtask to a distinct positive integer. Note that two different subtasks in two different subsets Θ_e and $\Theta_{e'}$ may be mapped to the same integer; however, we treat these two subsets separately. Using f_e, we can map every path in Θ_e to a sequence of integers, denoted by S_e. Let Γ_e collect all sequences of integers for all paths in Θ_e, so $|\Theta_e| = |\Gamma_e|$. Moreover, all sequences of integers in Γ_e are permutations of each other and we denote this base set by $X_e = [|\mathcal{T}(\Theta_e)|]$. For every sequence $S_e \in \Gamma_e$, let $S_e[i]$ denote its i-th entry. We define a linear order $L_{X_e} = (X_e, <_L)$ such that $S_e[i] <_L S_e[j]$ if $i < j$. In other words, the subtask $S_e[i]$ should be completed prior to $S_e[j]$. Then, let Ξ_e collect all linear orders over X_e that can be defined from sequences in Γ_e. A poset $P_e = (X_e, <_{P_e})$ containing the maximum number of linear orders in Ξ_e can be found using the algorithm proposed in [42] for the partial cover problem, where the order represents the precedence relation. Note that Ξ_e may not be identical to Ξ_{P_e}, the set of all linear extensions of P_e. Thus, after obtaining poset P_e, each of the remaining linear orders in Ξ_e that are not covered by P_e are treated as separate totally ordered sets, that are posets as well. In this way, we do not discard any posets.

Finally, given a partition $\{\Theta_e\}$ and a corresponding set of posets $\{P_e\}$, we sort $\{P_e\}$ lexicographically first in descending order in terms of the width of posets and then in ascending order in terms of the height. Recall that the width of a poset is the cardinality of its maximal antichain, and its height is the cardinality of its maximal chain; see Section II-B. Intuitively, the wider a poset is, the more temporally independent subtasks it contains. The shorter a poset is, the fewer subtasks it has. We consider first wider posets since they impose less restrictions on the high-level plans compared to shorter posets. Every linear extension of subtasks in a poset produces a simple path connecting v_0 and v_{accept} in $\mathcal{A}^-_{\text{subtask}}$.

Example 1: continued (Temporal constraints) For task (i), there are two simple paths in $\mathcal{A}^-_{\text{subtask}}$ leading to v_6 and all have the same set of four edges, thus, $\Theta_1 = \{v_{\text{init}}, v_1, v_4, v_3, v_6; v_{\text{init}}, v_1, v_2, v_3, v_6\}$; see Fig 7(a). The design of equivalent subtasks, mapping function, integer sequence and the poset are shown in Fig. 7(b). The temporal relation implies that subtasks (v_1, v_4) and (v_1, v_2) are independent, which agrees with our observation. For task (ii), the NBA $\mathcal{A}^-_{\text{subtask}}$ in Fig. 5(b) only has one path of two subtasks that generates a totally ordered set where every two subtasks are comparable.

Remark 4.4: If the size of sub-NBA $\mathcal{A}^-_{\text{subtask}}$ is still large, leading to large number of simple paths, we can select a fixed number of simple paths, similar to finding a fixed number of runs in [28]. This will not severely compromise the diversity of the selected simple paths since a lot of simple paths are combinations of the same set of elementary subtasks.

V. Design of High-Level Task Allocation Plans and Low-Level Executable Paths

In this section, we synthesize plans that satisfy the LTL specification ϕ by first generating a time-stamped task allocation plan that respects the temporal order between subtasks that need to be satisfied in order to satisfy the specification, and then obtaining a low-level executable path that also satisfies the negative literals that we removed from $\mathcal{A}_{\text{relax}}$ in Section IV-A. In what follows, we discuss the synthesis of a prefix path; a similar process is used to synthesize the suffix path in Appendix A-C in [40]. Specifically, to synthesize high-level prefix plans, we iterate over the sorted set of posets $\{P_{\text{pre}}\}$, where P_{pre} is a poset corresponding to a simple prefix path in $\mathcal{A}^{-}_{\text{subtask}}$ and, for every poset in $\{P_{\text{pre}}\}$ we formulate a MILP to assign robots to tasks and determine a high-level plan, i.e., a sequence of time-stamped waypoints, that the robots need to visit to satisfy the subtasks in the corresponding simple path in $\mathcal{A}^{-}_{\text{subtask}}$. Note that, given a poset $P \in \{P_{\text{pre}}\}$, every element in the corresponding base set X_P is an integer associated with an edge/subtask in the NBA $\mathcal{A}^{-}_{\text{subtask}}$. Since a solution to the proposed MILP is effectively a linear extension of the poset P, the corresponding plan sequentially satisfies the vertex and edge labels of all subtasks in $\mathcal{A}^{-}_{\text{subtask}}$ associated with the elements in X_P. Therefore, this plan produces a simple path in $\mathcal{A}^{-}_{\text{subtask}}$ that connects v_0 and v_{accept}. To obtain the low-level executable path, for every subtask in this simple path, we formulate a generalized multi-path robot planning problem that considers the negative literals that were removed from $\mathcal{A}_{\text{relax}}$ in Section IV-A.

The proposed MILP is inspired by the vehicle routing problem (VRP) with temporal constraints [12]. In the VRP, a fleet of vehicles traverses a given set of customers such that all vehicles depart from and return to the same depot, and each customer is visited by exactly one vehicle. Compared to the VRP with temporal constraints [12], the LTL-MRTA problem is significantly more complicated. First, robots are not required to return to their initial locations. Instead, there may exist robots that need to execute the task forever corresponding to the "always" LTL operator. Second, there may exist labeled regions that do not need to be visited at all and others that need to be visited exactly once, more than once, or infinitely many times. Finally, visits of regions and visiting times are subject to logical constraints induced by the NBA $\mathcal{A}^{-}_{\text{subtask}}$.

A. Construction of the prefix routing graph

To formulate the proposed MILP, we first construct a routing graph $\mathcal{G} = (\mathcal{V}_{\mathcal{G}}, \mathcal{E}_{\mathcal{G}})$, where each vertex represents an initial robot location or a specific region that is associated with a specific literal in a specific label of a subtask in X_P. Each vertex/region in $\mathcal{V}_{\mathcal{G}}$ is visited by at most one robot. Simultaneous visits of multiple vertices in $\mathcal{V}_{\mathcal{G}}$ by a fleet of robots satisfy a literal, a clause, or a label in $\mathcal{A}^{-}_{\text{subtask}}$. The time of visits reflects the temporal relation among subtasks. We first construct the vertex set and then the edge set of the routing graph $\mathcal{G}$. Both constructions consist of four layers that iterate over the edges, then the labels, then the clauses, and finally, the literals in $\mathcal{A}^{-}_{\text{subtask}}$; see also Alg. 2.

Algorithm 2: Construct the routing graph

Input: Poset P

; ▷ Create the vertex set

1 Create the vertex set $\mathcal{V}_{\text{init}}$ for initial locations ;

; ▷ vertices for labels

2 **for** $e = (v_1, v_2) \in X_P$ **do**

3 **if** $\gamma(v_1, v_2) \neq \top$ **then**

4 **for** $\mathcal{C}_p^{\gamma} \in \mathsf{cls}(\gamma)$ **do**

5 **for** $\pi_{i,j}^{k,\chi} \in \mathsf{lits}^{+}(\mathcal{C}_p^{\gamma})$ **do**

6 Create i vertices ;

7 **if** $\gamma(v_1) \neq \top, \bot$ **then**

8 Create vertices by following lines 4-6 ;

; ▷ Create the edge set

9 **for** $e = (v_1, v_2) \in X_P$ **do**

10 **if** $\gamma(v_1, v_2) \neq \top$ **then**

11 **for** $\mathcal{C}_p^{\gamma} \in \mathsf{cls}(\gamma)$ **do**

12 **for** $\pi_{i,j}^{k,\chi} \in \mathsf{lits}^{+}(\mathcal{C}_p^{\gamma})$ **do**

13 (i) Vertices of initial robot locations ;

14 (ii) Vertices of prior subtasks ;

15 (iii) Vertices associated with $\gamma(v_1)$;

16 **if** $\gamma(v_1) \neq \top, \bot$ **then**

17 **if** $S_2^e = \emptyset$ **then**

18 Create edges by following lines 11-13 ;

19 **else if** $S_2^e \neq \emptyset$ **then**

20 Create edges from vertices associated with subtasks in S_2^e ;

21 **if** $X^e_{\prec_P} = \emptyset$ **and** $X^e_{\parallel_P} \neq \emptyset$ **then**

22 Create edges from vertices associated with initial robot locations ;

1) Construction of the vertex set: The vertex set $\mathcal{V}_{\mathcal{G}}$ consists of three types of vertices, namely, location vertices related to initial robot locations, literal vertices related to edge labels in the sub-NBA $\mathcal{A}^{-}_{\text{subtask}}$, and literal vertices related to vertex labels in the sub-NBA $\mathcal{A}^{-}_{\text{subtask}}$. Specifically, we construct the location vertices as follows.

a) Location vertices associated with initial robot locations: First we create n vertices, collected in the set $\mathcal{V}_{\text{init}} \subseteq \mathcal{V}_{\mathcal{G}}$ such that each vertex points to the initial location $s^0_{r,j}$ of robot $[r, j] \in \mathcal{K}_j, \forall j \in [m]$ [line 1, Alg. 2] (see blue dots in Fig. 8).

To obtain the set of literal vertices in $\mathcal{V}_{\mathcal{G}}$, we iterate over subtasks in X_P. Given a subtask $e = (v_1, v_2) \in X_P$, we construct vertices for the edge label $\gamma(v_1, v_2)$ and the starting vertex label $\gamma(v_1)$, if they are neither $\top$ nor $\bot$. Specifically, we take the following steps.

b) Literal vertices associated with edge labels: If $\gamma(v_1, v_2) \neq \top$, we operate on $\gamma(v_1, v_2) = \bigvee_{p \in \mathcal{P}} \bigwedge_{q \in \mathcal{Q}_p} \pi_{i,j}^{k,\chi}$ starting by iterating over the clauses $\mathcal{C}_p^{\gamma} \in \mathsf{cls}(\gamma)$ in the label, and then over the literals in each clause $\mathcal{C}_p^{\gamma}$ [lines 2-6, Alg. 2]. The literal $\pi_{i,j}^{k,\chi} \in \mathsf{lits}^{+}(\mathcal{C}_p^{\gamma})$ implies that at least $\langle i, j \rangle$, i.e., i robots of type j, should visit the target region ℓ_k simultaneously. Hence, we create i vertices in $\mathcal{V}_{\mathcal{G}}$ all associated with region ℓ_k. If $\langle i, j \rangle$ visit these i vertices simultaneously, one robot per vertex, then $\pi_{i,j}^{k,\chi}$ is true. Note that if $\chi \neq 0$, the robots visiting these i vertices should be the same as those visiting another i vertices associated with another literal with the same nonzero connector, which is ensured by the MILP formulation; see the red, yellow, and green dots in Fig. 8.

c) Literal vertices associated with starting vertex labels: After vertices in $\mathcal{V}_{\mathcal{G}}$ associated with the edge label $\gamma(v_1, v_2)$ of subtask e have been constructed, vertices in $\mathcal{V}_{\mathcal{G}}$ associated with the starting vertex label $\gamma(v_1)$ can be constructed in the same manner if $\gamma(v_1)$ is neither $\top$ nor $\bot$ [lines 7-8, Alg. 2].

Repeating steps b) and c) for all subtasks in X_P completes the construction of the vertex set $\mathcal{V}_{\mathcal{G}}$. Note that each vertex in $\mathcal{V}_{\mathcal{G}} \setminus \mathcal{V}_{\text{init}}$ is associated with a literal of a certain subtask in X_P. Also, each literal of a certain subtask in X_P is associated

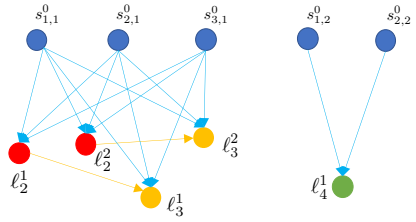


Fig. 8. Routing graph $\mathcal{G}$ for task (i). $s^0_{1,1}$, $s^0_{2,1}$ and $s^0_{3,1}$ are initial locations of three robots of type 1 and $s^0_{1,2}$ and $s^0_{2,2}$ are initial locations of two robots of type 2 (see case a)). Red dots ℓ^1_2 and ℓ^2_2 correspond to the edge label $\pi^{2,1}_{2,1}$ of element 3, i.e., edge (v_1, v_2) in X_P; see Fig. 7(b). Yellow dots ℓ^1_3, ℓ^2_3 correspond to the edge label $\pi^{3,1}_{2,1}$ of element 4, and green dot ℓ^1_4 corresponds to the edge label $\pi^4_{1,2}$ of element 2 (see case b)). No dots correspond to vertex labels since all vertex labels are either $\top$ or $\bot$. The edges from ℓ^1_2 to ℓ^1_3 and from ℓ^2_2 to ℓ^2_3 are due to $3 <_P 4$.

with one or more vertices in $\mathcal{V}_\mathcal{G} \setminus \mathcal{V}_{\text{init}}$, and the literal specifies the region and the robot type associated with these vertices. To capture this correspondence, let $\mathcal{M}^\mathcal{V}_e : \mathcal{V}_\mathcal{G} \setminus \mathcal{V}_{\text{init}} \to X_P$ and $\mathcal{M}^\mathcal{V}_{\mathsf{lits}} : \mathcal{V}_\mathcal{G} \setminus \mathcal{V}_{\text{init}} \to \prod_{\mathsf{lits}}$ map a vertex in $\mathcal{V}_\mathcal{G} \setminus \mathcal{V}_{\text{init}}$ to its associated subtask and literal, respectively, where $\prod_{\mathsf{lits}}$ is the cartesian product $X_P \times \{0,1\} \times \mathcal{P} \times \mathcal{Q}_p$, and 0, 1 represent the label type, 0 for vertex label and 1 for edge label. Furthermore, let $\mathcal{M}^{\mathsf{lits}}_\mathcal{V} : \prod_{\mathsf{lits}} \to 2^{\mathcal{V}_\mathcal{G}}$ and $\mathcal{M}^{\mathsf{cls}}_\mathcal{V} : \prod_{\mathsf{cls}} \to 2^{\mathcal{V}_\mathcal{G}}$ map a literal and clause to the associated vertices in $\mathcal{G}$, respectively, where $\prod_{\mathsf{cls}}$ is the cartesian product $X_P \times \{0,1\} \times \mathcal{P}$. We also define $\mathcal{M}^\mathcal{V}_\mathcal{L} : \mathcal{V}_\mathcal{G} \to \mathcal{L}$ and $\mathcal{M}^\mathcal{V}_\mathcal{K} : \mathcal{V}_\mathcal{G} \to \{\mathcal{K}_j\}$ that map a vertex in $\mathcal{V}_\mathcal{G}$ to its associated region and robot type. Finally, if $\chi \neq 0$, we define $\mathcal{M}^\chi_\gamma : \mathbb{N}^+ \to 2^{X_P \times \{0,1\}}$ to map χ to all labels in X_P that have literals with the same connector χ.

Example 1: continued (Mappings for task (i)) The mappings in Fig. 8 associated with the vertex ℓ^1_2 are: $\mathcal{M}^\mathcal{V}_e(\ell^1_2) = (v_1, v_2) = 3$ and $\mathcal{M}^\mathcal{V}_{\mathsf{lits}}(\ell^1_2) = ((v_1, v_2), 1, 1, 1)$ since the vertex ℓ^1_2 corresponds to the first literal $\pi^{2,1}_{2,1}$ of the first clause of the edge label of subtask (v_1, v_2) in X_P; see also Fig. 7. $\mathcal{M}^\mathcal{V}_\mathcal{L}(\ell^1_2) = \ell_2$ and $\mathcal{M}^\mathcal{V}_\mathcal{K}(\ell^1_2) = \mathcal{K}_1$ since the literal $\pi^{2,1}_{2,1}$ requires two robots of type 1 to visit region ℓ_2.

Furthermore, the literal/clause-to-vertex mappings are: $\mathcal{M}^{\mathsf{lits}}_\mathcal{V}(((v_1, v_2), 1, 1, 1)) = \mathcal{M}^{\mathsf{cls}}_\mathcal{V}(((v_1, v_2), 1, 1)) = \{\ell^1_2, \ell^2_2\}$; $\mathcal{M}^{\mathsf{lits}}_\mathcal{V}(((v_1, v_4), 1, 1, 1)) = \mathcal{M}^{\mathsf{cls}}_\mathcal{V}(((v_1, v_4), 1, 1)) = \{\ell^1_4\}$ since the literal $\pi^4_{1,2}$, the first literal of the first clause of the edge label of subtask (v_1, v_4), requires one robot to visit region ℓ_4. Finally, the connector-to-label mapping is: $\mathcal{M}^\chi_\gamma(1) = \{((v_1, v_2), 1), ((v_3, v_6), 1)\}$ since the connector 1 appears in the edge label of subtasks (v_1, v_2) and (v_3, v_6).

2) Construction of the edge set: The edges in $\mathcal{G}$ respect the partial order among subtasks captured by the poset P. We construct the edge set $\mathcal{E}_\mathcal{G}$ by following a similar procedure as that used to construct the vertex set $\mathcal{V}_\mathcal{G}$. Specifically, we iterate over the elements in X_P. For every subtask $e = (v_1, v_2) \in X_P$, if $\gamma(v_1, v_2) \neq \top$, we first operate on the edge label $\gamma(v_1, v_2) = \bigvee_{p \in \mathcal{P}} \bigwedge_{q \in \mathcal{Q}_p} \pi^{k,\chi}_{i,j}$ starting by iterating over the clauses $\mathcal{C}^\gamma_p \in \mathsf{cls}(\gamma)$, and then over the literals in each clause $\mathcal{C}^\gamma_p$ [lines 9-15, Alg. 2]. Specifically, recall from Section V-A1 that the literal $\pi^{k,\chi}_{i,j} \in \mathsf{lits}^+(\mathcal{C}^\gamma_p)$ corresponds to i vertices in $\mathcal{V}_\mathcal{G}$ that are associated with region ℓ_k that should be visited by i robots. In what follows, we identify three types of *leaving vertices* in $\mathcal{V}_\mathcal{G}$ from where i robots can depart to reach these i vertices that satisfy literal $\pi^{k,\chi}_{i,j}$.

a) Location vertices: The location vertices in $\mathcal{V}_{\text{init}}$ associated with robots of type j are leaving vertices. We add an edge from all initial vertices to every vertex associated with literal $\pi^{k,\chi}_{i,j}$ (blue edges in Fig. 8). Intuitively, robots depart from initial locations to undertake certain subtasks. These edges are associated with a weight T^* equal to the shortest travel time from the initial location to ℓ_k and another weight d equal to the smallest traveling cost between the initial location and ℓ_k.

b) Leaving vertices associated with prior subtasks: Let $X^e_{<_P}$, $X^e_{\prec_P}$ and $X^e_{\|_P}$ denote the sets that collect subtasks in X_P that are smaller than, covered by, and incomparable to subtask e, respectively (see Section II-B). In words, $X^e_{<_P}$ contains subtasks in X_P that should be completed prior to e, $X^e_{\prec_P} \subseteq X^e_{<_P}$ contains subtasks in $X^e_{<_P}$ that can be completed right before e, and $X^e_{\|_P}$ contains subtasks independent from e. To find leaving vertices, we iterate over $S^e_1 = X^e_{<_P} \cup X^e_{\|_P}$ that includes all subtasks that can be completed prior to e, respecting the partial order between subtasks. Given a subtask $e' = (v'_1, v'_2) \in S^e_1$, if its edge label $\gamma'(v'_1, v'_2) \neq \top$, we iterate over all clauses in γ' and then over all literals in each clause. Specially, given a clause $\mathcal{C}^{\gamma'}_{p'} \in \mathsf{cls}(\gamma')$, for any literal $\pi^{k',\chi'}_{i',j'} \in \mathsf{lits}^+(\mathcal{C}^{\gamma'}_{p'})$, if $j' = j$, then literal vertices in $\mathcal{V}_\mathcal{G}$ associated with this literal are leaving vertices. If further $i' = i$, we randomly create i one-to-one edges starting from these i vertices and ending at the i vertices associated with $\pi^{k,\chi}_{i,j}$ (see the orange edges in Fig. 8). Because there are exactly i robots of type j, it suffices to build i one-to-one edges. Furthermore, if $\chi = \chi' \neq 0$, then literals $\pi^{k,\chi}_{i,j}$ and $\pi^{k',\chi'}_{i',j'}$ must have the same number of vertices. Building i one-to-one edges can guarantee that the same i robots of type j satisfy these two literals. Otherwise, if $i' \neq i$, we add $i \times i'$ edges to $\mathcal{E}_\mathcal{G}$ by creating an edge from any vertex associated with $\pi^{k',\chi'}_{i',j'}$ to any vertex of $\pi^{k,\chi}_{i,j}$. Finally, since each region may span multiple cells, the weights T^* and d of these edges are set as the shortest travel time and lowest traveling cost from $\ell_{k'}$ to ℓ_k. After creating edges associated with the edge label $\gamma'(v'_1, v'_2)$ of e', we identify leaving vertices among literal vertices in $\mathcal{V}_\mathcal{G}$ associated with the starting vertex label $\gamma(v'_1)$ of e' and build edges in the same manner.

c) Leaving vertices associated with $\gamma(v_1)$ of e: When the iteration over S^e_1 is completed, we identify leaving vertices among literal vertices associated with the starting vertex label $\gamma(v_1)$ of the current subtask e by following the procedure in case b) for the prior subtasks. This is because $\gamma(v_1)$ becomes true before $\gamma(v_1, v_2)$.

So far we have constructed three types of leaving vertices corresponding to the literal $\pi^{k,\chi}_{i,j}$ in $\mathsf{lits}^+(\mathcal{C}^\gamma_p)$ of the edge label $\gamma(v_1, v_2)$ [lines 13-15, Alg. 2]. We continue constructing leaving vertices for all other literals in $\mathsf{lits}^+(\mathcal{C}^\gamma_p)$ [line 12, Alg. 2] and clauses in $\mathsf{cls}(\gamma)$ [line 11, Alg. 2]. After constructing all edges pointing to vertices associated with literals in the edge label $\gamma(v_1, v_2)$ of the current subtask e [line 10, Alg. 2], we construct edges pointing to vertices associated with literals in the starting vertex label $\gamma(v_1)$, by identifying leaving vertices among location vertices and literal vertices associated with prior subtasks. Specifically, let $S^e_2 = X^e_{\prec_P} \cup X^e_{\|_P}$ be the set

that collects all subtasks that can occur immediately prior to subtask e. The satisfaction of edge labels of subtasks in S_2^e can directly lead to the starting vertex v_1 of e. We consider the following cases.

(1) $S_2^e = \emptyset$: In this case, no subtask can be completed before subtask e, i.e., the subtask e should be the first one among all in X_P to be completed. Thus, v_1 is identical to the initial vertex v_0. In this case, we only identify location vertices as leaving vertices, as in case a) [lines 18, Alg. 2].

(2) $S_2^e \neq \emptyset$: We identify leaving vertices associated with prior subtasks in S_2^e. Given a subtask $e' = (v_1', v_2') \in S_2^e$, we find all clauses $\mathcal{C}_{p'}^{\gamma'} \in \mathsf{cls}(\gamma')$ in the edge label γ' of e' such that, for the considered clause $\mathcal{C}_p^{\gamma} \in \mathsf{cls}(\gamma)$ in the starting vertex label of subtask e, its corresponding clause $(\mathcal{C}_p^{\gamma})_\phi$ in $\mathcal{A}_\phi$ is the subformula of their corresponding clauses $(\mathcal{C}_{p'}^{\gamma'})_\phi$ in $\mathcal{A}_\phi$. Next, for each literal $\pi_{i,j}^{k,\chi} \in \mathsf{lits}^+(\mathcal{C}_p^{\gamma})$ we create i one-to-one edges, starting from those i vertices associated with the counterpart of literal $\pi_{i,j}^{k,\chi}$ in the found clause $\mathcal{C}_{p'}^{\gamma'} \in \mathsf{cls}(\gamma')$ and ending at the i vertices associated with $\pi_{i,j}^{k,\chi}$ [lines 20, Alg. 2]. We create such one-to-one edges based on condition (d) in Definition 3.8 and condition (b) in Definition 3.10. That is, the edge label strongly implies its end vertex label, the satisfied clause in the edge label implies the satisfied clause in the end vertex label, and the fleet of robots satisfying the clause in the vertex label belongs to the fleet of robots satisfying the clause in the incoming edge label. This is also the reason why we consider prior subtasks in S_2^e rather than S_1^e as in case b).

(3) $X_{\prec_P}^e = \emptyset$ and $X_{\|_P}^e \neq \emptyset$: In this case, the subtask e can be the first one among all to be completed. If so, its starting vertex label $\gamma(v_1)$ should be satisfied at the beginning. However, robots cannot depart from leaving vertices that are literal vertices (see case (2)), because these edges are enabled after subtask e. Therefore, for the vertex label $\gamma(v_1)$, we additionally identify leaving vertices pointing to initial robot locations, as in case a) [lines 22, Alg. 2]. Note that, if $X_{\prec_P}^e \neq \emptyset$, there are no leaving vertices associated with initial locations since there exists a subtask that should be completed before e and, therefore, subtask e can not be the first one. When the iteration over all subtasks in X_P is over, we finish the construction of the edge set $\mathcal{E}_\mathcal{G}$ [line 9, Alg. 2].

B. Construction of the robot prefix plans

Given the routing graph constructed in Section V-A, the proposed MILP contains five types of constraints including routing constraints, scheduling constraints, logical constraints, temporal constraints, and transition constraints; see Appendix A-B in [40]. The feasibility of the MILP and the properties of the resulting solutions are analyzed in Lemmas 6.6 and 6.7. Given the solution to the MILP, we first define a time axis that includes the sorted completion times of all subtasks in X_P. This time axis produces a linear extension of the poset P and the plan generated by this linear extension satisfies the vertex and edge labels in a given simple path in $\mathcal{A}_{\text{subtask}}^-$. Next, we extract a time-stamped task allocation plan for each robot that is augmented with the completion time of each subtask, and can be used to generate low-level paths satisfying the specification ϕ.

1) Time axis: The progress made in $\mathcal{A}_{\text{subtask}}^-$ is directly linked to the satisfaction of edge labels which, by condition (d) in Definition 3.8, implies the satisfaction of their end vertex labels, excluding v_{accept}. Therefore, we collect the completion times of all subtasks in X_P (the time when edges are enabled) and sort them in an ascending order to form a single increasing time axis, denoted by $\vec{t}$. We note that there are no identical time instants in the time axis since, by construction, the solution to the MILP produces a simple path in $\mathcal{A}_{\text{subtask}}^-$ and subtasks in any simple path are completed at different times.

2) High-level robot plans: Next we extract a high-level plan for each robot, which is a sequence of waypoints that the robots need to visit to complete the subtasks in X_P along with the time instants of these visits. Specifically, for each robot $[r, j]$, let $p_{r,j}$ denote its corresponding high-level plan and let $t_{r,j}$ denote its timeline. Consider also a vertex $v_0^* \in \mathcal{V}_{\text{init}}$ in the routing graph $\mathcal{G}$ that is associated with the initial location of robot $[r, j]$ and let v_1^* be the vertex that robot r traverses to. Note that robot r can only travel along one outgoing edge of v_0^*. Note also that each vertex in the routing graph $\mathcal{G}$ is associated with a label captured in the mapping $\mathcal{M}_{\mathsf{lits}}^{\mathcal{V}}$. If the label associated with v_1^* is a vertex label, then we proceed to the next vertex v_2^* that robot r reaches from v_1^*, until a vertex $v^* \in \mathcal{V}_\mathcal{G}$ associated with an edge label is found. Then, the region associated with this vertex v^*, captured by the mapping $\mathcal{M}_{\mathcal{L}}^{\mathcal{V}}(v^*)$, constitutes the first waypoint robot r needs to visit to complete a subtask. We add this region $\mathcal{M}_{\mathcal{L}}^{\mathcal{V}}(v^*)$ to the plan $p_{r,j}$. Next, the corresponding visit time indicates the completion time of the associated subtask that is captured by the mapping $\mathcal{M}_e^{\mathcal{V}}(v^*)$. We add this time instance to timeline $t_{r,j}$. Since each time instant on the time axis $\vec{t}$ corresponds to the completion of one subtask, this visit time in $t_{r,j}$ corresponds to the time instant on $\vec{t}$ that the subtask $\mathcal{M}_e^{\mathcal{V}}(v^*)$ is completed. Continuing this process, we can construct for robot $[r, j]$ a sequence of waypoints and the corresponding timeline whose time instants appear on the time axis $\vec{t}$. Given this high-level plan $\{p_{r,j}\}$, we can design low-level executable paths that reconsider the negative literals that were originally removed from the NBA $\mathcal{A}_{\text{relax}}$.

Example 1: continued (Time-stamped task allocation plan) After solving the MILP for the workspace in Fig. 1, the high-level plans and the associated timelines for robots are as follows: $p_{2,1} = p_{3,1} = \{\ell_2, \ell_3\}, t_{2,1} = t_{3,1} = \{6, 16\}, p_{2,2} = \{\ell_4\}$, $t_{2,2} = \{10\}$. That is, robots $[2, 1]$ and $[3, 1]$ visit the office building ℓ_2 at time instant 6, then robot $[2, 2]$ visits the control room ℓ_4 at time instant 10, and finally robots $[2, 1]$ and $[3, 1]$ visit the delivery site ℓ_3 at time instant 16. The remaining robots remain idle. The induced simple path in $\mathcal{A}_{\text{subtask}}^-$ in Fig. 7(a) is $v_{\text{init}}, v_1, v_2, v_3, v_6$. The associated time axis is $\vec{t} = \{0, 6, 10, 16\}$, one time instant per subtask. In words, the subtask (v_{init}, v_1) is completed at time instant 0 and the subtask (v_1, v_2) is completed at time instant 6, which corresponds to the event that robots $[2, 1]$ and $[3, 1]$ visit the office building ℓ_2.

C. Design of low-level prefix paths

In this section we discuss the correction stage that reintroduces the negative literals to the NBA and corrects the

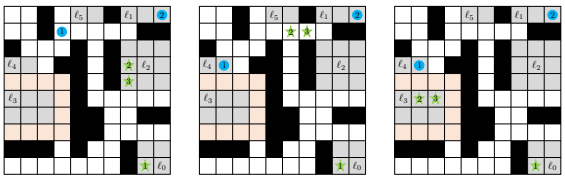
high-level plans designed in Section V-B (if needed) so that they satisfy the specification ϕ. To this end, we first find the simple path in the NBA $\mathcal{A}^-_{\text{subtask}}$ connecting v_0 and v_{accept} using the time axis and the time-stamped task allocation plan. To satisfy the specification ϕ, for every subtask in the simple path, we formulate a generalized multi-robot path planning (GMRPP) problem. Each GMRPP is essentially a generalization of the multi-robot point-to-point navigation problem, whose goal is to determine a collection of executable paths that allow the robots to complete the current subtask (by enabling the edge label at the end while respecting the starting vertex en route) and automatically activate the next subtask, since the satisfaction of the edge label leads to the satisfaction of the starting vertex of the next subtask. The details can be found in Appendix B in [40], that also discusses different implementations of the proposed GMRPP that depend on whether all or a subset of robots are allowed to move during the execution of the current subtask, since not all robots are responsible for the completion of this subtask, and whether the completion times of subtasks are disjoint or partially overlapping. Finally, the feasibility of the proposed GMRPP is analyzed Lemma C.11 in [40].

D. Obtaining the best prefix-suffix path

After obtaining the prefix path corresponding to a poset $P \in \{P_{\text{pre}}\}$ for the given pair v_0 and v_{accept}, next we find the suffix path around v_{accept}. For this, we can follow a similar process as this described in Sections V-A$\sim$V-C to find the prefix path for poset $P \in \{P_{\text{pre}}\}$, with the difference that now we treat the accepting vertex v_{accept} as both the initial vertex v_0 and the accepting vertex v_{accept}. This is because the suffix path is essentially a loop, i.e., the final locations in the suffix path are identical to the initial locations in the suffix path, which are also the final locations in the prefix path.

Specifically, given the pair v_0 and v_{accept}, we solve one MILP for each poset $P' \in \{P_{\text{suf}}\}$ to obtain a corresponding suffix path; these MILPs can be infeasible if there are no feasible paths that induce simple paths corresponding to the poset P'. Then, among all suffix paths for all posets $P' \in \{P_{\text{suf}}\}$ we select the one with the lowest cost. This best suffix path corresponds to the prefix path generated from a poset $P \in \{P_{\text{pre}}\}$ for the given pair v_0 and v_{accept}. Combining this suffix path with the corresponding prefix path we obtain the best total path associated with the poset P for the given pair v_0 and v_{accept}. Then, using cost function (2), we select the best total path over all posets in $\{P_{\text{pre}}\}$ for the given pair v_0 and v_{accept}. Finally, by iterating over all pairs of initial and accepting vertices with finite total length, we can obtain the best total path. We highlight that our method can terminate anytime once a feasible path is found, but running the algorithm longer can lead to more optimal feasible paths. Note also that by iterating over the pairs v_0 and v_{accept} and the corresponding posets in the ascending order discussed in Section IV-C, it is more likely that the first solutions have low cost since they involve fewer subtasks that need to be accomplished, which is validated numerically in Section VII.

Example 1: continued (Low-level paths) When generating paths for task (i), collision avoidance is considered. Fig. 9



(a) $t = 6$ (b) $t = 10$ (c) $t = 18$

Fig. 9. Key frames demonstrating the execution of low-level paths that satisfy task (i). The initial configuration is shown in Fig. 1. Fig. 9(a) shows that at time instant 6, robots $[2, 1]$ and $[3, 1]$ reach the office building ℓ_2, while robot $[1, 2]$ is on the way to the control room ℓ_4. Fig. 9(b) shows at time instant 10, robot $[1, 2]$ reaches the control room ℓ_4 while robots $[2, 1]$ and $[3, 1]$ head towards the delivery site ℓ_3. Finally, they reach ℓ_3 in Fig. 9(c) at time instant 18. Robots $[1, 1]$ and $[2, 2]$ remain idle throughout the process.

shows an array of three key frames where different subtasks are completed. Observe that task (i) is completed at time 18, longer than 16 given by the time-stamped task allocation plan since the high-level plan uses the shortest travel time between regions and does not consider collision avoidance.

VI. Theoretical Analysis

In this section, we analyze the completeness and soundness of our method. First we present important lemmas and then show that, with mild assumptions, our method is complete for LTL^0 specifications. We start by providing necessary notation.

A. Notation

Given a NBA $\mathcal{A}$, e.g., $\mathcal{A}_\phi, \mathcal{A}_{\text{relax}}$ and $\mathcal{A}_{\text{subtask}}$, we define by $\mathcal{L}_E(\mathcal{A})$ the set of words in $\mathcal{L}(\mathcal{A})$ that can be realized by robot paths. Recall that we can always map a run in $\mathcal{A}$ to its counterpart in $\mathcal{A}_\phi$. If $\mathcal{A} = \mathcal{A}_\phi$, then the counterpart of a run is the run itself. We define by $\mathcal{L}^\phi_E(\mathcal{A})$ the set of words in $\mathcal{L}_E(\mathcal{A})$ such that for any word $w \in \mathcal{L}^\phi_E(\mathcal{A})$ that induces an accepting run in $\mathcal{A}$, the counterpart of this accepting run in $\mathcal{A}_\phi$ is a restricted accepting run. In words, if $\mathcal{A} = \mathcal{A}^-_{\text{subtask}}$, and if a path τ generates a word w in $\mathcal{L}^\phi_E(\mathcal{A}^-_{\text{subtask}}) \subseteq \mathcal{L}_E(\mathcal{A}^-_{\text{subtask}})$ and w induces a run ρ in $\mathcal{A}^-_{\text{subtask}}$ connecting a pair v_0 and v_{accept}, then, we can obtain the corresponding run ρ_ϕ in $\mathcal{A}_\phi$ that is the counterpart of the run ρ in $\mathcal{A}^-_{\text{subtask}}$. This motivates us to modify the path τ that satisfies $\mathcal{A}^-_{\text{subtask}}$ to get another path that can produce this run ρ_ϕ in $\mathcal{A}_\phi$. Additionally, let $\tilde{\mathcal{L}}^\phi_E(\mathcal{A}) \subseteq \mathcal{L}^\phi_E(\mathcal{A})$ collect those words in $\mathcal{L}^\phi_E(\mathcal{A})$ that can be generated by paths that satisfy Assumption 3.11. Next, we consider the prefix and suffix parts separately. Given a pair of initial and accepting vertices, v_0 and v_{accept}, let $\mathcal{L}^{\phi, v_0 \to v_{\text{accept}}}_E(\mathcal{A})$ be the set that collects finite realizable words that can generate a run in $\mathcal{A}$ connecting v_0 and v_{accept}, and further the corresponding run in $\mathcal{A}_\phi$ satisfies the requirements on the prefix part of a restricted accepting run (see conditions (a)-(d) in Definition 3.8).

B. Completeness and soundness

The following proposition states that paths exist that can induce accepting runs in the pruned sub-NBA $\mathcal{A}^-_{\text{subtask}}$. This result will be used to show the feasibility of the MILP for the time-stamped task allocation plan.

Proposition 6.1: (Feasible prefix paths for the sub-NBA $\mathcal{A}^-_{\text{subtask}}$) Given a workspace satisfying Assumption 3.5 and a valid specification $\phi \in$ *LTL*$^\chi$, if there exists a path $\tau = \tau^{\text{pre}}[\tau^{\text{suf}}]^\omega$ inducing a restricted accepting run $\rho = \rho^{\text{pre}}[\rho^{\text{suf}}]^\omega = v_0, \ldots, v_{\text{prior}}, v_{\text{accept}}[v_{\text{next}}, \ldots, v'_{\text{prior}}, v_{\text{accept}}]^\omega$ in

$\mathcal{A}_\phi$ and this path satisfies Assumption 3.11, then there exists another path $\overline{\tau} = \overline{\tau}^{\text{pre}}[\overline{\tau}^{\text{suf}}]^\omega$ such that $\overline{\tau}^{\text{pre}}$ generates a word in $\tilde{\mathcal{L}}_E^{\phi,v_0\to v_{\text{accept}}}(\mathcal{A}^-_{\text{subtask}}) \neq \emptyset$.

To prove Proposition 6.1, we recall the main steps to obtain the sub-NBA $\mathcal{A}^-_{\text{subtask}}$ in Sections IV-A and IV-B and characterize the relations between the different NBAs; see Lemmas 6.2-6.5. The first lemma shows that the pruning steps in Section IV-A do not affect the set of restricted accepting runs in $\mathcal{A}_\phi$ that can be induced by realizable words.

Lemma 6.2: ($\mathcal{A}_\phi$ and $\mathcal{A}^-_\phi$) The pruning steps in Section IV-A satisfy $\mathcal{L}_E^\phi(\mathcal{A}^-_\phi) = \mathcal{L}_E^\phi(\mathcal{A}_\phi)$.

Note that any word in $\mathcal{L}_E^\phi(\mathcal{A}_\phi)$ induces a restricted accepting run in $\mathcal{A}_\phi$. A direct consequence of Lemma 6.2 is that, any path generating a word $w \in \mathcal{L}_E^\phi(\mathcal{A}^-_\phi)$ satisfies ϕ since the word w also belongs to $\mathcal{L}_E^\phi(\mathcal{A}_\phi)$. The following lemma shows that ignoring negative literals expands the set of realizable words accepted by $\mathcal{A}_{\text{relax}}$ compared to that of $\mathcal{A}^-_\phi$.

Lemma 6.3: ($\mathcal{A}^-_\phi$ and $\mathcal{A}_{\text{relax}}$) The relaxation stage that replaces all negative literals with $\top$ in Section IV-A, satisfies $\mathcal{L}_E^\phi(\mathcal{A}^-_\phi) \subseteq \mathcal{L}_E^\phi(\mathcal{A}_{\text{relax}})$.

Lemma 6.3 implies that a word in $\mathcal{L}_E^\phi(\mathcal{A}_{\text{relax}})$ may not belong to $\mathcal{L}_E^\phi(\mathcal{A}^-_\phi)$. Hence, a path generating a word in $\mathcal{L}_E^\phi(\mathcal{A}_{\text{relax}})$ may not satisfy the specification ϕ since $\mathcal{A}_{\text{relax}}$ ignores the negative literals. The following two lemmas show that extraction and pruning of the sub-NBA $\mathcal{A}_{\text{subtask}}$ for the prefix part do not empty the subset of words in $\mathcal{L}_E^{\phi,v_0\to v_{\text{accept}}}(\mathcal{A}^-_{\text{subtask}})$ that can be generated by feasible paths satisfying Assumption 3.11.

Lemma 6.4: ($\mathcal{A}_{\text{relax}}$ and $\mathcal{A}_{\text{subtask}}$) The extraction of the sub-NBA $\mathcal{A}_{\text{subtask}}$ in Section IV-B2 satisfies $\mathcal{L}_E^{\phi,v_0\to v_{\text{accept}}}(\mathcal{A}_{\text{relax}}) = \mathcal{L}_E^{\phi,v_0\to v_{\text{accept}}}(\mathcal{A}_{\text{subtask}})$.

Lemma 6.5: ($\mathcal{A}_{\text{subtask}}$ and $\mathcal{A}^-_{\text{subtask}}$) The pruning steps in Section IV-B3 satisfy $\mathcal{L}_E^{\phi,v_0\to v_{\text{accept}}}(\mathcal{A}^-_{\text{subtask}}) \subseteq \mathcal{L}_E^{\phi,v_0\to v_{\text{accept}}}(\mathcal{A}_{\text{subtask}})$. Additionally, if there exists a path $\tau = \tau^{\text{pre}}[\tau^{\text{suf}}]^\omega$ inducing a restricted accepting run in $\mathcal{A}_\phi$ and this path satisfies Assumption 3.11, then there exists a path $\overline{\tau}^{\text{pre}}$, modified from τ^{pre}, that can generate a word in $\tilde{\mathcal{L}}_E^{\phi,v_0\to v_{\text{accept}}}(\mathcal{A}^-_{\text{subtask}})$, i.e., $\tilde{\mathcal{L}}_E^{\phi,v_0\to v_{\text{accept}}}(\mathcal{A}^-_{\text{subtask}}) \neq \emptyset$.

Finally, Lemma 6.6 states that, if the poset P is inferred from a set of simple paths that includes a simple path associated with a feasible prefix path, then the MILP associated with this poset P is feasible; Lemma 6.7 states that a simple path in $\mathcal{A}^-_{\text{subtask}}$ can be extracted from the solution to the MILP and discusses the temporal properties associated with this path.

Lemma 6.6: (Feasibility of the prefix MILP) If there exists a path $\overline{\tau}^{\text{pre}}$ generating a finite word $\overline{w}^{\text{pre}} \in \tilde{\mathcal{L}}_E^{\phi,v_0\to v_{\text{accept}}}(\mathcal{A}^-_{\text{subtask}})$, and the word $\overline{w}^{\text{pre}}$ induces a simple path $\overline{\theta}^{\text{pre}}$ in $\mathcal{A}^-_{\text{subtask}}$ that belongs to the set of simple paths that generate the poset P, then the prefix MILP associated with this poset P is feasible.

Lemma 6.7: (Properties of the simple path) If the MILP for the prefix path associated with the poset P produces a solution, then a simple path $\tilde{\theta}$ belonging to the set of simple paths that generate the poset P, can be extracted from the sub-NBA $\mathcal{A}^-_{\text{subtask}}$. Additionally, the following properties hold:

(a) The first subtask in the simple path $\tilde{\theta}$ begins at time 0;

(b) For any subtask $e \in \tilde{\theta}$, if its starting vertex has a self-loop, then the completion time of the subtask e is no earlier than the activation of its starting vertex label, and at most one time step after the completion of its starting vertex label;

(c) For any two consecutive subtasks $e, e' \in \tilde{\theta}$, the latter subtask e' is activated at most one time step after the completion of the former subtask e.

Property (a) guarantees the initialization of the sequence of subtasks in $\tilde{\theta}$, property (b) ensures that each subtask in $\tilde{\theta}$ is correctly executed, and property (c) prevents gaps when transitioning between consecutive subtasks. Combined these three properties establish that once the first subtask is activated at time 0, each subsequent subtask is completed successfully and inter-subtasks transitions occur seamlessly, until the completion of the last subtask.

Theorem 6.8: (Completeness) Consider a discrete workspace satisfying Assumption 3.5, a team of n robots of m types and a valid specification $\phi \in$ *LTL*^{0}. Assume also that there exists a path $\tau = \tau^{\text{pre}}[\tau^{\text{suf}}]^\omega$ that induces a restricted accepting run $\rho = \rho^{\text{pre}}[\rho^{\text{suf}}]^\omega = v_0, \ldots, v_{\text{prior}}, v_{\text{accept}}$ $[v_{\text{next}}, \ldots, v'_{\text{prior}}, v_{\text{accept}}]^\omega$ in the pre-processed NBA $\mathcal{A}_\phi$ and satisfies Assumption 3.11. Then, our method can find a robot path $\tilde{\tau} = \tilde{\tau}^{\text{pre}}[\tilde{\tau}^{\text{suf}}]^\omega$ that satisfies the specification ϕ.

The key idea in the proof of Theorem 6.8 is to first show that, feasible paths still exist in $\mathcal{A}^-_{\text{subtask}}$ (see Proposition 6.1) and then use this fact to show feasibility of the MILP and GMRPP problems. The detailed proof can be found in Appendix C in [40]. We emphasize that the completeness result in Theorem 6.8 is ensured for LTL^0 rather than LTL$^\chi$ formulas. This is because task allocations captured by induced atomic propositions in the prefix part may not lead to feasible allocations in the suffix part. However, when the LTL$^\chi$ specification can be satisfied by finite-length paths, such as co-safe LTL [46] or LTL$_f$ [47], then our method is complete also for LTL$^\chi$ specifications.

Remark 6.9: The path $\tilde{\tau}$ constructed by our approach may not satisfy Assumption 3.11 that requires that robots close their suffix loops at the same time the NBA $\mathcal{A}_\phi$ transitions to v_{accept}. However, in our method, when the NBA $\mathcal{A}_\phi$ transitions to v_{accept}, only those robots involved in the completion of the last subtask in the prefix part return to regions corresponding to their initial locations. Thereafter, trajectories are closed.

The soundness follows directly Theorem 6.8.

Corollary 6.10: (Soundness) Consider a discrete workspace, a team of n robots of m types and a valid specification $\phi \in$ *LTL*$^\chi$. Then, the path returned by the GMRPP satisfies the specification ϕ. Also, the specific implementation of the GMRPP is not important.

VII. Numerical Experiments

In this section we present three case studies, implemented in Python 3.6.3 on a computer with 2.3 GHz Intel Core i5 and 8G RAM, that illustrate the correctness and scalability of our method. The MILP is solved using Gurobi [48] with big-M $M_{\text{max}} = 10^5$. First, we compare with the optimal solution to examine the suboptimality of our proposed method when the NBA can be captured by one poset (thus, only one solution). Second, we generate multiple solutions for

TABLE I
STATISTICS ON THE OPTIMAL COST AND SOLUTIONS

task	J^*	NoCol+Seq		Col+Sim	
		cost	horizon	cost	horizon
(i)	36.0±5.1	36.4±5.3 (35)	23.9±4.1	38.8±5.5	19.6±2.8
(ii)	25.4±2.8	28.6±3.1 (8)	29.2±2.9	28.6±3.1	29.2±2.9

Case Study I: Column "NoCol+Seq" represents the case where collision avoidance is ignored and robots move sequentially, and column "Col+Sim" incorporates collision avoidance and simultaneous execution. The notation J^* denotes the optimal cost without considering collision avoidance. The number of trials out of 50 trials where the cost cost is equivalent to the optimal cost J^* are shown inside the parentheses.

specifications with multiple posets, and compare the cost of the first solution corresponding to the widest poset to that of the subsequent solutions. We observe that the quality of the first solution obtained for the widest poset is generally very good. Finally, we compare our method to the approach proposed in [39] for large workspaces and numbers of robots and show that our method outperforms the approach in [39] in terms of optimality and scalability. We emphasize that the sets of restricted accepting runs of all specifications $\phi_1 - \phi_{10}$ considered in the following simulations, are nonempty, which shows that this assumption is not restrictive in practice.

A. Case study I: Suboptimality

In this case study, we examine the quality of the paths constructed for the two tasks in the Example 1. Observe that in Fig. 7(a), a unique poset corresponds to the sub-NBA $\mathcal{A}^-_{\text{subtask}}$ for task (i). A similar observation can be made for the sub-NBA $\mathcal{A}^-_{\text{subtask}}$ in Fig. 5(b) for task (ii). In the workspace shown in Fig. 9, we randomly generate the initial locations of all robots inside label-free cells. To measure the suboptmality of our solution in terms of path length (travelled distance), we use brute-force search to find the optimal cost.

Next, given the same randomly generated initial robot locations, we implement our proposed method in the following two different ways. First, we implement GMRPP without collision avoidance and with sequential execution (see Appendix B-B1 in [40]). Using sequential execution, only robots participating in the subtask under consideration are assigned target regions and the rest of the robots just move out of their way, whereas in the case of the simultaneous execution (see Appendix B-D2 in [40]), multiple subtasks can be undertaken at the same time, and robots that do not participate in the current subtasks simultaneously move towards their target points for subsequent subtasks. Second, we implement GMRPP with collision avoidance and with simultaneous execution. Both implementations employ the full execution (see Appendix B-B1 in [40]), in which all robots are allowed to move. Note that in the partial execution (see Appendix B-D3 in [40]), only necessary robots participating in the current subtask are allowed to move and the remaining robots are treated as obstacles. Table I shows statistical results on the path costs and path time horizons (number of time stamps), averaged over 50 trials. For task (i) , the MILP for the high-level plan includes 105 variables and 183 constraints; for task (ii), it includes 33 variables and 61 constraints to find the prefix plan and 94 variables and 162 constraints to find the suffix plan.

Without considering collision avoidance, the cost is close to the optimal cost, especially for task (i) that only requires paths of finite length. In 35 out of 50 trials, our method can identify the exact optimal solutions. For task (ii), the additional cost arises from planning separately for the prefix and suffix parts. In the prefix part, the robot can visit the cell in region ℓ_2 that is the closest to its initial location, however, it may incur additional cost to return to this cell in the suffix part. The costs when considering collision avoidance are also close to the optimal cost, indicating that often robots follow the shortest path. As for the path horizon, observe that, for task (i), simultaneous execution results in shorter horizons since one robot of type 2 can move towards ℓ_4 while two robots of type 1 leave from their initial locations for ℓ_2. For task (ii), the horizon remains almost the same, since corresponding subtasks cannot be executed in parallel by the same robot.

B. Case study II: Quality of the first solution

Common to two specifications above is that the sub-NBA $\mathcal{A}^-_{\text{subtask}}$ for the prefix and suffix parts can be concisely captured by one poset, which may not be the case for most specifications. Here, we consider various specifications that can produce many posets and examine the quality of the first solutions obtained for the widest poset by comparing to subsequent solutions obtained for subsequent posets. We use the same workspace and robot team as in Example 1. The considered specifications are as follows:

$$\phi_3 = \square\lozenge(\pi^{2,1}_{2,1} \wedge \lozenge(\pi^{3,1}_{2,1} \wedge \lozenge(\pi^{4,1}_{2,1} \wedge \lozenge\pi^{5,1}_{2,1}))),$$
$$\phi_4 = \square\lozenge(\pi^{2,1}_{2,1} \wedge \lozenge\pi^{3,1}_{2,1}) \wedge \square(\pi^{5,2}_{1,1} \Longrightarrow \bigcirc(\pi^{5,2}_{1,1} \,\mathcal{U}\, \pi^{4}_{1,2})) \wedge \square\neg\pi^{4}_{2,1}$$
$$\phi_5 = \lozenge(\pi^{4,1}_{1,2} \wedge \bigcirc(\pi^{4,1}_{1,2} \,\mathcal{U}\, \pi^{3}_{2,1})) \wedge \square\lozenge(\pi^{4,1}_{1,2} \wedge \lozenge\pi^{3,1}_{1,2}),$$
$$\phi_6 = \square\lozenge(\pi^{3,1}_{1,1} \vee \pi^{5,1}_{1,1}) \wedge \square\lozenge\pi^{2,1}_{1,1} \wedge \square\lozenge(\pi^{3}_{2,2} \vee \pi^{5}_{2,2}) \wedge \square\neg\pi^{4}_{2,1} \wedge \square\neg\pi^{4}_{2,2},$$
$$\phi_7 = \square\lozenge(\pi^{4}_{1,2} \wedge \bigcirc(\lozenge\neg\pi^{4}_{1,2})) \wedge \square\lozenge(\pi^{5}_{1,1} \wedge \bigcirc(\lozenge\neg\pi^{5}_{1,1})) \wedge \lozenge(\pi^{3}_{3,1} \wedge \pi^{3}_{2,2}),$$
$$\phi_8 = \square\lozenge(\pi^{4,1}_{2,2} \wedge \lozenge(\pi^{2,1}_{2,2} \wedge \lozenge\pi^{5,1}_{2,2})) \wedge \neg\pi^{2}_{1,2} \,\mathcal{U}\, \pi^{4,1}_{2,2} \wedge \neg\pi^{5}_{1,2} \,\mathcal{U}\, \pi^{4,1}_{2,2}$$
$$\wedge (\square\lozenge\pi^{5}_{2,1} \vee \square\lozenge\pi^{3}_{2,1}).$$

These specifications involve various operators and are representative of commonly used complex tasks in robotics applications. For example, ϕ_3 can capture surveillance and data gathering tasks [4], and the subformula $\square\lozenge(\pi^{3,1}_{1,1} \vee \pi^{5,1}_{1,1})$ in ϕ_6 can specify intermittent connectivity tasks where robots are required to meet at communication regions infinitely often [5]. Furthermore, subformula $\square\neg\pi^{4}_{2,2}$ in ϕ_6 can be used to represent collision avoidance among robots and $\neg\pi^{2}_{1,2} \,\mathcal{U}\, \pi^{4,1}_{2,2}$ in ϕ_8 can prioritize certain subtasks to others.

We executed our method 20 times for each specification. In each trial, we randomly generated initial robot locations inside the label-free cells such that no two robots occupy the same cell. We considered collision avoidance, as well as full and simultaneous execution. In Table II, we report the number of pairs of initial and accepting vertices in the NBA $\mathcal{A}_\phi$ before pre-processing, the size (number of vertices and edges) of the NBA $\mathcal{A}_\phi$ before and after pre-processing, and the size of the sub-NBA $\mathcal{A}^-_{\text{subtask}}$ for the prefix and suffix parts from which the first solutions are obtained. The size (number of variables and constraints) of the MILP for the prefix and suffix part of the first solution is also displayed.[1] We terminate our method until all solutions or the first 10 solutions are generated, whichever

[1]The size of the MILP differs for different solutions since they may be generated from different posets of subtasks. We only report the results for the first solution since we aim to examine the quality of the first solution.

TABLE II
RESULTS FOR SPECIFICATIONS $\phi_3 - \phi_8$

Task	N_{pair}	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{A}_\phi\rvert$	$\lvert\mathcal{A}^{-,\text{pre}}_{\text{subtask}}\rvert$	$\lvert\mathcal{A}^{-,\text{suf}}_{\text{subtask}}\rvert$	MILP$^{\text{pre}}$	MILP$^{\text{suf}}$	$N_{\text{sol}}=1$		$N_{\text{sol}}=5$		$N_{\text{sol}}=10$	
								cost	time(sec)	cost	time(sec)	cost	time(sec)
ϕ_3	8	(20, 142)	(20, 49)	(3, 2)	(5, 5)	(45, 78)	(276, 397)	66.4±4.7	1.7±0.2	—	—	—	—
ϕ_4	4	(10, 57)	(10, 31)	(3, 2)	(3, 3)	(45, 78)	(130, 210)	61.4±4.8	1.4±0.2	—	—	—	—
ϕ_5	2	(11, 31)	(11, 25)	(10, 19)	(3, 3)	(78, 141)	(76, 142)	17.9±5.7	0.5±0.1	17.9±5.7	2.2±0.7	—	—
ϕ_6	1	(4, 9)	(4, 8)	(4, 5)	(4, 5)	(117, 203)	(204, 308)	33.3±7.1	1.2±0.5	30.8±5.7	3.2±1.1	30.8±5.7	4.5±1.5
ϕ_7	3	(24, 140)	(24, 104)	(22, 57)	(9, 18)	(124, 194)	(93, 164)	45.4±7.1	2.5±0.3	45.4±7.1	2.9±0.3	45.4±7.1	3.9±0.3
ϕ_8	4	(15, 83)	(15, 41)	(8, 13)	(8, 14)	(120, 210)	(201, 325)	74.0±6.2	1.6±0.2	74.0±6.2	8.7±0.5	74.0±6.2	22.4±1.2

Case Study II: N_{pair} is the number of pairs of initial and accepting vertices, $|\mathcal{A}|$, $|\mathcal{A}_\phi|$, $|\mathcal{A}^{-,\text{pre}}_{\text{subtask}}|$ and $|\mathcal{A}^{-,\text{suf}}_{\text{subtask}}|$ are the size of the NBA before and after pre-processing, for the prefix and suffix parts from which the first solutions are obtained, respectively. MILP$^{\text{pre}}$ and MILP$^{\text{suf}}$ are the size of MILP of the first solution. The symbol "—" means that only one solution found for ϕ_3 and ϕ_4, and less than or equal to 5 solutions found for ϕ_5.

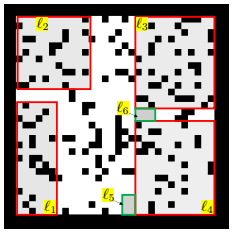


Fig. 10. Grid world from [39].

comes first. We record the smallest cost and runtimes achieved by the first solution, after the first 5 and 10 solutions.

In Table II, the size of sub-NBA $\mathcal{A}^{-}_{\text{subtask}}$ is dramatically reduced compared to the size of NBA before pre-processing, especially for specifications ϕ_3, ϕ_7 and ϕ_8, considerably reducing the computation times. It takes about 20 seconds to get 10 solutions for specification ϕ_8. Except for specification ϕ_6, the first solution returned by our method is also the lowest cost solution. For specification ϕ_6, the best solution corresponds to one of the first 5 posets. This is because our optimization-based method sorts the set of posets in part according to their height so that posets with smaller numbers of subtasks are considered first (see Section IV-C). Therefore, we can terminate our method only after a few solutions have been obtained, which is especially important when the complexity of the planing problem increases.

C. Case study III: Scalability

In this case study, we examine the scalability with respect to the size of the workspace and the number of robots.

1) Comparison with the BMC method: Similar to our method, [39] also adopts a hierarchical framework, which improves the scalability of the methods in [35, 36] that focus on feasibility of control synthesis over LTL^0. For the purpose of comparison, we borrow the workspace used in [39], a 30-by-30 grid world containing 6 regions $\ell_i, i = 1, \ldots, 6$; shown in Fig. 10. At each trial, 20% of cells are randomly selected as obstacles. Consider a team of n robots of the same type whose initial locations are randomly sampled inside region ℓ_1. The specification is given by [39]:

$$\phi_9 = \square\Diamond\pi^2_{n,1} \wedge \square\Diamond\pi^3_{n/2,1} \wedge \square\Diamond\pi^4_{n/2,1} \wedge \neg\pi^4_{1,1}\,\mathcal{U}\,(\pi^5_{1,1} \wedge \pi^6_{1,1}).$$

The size of the NBA is independent from the number of robots. The NBA $\mathcal{A}_\phi$ has one pair of initial and accepting vertices, 5 vertices and 10 edges (excluding self-loops). The sub-NBA $\mathcal{A}^{-}_{\text{subtask}}$ for the prefix part has 5 vertices and 5 edges and for the suffix part has 4 vertices and 5 edges. We employ the full and simultaneous execution. We record runtimes and cost of the first feasible solutions, where the cost is the sum of the cost of the prefix and suffix parts. Both methods consider collision avoidance. The horizon increases by 10 when no solution exists for the GMRPP, until the considered horizon exceeds the initial horizon by 100. The work [39] can address robots of the same type. The statistical results averaged over 10 trials are shown in Table III. For $n = 30$ robots, the MILP to find the prefix plan includes 89179 variables and 91244 constraints and the MILP to find the suffix plan includes 112519 variables and 115482 constraints.

TABLE III
RESULTS WITH RESPECT TO THE NUMBER OF ROBOTS

n	Our method		BMC method	
	cost	time(sec)	cost	time(sec)
4	270.6±4.4	62.4±1.4	944.4±21.2	76.5±13.8
8	513.0±30.2	124.9±9.2	1819.0±149.9	334.9±153.9
12	794.6±11.1	187.4±9.1	2217.0±163.8	704.3±178.0
16	1080.2±14.7	502.0±225.4	2725.8±149.2	1135.8±123.7
30	2509.4±168.9	4072.1±985.4	—	—

Observe in Table III that our method outperforms the BMC method both in terms of runtimes and optimality of the solutions. Specifically, as the number of robots increases, the runtime of our method is about half the runtime of the BMC method but the cost returned by our method is about 1/3 of the cost of the solutions obtained using the BMC method. The reason is that we optimize the cost at both the high level and the low level, while the BMC method only considers feasibility. For $n = 30$ robots, the BMC method did not produce a solution within 2 hours. Furthermore, the efficiency of the low-level path planner has significant impact on the runtime. In our method, the number of times that the path planner is invoked is the same or smaller than the number of subtasks in the simple path extracted from the high-level plan. On the other hand, the BMC method abstracts the given environment by aggregating states with the same observation, where transitions between abstract states are defined by whether they share the same boundary. Then, each transition in the high-level plan obtained by the BMC method is converted into one instance of multi-robot path planning problem. Obviously, the number of transitions in the BMC method is larger than the number of subtasks in our method, since each subtask may take multiple transitions.

2) Full vs. partial GMRPP execution: We use the same workspace as in Fig. 10 and consider a team of n homogeneous robots that are subject to the specification:

TABLE IV
RESULTS WITH RESPECT TO THE NUMBER OF ROBOTS.

n	Full execution		Partial execution	
	cost	time(sec)	cost	time(sec)
4	181.4±17.7	89.5±5.0	180.4±20.1	65.8±10.1
8	356.6±16.0	198.9±12.3	354.2±15.2	129.3±4.9
12	573.5±63.3	350.7±25.4	554.3±49.4	192.5±10.4
16	774.2±59.0	561.0±44.4	763.0±50.7	278.9±8.9
32	1560.4±160.7	1886.8±696.0	1524.6±30.6*	778.1±134.9

* 3 out of 10 trials failed.

$$\phi_{10} = \Diamond(\pi^5_{3,1} \vee \pi^6_{3,1}) \wedge \Box\Diamond(\pi^{2,1}_{n/2,1} \wedge \Diamond\pi^{4,1}_{n/2,1}) \wedge \Box\Diamond\pi^3_{n/4,1} \wedge \Box\neg\pi^6_{4,1}.$$

Before pre-processing, there are two pairs of initial and accepting vertices in the NBA $\mathcal{A}_\phi$ that contains 8 vertices and 27 edges. After pre-processing, the NBA $\mathcal{A}_\phi$ has 8 vertices and 20 edges. For the first pair of initial and accepting vertices, the sub-NBA $\mathcal{A}^-_{\text{subtask}}$ associated with the prefix part has 7 vertices and 10 edges, and the sub-NBA associated with suffix part has 5 vertices and 7 edges. We compare the performance of our method for the full and partial execution in the GMRPP problem and for an increasing number of robots up to 32. The results averaged over 10 trials are shown in Table IV. For $n = 32$ robots, the MILP to find the prefix plan includes 41428 variables and 42533 constraints and the MILP to find the suffix plan includes 78625 variables and 81200 constraints. It can be seen that our method with partial execution in the GMRPP problem takes less time than with full execution. This advantage becomes more significant as the number of robots increases since in this case, a larger number of robots that do not participate in the current subtask can remain idle and can be treated as obstacles in the GMRPP. For example, the subtask requiring that at least 3 robots meet at region ℓ_5 or ℓ_6, only involves 3 robots no matter how large the robot team is. On the other hand, the full execution of the GMRPP problem results in slightly larger cost which suggests that even though all robots are allowed to move, those robots that do not participate in the specific subtask rarely move because our method optimizes the cost. Observe that for the partial execution and for 32 robots, no solutions are generated in 3 out of 10 trials. This is due to the fact that robots treated as obstacles affect the obstacle-free workspace and, therefore, may make the GMRPP infeasible. Thus, the partial execution of the GMRPP problem can be more effective in large workspaces with few robots, where a few idle robots do not significantly alter the obstacle-free environment.

VIII. CONCLUSION

In this work, we consider the problem of allocating tasks, expressed as global LTL specifications, to teams of heterogeneous mobile robots. We proposed a hierarchical approach to solve this problem that first solves an MILP to obtain a high-level time-stamped allocation of robots to tasks and then formulates a sequence of multi-robot path planning problems to obtain the low-level executable paths. We proved that, with mild assumptions, the proposed method is complete and we provided extensive simulations that showed that our method outperforms the state-of-the-art BMC method in terms of optimality and scalability. Scalability of our method is primarily due to a clever relaxation of the NBA that captures the LTL specification, that involves removing the negative literals. This relaxation is motivated by "lazy collision checking" methods for point-to-point navigation, and significantly simplifies the high-level planning problem as constraint violation is not considered during planning and instead it is only checked during execution when needed. To the best of our knowledge, this is the first time that "lazy collision checking" methods are used and shown to be effective for high-level planning tasks.

REFERENCES

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[2] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press Cambridge, 2008.

[3] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for mobile robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain, 2005, pp. 2020–2025.

[4] M. Guo and M. M. Zavlanos, "Distributed data gathering with buffer constraints and intermittent communication," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 279–284.

[5] Y. Kantaros and M. M. Zavlanos, "Distributed intermittent connectivity control of mobile robot networks," *IEEE Transactions on Automatic Control*, vol. 62, no. 7, pp. 3109–3121, 2017.

[6] K. Leahy, D. Zhou, C.-I. Vasile, K. Oikonomopoulos, M. Schwager, and C. Belta, "Persistent surveillance for unmanned aerial vehicles subject to charging and temporal logic constraints," *Autonomous Robots*, vol. 40, no. 8, pp. 1363–1378, 2016.

[7] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, and M. Fisher, "Formal specification and verification of autonomous robotic systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–41, 2019.

[8] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning under temporal logic constraints," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 3288–3293.

[9] 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.

[10] 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.

[11] G. Sánchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *Robotics research*. Springer, 2003, pp. 403–417.

[12] D. Bredström and M. Rönnqvist, "Combined vehicle routing and scheduling with temporal precedence and synchronization constraints," *European journal of operational research*, vol. 191, no. 1, pp. 19–31, 2008.

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

[14] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe LTL specifications," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 1525–1532.

[15] Y. Kantaros and M. M. Zavlanos, "Temporal logic optimal control for large-scale multi-robot systems: 10^{400} states and beyond," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 2519–2524.

[16] X. Luo, Y. Kantaros, and M. M. Zavlanos, "An abstraction-free method for multirobot temporal logic optimal control synthesis," *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1487–1507, 2021.

[17] M. Kloetzer, X. C. Ding, and C. Belta, "Multi-robot deployment from LTL specifications with reduced communication," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*. IEEE, 2011, pp. 4867–4872.

[18] Y. Shoukry, P. Nuzzo, A. Balkan, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 1132–1137.

[19] S. Moarref and 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)*. IEEE, 2017, pp. 17–23.

[20] B. Lacerda and P. U. Lima, "Petri net based multi-robot task coordination from temporal logic specifications," *Robotics and Autonomous Systems*, vol. 122, p. 103289, 2019.

[21] J. Tumova and D. V. Dimarogonas, "Decomposition of multi-agent planning under distributed motion and task LTL specifications," in *2015 54th IEEE Conference on Decision and Control (CDC)*. IEEE, 2015, pp. 7448–7453.

[22] Y. Kantaros and M. M. Zavlanos, "Distributed communication-aware coverage control by mobile sensor networks," *Automatica*, vol. 63, pp. 209–220, 2016.

[23] X. Luo and M. Zavlanos, "Transfer planning for temporal logic tasks," in *Proc. of the 58th IEEE Conference on Decision and Control*, France, Nice, 2019.

[24] A. Camacho, E. Triantafillou, C. J. Muise, J. A. Baier, and S. A. McIlraith, "Non-deterministic planning with temporally extended goals: Ltl over finite and infinite traces." in *AAAI*, 2017, pp. 3716–3724.

[25] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Hierarchical LTL-task mdps for multi-agent coordination through auctioning and learning," *The International Journal of Robotics Research*, 2019.

[26] S. Karaman and E. Frazzoli, "Linear temporal logic vehicle routing with applications to multi-uav mission planning," *International Journal of Robust and Nonlinear Control*, vol. 21, no. 12, pp. 1372–1395, 2011.

[27] 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.

[28] M. Kloetzer and C. Mahulea, "Path planning for robotic teams based on LTL specifications and petri net models," *Discrete Event Dynamic Systems*, vol. 30, no. 1, pp. 55–79, 2020.

[29] 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.

[30] F. Faruq, D. Parker, B. Laccrda, and N. Hawes, "Simultaneous task allocation and planning under uncertainty," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3559–3564.

[31] C. Banks, S. Wilson, S. Coogan, and M. Egerstedt, "Multi-agent task allocation using cross-entropy temporal logic optimization," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7712–7718.

[32] Y. Kantaros and M. M. Zavlanos, "Sampling-based optimal control synthesis for multirobot systems under global temporal tasks," *IEEE Transactions on Automatic Control*, vol. 64, no. 5, pp. 1916–1931, 2018.

[33] ——, "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.

[34] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," in *International Conference on Computer Aided Verification*. Springer, 2002, pp. 359–364.

[35] Y. E. Sahin, P. Nilsson, and N. Ozay, "Synchronous and asynchronous multi-agent coordination with cLTL+ constraints," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 335–342.

[36] ——, "Multirobot coordination with counting temporal logics," *IEEE Transactions on Robotics*, 2019.

[37] A. Biere, K. Heljanko, T. Junttila, T. Latvala, and V. Schuppan, "Linear encodings of bounded LTL model checking," *Logical Methods in Computer Science*, vol. 2, no. 5:5, pp. 1–64, 2006.

[38] A. M. Jones, K. Leahy, C. I. Vasile, S. Sadradinni, Z. Serlin, R. Tron, and C. Belta, "Scalable and Robust Deployment of Heterogenenous Teams from Temporal Logic Specifications," in *International Symposium on Robotics Research (ISRR)*, Hanoi, Vietnam, October 2019.

[39] Y. E. Sahin, N. Ozay, and S. Tripakis, "Multi-agent coordination subject to counting constraints: A hierarchical approach," in *Distributed Autonomous Robotic Systems*. Springer, 2019, pp. 265–281.

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

[41] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," in *1st Symposium in Logic in Computer Science (LICS)*. IEEE Computer Society, 1986.

[42] L. S. Heath and A. K. Nema, "The poset cover problem," *Open Journal of Discrete Mathematics*, vol. 3, no. 03, p. 101, 2013.

[43] E. Nunes, M. Manner, H. Mitiche, and M. Gini, "A taxonomy for task allocation problems with temporal and ordering constraints," *Robotics and Autonomous Systems*, vol. 90, pp. 55–70, 2017.

[44] P. Gastin and D. Oddoux, "Fast LTL to büchi automata translation," in *International Conference on Computer Aided Verification*. Springer, 2001, pp. 53–65.

[45] A. Stefanescu, "Automatic synthesis of distributed transition systems," 2006.

[46] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, no. 3, pp. 291–314, 2001.

[47] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[48] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2018. [Online]. Available: http://www.gurobi.com

Xusheng Luo (S'19) received the B.Eng. and M.S.E. degrees in aerospace engineering from the Harbin Institute of Technology, Harbin, China, in 2015 and 2017, respectively, and the Ph.D. degree in mechanical engineering from Duke University, Durham, NC, in 2020. His research interest focuses on the formal control synthesis under high-level specifications with applications in robotics.

Michael M. Zavlanos (S'05M'09SM'19) received the Diploma in mechanical engineering from the National Technical University of Athens, Greece, in 2002, and the M.S.E. and Ph.D. degrees in electrical and systems engineering from the University of Pennsylvania, Philadelphia, PA, in 2005 and 2008, respectively. He is currently an Associate Professor in the Department of Mechanical Engineering and Materials Science at Duke University, Durham, NC. His research focuses on control theory, optimization, and learning and, in particular, autonomous systems and robotics, networked and distributed control systems, and cyber-physical systems. Dr. Zavlanos is a recipient of various awards including the 2014 ONR YIP Award and the 2011 NSF CAREER Award.