2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2025)
October 19-25, 2025. Hangzhou, China

2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 979-8-3315-4393-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/IROS60139.2025.11246879

MaxAuc: A Max-Plus-Based Auction Approach for Multi-Robot Allocations for Time-Ordered Temporal Logic Tasks

Mengjie Wei, Yuda Li, Siqi Wang, Shaoyuan Li and Xiang Yin

***Abstract*— In this paper, we investigate a multi-robot task allocation problem where a team of heterogeneous robots operates in a discrete workspace to achieve a set of tasks expressed by linear temporal logic formulas. In contrast to existing works, we further consider inter-task-time-order constraints, which are imposed on the start or end times of each task. Solving such problems generally requires combinatorial search, which is not scalable. Inspired by the efficiency of max-plus algebra in handling time constraints, we propose a novel approach called MaxAuc, which integrates Auction-based task allocation with Max-plus algebra in a novel manner. Specifically, max-plus computations are performed to approximate task priorities in the auction without explicitly solving the constraint optimization problem. Our numerical results demonstrate that MaxAuc is highly scalable with respect to both the number of robots and the number of tasks, while maintaining a tolerable performance trade-off compared to the baseline's optimal yet exhaustive solution.**

I. INTRODUCTION

Task allocation and planning are central challenges in autonomous systems, such as mobile robots and manufacturing systems [1], [2]. In recent years, with the increasing complexity and functionality of autonomous systems, there has been growing interest in task planning for high-level specifications described by temporal logic. Among these, Linear Temporal Logic (LTL) has emerged as one of the most widely used formal languages for specifying system behavior [3]. By extending Boolean logic with temporal operators, LTL provides a rigorous yet user-friendly framework for describing complex tasks that involve both temporal dependencies and logical constraints. LTL-based task planning has been successfully applied in various systems; see, e.g., recent surveys [4] for extensive literature.

In the context of LTL planning, the basic approach is the automata-theoretical framework, where the system model is synchronized with the automaton representation of the LTL formula, reducing the planning problem to a graph-search problem [5]–[7]. For multi-robot systems, the planning problem can, in principle, be solved by synchronizing the behavior of each robot [8], [9]. However, this monolithic approach faces significant challenges. One major challenge is scalability, as the size of the product state space grows exponentially with the number of robots. To address this, many recent works have focused on developing efficient planning methods that avoid constructing the monolithic model from the outset [10]–[15].

Another challenge, more specific to multi-robot systems, is the need to handle multiple sub-tasks. Particularly, decision-making in such systems often involves assigning specific sub-tasks to individual robots. These sub-tasks may either be provided directly as part of the overall task or decomposed from it [16]–[19]. Hence, one must determine at what time, which robot should execute which sub-task, leading to a highly combinatorial problem. An example is simultaneous task allocation and planning, which leverages transition relations in LTL-derived automaton to dynamically partition tasks and assign sub-tasks to agents [20], [21]. Auction mechanisms, as a well-established allocation tool, are often employed here to optimize task distribution efficiently [22], [23].

In this paper, we address a new class of multi-robot temporal logic task allocation and planning problems. Specifically, we consider a scenario where a team of n robots aims to accomplish a set of m tasks, each represented by a co-safe LTL (scLTL) formula. The planner must assign each robot to tasks by fulfilling their underlying atomic propositions, ensuring that all tasks are ultimately completed. Unlike most existing works, we further incorporate the starting and finishing times of each task into the problem formulation. A new constraint, referred to as the *inter-task-time-order constraint*, is introduced to impose the temporal relationships between the starting and finishing times of the tasks. The overall objective is to allocate the team of robots to tasks, fulfilling the LTL specifications in the least possible time while satisfying the task-time-order constraints. This problem formulation is motivated by many practical scenarios. For example, consider a team of field robots to complete three tasks: ϕ_1 "assemble a generator", ϕ_2 "lay pipes", and ϕ_3 "turn on a machine in the right steps". Each task is described by an LTL formula over basic operations. In this scenario, ϕ_3 can only start after ϕ_1 finishes, as the machine requires electricity generated during the turning-on steps. However, ϕ_2 can be started at any time but must be completed before ϕ_3 finishes to prevent the machine from running idle.

To solve this problem, we propose a novel algorithm called MaxAuc, a **Max**-plus-based **Auc**tion approach for multi-robot allocations for time-ordered temporal logic tasks. Our approach begins by defining a set of auction items based on the progress of each task in its automaton representation. While arranging these items in a time-optimal manner typically reduces to the naive monolithic solution, we propose a computationally more efficient method by leveraging max-plus algebra, a technique widely used in discrete event

This work was supported by the National Natural Science Foundation of China (62173226, 92367203).

M. Wei, Y. Li, S. Wang, S. Li and X. Yin are with the School of Automation and Intelligent Sensing, Shanghai Jiao Tong University, Shanghai 200240, China. (Corresponding Author: Xiang Yin.) E-mail: {mj.wei,yuda.li,sq_wang,syli,yinxiang}@sjtu.edu.cn.

systems for handling makespans [24], to estimate the priority of each task and the marginal contribution of each auction item. Using this heuristic information informed by max-plus algebra, we generate an allocation plan incrementally, ensuring that all tasks are fulfilled while satisfying the inter-task-time-order constraints. We conduct numerical experiments to validate the efficiency of our approach. Specifically, we show that our method is highly scalable as the number of tasks, robots, or workspace size increases, while only sacrificing a small performance compared to the naive baseline algorithm, which can easily become infeasible for larger-scale problems.

II. Preliminaries

A. Linear Temporal Logic Specifications

Let Σ be an alphabet. We denote by Σ^ω and $\Sigma^\star$ the set of all infinite words and the set of all finite words over Σ, respectively. For any finite word $\rho = \sigma_1\sigma_2\dots\sigma_n \in \Sigma^\star$, we define $|\rho| = n$ as its length with $|\epsilon| = 0$ for empty word ϵ.

Let $\mathcal{AP}$ be a set of atomic propositions. The syntax of linear temporal logic formulae is defined recursively by:

$$\varphi ::= \top \mid \pi \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2, \tag{1}$$

where $\top$ denotes the Boolean constant *true*, $\pi \in \mathcal{AP}$ is an atomic proposition, $\wedge$ and $\neg$ are Boolean operators "conjunction" and "negation", respectively, and $\bigcirc$ and $\mathcal{U}$ are temporal operators "next" and "until", respectively. Note that, one can further induce other temporal operators such as "eventually" by $\Diamond\varphi := \top\mathcal{U}\varphi$. In general, LTL formulae are evaluated over infinite sequences over $2^{\mathcal{AP}}$. For word $\rho \in (2^{\mathcal{AP}})^\omega$, we denote by $\rho \models \varphi$ if it satisfies φ; the readers are referred to [25] for more details on the semantics of LTL.

Since we focus on finite-horizon tasks, we restrict our attention to a fragment known as syntactically co-safe LTL (scLTL). Compared to the full LTL fragment, the main difference is that negation can only be applied to atomic propositions. As a result, temporal operators such as "always" cannot be used. For any finite word $\rho_F \in (2^{\mathcal{AP}})^\star$, we denote $\rho_F \models \varphi$ if $\rho_F\rho \models \varphi$ for any infinite extension $\rho \in (2^{\mathcal{AP}})^\omega$. A key property of scLTL is that for any infinite word satisfying φ, there exists a "good prefix" satisfying φ.

Definition 1 (DFA): A deterministic finite automaton (DFA) is a five-tuple $\mathcal{D} = (Q, \Sigma, \delta, q_0, \mathcal{F})$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is a finite set of alphabets, $\delta : Q \times \Sigma \to Q$ is the transition function, and $F \subseteq Q$ is the set of accepting states.

The transition function can be extended to $\delta : X \times \Sigma^\star \to X$ recursively by: (i) $\delta(q, \epsilon) = q$; and (ii) for any $\rho \in \Sigma^*, \sigma \in \Sigma$, we have $\delta(q, \rho\sigma) = \delta(\delta(q, \rho), \sigma)$. We say a finite word $\rho \in \Sigma^\star$ is accepted by $\mathcal{D}$ if $\delta(q_0, \rho) \in \mathcal{F}$. It is well-known that for any scLTL formula φ, there is a DFA, denoted by $\mathcal{D}_\varphi$, that exactly accepts all good prefixes of φ. For any state $q \in Q$, we denote by $d_{\mathcal{D}}(q)$ its shortest distance to accepting states in DFA $\mathcal{D}$, i.e., $d_{\mathcal{D}}(q) = \min\{|\rho| : \delta(q, \rho) \in \mathcal{F}\}$.

B. Max-Plus Algebra

A max-plus algebra is an algebraic structure defined on the set $\mathbb{Z} \cup \{+\infty\}$ or $\mathbb{R} \cup \{+\infty\}$ with the algebraic addition $\oplus$ defined by $x \oplus y = \max(x, y)$ and multiplication $\otimes$ defined by $x \otimes y = x + y$. We denote for simplicity $-\infty$ as ε, and $\mathbb{Z} \cup \{-\infty\}$ as $\mathbb{Z}_\varepsilon$. For $n \in \mathbb{N}$, we denote $\mathbb{Z}_\epsilon^n$ as the Cartesian product of n copies of $\mathbb{Z}_\varepsilon$, together with a partial order $\preceq$ induced by the order $\leq$ on $\mathbb{Z}_\varepsilon$.

We denote $\mathbb{Z}_\varepsilon^{m\times n}$ as the set of matrices of size $m \times n$ with entries in $\mathbb{Z}_\varepsilon$. The operations of max-plus algebraic addition ($\oplus$) and multiplication ($\otimes$) on matrices are defined as: $[A \oplus B]_{ij} = a_{ij} \oplus b_{ij} = \max(a_{ij}, b_{ij})$, $[A \otimes C]_{ij} = \bigoplus_{k=1}^n a_{ik} \otimes c_{kj} = \max_{k=1,\dots,n}(a_{ik} + c_{kj})$. Here, a_{ij} denotes the element in the i-th row and j-th column of the matrix A. The natural order $\preceq$ on the set of matrices $\mathbb{Z}^{n\times m}$ is defined by $A \preceq B \Leftrightarrow A \oplus B = B$. We denote $I_n \in \mathbb{Z}_\varepsilon^{n\times n}$ as the identity matrix in $\mathbb{Z}_\varepsilon^{n\times n}$, which is characterized by having 0 on its diagonal and ε elsewhere. We define in addition for $x \in \mathbb{Z}_\varepsilon^n$, and for $A \in \mathbb{Z}_\varepsilon^{m\times n}$, $[A \otimes x]_i = \bigoplus_{1\leq j\leq n} A_{ij} \otimes x_j$ for $1 \leq i \leq m$.

In this paper, we will encounter the typical problem of finding the smallest element of the set $\{x \in \mathbb{Z}_\varepsilon^n \mid A \otimes x \oplus b \preceq x\}$ where A is a matrix in $\mathbb{Z}^{n\times n}$ and $b \in \mathbb{Z}_\varepsilon^n$. Such a problem has been solved by [26], which states that the smallest element exists and is given explicitly by the following form

$$x^* = \left(\bigoplus_{k\in\mathbb{Z}} A^{\otimes k}\right) \otimes b,$$

where $A^{\otimes k}$ is defined recursively as $A^{\otimes 0} = I_n$, $A^{\otimes k+1} = A \otimes (A^{\otimes k})$. The notation $\bigoplus_{k\in\mathbb{Z}} A^{\otimes k}$ denotes the minimal upper bound for the sequence $\{A^{\otimes k}\}_k$ in the set of matrices $\mathbb{Z}_\varepsilon^{n\times n}$. The computation complexity of x^* is $O(n^3)$ [27].

III. Problem Formulation

A. System Model

The workspace of the multi-robot system is modeled by a labeled graph

$$G = (V, E, \mathcal{AP}, L),$$

where V is the set of vertices, each representing a physical region in the workspace, $E \subseteq V \times V$ is the set of edges, representing the mobility of the robots between regions, $\mathcal{AP}$ is the set of atomic propositions, $L : V \to 2^{\mathcal{AP}}$ is a labeling function that assigns to each vertex $v \in V$ the set of atomic propositions that can be executed by the robots when they are in region v.

We consider a team of n homogeneous robots $R_1, \dots, R_n$. Each robot $r = 1, \dots, n$ is modeled by a transition system

$$R_r = (V, v_{0,r}, A, \xi), \tag{2}$$

where V is the set of all regions as in G, $v_{0,r} \in V$ is the initial state of robot r, $A = \mathcal{AP} \cup E$ is the set of actions, and the transition function $\xi : V \times A \to V$ is defined by: for each $v \in V, a \in A$, we have

$$\xi(v, a) = \begin{cases} v & \text{if} \quad a \in L(v), \\ v' & \text{if} \quad a = (v, v') \in E. \end{cases} \tag{3}$$

Essentially, this model assumes that, at each time instant, each robot has the flexibility to either execute an atomic

proposition available at its current vertex (region) or move to an adjacent region by traversing an edge in the graph.

Finally, we assume that all robots operate synchronously, and each transition in ξ consumes one time unit. This means that at each time instant, every robot selects an action to execute and all robots complete their transitions simultaneously before the next action period begins.

B. Task Allocations

We assume that n robots need to accomplish m tasks $\Phi = \{\varphi_1, \ldots, \varphi_m\}$, where each φ_i is a scLTL formula over $\mathcal{AP}$. We denote by $I = \{1, \ldots, m\}$ the index set of tasks. Note that two tasks φ_i and φ_j may share some common atomic propositions. As a result, when a robot $r \in \{1, \ldots, n\}$ takes an action $a \in \mathcal{AP}$ at time instant t, it is necessary to specify which task this atomic proposition is being executed for. To this end, we introduce the following definitions.

Definition 2 (Decisions, Plans and Projections): Let $t = 1, 2, \ldots$ be a time instant and $r \in \{1, \ldots, n\}$ be a robot.

- The **local decision** of robot r at time instant t is an action-task pair of form $\gamma_r^t = (a_r^t, i_r^t)$, where $a_r^t \in A$ is the action taken by robot r at time t and $i_r^t \in I \cup \{\perp\}$ is the task allocation for this action. Here, $i_r^t = \perp$ indicates that the action a_r^t is task-unrelated (e.g., a movement or an action not contributing to any specific task). We denote by $\Gamma = A \times (I \cup \{\perp\})$ the set of local decisions.
- The **global decision** of the multi-robot system at time instant t is represented as an n-tuple $\gamma^t = (\gamma_1^t, \ldots, \gamma_n^t)$, where each component γ_r^t corresponds to the local decision of robot r at time t. We denote by $\bar{\Gamma} = \Gamma \times \cdots \times \Gamma$ the set of all possible global decisions.
- The **allocation plan** is defined as a finite sequence of global decisions of the form

$$\kappa = \underbrace{(\gamma_1^1, \ldots, \gamma_n^1)}_{\gamma^1} \underbrace{(\gamma_1^2, \ldots, \gamma_n^2)}_{\gamma^2} \cdots \underbrace{(\gamma_1^k, \ldots, \gamma_n^k)}_{\gamma^k} \in (\bar{\Gamma})^\star, \tag{4}$$

  which contains the decision information for all robots at each time step.
- The **projection** of an allocation plan κ of the form (4) to a specific task $i \in I$ is defined as the sequence of atomic propositions that are allocated to task i over time, i.e.,

$$\kappa\restriction_i = \gamma^1\restriction_i \, \gamma^2\restriction_i \cdots \gamma^k\restriction_i, \tag{5}$$

  where $\gamma^1\restriction_i = \{a_r^t \mid \gamma_r^t = (a_r^t, i)\}$ is the set of all atomic propositions executed and allocated to task i by some robot r at time instant $t = 1, \ldots, k$.

With the above notations, we say an allocation plan κ satisfies the set of tasks $\Phi = \{\varphi_i, \ldots, \varphi_m\}$ if its projection to each task satisfies the corresponding scLTL formula, i.e.,

$$\kappa \models \Phi \quad \Leftrightarrow \quad \forall i \in I : \kappa\restriction_i \models \varphi_i. \tag{6}$$

C. Task-Time-Order Constraints

Let $\kappa \in (\bar{\Gamma})^\star$ be an allocation plan of the form (4) such that $\kappa \models \Phi$. To characterize the execution timeline of each task, we define:

- The **starting time** of task $i \in I$ in κ, denoted by $t_s(i, \kappa)$, as the first time instant at which an atomic proposition is allocated to task i, i.e., $t_s(i, \kappa) = \min\{t \mid \gamma^t\restriction_i \neq \emptyset\}$;
- The **ending time** of task $i \in I$ in κ, denoted by $t_e(i, \kappa)$, as the last time instant at which an atomic proposition is allocated to task i, i.e., $t_e(i, \kappa) = \max\{t \mid \gamma^t\restriction_i \neq \emptyset\}$;
- The **makespan** of κ, denoted by $M(\kappa)$, is then defined as the latest ending time among all tasks, i.e., $M(\kappa) = \max\{t_e(i, \kappa) \mid i \in I\}$.

The inter-task dependencies between the start and end time of each task can be defined as follows.

Definition 3 (Inter-Task-Time-Order Constraint): Let $I = \{1, \ldots, m\}$ be the index set of all tasks. The inter-task-time-order constraint (referred to as time constraints hereafter) is defined as a partial-order relation:

$$\mathcal{C} \subseteq (\{s, e\} \times I) \times (\{s, e\} \times I). \tag{7}$$

Specifically, a constraint $\langle(\alpha, i), (\beta, j)\rangle \in \mathcal{C}$ represents the requirement that: $t_\alpha(i, \kappa) < t_\beta(j, \kappa)$, where $t_\alpha(i, \kappa)$ and $t_\beta(j, \kappa)$ denote the starting or ending times of tasks i and j, depending on the values of α and β.

For an allocation plan κ, we denote by $\kappa \models \mathcal{C}$ if it satisfies constraints $\mathcal{C}$. For $\theta \in \{s, e\}$ and $k \in I$, we define

$$\begin{aligned} \mathcal{C}_L[\theta, k] &= \{\langle(\theta, k), (\alpha, i)\rangle \in \mathcal{C} \mid \alpha \in \{s, e\},\ i \in I\}, \\ \mathcal{C}_R[\theta, k] &= \{\langle(\alpha, i), (\theta, k)\rangle \in \mathcal{C} \mid \alpha \in \{s, e\},\ i \in I\}, \end{aligned} \tag{8}$$

to represent constraints where $t_\theta(k, \kappa)$ appears on the LHS and RHS of the inequality $t_\alpha(i, \kappa) < t_\beta(j, \kappa)$, respectively.

D. Problem Statement

Problem 1: Given a team of n homogeneous robots $R_1, \ldots, R_n$ operating in workspace G, and a set of scLTL tasks Φ with time constraints $\mathcal{C}$, find an allocation plan κ that satisfies Φ and $\mathcal{C}$, while minimizing, i.e.,

$$\begin{aligned} \min_{\kappa} \quad & M(\kappa) \\ \text{s.t.} \quad & \kappa \models \Phi \wedge \kappa \models \mathcal{C}. \end{aligned} \tag{9}$$

IV. A Naive Baseline Approach

Problem 1 can be solved using a naive approach by synchronizing all robot models and scLTL DFAs in a monolithic manner. Based on the global product space, one can simply treat states violating the time constraints as illegal and solve the optimal reach-avoid problem. While the objective of this work is to avoid using this naive approach due to its lack of scalability, we will use it as a baseline in our comparison experiments to find the optimal solution. Therefore, we briefly outline the sketch of this naive approach without delving into detail.

Synchronize the DFAs: First, we convert each scLTL task to DFA $\mathcal{D}_i = (Q_i, q_i^{(0)}, \Sigma_i, \delta_i, \mathcal{F}_i)$, and build their synchronization automaton denoted by

$$\mathcal{D}_\times = (Q_\times, q_\times^{(0)}, \Sigma_\times, \delta_\times, \mathcal{F}_\times), \tag{10}$$

where we have $Q_\times = Q_1 \times \cdots \times Q_N$ and $\Sigma_\times = \Sigma_1 \times \cdots \times \Sigma_N$; we denote $[q_\times]_i, [\sigma_\times]_i$ as the i-th elements of $q_\times, \sigma_\times$.

Algorithm 1: Check Time Constraints

Input: $(q_\times, \sigma_\times, q'_\times)$, $\mathcal{C}(q_\times)$
Output: `Passed` or `Failed`

$\mathcal{C}_- \leftarrow \emptyset$
for $i \in I$ **do**
  if $([q_\times]_i = q_i^{(0)} \wedge [q'_\times]_i \notin q_{0_i})$ **or** $([q_\times]_i \notin \mathcal{F}_i \wedge [q'_\times]_i \in \mathcal{F}_i)$ **then**
    $\alpha \leftarrow s$ **if** $[q'_\times]_i \in \mathcal{F}_i$ **else** $\alpha \leftarrow e$
    if $\mathcal{C}(q_\times)_r[\alpha, i] \neq \emptyset$ **then**
      return `Failed`
    $\mathcal{C}_- \leftarrow \mathcal{C}_- \cup \mathcal{C}(q_\times)_l[\alpha, i]$
$\mathcal{C}(q'_\times) \leftarrow \mathcal{C}(q_\times) \setminus \mathcal{C}_-$
return `Passed`

A path from $q_\times^{(0)}$ to $q'_\times \in \mathcal{F}_\times$ corresponds to an accepting path for each $\mathcal{D}_i$.

Constraint Check: Note that since our time constraints are imposed on the starting and ending times of each task, the satisfaction of these constraints can be determined based on the global state in $\mathcal{D}_\times$. Specifically, for each state $q_\times \in Q_\times$, we denote by $\mathcal{C}(q_\times) \subseteq \mathcal{C}$ the set of constraints that are still effective. Suppose the system aims to move from state $q_\times$ to $q'_\times$ via $\sigma_\times$. We use Algorithm 1 to check whether this transition satisfies the current constraints. If it does, we also compute the constraints $\mathcal{C}(q'_\times)$ that are effective at the new state as shown in Algorithm 1. Furthermore, we use a depth-first search in $\mathcal{D}_\times$ to compute all effective constraints for each state, starting from the initial global state with $\mathcal{C}(q_i^{(0)}) = \mathcal{C}$. We then remove all transitions that violate the time constraints and take the accessible part of the product DFAs. For simplicity, we still denote the remaining pruned structure by $\mathcal{D}_\times$.

Global Search: To capture the mobility of all robots, we construct the synchronous product of each robot R_r, denoted by $R_\times$. We then take the product of $D_\times$ and $R_\times$ based on the state labels. Consequently, finding a minimal makespan task allocation reduces the problem of searching for the shortest path from the initial state to an accepting state in the product of $D_\times$ and $R_\times$. While this approach is effective, its complexity grows exponentially with the number of tasks and robots, making it infeasible for large-scale problems.

V. Max-Plus-Based Auction Approach

In this section, we provide our main algorithm MaxAuc, which follows an auction-based mechanism without explicitly building the entire product space.

A. Basic Ideas

Note that the automaton $\mathcal{D}_\times$ will generate only words that complete the tasks while satisfying the constraints. The main idea here is to incrementally generate a sequence in $\mathcal{D}_\times$ and the allocation plan without explicitly building the product space, as done in the naive approach.

Specifically, we start from the initial state ${q_\times}|^0 = q_\times^{(0)}$ with an empty plan $\kappa = \epsilon$ and an empty word $\rho = \epsilon$. At each stage, we proceed as follows. Suppose we are at state ${q_\times}|^k \in Q_\times$, representing the progress of all tasks. We aim to find a global action $\sigma_\times$ available at ${q_\times}|^k$, i.e., $\delta_\times({q_\times}|^k, \sigma_\times)$ is defined. We choose one such action and determine an allocation plan $\Delta\kappa$ to realize it. We then append $\Delta\kappa$ to the end of κ and $\sigma_\times$ to the end of ρ. The next state ${q_\times}|^{k+1}$ is derived by $\delta_\times({q_\times}|^k, \sigma_\times)$. This process is repeated until we reach a state $q_\times^f \in \mathcal{F}_\times$. The plan κ will then be a feasible solution to Problem 1.

The key question now is how to determine the task allocation for each robot at each instant such that it has the potential to reduce the overall makespan. Here, we use an auction-based approach to heuristically find a good allocation without building and searching through the global space.

- **Auction Items:** In our auction-based approach, the basic item, denoted by τ, is a local decision (a, i) for a task. At each state ${q_\times}|^k$, the auction mechanism determines which robot r needs to execute an auction item τ and plans $\Delta\kappa_r(\tau)$ for robot r to realize this item.
- **Bidding Strategies:** The robot places a bid with the goal of minimizing the makespan of the allocation plan. For each item τ, the bidding strategy consists of two parts: (1) The marginal cost associated with realizing the item, and (2) the max-plus-based task priority as heuristic information. The strategy returns the bid value for τ as well as the allocation plan $\Delta\kappa_r(\tau)$, which outlines how the winning robot will realize a based on the current κ.

B. Auction Pipeline

Note that, since the effective constraints $\mathcal{C}(q_\times)$ are recursively computed in Section IV, we can utilize this information while incrementally constructing the allocation plan. Additionally, with a slight abuse of notation, we write $\{a\}$ directly as a if only a single atomic proposition $a \in \mathcal{AP}$ holds.

Definition 4 (Auction Item): Let $q_\times \in \mathcal{D}_\times$ be the current global task state. An auction item available at $q_\times$ is a local decision $\tau = (a, i) \in \mathcal{AP} \times I$ such that the $q'_i = \delta_{\mathcal{D}_i}([q_\times]_i, a)$ satisfies (i) if $[q_\times]_i = q_i^{(0)}$, then $\mathcal{C}(q_\times)_R[s, i] = \emptyset$; and (ii) if $q'_i \in \mathcal{F}_i$, then $\mathcal{C}(q_\times)_R[e, i] = \emptyset$.

We denote by $\sigma_\times^\tau \in \Sigma_\times$ the action associated to τ as $\sigma_\times^\tau = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ where $\sigma_i = a$, and for all $j \in I \wedge j \neq i, \sigma_j = \epsilon$. At iteration k of the progress described in Section V-A, adding an auction item to ρ refers to appending $\sigma_\times^\tau$ to the end of ρ. We then denote $q'_\times \in Q_\times$, s.t. $\left(q'_\times\right)_i = \delta_i((q_\times)_i, a)$ and $\left(q'_\times\right)_j = (q_\times)_j$ for all $j \neq i$. Since the auction item is defined in such a way that $(q_\times, \sigma_\times, q'_\times)$, $\mathcal{C}(q_\times)$ will always return `Passed`, it is obvious that the concatenation of ρ and $\sigma_\times^\tau$ still be a prefix of an accepted word of the automaton $\mathcal{D}$. For $q_\times$, we denote $AI(q_\times)$ as the set of auction items satisfying def. 4.

The max-plus-based auction approach is outlined in Algorithm 2, where the maximum execution duration for the previous items is $t_g = \max\{k_{\kappa_r} \mid r \in 1, 2, \ldots, n\}$, k_κ

Algorithm 2: Max-Plus-Based Auction Approach

Input: Tasks and constraints $\Phi, \mathcal{C}$

Output: Allocation plan κ

$q_\times \leftarrow q_\times^{(0)}, t_g \leftarrow 0, \kappa_r \leftarrow \epsilon \ \forall r = 1, 2, \ldots, n$

while $q_\times \notin \mathcal{F}_\times$ **do**

  for $\hat{\tau} = (\hat{a}, \hat{i}) \in AI(q_\times)$ **do**

    for $r \in 1, \ldots, n$ **do**

      $b_r(\hat{\tau}), \Delta\kappa_r(\hat{\tau}) \leftarrow \texttt{Bid}(\hat{\tau}, t_g, \kappa_r)$

  $r^*, \tau^* \leftarrow \arg\min_{r,\tau} b_r(\tau)$

  $\kappa_r \leftarrow \kappa_r + \Delta\kappa_{r^*}(\tau^*)$

  $(q_\times)_{i^*} \leftarrow \delta_{i^*}((q_\times)_{i^*}, a^*)$

  $t_g \leftarrow k_{\kappa_{r^*}}$

return $\kappa = \cup_{r=1,2,\ldots,n}\kappa_r$

refers to the length of κ in the form of (4). The function $\texttt{Bid} : \mathcal{T} \times \mathbb{Z} \times (\bar{\Gamma})^\star \to \mathbb{R}_+ \times (\bar{\Gamma})^\star$ represents the bidding strategy of the robots for a specific item under time limit t_g and the previous allocation plan κ_r. It returns the bid value $b_r(\tau)$ and the corresponding allocation plan $\Delta\kappa_r(\tau)$ to fulfill the bid. The detailed construction of `Bid` will be provided later.

Algorithm 2 incrementally generates allocation plans κ_r, for robot $r \in \{1, \ldots, n\}$, where $\gamma_r^t = \epsilon, \ \forall j \in \{1, \ldots, n\}$ and $j \neq r$. The winning robot of each auction round is the robot r^* with the lowest bid value for the auction item $\tau^* = (a^*, i^*)$. The corresponding part $\Delta\kappa_{r^*}(\tau^*)$ is then appended to the end of κ_{r^*}. After repeating until the termination condition, the synchronization of all κ_r is returned.

In detail, the bidding strategy consists of two parts: computing the task priority $\boldsymbol{u} \in [0, 1]^m$ and the marginal cost $MC(\tau, \kappa_r, t_g)$. The bid value is given by:

$$b_r = \omega_m \cdot MC(\tau, \kappa_r, t_g) + \omega_u \cdot u_i, \tag{11}$$

where u_i is the task priority for task φ_i, $\omega_m, \omega_u \geq 0$ are the weights for the marginal cost and task priority, respectively. The allocation plan $\Delta\kappa_r(\tau)$ is derived from the solution of the marginal cost. We will elaborate on the two parts of bidding in the following sections.

C. Computing Marginal Cost of Auction Item

Having the robot with the shortest plan length win the item is beneficial for minimizing the team's makespan. To achieve this, we introduce the marginal cost of bidding, which represents the additional time consumption for robot r to complete the item, considering its previous allocation plan and the time limit t_g.

We assume the existence of a shortest path planner $P : V \times V \to (\bar{\Gamma})^\star$, which provides an allocation plan for a robot to move from v to v' with minimal number of transfers. Let $v_{c,r}$ be the state where robot r moves from $v_{0,r}$ after executing the allocation plan κ_r, and let $\Delta\kappa_r^p(\tau = (a, i)) = P(v_{c,r}, v')$ where v' is a state such that $L(v') = a$. The length of $\Delta\kappa_r^p(\tau)$ is denoted as $k_\Delta(r, \tau)$.

If a robot r doesn't have any actions assigned in the time interval $[k_r, t_g]$, i.e. its allocation plan $\gamma_r^t = \epsilon, \ \forall t \in [k_r + 1, t_g]$, it can start transferring before t_g to reduce makespan. It's still essential for the robot to finish the item it bids after time t_g to ensure that the projection of the plan aligns with the word we selected step by step. We define thus the marginal cost function $MC : \mathcal{T} \times (\bar{\Gamma})^\star \times \mathbb{N} \to \mathbb{N}$ as

$$MC(\tau, \kappa_r, t_g) = \max(\Delta t(r, \tau), 1), \tag{12}$$

where $\Delta t(r, \tau) = k_\Delta(r, \tau) - t_g + k_{\kappa_r}$. Subsequently, we denote $MC(\tau, \kappa_r, t_g)$ as $MC_r(\tau)$. If $\Delta t(r, \tau) < 1$, the robot needs to wait for $1 - \Delta t(r, \tau)$ before starting to fulfill the bid. In this case, we define $\Delta\kappa_r^t(\tau) = \gamma^{k_r+1} \ldots \gamma^{1-\Delta t(r,\tau)}$, where $\gamma_r^t = ((v, v), \perp)$. Otherwise, $\Delta\kappa_r^t(t_s) = \epsilon$.

Finally, the complete allocation plan for bid τ is given by

$$\Delta\kappa_r(\tau) = \Delta\kappa_r^t(\tau)\Delta\kappa_r^p(\tau). \tag{13}$$

D. Computing Max-Plus-Based Task Priority

The max-plus-based task priority will assign a global execution preference to each task, aiming to mitigate the impact of local greediness in each auction that could lead to a solution of large makespan. The task priority is derived by the fastest end time defined as follows.

Definition 5 (Fastest-End Time): For task φ_i, we define the fastest-end time $t_e^-(i) \in \mathbb{N}$ as a time stamp such that $t_e^-(i) = \min\{t_e(i, \kappa) \mid \kappa \text{ s.t. } \kappa \models \Phi \wedge \kappa \models \mathcal{C}\}$.

Intuitively, to reduce the makespan from the global, tasks with an earlier fastest-end time correspond to a lower priority value, which attempts to minimize the difference between the actual end time and the fastest-end time of each task.

In general, explicitly calculating the fastest-end times for all tasks in a multi-robot, multi-task scenario is an NP-hard combinatorial optimization problem. To enhance the efficiency, we propose a max-plus-based method with polynomial complexity to provide a lower approximation for the fastest-end time. To model $\mathcal{C}(q_\times)$, we define task end times as a vector $\boldsymbol{x} = (x_1, \ldots, x_m)$ and the start time vector $\boldsymbol{y} = (y_1, \ldots, y_m)$ is defined by

$$y_i = x_i - d_{\mathcal{D}_i}([q_\times]_i), \tag{14}$$

where $d_{\mathcal{D}_i}(q)$ denotes the shortest distance from q to accepting states in $\mathcal{D}_i$ as introduced in Section II-A. For constraint $\langle(\alpha, \beta), (i, j)\rangle \in \mathcal{C}(q_\times)$, we have

$$x_i - 1_s(\alpha) \cdot d_{\mathcal{D}_i}([q_\times]_i) + 1 \leq x_j - 1_s(\beta) \cdot d_{\mathcal{D}_j}([q_\times]_j), \tag{15}$$

where $1_s(\alpha)$ is 1 if $\alpha = s$, and is 0 if $\alpha = e$. We then reform Eq. (15) into the max-plus algebraic form as

$$(v_{\alpha\beta})_{ij} \otimes x_i \preceq x_j, \tag{16}$$

where the parameter $(v_{\alpha\beta})_{ij}$ is defined as

$$(v_{\alpha\beta})_{ij} = 1_s(\beta) \cdot d_{\mathcal{D}_j}([q_\times]_j) - 1_s(\alpha) \cdot d_{\mathcal{D}_i}([q_\times]_i) + 1. \tag{17}$$

To ensure the starting time $\boldsymbol{y}$ positive, we set

$$\boldsymbol{x} \succeq D = \left(d_{\mathcal{D}_1}([q_\times]_1), \ldots, d_{\mathcal{D}_m}([q_\times]_m)\right). \tag{18}$$

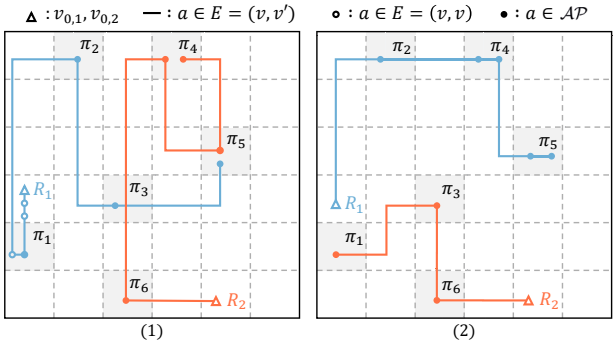

Fig. 1: Illustrative case study, where (1) shows the allocation κ generated without considering the max-plus-based task priority and (2) shows the allocation κ' generated with the max-plus-based task priority.

Also, for each i, j, using the max-plus algebraic operators, we consolidate the constraints into a single parameter that

$$\underline{v}_{i,j} = \bigoplus_{\alpha,\beta \in \{s,e\}} (v_{\alpha\beta})_{ij}, \tag{19}$$

Then Eq. (17) together with Eq. (18) can be reformed into the matrix form

$$\boldsymbol{x} \succeq V \otimes \boldsymbol{x} \oplus D, \tag{20}$$

where the element at the i^{th} row and j^{th} column of the matrix $V \in \mathbb{Z}_{\varepsilon}^{n \times n}$ is $\underline{v}_{i,j}$. According to the theory reviewed in Section II-B, the optimal solution $\boldsymbol{x}^*$ is

$$\boldsymbol{x}^* = \bigoplus_{i \in \mathbb{N}} V^{\otimes i} \otimes D. \tag{21}$$

Note that the complexity of solving Eq. (21) is $O(m^3)$ [27].

After obtaining the fastest end time vector $\boldsymbol{x}^*$, the vector indicator $\boldsymbol{u}$ for the task priorities is defined by $\boldsymbol{u} = (x_1^*, \ldots, x_m^*)/\max(x_1^*, \ldots, x_m^*, 1)$.

VI. Experimental Results

In this section, we first conduct a case study simulation to better illustrate our method. Then we conduct numerical experiments to illustrate the scalability of our method when the size of the problem increases. All experiments are implemented by `Python 3.12` and we use `SPOT` [28] to translate scLTL formulas to DFA.

A. Illustrative Case Study

We consider a team of two robots, R_1 and R_2, operating in a 6×6 grid-world workspace, as shown in Fig. 1. The initial positions of the robots are also indicated as triangles in the figure. Within this workspace, there are six grid cells of particular interest, which are marked in gray. Let $\mathcal{AP} = \{\pi_i \mid i = 1, 2, \ldots, 6\}$ represent the set of atomic propositions, where each π_i corresponds to a specific property or task associated with the gray grid cells. The assignment of these atomic propositions to the grid cells is illustrated in the figure.

We assume that the two robots need to accomplish three tasks $\Phi = \{\varphi_1, \varphi_2, \varphi_3\}$, where

$$\begin{aligned} \varphi_1 &= \Diamond(\pi_2 \wedge \Diamond(\pi_3 \wedge \Diamond\pi_4)), \\ \varphi_2 &= \Diamond(\pi_1 \wedge \Diamond\pi_5), \\ \varphi_3 &= \Diamond(\pi_6 \wedge \Diamond(\pi_4 \wedge \Diamond\pi_5)). \end{aligned} \tag{22}$$

The time constraints of set Φ are given by:

$$\mathcal{C} = \{\langle (s,3), (s,2) \rangle, \langle (e,1), (e,2) \rangle\}.$$

This means (i) task φ_3 should start earlier than task φ_2; and (ii) task φ_1 should finish earlier than task φ_2. Note that there are no specific time constraints between tasks φ_1 and φ_3.

Here, we use the second round of the auction as an example to illustrate our auction process and compare it with the case where the max-plus-based task priority is excluded from the robot's bidding strategy. For convenience, the action set of each robot is denoted by $A = \{\rightarrow, \leftarrow, \uparrow, \downarrow, \circlearrowleft, \pi_1, \ldots, \pi_6\}$. where the arrows $(\rightarrow, \leftarrow, \uparrow, \downarrow)$ represent movement in different directions, and $\circlearrowleft$ denotes the robot staying in the same state.

After the first round of auction, we can obtain that $\kappa_1 = \epsilon$, $\kappa_2 = (\epsilon, (\leftarrow, \perp))(\epsilon, (\leftarrow, \perp))(\epsilon, (\pi_6, 3))$, where $k_{\kappa_1} = 0, k_{\kappa_2} = 3$, so $t_g = 3$. The set of auction items available for the second round of auction is $\{(\pi_1, 2), (\pi_2, 1), (\pi_4, 3)\}$. For Robot 1, it requires 5 actions to complete π_2 from its initial state, specifically $(\uparrow, \uparrow, \uparrow, \rightarrow, \pi_2)$. However, since it isn't assigned actions during time of $0 \sim 3$, the marginal cost for $(\pi_2, 1)$ is $\max(5-3, 1) = 2$. For π_1, Robot 1 only needs 2 actions, so the marginal cost for item $(\pi_1, 2)$ is $\max(2-3, 1) = 1$, ensuring that the second auction's assigned item is fulfilled after the first auction's assigned item. Similarly, Robot 1's marginal cost for $(\pi_4, 3)$ is 4. For Robot 2, the marginal cost for each item is the actual length of its action sequence, as it is assigned an action at exactly t_g. Thus, $MC_2((\pi_1, 2)) = 4$, $MC_2((\pi_2, 1)) = 7$, and $MC_2((\pi_4, 3)) = 7$.

To calculate the task priority, we set $\boldsymbol{x} = (x_1, x_2, x_3)$ as the fastest end times for each task. The start times are $\boldsymbol{y} = (x_1 - 3, x_2 - 2, x_3 - 2)$, as per Eq. (14). After Robot 2 executed π_6 at t_g, the effective constraint now is only $\langle (e,1), (e,2) \rangle$. The optimality problem in max-plus algebra becomes to find the minimum $\boldsymbol{x}$ such that

$$\begin{bmatrix} \varepsilon & \varepsilon & \varepsilon \\ 1 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon \end{bmatrix} \otimes \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \oplus \begin{bmatrix} 3 \\ 2 \\ 2 \end{bmatrix} \preceq \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}. \tag{23}$$

The solution to the above is $\boldsymbol{x} = (3, 4, 2)$, and the task priority is $\boldsymbol{u} = (0.75, 1, 0.5)$ after normalization.

Greedy-Based Auction: If we only consider the marginal cost of item as bids, the lowest bid is $MC_1((\pi_1, 2))$. After the second auction, the allocation plan for Robot 1 becomes:

$$\kappa_1 = ((\circlearrowleft, \perp), \epsilon)((\circlearrowleft, \perp), \epsilon)((\downarrow, \perp), \epsilon)((\pi_1, 2), \epsilon),$$

while κ_2 remains unchanged. Continuing the auction until all tasks are fulfilled, we obtain the allocation plan κ without considering the max-plus-based task priority. The graphical

representation of κ is shown in Fig. 1(1). In the figure, solid circles represent the robot spending one clock cycle to execute proposition $L(v)$, hollow circles represent the robot staying in the same state for a clock cycle, and straight lines represent state transitions. With the clock cycle set to 1 second, the resulting makespan is $M(\kappa) = 20s$.

Max-Plus-Based Auction: When we incorporate the max-plus-based task priority into the bidding strategy, with $\omega_m = 1$ and $\omega_u = 8$, Robot 1 places the lowest bid for $(\pi_2, 1)$. After the second auction, the allocation plan for Robot 1 becomes:

$$\kappa_1' = ((\uparrow, \perp), \epsilon)((\uparrow, \perp), \epsilon)((\uparrow, \perp), \epsilon)((\rightarrow, \perp), \epsilon)((\pi_2, 1), \epsilon),$$

while κ_2' remains unchanged. The allocation plan κ' is illustrated in Fig. 1(2), with a makespan of $M(\kappa') = 14s$. The reason κ' achieves a lower makespan than κ is that Robot 1 is guided by the task priority to first execute task φ_1. This allows π_4 and π_6, which appear in multiple tasks, to be executed consecutively, avoiding the time lost from revisiting the states of π_4 and π_6 multiple times.

B. Numerical Experiments

To demonstrate the scalability of our algorithm and analyze the tradeoff between the resulting makespan $M(\kappa)$ and the planning time T, we conduct numerical experiments on randomly generated systems.

Specifically, we consider an $N \times N$ grid-world workspace with n robots. The task set Φ consists of m scLTL formulae, each in the form of Eq. (22). The atomic propositions are randomly assigned to grids in the workspace, and the time constraints $\mathcal{C}$ are also generated randomly. Cases where no feasible plans exist are excluded from the experiments.

We evaluate the scalability of our algorithm in terms of three key parameters: (i) the size of the workspace N, (ii) the number of robots n, and (iii) the number of tasks m. To minimize the effect of randomness, for each combination of parameters, we repeat the experiment 20 times and compute the average statistics for both the makespan $M(\kappa)$ and the planning time T.

Scalability in Workspace Size: First, we fix the number of robots to $n = 2$ and the number of tasks to $m = 2$. We then increase the size of the workspace from $N = 4$ to $N = 8$. For each randomly generated case, we apply both the naive baseline algorithm, which provides the optimal makespan, and the proposed MaxAuc algorithm.

The average makespan and planning time for each algorithm and system size are shown in Fig. 2. From the results, it is evident that the planning time of the naive baseline algorithm grows exponentially as the workspace size increases. In contrast, the planning time of the MaxAuc algorithm remains very stable with respect to the workspace size. This is because the bidding strategy focuses only on states of $L(v) \in \mathcal{AP}$ within the workspace, making the growth of planning time with respect to size N less significant. Despite this significant reduction in planning time, the makespan performance of the MaxAuc algorithm remains comparable to that of the baseline algorithm. These results illustrate

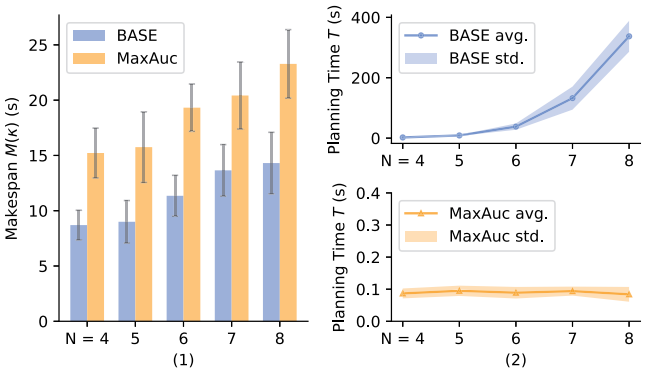

Fig. 2: Scalability test in the size of the workspace.

TABLE I: Scalability test in the number of robots.

n	BASE		MaxAuc	
	$M(\kappa)$ (s)	T (s)	$M(\kappa)$ (s)	T (s)
1	33.6 ± 1.62	1.00 ± 0.02	42.17 ± 4.56	0.25 ± 0.02
2	16.30 ± 1.34	338.73 ± 13.29	28.16 ± 3.20	0.25 ± 0.03
3	–	–	24.19 ± 2.32	0.24 ± 0.05
4	–	–	23.71 ± 2.81	0.27 ± 0.03
5	–	–	19.08 ± 2.07	0.26 ± 0.04

that the proposed MaxAuc algorithm effectively achieves a tradeoff between planning time and makespan performance. It provides a scalable solution for larger workspaces while maintaining near-optimal task completion times, making it suitable for real-time applications in complex environments.

Scalability in Robot Number: Next, we fix the size of the workspace to $N = 4$ and the number of tasks to $m = 5$ and increase the number of robots from $n = 1$ to $n = 5$. The statistical results are shown in Table I. We can observe that the naive baseline algorithm struggles to scale, resulting in timeouts ($T \geq 600$ s) for systems with more than two robots. In contrast, the proposed MaxAuc algorithm effectively handles up to five robots, with the planning time remaining within 0.3 seconds. It is worth noting that, during the auction process, the robot team needs to bid for the same set of auction items independently. Therefore, we adopt a parallel approach to handle this. As the number of robots increases, more computational resources are allocated to enable robots to bid simultaneously. This explains why the planning time of our method does not increase significantly with the number of robots. Without the parallel approach, the planning time would increase linearly with the number of robots, but it remains highly efficient with our method.

Scalability in Task Number: Finally, we fix the size of the workspace to $N = 4$ and the number of robots to $n = 2$ and increase the number of tasks from $m = 2$ to $m = 6$. The statistical results are shown in Table II. We observe that the naive baseline algorithm results in timeouts for systems with more than five tasks. In contrast, the proposed MaxAuc algorithm can still handle six tasks within 0.3 s.

To further summarize the scalability analysis, we consider a more complex workspace of size 10×10. The computation time for various combinations of the number of robots and

TABLE II: Scalability test in the number of tasks.

m	BASE		MaxAuc	
	$M(\kappa)$ (s)	T (s)	$M(\kappa)$ (s)	T (s)
2	8.70 ± 1.34	2.27 ± 0.40	15.22 ± 2.25	0.09 ± 0.02
3	11.75 ± 1.30	20.68 ± 1.98	20.68 ± 2.43	0.16 ± 0.02
4	12.00 ± 0.63	123.41 ± 4.44	21.22 ± 2.75	0.18 ± 0.03
5	16.30 ± 1.34	338.73 ± 13.29	28.71 ± 3.92	0.25 ± 0.04
6	–	–	38.92 ± 3.02	0.30 ± 0.04

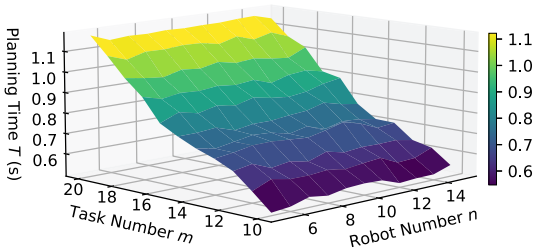

Fig. 3: The planning time with respect to different numbers of robots and tasks for MaxAuc in 10×10 workspace.

tasks is shown in Fig. 3. Our approach can handle 15 robots with 20 tasks in approximately 1 second. Notably, the planning time remains relatively stable with respect to the number of robots, as we employ a parallel method for each robot. Additionally, the planning time scales at a relatively slow rate with the number of tasks.

VII. Conclusion

In this paper, we investigate a new class of temporal logic task allocation problems for multi-robot systems under inter-task-time-order constraints. Inspired by max-plus algebra in handling time constraints, we propose MaxAuc, a max-plus-based auction approach. Experimental results demonstrate that our algorithm is highly scalable with respect to the number of robots and tasks, while maintaining a tolerable performance trade-off. A limitation is that we focus only on single atomic propositions when selecting auction items, excluding cases where multiple robots can contribute to the same task. Although we could extend our approach by expanding the auction item space, this might lead to scalability issues. In future work, we plan to investigate this more general scenario in detail, focusing on developing scalable methods to efficiently allocate tasks among multiple robots while allowing for collaborative contributions.

References

[1] H. Kress-Gazit, M. Lahijanian, and V. Raman, "Synthesis for robots: Guarantees and feedback for robot behavior," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 211–236, 2018.

[2] J. Zhao, K. Zhu, M. Feng, S. Li, and X. Yin, "No-regret path planning for temporal logic tasks in partially-known environments," *The International Journal of Robotics Research*, 2025.

[3] C. Belta and S. Sadraddini, "Formal methods for control synthesis: An optimization perspective," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 115–140, 2019.

[4] X. Yin, B. Gao, and X. Yu, "Formal synthesis of controllers for safety-critical autonomous systems: Developments and challenges," *Annual Reviews in Control*, vol. 57, p. 100940, 2024.

[5] S. L. Smith, J. Tümová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[6] A. Ulusoy, S. L. Smith, and C. Belta, "Optimal multi-robot path planning with ltl constraints: guaranteeing correctness through synchronization," in *Distributed Autonomous Robotic Systems: The 11th International Symposium*, pp. 337–351, Springer, 2014.

[7] B. Cui, K. Zhu, S. Li, and X. Yin, "Security-aware reinforcement learning under linear temporal logic specifications," in *IEEE International Conf. Robotics and Automation*, pp. 12367–12373, 2023.

[8] 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.

[9] X. Yu, X. Yin, S. Li, and Z. Li, "Security-preserving multi-agent coordination for complex temporal logic tasks," *Control Engineering Practice*, vol. 123, p. 105130, 2022.

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

[11] P. Lv, G. Luo, Z. Ma, S. Li, and X. Yin, "Optimal multi-robot path planning for cyclic tasks using Petri nets," *Control Engineering Practice*, vol. 138, p. 105600, 2023.

[12] R. Liu, S. Li, and X. Yin, "Nngtl: Neural network guided optimal temporal logic task planning for mobile robots," in *IEEE International Conf. Robotics and Automation*, pp. 10496–10502, 2024.

[13] Y. E. Sahin, P. Nilsson, and N. Ozay, "Multirobot coordination with counting temporal logics," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1189–1206, 2020.

[14] Z. Liu, M. Guo, W. Bao, and Z. Li, "Fast and adaptive multi-agent planning under collaborative temporal logic tasks via poset products," *Research*, vol. 7, p. 0337, 2024.

[15] B. Cui, F. Huang, S. Li, and X. Yin, "Robust temporal logic task planning for multirobot systems under permanent robot failures," *IEEE Transactions on Control Systems Technology*, 2024.

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

[17] X. Luo and M. M. Zavlanos, "Temporal logic task allocation in heterogeneous multirobot systems," *IEEE Transactions on Robotics*, vol. 38, no. 6, pp. 3602–3621, 2022.

[18] L. Li, Z. Chen, H. Wang, and Z. Kan, "Fast task allocation of heterogeneous robots with temporal logic and inter-task constraints," *IEEE Robotics and Automation Letters*, vol. 8, no. 8, pp. 4991–4998, 2023.

[19] Z. Chen and Z. Kan, "Real-time reactive task allocation and planning of large heterogeneous multi-robot systems with temporal logic specifications," *The International Journal of Robotics Research*, vol. 44, no. 4, pp. 640–664, 2025.

[20] 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.

[21] X. Luo and C. Liu, "Simultaneous task allocation and planning for multi-robots under hierarchical temporal logic specifications," *arXiv preprint arXiv:2401.04003*, 2024.

[22] M. G. Lagoudakis, E. Markakis, D. Kempe, P. Keskinocak, A. J. Kleywegt, S. Koenig, C. A. Tovey, A. Meyerson, and S. Jain, "Auction-based multi-robot routing.," in *Robotics: Science and Systems*, vol. 5, pp. 343–350, Rome, Italy, 2005.

[23] F. Quinton, C. Grand, and C. Lesire, "Market approaches to the multi-robot task allocation problem: a survey," *Journal of Intelligent & Robotic Systems*, vol. 107, no. 2, p. 29, 2023.

[24] S. Kubo and K. Nishinari, "Applications of max-plus algebra to flow shop scheduling problems," *Discrete Applied Mathematics*, vol. 247, pp. 278–293, 2018.

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

[26] F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat, *Synchronization and linearity*, vol. 115. Wiley New York, 1992.

[27] L. Hardouin, B. Cottenceau, Y. Shang, J. Raisch, *et al.*, "Control and state estimation for max-plus linear systems," *Foundations and Trends® in Systems and Control*, vol. 6, no. 1, pp. 1–116, 2018.

[28] A. Duret-Lutz, E. Renault, M. Colange, F. Renkin, A. G. Aisse, P. Schlehuber-Caissier, T. Medioni, A. Martin, J. Dubois, C. Gillard, and H. Lauko, "From Spot 2.0 to Spot 2.10: What's new?," in *34th International Conf. Computer Aided Verification*, pp. 174–187, 2022.