Improving Multi-Robot Behavior Using Learning-Based Receding Horizon Task Allocation

Philipp Schillinger*†, Mathias Bürger* and Dimos V. Dimarogonas†

*Bosch Center for Artificial Intelligence, Renningen, Germany.
Email: {philipp.schillinger, mathias.buerger}@de.bosch.com

†KTH Centre for Autonomous Systems and ACCESS Linnaeus Center
EECS, KTH Royal Institute of Technology, Stockholm, Sweden.
Email: {schillin, dimos}@kth.se

***Abstract*—Planning efficient and coordinated policies for a team of robots is a computationally demanding problem, especially when the system faces uncertainty in the outcome or duration of actions. In practice, approximation methods are usually employed to plan reasonable team policies in an acceptable time. At the same time, many typical robotic tasks include a repetitive pattern. On the one hand, this multiplies the increased cost of inefficient solutions. But on the other hand, it also provides the potential for improving an initial, inefficient solution over time. In this paper, we consider the case that a single mission specification is given to a multi-robot system, describing repetitive tasks which allow the robots to parallelize work. We propose here a decentralized coordination scheme which enables the robots to decompose the full specification, execute distributed tasks, and improve their strategy over time.**

I. Introduction

Behavior planning methods enable the use of robots for increasingly sophisticated tasks. Actions can be planned in a way that the resulting behavior is guaranteed to satisfy a given specification. *Linear Temporal Logic* (LTL) [3, 5] is an established formalism to describe logical specifications including temporal dependencies in a mathematical way. Such a specification provides a solid basis for synthesizing verifiably correct behaviors for the available robots.

We propose here a decentralized receding horizon algorithm that efficiently controls a team of robots from a single LTL specification as input. More precisely, the proposed approach uses *Markov Decision Process* (MDP) [31] models of the robots and an LTL specification to describe safety constraints and tasks that need to be satisfied repeatedly. To handle the computational complexity of planning with probabilistic models, we employ *options* [37] as an abstraction for planning and show how such options can be derived from the given LTL formula. In this setting, an option is defined by a set of terminal states as well as some safety constraints. Policies for the single options can then be found as the solution of a stochastic shortest path problem.

To coordinate the allocation of options to different robots, we employ a simple auctioning scheme [34] where robots greedily bid for the next cheapest option. However, to incorporate a long-term perspective in this allocation process, we include an estimate of the *cost-to-go* in the bids of each robot. This cost-to-go estimate is continuously improved while the repetitive tasks as described by LTL are being followed. We argue that, in the considered multi-robot setting, the cost-to-go can be approximated sufficiently well by learning a value which only depends on the LTL progress while abstracting from the current and future positions of the robots. With this setting, the cost-go-go values can be updated using a standard *Q-learning* approach [36].

The task execution is done in a receding horizon manner. The robots execute the allocation procedure and follow the policy of their respective allocated option until the first robot achieves some progress with respect to the LTL goal. Then, the allocation procedure is repeated to adjust for recent observations. This provides an efficient coordination between the robots and enables to prepare future subgoals in advance by following some options only partially.

This work substantially extends the previous work [34]. First, the notion of options is significantly simplified, as here an option corresponds to exactly one transition in the LTL automaton. Considering the co-to-go approximations, we show convergence even with these simple option definitions. Second, the cost-to-go incorporates a perspective of long-term performance, particularly relevant for repetitive tasks in which costs accumulate and inefficiency is expensive.

A. Related Work

The problem at hand requires to find policies for multiple robots in a way that guarantees satisfaction of a temporally extended, repetitive task which is specified by a temporal logic formula. At the intersection of temporal logics and multi-robot systems, computational complexity is a particular challenge and motivates distributed approaches. For example, [13, 14] plan for each robot individually by assuming locally assigned tasks and then reconfigure these plans online to achieve cooperation. [40] takes a similar approach and decomposes the problem into smaller, finite horizon problems. [28] does not synthesize controllers in advance, but ensure that the continuous behavior of the robots is guaranteed to comply with specifications given in an

assume-guarantee structure [27]. However, these approaches not only assume that an assignment of tasks to robots is known in advance, but also do not consider the efficiency of mission satisfaction. In contrast, [41] forms a centralized product between the robots to optimize repeated satisfaction of a single goal proposition. [35] avoids the expensive product by decomposing the mission into independent tasks, but only covers finite missions. Still, both approaches are centralized and assume deterministic actions.

To find policies in increasingly complex and uncertain environments, temporal logics can be used to guide a reinforcement learning process [22]. For example, [32] demonstrates a temporal difference learning approach to satisfy a given LTL specification. [15, 1] use batch Q-learning to increase the robustness of satisfying a temporal logic goal. Similarly, [21] converts a temporal logic goal into a reward function. [20] defines options [37] to satisfy single atomic propositions in order to reduce the complexity of learning. [26] defines options and uses neural networks to learn low-level controllers as well as high-level option activations. However, these approaches are limited to a single agent and do not consider the effects of multi-agent dependencies.

When extending learning to incorporate multiple agents, the varying policies of other agents are challenging for purely distributed learning approaches [8]. While this effect can be utilized when training a single agent [4], it generally needs to be addressed for finding multi-agent policies. [12] presents with the MAXQ framework a hierarchical approach to multi-agent reinforcement learning. [7] employs a decentralized Monte Carlo Tree Search while agents communicate their policies. [23] proposes that the agents learn a set of policies with full knowledge about the policies of other agents and then estimate which policy is followed by the others. [9] demonstrates a model identification approach while conforming with temporal logic constraints. However, these approaches do not support goals given by a temporal logic formula and often require a learning phase in advance. To formally model the dependencies on other agents instead of learning them, [2] proposes to use the DecPOMDP formalism [6] and presents a way to incorporate options in this framework as opposed to considering an LTL specification. [25] extends the approach by incorporating planning of policies for the explicitly modeled options.

Another efficient method to coordinate the allocation of tasks between multiple robots are decentralized auctions [19, 10]. For example, [42] proposes an auction approach to cooperate in overlapping tasks based on a hierarchical structure. [16] presents sequential single-item auctions as a particularly efficient coordination method. In our previous work [34], we consider dependencies of auction tasks derived from temporal logic specifications. Specifically, efficient concurrent execution of dependent tasks requires attention. [29] discusses three different termination schemes of concurrent action execution and [30] presents an approach for generating concurrent plans in MDPs. [38, 24] model concurrent cooperation of a two-arm manipulator with asynchronous activities. Still, optimizing concurrent execution while considering their interdependency is computationally expensive for an increasing number of agents.

To combine the advantages of both learning coordination and the auctions, we propose in this work a receding horizon scheme. The robots coordinate their next actions by auctioning while learning the desirability of their assignment for the team beyond the horizon to improve over time.

B. Preliminaries

A *Linear Temporal Logic* (LTL) formula ϕ over the propositions $\pi \in \Pi$ with $\pi \in \{\top, \bot\}$ is given inductively by the syntax [3, 5]

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

where ϕ_1, ϕ_2 are itself LTL formulas. $\top$, $\bot$ denote the Boolean constants *true* and *false*, respectively. The operators $\neg$ (negation), $\wedge$ (conjunction), and $\vee$ (disjunction) are Boolean and are extended by the temporal operators $\bigcirc$ (next) and $\mathcal{U}$ (until).

In LTL, the semantics of these operators are defined over sequences of sets of propositions $\sigma\colon \mathbb{N} \to 2^{\Pi}$. We say that a sequence $\sigma(t)$ *satisfies* ϕ at time t as given in the following and denoted by the operator $\vDash$.

- $\sigma(t) \vDash \top$ trivially holds
- $\sigma(t) \vDash \pi$ if proposition $\pi \in \sigma(t)$
- $\sigma(t) \vDash \neg\phi_1$ if not $\sigma(t) \vDash \phi_1$
- $\sigma(t) \vDash \phi_1 \wedge \phi_2$ if both $\sigma(t) \vDash \phi_1$ and $\sigma(t) \vDash \phi_2$
- $\sigma(t) \vDash \phi_1 \vee \phi_2$ if any of $\sigma(t) \vDash \phi_1$ or $\sigma(t) \vDash \phi_2$
- $\sigma(t) \vDash \bigcirc\phi_1$ if next $\sigma(t+1) \vDash \phi_1$
- $\sigma(t) \vDash \phi_1 \,\mathcal{U}\, \phi_2$ if eventually $\exists T \geq t\colon \sigma(T) \vDash \phi_2$ and until then $\forall t' \in [t, T)\colon \sigma(t') \vDash \phi_1$

In addition to the above basic operators, we define the derived operators $\Rightarrow$ (implication) as $\phi_1 \Rightarrow \phi_2 := \neg\phi_1 \vee \phi_2$, $\Diamond$ (eventually) given by $\Diamond\phi_1 := \top \,\mathcal{U}\, \phi_1$, and $\Box$ (always) as $\Box\phi_1 := \neg\Diamond\neg\phi_1$.

It is well-known [3, 5] that an LTL formula ϕ can be translated into a non-deterministic *Büchi Automaton* (BA) defined as follows.

Definition 1 (Büchi Automaton)**.** *A Büchi automaton is a tuple $\mathcal{B} := (Q_\mathcal{B}, Q_{\mathcal{B},0}, \alpha_\mathcal{B}, \delta_\mathcal{B}, F_\mathcal{B})$, consisting of **(1)** a finite set of states $Q_\mathcal{B}$, **(2)** a set of initial states $Q_{\mathcal{B},0} \subset Q_\mathcal{B}$, **(3)** an input alphabet $\alpha_\mathcal{B}$, **(4)** a non-deterministic transition relation $\delta_\mathcal{B} \subseteq Q_\mathcal{B} \times \alpha_\mathcal{B} \times Q_\mathcal{B}$, **(5)** a set of accepting states $F_\mathcal{B} \subset Q_\mathcal{B}$.*

A run $\rho\colon \mathbb{N}_0 \to Q$ is a sequence of states $\rho = q_0 q_1 q_2 \ldots$ and we say that a sequence σ *describes* a run ρ if it holds that $(\rho(i-1), \sigma(i), \rho(i)) \in \delta_\mathcal{B}$ for all $i \in \mathbb{N}$. The construction of a BA $\mathcal{B}$, e.g. [11], ensures that the following relation exists to its corresponding LTL formula ϕ.

Definition 2 (Büchi Acceptance)**.** *A sequence σ is* accepted *by the BA $\mathcal{B}$ constructed from ϕ if and only if there exists a run ρ described by σ such that $\rho(0) \in Q_{\mathcal{B},0}$ and the set of accepting states $F_\mathcal{B}$ is traversed infinitely often. In this case, ρ is called* accepting *and it holds that $\sigma \vDash \phi$.*

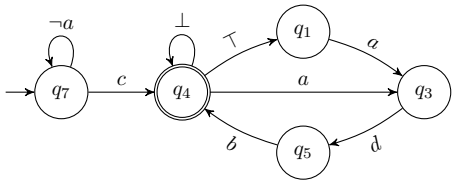

Fig. 1. Example BA for the formula $\phi = \neg a \,\mathcal{U}\, c \wedge \square\big(\lozenge(a \wedge \lozenge b) \wedge \lozenge d\big)$. For a simplified illustration, propositions $\{a, b, c, d\}$ are assumed to be mutually exclusive and infeasible transitions are not drawn. Also, self-transitions which are trivially true are omitted.

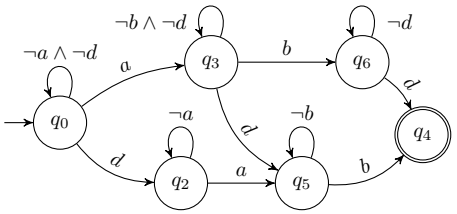

Fig. 2. CFA corresponding to the BA in Fig. 1, i.e., for the formula $\phi_\mathrm{r} = \lozenge(a \wedge \lozenge b) \wedge \lozenge d$.

In particular, a run can be formulated as the concatenation of a finite *prefix* and an infinite *suffix* [3]. We call a prefix *safe* if there is a suffix such that the prefix can be extended to an accepting run. There exists a fragment of LTL called *syntactically co-safe LTL* (scLTL) [17] for which a run ρ is accepting if its prefix is safe for every suffix.

II. Model Construction

LTL is an established formalism to specify complex logical and temporally extended goals for a given system. As a basis for the presented algorithms, we first introduce how to relate such a goal with a probabilistic model of the system suitable for planning, learning, and multi-agent coordination.

A. Mission Formulation

We restrict mission formulations to the fairly general subset of LTL which can be expressed as

$$\phi = \phi_\mathrm{i} \wedge \square \phi_\mathrm{r} \tag{1}$$

where both ϕ_i and ϕ_r are scLTL formulas. ϕ_i captures initial tasks which only have to be completed once. ϕ_r formulates repetitive tasks which the robots need to satisfy repeatedly.

We construct finite automata for ϕ_i and ϕ_r independently. Specifically, we introduce a so-called *Cyclic Finite Automaton* (CFA) as follows in order to resemble the suffix-part of a BA and construct the CFA directly from ϕ_r.

Definition 3 (Cyclic Finite Automaton)**.** *The Cyclic Finite Automaton $\mathcal{C}$ is a tuple $\mathcal{C} = (Q, q_0, \alpha, \delta, F)$, given by **(1)** a finite set of states Q, **(2)** an initial state q_0, **(3)** an input alphabet α, **(4)** a deterministic transition relation $\delta \subseteq Q \times \alpha \times Q$, **(5)** a set of accepting states F.*

When a run ρ described by a sequence σ reaches an accepting state, we say that the CFA *accepts* σ and ρ is reset to the initial state q_0 of the CFA. If ρ is reset multiple times (or infinitely often), we say that σ is accepted multiple times (or infinitely often). An example of a BA and the CFA for the respective ϕ_r is shown in Figures 1 and 2.

The relation between a CFA $\mathcal{C}_\phi$ for ϕ_r and a BA $\mathcal{B}_\phi$ for ϕ with the structure (1) can formally be shown as follows.

Proposition 1 (Equivalent Acceptance)**.** *Let $\sigma = \sigma_\mathrm{p}\sigma_\mathrm{s}$ be a sequence with a finite prefix σ_p and an infinite suffix σ_s. Furthermore, let σ_p be a safe prefix for ϕ and $\sigma_\mathrm{p} \vDash \phi_\mathrm{i}$. Then, σ is accepted by $\mathcal{B}_\phi$ if and only if σ_s is accepted by $\mathcal{C}_\phi$ infinitely often.*

Proof: First, observe that $\sigma_\mathrm{p} \vDash \phi_\mathrm{i}$ if and only if there exists a run $\rho_\mathrm{p}^{\mathcal{B}}$ described by σ_p which reaches an accepting state of $\mathcal{B}_\phi$. For the *only if*-direction, assume that σ is accepted by $\mathcal{B}_\phi$. Then, every consecutive accepting loop in $\mathcal{B}_\phi$ described by the continuation of σ_p fulfills ϕ_r and consequently, describes an accepting run in $\mathcal{C}_\phi$. For the *if*-direction, assume that σ_s is accepted by $\mathcal{C}_\phi$ infinitely often and consider that $\sigma = \sigma_\mathrm{p}\sigma_\mathrm{s}$ would not be accepted by $\mathcal{B}_\phi$. Since $\rho_\mathrm{p}^{\mathcal{B}}$ reaches an accepting state, this would imply that σ_s violates $\square\phi_\mathrm{r}$ eventually and thus, contradict the assumption. Consequently, the equivalence follows. ■

Throughout this paper, we do not use the BA of an LTL mission directly. Instead, we focus for clarity of presentation and without loss of generality on the suffix as represented by a CFA. Nevertheless, note that the prefix of ϕ can be trivially incorporated when considering a different automaton structure during the first iteration and then continuing with the CFA. In particular, the automaton for the first iteration is obtained by following the BA to an accepting state and then resetting to the initial CFA state.

Remark. We make the limitation to (1) for the following technical reason: most common LTL-to-automaton translators reduce the size of a BA as much as possible and remove states which are redundant for acceptance. However, this optimization may exclude some trajectories which are desirable when considering each accepting loop separately. For example, in Figure 1, only the order a, d, b leads back to the accepting state while the order a, b, d is not recognized as an accepting loop until a further occurence of b.

B. System and Problem Definition

We model a robot in the environment as a probabilistic dynamical system. At each iteration $k \in \mathbb{N}$ the system has a state $s_k \in S$ and can select an action a_k from a set of possible actions A. The *transition kernel* $P_{a_k}(s_{k+1}, s_k) = \Pr(s_{k+1}|s_k, a_k)$ is the probability that action a_k in state s_k transitions to the subsequent state s_{k+1}. For each transition, the function $c\colon S \times A \to \mathbb{R}_{>0}$ is an expected immediate cost, i.e., $c(s_k, a_k)$ is the cost incurred when action a_k is chosen in state s_k. Finally, let Π be a set of atomic propositions. A *labeling function* $\lambda\colon S \to 2^\Pi$ assigns to every state a label. In the following, we will refer to the tuple $\mathcal{M} = (S, A, P, c, \Pi, \lambda)$ as a *labeled Markov Decision Process*.

Given an initial state $s_0 \in S$ and a sequence of actions $a_0, a_1, \ldots, a_{K-1}$, the dynamical system will generate a probabilistic trajectory $s_1, s_2, \ldots, s_K$. The corresponding *sequence* σ to this trajectory is then $\sigma = \lambda(s_1), \ldots, \lambda(s_K)$.

For the purpose of this paper, it is most useful to give a particular interpretation to the above dynamics, motivated by robot navigation problems. Specifically, we consider as cost $c(s_k, a_k)$ the duration of an action a_k. Note that this is a slight variation of the standard MDP setting [31], where usually one iteration corresponds to one time instant.

The notion of *time* needs particular attention in a multi-agent setting. For an analysis purpose, we introduce here a continuous, global reference time $t \in \mathbb{R}_{\geq 0}$. Suppose that, at global time t, a robot r is in state s_k^r and initiates action a_k^r. We denote $s^r(\tau) = s_k^r$ for all $\tau \in [t, t + c_k^r)$ where $c_k^r = c(s_k^r, a_k^r)$. That is, in the global time, we assume that a robot is in its originating state until the new state is reached.

A global time allows us to introduce a *team trajectory* $\mathrm{s}(t)$. Consider a team of n robots $R = \{r_1, \ldots, r_n\}$, each modeled by a labeled MDP $\mathcal{M}^{r_i}$ over the same set of atomic propositions Π. We assume throughout this work that the robots' MDPs are independent of each other, that is, we assume the dynamics of the robots are decoupled. The team state at global time t is thus $\mathrm{s}(t) = (s^1(t), \ldots, s^n(t))$. A sequence of time instances $\theta = t_0 t_1 \ldots$ then denotes the times t_k at which the state $\mathrm{s}(t)$ changes, that is, at least one robot reaches a new state. Consequently, a team trajectory is a sequence of states $\mathrm{s}(t_0)\mathrm{s}(t_1)\ldots$ at times t_k. Finally, a *team sequence* is given accordingly by $\sigma = \big(\lambda(s^1(t_0)), \ldots, \lambda(s^n(t_0))\big)\big(\lambda(s^1(t_1)), \ldots, \lambda(s^n(t_1))\big)\ldots$.

We conclude by formulating the objective addressed in this paper based on the above definition of a team sequence. Let $\sigma(t_k)$ be the team sequence from the start t_0 to time instant t_k, and let $\rho(t_k)$ be a run in the CFA $\mathcal{C}$ of the mission ϕ_r described by $\sigma(t_k)$. Further, define a sequence of time instances $\Theta := T_0, T_1, \ldots$ with Θ being a strict subsequence of θ where T_ℓ is the time such that the run $\rho(T_\ell)$ reaches an accepting state $q \in F$ for the ℓ-th time and $T_0 := t_0$.

Problem 1. *Given a team of robots R, where each robot is modeled as a labeled MDP $\mathcal{M}^{r_i}$, and an LTL mission specification ϕ as in Equation* (1)*; minimize the time between repeated visits of an accepting state, given by*

$$\Theta_\Delta(\ell) = T(\ell) - T(\ell - 1). \tag{2}$$

C. Option Definition

To achieve the above objective in a computationally tractable way, we introduce a layer of abstraction which allows to plan specific actions for the robots individually on the lower layer while coordinating their progress towards goal satisfaction on the higher layer. Instead of forming a product between the MDP and an abstract automaton like the CFA, as for example done by [18], we follow the idea of *options* defined over an MDP [37].

A policy $\pi\colon S \to A$ for an MDP $\mathcal{M}$ defines a mapping from a state $s \in S$ to a desirable action $a \in A$ and consequently, produces a sequence of actions $a_0, a_1, \ldots, a_{K-1}$ given an initial state, as discussed before. We call a policy *safe* if it guarantees that a given Boolean constraint φ_c is never violated and a Boolean goal φ_g is satisfied eventually.

In this paper, we are specifically interested in condition pairs $(\varphi_\mathrm{c}, \varphi_\mathrm{g})$ which reflect transitions in the CFA. For a transition from q to q', the condition φ_c is given by the self-transition at q to guarantee that q can remain the active state until the transition has been completed, i.e., until the policy terminates. Accordingly, the goal φ_g is given by the respective condition to transition to q'.

We call a transition in the CFA *admissible* in the case that it admits a policy which is safe with probability 1. We assume that for every state q there exists at least one admissible transition and do not consider non-admissible transitions in the following. For an admissible transition, a safe policy to the states in $\mathcal{M}$ which satisfy φ_g can be constructed by removing all actions from $\mathcal{M}$ which have a non-zero probability of ending in a state that violates φ_c.

Starting at a state $s_0 \in S$ with the safe policy π, a probabilistic duration can be stated as

$$d(s_0) = \sum_{k=0}^{K-1} c(s_k, \pi(s_k)) \tag{3}$$

with the final state s_K denoting a state which satisfies the goal condition φ_g. To summarize, we define one *option* for each transition in the CFA as follows.

Definition 4 (Option)**.** *An option is given by the tuple $o = (q, q', \beta, \pi, d)$ where **(1)** $q \in Q$ is the originating state in the CFA with a self-transition condition φ_c, **(2)** $q' \in Q$ is the goal state in the CFA to which a transition is permitted when a condition φ_g is satisfied, **(3)** $\beta \subseteq S$ is the termination set of the option, given by $\beta = \{s \in S\colon \lambda(s) \vDash \varphi_\mathrm{g}\}$, **(4)** π is a safe policy ensuring that β will be reached and φ_c holds until then, **(5)** $d\colon S \to \mathbb{R}_{\geq 0}$ is a random function, mapping an initial state distribution over S to the expected duration of this option as given by Equation* (3)*.*

III. Task Coordination

Based on the above definitions, we now describe the distributed option allocation approach, synchronously followed by all robots, to coordinate the subsequent asynchronous execution. The idea is similar to [34] and extends the finite *SSO auctions* presented there. In particular, we propose a sequential auctioning algorithm called *Cyclic Single-Option Auctions* (CSO), as summarized by Algorithm 1. Each robot plans a policy for each option and submits bids which incorporate both the expected option duration and their approximated cost-to-go at the target state of the option. In the next auction round, the robots then consider the updated CFA state and allocate subsequent options until all robots are assigned at least one option.

Let t_0 be the time instant at which the allocation procedure starts. Each robot is in a state $s^r(t_0)$ of its MDP $\mathcal{M}^r$ and the whole team is in a joint CFA state $q(t_0)$, i.e., the initial system state is $\big(q(t_0), s^1(t_0), \ldots, s^n(t_0)\big)$. Throughout subsequent auction rounds of the algorithm, the robots track a predicted CFA state q_c to anticipate completion of options which have been allocated in previous rounds. Furthermore, the robots predict future MDP states as

Algorithm 1 Cyclic Single-Option Auctions (CSO)

Model: Robot MDP $\mathcal{M}$, CFA $\mathcal{C}$, team of robots R
Input: CFA state q_c, robot state s_0
Output: Next option o for the executing robot ρ
Notation Remarks:
$\rho \in R$ – the robot which executes this algorithm
D_r – total duration of options assigned to robot $r \in R$

1: $\widehat{s}(s_0) \leftarrow 1$; $\widehat{s}(s) \leftarrow 0, \forall s \neq s_0$
2: $D_r \leftarrow 0, \forall r \in R$
3: **while** $\exists r \in R : D_r = 0$ **do**
▶ Evaluate available options, c.f. Fig. 3
4: **for all** $q_i \in Q$: $(q_c, \cdot, q_i) \in \delta$ **do**
5: $o_i \leftarrow constructOption(q_c, q_i)$ ▷ Def. 4
6: $\Delta_i^\rho \leftarrow \max\big(\bigcup_{r\neq\rho}\{D_r\} \cup \{D_\rho + \bar{d}^\rho_{o_i}\}\big)$
▶ Select best assignment of next task
7: $B \leftarrow syncBids\big(\{b_i^\rho := \Delta_i^\rho + \mathbf{V}(q_i), \forall i\}\big)$ ▷ Eq. (4)
8: $r^*, i^* \leftarrow \mathrm{argmin}(B)$
▶ Predict state after executing the assigned task
9: $q_c \leftarrow q_{i^*}$; $D_{r^*} \leftarrow \Delta^{r^*}_{i^*}$
10: **if** $r^* = \rho$ **then**
11: **if** o not set **then** $o \leftarrow o_i$
12: $\widehat{s} \leftarrow updateState(\widehat{s})$
13: **return** o

resulting from assigned options during the procedure. This prediction is denoted by $\widehat{s}\colon S \to [0,1]$ with $\sum_{s\in S}\widehat{s}(s) = 1$ and is initialized with $\widehat{s}(s(t_0)) := 1$.

In each round of the auctioning process, each robot r constructs an option o_i for each q_i to which a transition from q_c exists and estimates the expected duration $\bar{d}^r_{o_i} := \mathbb{E}_{\widehat{s}}\left[d^r_{o_i}(s)\right]$ over the state estimate $\widehat{s}^r$. Figure 3 illustrates the different options which form the bids for an auction round. To calculate the corresponding bid, for example for option o_2^1, a safe policy π_2^1 is planned for the constraints φ_{11} and the goal condition φ_{12}. This policy is then used along with the state estimate $\widehat{s}^1$ to calculate the option duration $\bar{d}_2^1$.

To reflect potential dependencies on previous options, we follow the idea of [34, Lem. 4] and calculate an expected total duration Δ_i^r. In the above example, the duration Δ_2^1 after which option o_2^1 is expected to terminate is given by $\Delta_2^1 = \max\{D_1 + \bar{d}_2^1, D_2\}$ with D_r denoting the duration of all options already assigned to robot r. This process of determining the expected durations Δ_i^r is done in parallel by each robot r for all of the target states q_i.

To select the winning bid, we consider here an additional parameter $\mathbf{V}\colon Q \to \mathbb{R}_{\geq 0}$ that will later on allow us to consider the long-term performance. In particular, we associate to each CFA state q a single $\mathbf{V}(q)$ to estimate how preferable it is to transition to the respective q. Further details on $\mathbf{V}$ and how the value is determined is discussed in Section IV.

Consequently, the bid of robot r for target q_i is given by

$$b_i^r = \Delta_i^r + \mathbf{V}(q_i) \tag{4}$$

where q_i denotes the logic end state of an option. The bids b_i^r are then communicated between the robots, as

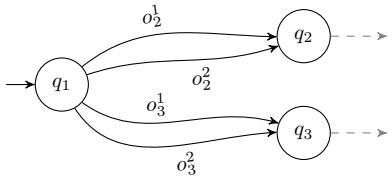

Fig. 3. Illustration of the considered options with o_i^r denoting the option by robot r to reach q_i, here for two robots and two target states.

denoted by the $syncBids(\cdot)$ directive, and a winning option is determined. Finally, the anticipated team progress q_c is updated and the assigned robot updates the physical state in which the assigned option is expected to terminate, denoted by $updateState(\cdot)$. Note that this anticipation of q_c requires for consistency during execution that only the first assigned option is allowed to terminate, while other robots can only "prepare" by following their options up to a certain extent. However, a discussion thereof is deferred to Section V.

Furthermore, note that reaching an accepting state does not terminate Algorithm 1. Instead, the CFA is reset to its initial state and is traversed again as discussed in Proposition 1. The algorithm terminates only if all robots are assigned at least one option. The following result shows that this happens always after a finite number of iterations.

Theorem 1 (Termination)**.** *Under the assumption that (1) the duration of an option is bounded from below by some $d_{\min} > 0$, (2) for every robot there exists at least one admissible option, and (3) the value function $\mathbf{V}$ is nonnegative and finite, then, after a finite number of iterations of Algorithm 1, $D_r > 0$ for all $r \in R$ and the algorithm terminates.*

Proof: The total cost $D = \sum_{r\in R} D_r$, given by the sum of the costs of all robots, increases by a minimum amount during each iteration as given by $d_{\min}$. A robot ρ with $D_\rho = 0$ will then have the winning option after a finite number of iterations of Algorithm 1 if there exists an option o_i^ρ of this robot with finite duration $\bar{d}^\rho_{o_i}$ and finite value $\mathbf{V}(q_i)$ of the target state q_i. Consequently, $D_r > 0$ for all r and Algorithm 1 terminates as given by line 3. ■

Assumptions (1) and (2) are clearly natural in a physical system and do not limit practical applicability of the algorithm. Assumption (3) is given by the update design of $\mathbf{V}$ as discussed in the following section.

IV. Cost-to-Go Approximation

For the auction process in Algorithm 1, each agent computes for all its available options the duration d_i^r and the cost-to-go estimate $\mathbf{V}(q_i)$ of target state q_i. Specifically, d_i^r is the expected duration of the policy execution starting at the current robot state s^r to a termination state of the respective option. Due to the stochasticity of the setting, subsequent physical states are generally a random variable, but the option terminates only in one specific logic state. In fact, even among multiple robots, only one specific next team CFA state q' follows from the parallel option execution of the robots. While this aspect is discussed in

detail in Section V, recall that Algorithm 1 sequentially assigns options to the robots while updating an anticipated progress. As such, only the option which is assigned first can cause a CFA transition upon termination.

Consequently, one can consider only the option which terminates first when approximating the cost-to-go $\mathbf{V}$ and thus, study single option execution rather than a parallel execution of options. The other options can instead be seen as changing the initial state of the subsequent options and only become relevant after future executions of Algorithm 1.

Following this idea, the randomness of an option duration d_i^r can then be understood as not only being a result of stochastic dynamics, but also according to an unknown underlying distribution regarding the initial state of r when the option starts. Here, this distribution is reflected by observed starting states during repeated executions of the mission and we use $\mathbf{V}$ in the following to estimate the long-term cost-to-go only over the logic states Q rather than over the full team product state space $Q \times S^1 \times \ldots \times S^n$, especially since the latter becomes intractable for large n.

In this setting, the problem reduces to a shortest path problem with random cost through the CFA. The duration of completing one transition in the CFA is the duration of an option with the expectation $\bar{d}_o = \mathbb{E}_s\left[d_o^r(s^r)\right]$. The Bellman equation for this shortest path problem is

$$\mathbf{V}^*(q) = \min_{o \in O(q)} \Big(\bar{d}_o + \mathbf{V}^*(q') \Big) \tag{5}$$

with boundary conditions $\mathbf{V}^*(q) = 0$ for all $q \in F$ and q' being the logic goal state of option $o \in O(q)$ where $O(q)$ denotes the set of possible options at CFA state q.

In our setting, the durations $\bar{d}_o$ are unknown and $\mathbf{V}^*$ can only be approximated by observing d_o^r from samples obtained online. In particular, we obtain samples for choosing a particular option at a logic state. Accordingly, we update a Q-function [36] given by $\mathbf{Q}\colon Q \times O \to \mathbb{R}_{\geq 0}$, which reflects the approximated cost-to-go when selecting an option o in state q. Then, an estimate $\mathbf{V}_k$ of the cost-to-go $\mathbf{V}^*$ at some iteration k can be derived from $\mathbf{Q}_k$ by

$$\mathbf{V}_k(q) = \min_{o \in O(q)} \mathbf{Q}_k(q, o) \tag{6}$$

for all CFA states q and options $o \in O(q)$ available at q. Note that we do not directly use $\mathbf{Q}_k(q, o)$ during the assignment in Algorithm 1 since the expected durations for the respective options are calculated explicitly.

Upon option completion, one duration sample $d_k \sim d_o^r$ is observed and a temporal difference update is performed as

$$\begin{aligned} \mathbf{Q}_{k+1}(q, o) = \mathbf{Q}_k(q, o) \\ + \alpha_k \Big[d_k + \mathbf{V}_k(q'_o) - \mathbf{Q}_k(q, o) \Big] \end{aligned} \tag{7}$$

with a nonnegative step-size parameter α_k that decreases for larger k. If $q \in F$, we set $\mathbf{Q}(q, o) := 0$ for all options o. The following can then be established [39, Lem. 9, Thm. 4].

Proposition 2 (Convergence)**.** *Suppose that $\mathbf{Q}_0(q, o) \geq 0$ for all q and o, $\mathbf{Q}_0(q, o) = 0$ for all o where $q \in F$, and $d_o^r(s) \geq 0$. Then, the sequence of iterates $\{\mathbf{Q}_k\}$ is bounded and $\mathbf{Q}_k(q, o)$ converges with probability 1 such that $\mathbf{V}^*(q) = \min_{o \in O(q)} \mathbf{Q}(q, o)$ for all q.*

Algorithm 2 Receding Horizon Concurrent Execution

Input: CFA $\mathcal{C}$, physical state s^r

$q_1 \leftarrow q_0$; $t_1 \leftarrow getCurrentTime()$
for $k \in 1, 2, \ldots$ **do**
  $o \leftarrow CSO(q_k, s^r)$ ▷ Alg. 1
  $\tilde{\pi} \leftarrow addWait(\pi)$
▶ Execute policy until termination set is reached
  while not interrupted **do**
    $s^r \sim P_{\tilde{\pi}}(\cdot, s^r)$ ▷ Follow policy
    if $s^r \in \beta$ **then**
      $sendInterrupt()$
      $q_{k+1} \leftarrow q'$ ▷ Team progress update
▶ Update cost-to-go estimate for all robots
      $t_{k+1} \leftarrow getCurrentTime()$
      $d_k \leftarrow t_{k+1} - t_k$
      $\mathbf{V}_k(q_k) \leftarrow update(d_k, o, q_k)$ ▷ Eq. (6), (7)

Please note that it is natural to assume in our setting that all durations are nonnegative and that the cost-to-go is zero for accepting states. In particular, a minimal duration $d_{\min} > 0$ can be assumed in a physical system like ours.

V. Receding Horizon Concurrent Execution

While the system is running, we propose to use the auction algorithm and the cost-to-go estimate in a receding horizon fashion. The robots follow the execution scheme described in Algorithm 2. At each iteration k, the robots observe the joint CFA state q_k as well as their own physical state s^r. Starting from these states, the robots perform the CSO algorithm (Alg. 1) to determine the next option.

To ensure consistency as noted in Section III, meaning that only the option assigned first is allowed to terminate and that no option which assumes an anticipated progress violates the LTL specification in the current state, the policy of each robot has to be slightly adjusted. This is done by adding so-called *wait* actions in the following way. For every state-action pair, it is checked whether one of its target states violates the self-loop condition φ_k of the current CFA state q_k. If so, the respective action is substituted by a wait action, which ensures that the robot stays in its respective state and does not violate φ_k. As a desired side-effect resulting from the determinism of the CFA, this guarantees that only the first option can terminate and consequently, adds robustness regarding communication delay. We denote this adjustment process $addWait(\cdot)$, which maps a policy π to a policy $\tilde{\pi}$ containing only actions not violating the condition φ_k.

Every robot then executes its option by following the policy $\tilde{\pi}$ as illustrated in Figure 4. Note that this applied interruption scheme corresponds to T_{any} in the formalism of [29]. When the first robot reaches a termination state $s \in \beta$, all other robots are sent an interruption signal, denoted by $sendInterrupt()$, to stop their execution. At the same time,

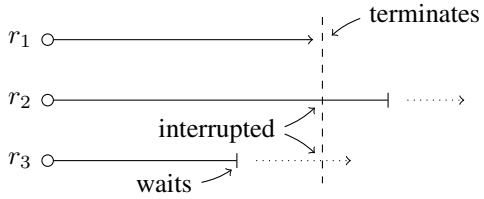

Fig. 4. Option interruption scheme for time progressing from left to right. Only the first option is allowed to terminate and will interrupt all others. The robots do not communicate before the interrupt signal and subsequently, option allocation is updated by following again the CSO algorithm.

the team CFA state of the next iteration q_{k+1} is set to the goal state q' of the respective option and is synchronized between the robots. Recall that in the CFA, the state is reset to the initial state whenever an accepting state is reached.

The value function $\mathbf{V}_k(q_k)$ is updated according to Equations (6) and (7) where the duration d_k is determined by measuring the time required for performing iteration k. For simplicity, we assume here that $\mathbf{V}$ and $\mathbf{Q}$ are updated by the robot which terminates its option. However, in practice, each robot can update these values based on the received interrupt signals without further communication.

Following the receding horizon scheme, the task coordination procedure (Alg. 1) is executed again after an interrupt and the next options are selected. This repeated coordination allows the robot to account for the stochastic dynamics and adjust their task allocation whenever reasonable.

Algorithm 2 produces a trajectory over team states $\mathrm{s}(t)$ and thus, a team sequence σ of the labels of states $\mathrm{s}(t)$ for all t while the robots are operating. Indeed, the generated team sequence satisfies the specified LTL mission formulation ϕ.

Theorem 2 (Correctness)**.** *From any initial team state $(s^1, \ldots, s^n)$ that does not violate the initial conditions of ϕ, Alg. 2 generates a team sequence which satisfies ϕ.*

Proof: By contradiction, suppose that an accepting state would not be reached. Then, observations $d_o^r(s) \geq d_{\min} > 0$ would update $\mathbf{Q}_k$ infinitely often by a positive increment, resulting in an unbounded $\mathbf{Q}_k$ and contradicting Proposition 2. It thus follows from Proposition 2 that an accepting state $q \in F$ is reached eventually and thus, acceptance of the sequence follows from Proposition 1. ∎

Furthermore, the team sequence σ is used to measure the performance. In particular, $\Theta_\Delta(\ell)$, as introduced in Equation (2), denotes the time of one repeated satisfaction of ϕ and our objective is to decrease $\Theta_\Delta(\ell)$, i.e., satisfy ϕ as often as possible in a certain time window.

In this context, it is most reasonable to discuss the expected satisfaction time $\mathbb{E}[\Theta_\Delta(\ell)]$ instead of particular realizations of $\Theta_\Delta(\ell)$ due to the uncertainty of the system. Specifically, the expectation is to be understood as

$$\mathbb{E}[\Theta_\Delta(\ell)] := \lim_{L\to\infty} \frac{1}{(L-\ell)} \sum_{l=\ell}^{L} \Theta_\Delta(l) \tag{8}$$

including uncertainty of option durations resulting from the underlying MDPs as well as uncertainty regarding the initial physical state of one iteration as observed during execution.

We now consider the long-term performance of the system with respect to $\mathbb{E}[\Theta_\Delta(\ell)]$. In particular, let ℓ be sufficiently large and assume convergence, as follows from Proposition 2, such that $\mathbf{V}$ approximately fulfills the Bellman equation (5), i.e., $\mathbf{V}(q) \approx \mathbf{V}^*(q)$ for all q. In particular, it then holds that

$$\mathbb{E}[\Theta_\Delta(\ell)] \approx \mathbf{V}(q_0) \tag{9}$$

that is, the expected satisfaction time is approximated by the cost-to-go at the initial state q_0.

More generally, consider in iteration ℓ some intermediate state q and let τ denote the time required for reaching q. The expected satisfaction time is then given by

$$\mathbb{E}[\Theta_\Delta(\ell)] \approx \tau + \bar{d}_o + \mathbf{V}(q') \tag{10}$$

where o denotes the option selected at state q and q' denotes the option's target state. In particular, recall from Equation (4) that Algorithm 1 chooses

$$o = \operatorname{argmin}\{\bar{d}_o + \mathbf{V}(q')\} \tag{11}$$

as the first option, i.e., with $D_r = 0$ for all r. Then, for any other option choice $\widetilde{o}$ at q, it immediately follows that

$$\bar{d}_o + \mathbf{V}(q') \leq \bar{d}_{\widetilde{o}} + \mathbf{V}(\widetilde{q}'). \tag{12}$$

In other words, convergence of $\mathbf{V}$ to the expected cost-to-go establishes that

$$\mathbb{E}\big[\Theta_\Delta(\ell)\big] \lesssim \mathbb{E}\big[\widetilde{\Theta}_\Delta(\ell)\big] \tag{13}$$

holds for all $\widetilde{\Theta}_\Delta(\ell)$ resulting from other option choices $\widetilde{o}$ and thus, Algorithm 1 selects the approximately best option.

Please note that, as discussed above, convergence has to be understood as convergence to a locally optimal cost-to-go, given the observed physical distribution of the robots. Nevertheless, the distribution is usually becoming more uniform when the number of robots in the system is increased.

VI. Evaluation

We implemented the presented approach as a behavior control framework in ROS, using *Spot*[1] [11] for LTL translation and *FlexBE*[2] [33] for action implementation. To demonstrate the ability of our proposed approach to improve over time, we show the following case study scenario[3] inspired by transportation tasks. The environment is given by the grid map illustrated in Figure 5, containing a set of target locations indicated by different colors and three robots. The following goal specification is given to the system:

$$\phi = \square\lozenge\big(red \wedge \lozenge blue \vee yellow \wedge \lozenge green\big).$$

This mission can either be fulfilled by first delivering to red and then to $blue$ or first to $yellow$ and then to $green$. Objects can be picked up at the locations $p1$ and $p2$. Furthermore, a robot can be damaged with probability 0.1 when performing

[1] http://spot.lrde.epita.fr [2] http://flexbe.github.io

[3] A video showing the case study is provided as media attachment.

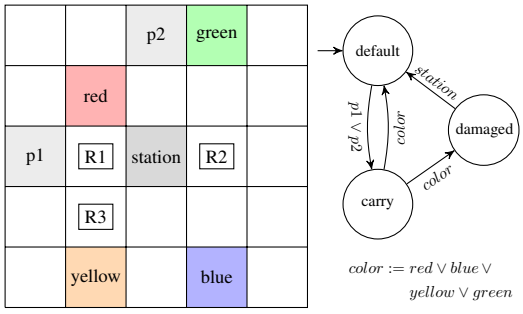

Fig. 5. Case study environment with robots R1, R2, and R3. Shown right are the states in which a robot can be during transportation, including transition conditions. The considered MDP is the product of both parts.

a delivery and while being damaged, no new object can be picked up. A damaged robot can be repaired at $station$.

We observe the following when comparing our approach ("learning") with a non-adaptive version ("static"), obtained by modifying the bid in Equation (4) to not include $\mathbf{V}(q_i)$. In the beginning, both approaches fulfill the mission by choosing red/$blue$ and "static" never changes this behavior. However, "learning" already chooses $yellow$/$green$ in the third iteration and sticks to this variant as the estimated completion time after delivering to $yellow$ is well below the one previously observed after delivering to red. This difference can especially be seen in Figure 6, which illustrates the grid occupancy during the experiments.

After seven repeated satisfactions, the robot which previously delivered to $green$ gets damaged. In the following re-allocation phase, a second robot helps with delivery to $green$, but still, this event results in a significantly increased time to satisfy the mission for the eighth iteration. As a consequence, the red/$blue$ choice is explored again, but still yields a higher duration. Finally, the robots converge to the $yellow$/$green$ behavior to achieve a lower average completion time and also another disturbance during a later iteration does not change this anymore.

Figure 7 shows the measured completion times for repeated satisfaction of the mission. Note that these times vary due to the different durations of the navigation actions, as well as specifically due to the varying relative timings caused by the emergent behavior of two robots following the more time consuming task in parallel. Nevertheless, the expected completion time $\mathbb{E}\left[\Theta_\Delta(\ell)\right]$ as defined in Equation (8) is indeed lower in the learning case and here converges to the optimal time. For an L bounded by the number of repeated simulations, we get an average $\mathbb{E}\left[\Theta^{\mathrm{s}}_\Delta(\ell)\right] = 50.8$ seconds

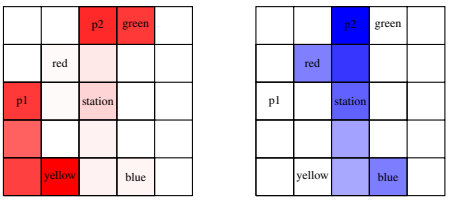

Fig. 6. Grid occupancy during execution of the learning case (red) and the static case (blue). A higher saturation indicates a longer total duration spent by any robot in the respective cell.

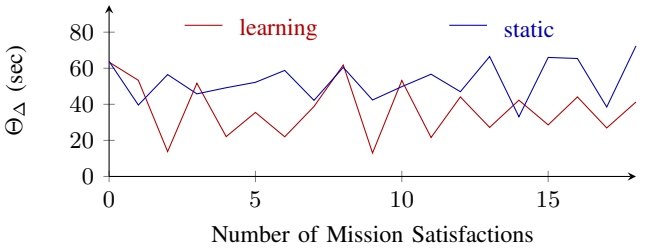

Fig. 7. Mission satisfaction times (in seconds) in the two different cases.

for "static" and $\mathbb{E}\left[\Theta^{\mathrm{l}}_\Delta(\ell)\right] = 34.0$ seconds for "learning".

Figure 8 shows the cost-to-go approximations $\mathbf{V}$ for CFA states q_3 ($yellow$ has been served) and q_2 (red has been served). Initially, $\mathbf{V}(q_i) = 0, \forall q_i$ and updates are performed online after each task progress (see Alg. 2). The increase in the estimate in iteration eight is particularly visible here. In addition, $\mathbf{V}(q_2)$ as approximated in the static case is visualized in Figure 8. Since the team sequence in the static case describes repeated CFA runs via q_2, the resulting value function provides a useful comparison to the learning case.

VII. Conclusions

We presented a framework which enables to synthesize multi-robot policies from a single LTL specification that describes repetitive tasks. Instead of planning one fixed policy, we proposed a receding horizon execution in which the task allocation can be revised to adjust for uncertainty in the environment. In particular, allocation is decided by following a decentralized coordination scheme based on auctions, considering not only the immediate task costs, but also an estimate of the long-term desirability of a particular assignment. This estimate is improved over time while the tasks are executed.

With LTL as a general formalism to describe a wide range of tasks, the presented results are applicable to numerous systems which can be modeled as MDPs, specifically in the area of logistics, factory automation, and service robotics.

This work was supported by the EU H2020 Research and Innovation Programme under GA No. 731869 (Co4Robots). The third author is supported by the H2020 ERC Starting Grant BUCOPHSYS, the Swedish Research Council (VR), the Swedish Foundation for Strategic Research (SSF), and the Knut och Alice Wallenberg Foundation (KAW).

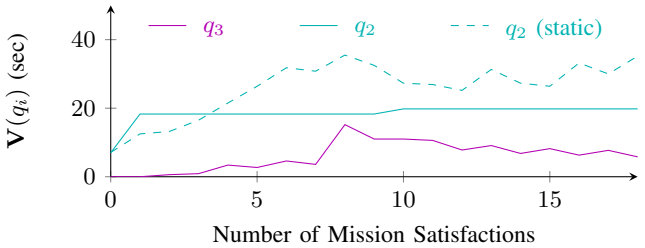

Fig. 8. Value function (expected cost-to-go) of CFA states in the "learning" case. Shown dashed is the value function of q_2 as approximated in the "static" case although it is not being considered for assignment there.

References

[1] Derya Aksaray, Austin Jones, Zhaodan Kong, Mac Schwager, and Calin Belta. Q-learning for robust satisfaction of signal temporal logic specifications. In *Conference on Decision and Control (CDC)*, pages 6565–6570. IEEE, 2016.

[2] Christopher Amato, George Konidaris, Ariel Anders, Gabriel Cruz, Jonathan P How, and Leslie P Kaelbling. Policy search for multi-robot coordination under uncertainty. *The International Journal of Robotics Research*, 35(14):1760–1778, 2016.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press Cambridge, 2008.

[4] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2017.

[5] Calin Belta, Boyan Yordanov, and Ebru Aydin Gol. *Formal Methods for Discrete-Time Dynamical Systems*, volume 89. Springer, 2017.

[6] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.

[7] Graeme Best, Oliver M Cliff, Timothy Patten, Ramgopal R Mettu, and Robert Fitch. Decentralised Monte Carlo tree search for active perception. In *Proc. of WAFR*, 2016.

[8] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews, 38 (2)*, 2008.

[9] Sandeep P Chinchali, Scott C Livingston, Marco Pavone, and Joel W Burdick. Simultaneous model identification and task satisfaction in the presence of temporal logic constraints. In *International Conference on Robotics and Automation (ICRA)*, pages 3682–3689. IEEE, 2016.

[10] Han-Lim Choi, Luc Brunet, and Jonathan P How. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 25(4):912–926, 2009.

[11] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Springer, October 2016.

[12] Mohammad Ghavamzadeh, Sridhar Mahadevan, and Rajbala Makar. Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 13(2):197–229, 2006.

[13] Meng Guo and Dimos V Dimarogonas. Multi-agent plan reconfiguration under local LTL specifications. *The International Journal of Robotics Research*, 34(2):218–235, 2015.

[14] Meng Guo and Dimos V Dimarogonas. Task and Motion Coordination for Heterogeneous Multiagent Systems With Loosely Coupled Local Tasks. *IEEE Transactions on Automation Science and Engineering*, 14(2):797–808, 2017.

[15] Austin Jones, Derya Aksaray, Zhaodan Kong, Mac Schwager, and Calin Belta. Robust Satisfaction of Temporal Logic Specifications via Reinforcement Learning. *arXiv preprint arXiv:1510.06460*, 2015.

[16] Sven Koenig, C Tovey, M Lagoudakis, V Markakis, David Kempe, Pinar Keskinocak, A Kleywegt, Adam Meyerson, and Sonal Jain. The power of sequential single-item auctions for agent coordination. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1625, 2006.

[17] Orna Kupferman and Moshe Y Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3): 291–314, 2001.

[18] Bruno Lacerda, David Parker, and Nick Hawes. Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1511–1516. IEEE, 2014.

[19] Michail G Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton J Kleywegt, Sven Koenig, Craig A Tovey, Adam Meyerson, and Sonal Jain. Auction-Based Multi-Robot Routing. In *Robotics: Science and Systems*, volume 5, pages 343–350, 2005.

[20] Xiao Li and Calin Belta. A Hierarchical Reinforcement Learning Method for Persistent Time-Sensitive Tasks. *arXiv preprint arXiv:1606.06355*, 2016.

[21] Xiao Li, Cristian-Ioan Vasile, and Calin Belta. Reinforcement Learning With Temporal Logic Rewards. *arXiv preprint arXiv:1612.03471*, 2016.

[22] Michael L Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-Independent Task Specifications via GLTL. *arXiv preprint arXiv:1704.04341*, 2017.

[23] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *arXiv preprint arXiv:1706.02275*, 2017.

[24] Thibaut Munzer, Marc Toussaint, and Manuel Lopes. Efficient behavior learning in human–robot collaboration. *Autonomous Robots*, pages 1–13, 2017.

[25] Shayegan Omidshafiei, Ali-Akbar Agha-Mohammadi, Christopher Amato, Shih-Yuan Liu, Jonathan P How, and John Vian. Decentralized control of multi-robot partially observable Markov decision processes using belief space macro-actions. *The International Journal of Robotics Research*, 36(2):231–258, 2017.

[26] Chris Paxton, Vasumathi Raman, Gregory D Hager, and Marin Kobilarov. Combining Neural Networks and Tree Search for Task and Motion Planning in Challenging Environments. *arXiv preprint arXiv:1703.07887*, 2017.

[27] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive (1) designs. In *Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.

[28] Vasumathi Raman and Hadas Kress-Gazit. Synthesis for multi-robot controllers with interleaved motion. In *International Conference on Robotics and Automation (ICRA)*, pages 4316–4321. IEEE, 2014.

[29] Khashayar Rohanimanesh and Sridhar Mahadevan. Learning to take concurrent actions. In *Advances in neural information processing systems (NIPS)*, pages 1651–1658, 2003.

[30] Khashayar Rohanimanesh and Sridhar Mahadevan. Coarticulation: An approach for generating concurrent plans in markov decision processes. In *International Conference on Machine Learning (ICML)*, pages 720–727. ACM, 2005.

[31] Sheldon M Ross. *Applied Probability Models with Optimization Applications*. Holden Day, San Francisco, 1970.

[32] Dorsa Sadigh, Eric S Kim, Samuel Coogan, S Shankar Sastry, and Sanjit A Seshia. A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In *Conference on Decision and Control*, pages 1091–1096. IEEE, 2014.

[33] Philipp Schillinger, Stefan Kohlbrecher, and Oskar von Stryk. Human-Robot Collaborative High-Level Control with Application to Rescue Robotics. In *IEEE International Conference on Robotics and Automation*, Stockholm, Sweden, May 2016.

[34] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. Auctioning over Probabilistic Options for Temporal Logic-Based Multi-Robot Cooperation under Uncertainty. In *IEEE International Conference on Robotics and Automation*. Brisbane, 2018.

[35] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. Simultaneous Task Allocation and Planning for Temporal Logic Goals in Heterogeneous Multi-Robot Systems. *The International Journal of Robotics Research*, 2018.

[36] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[37] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

[38] Marc Toussaint, Thibaut Munzer, Yoan Mollard, Li Yang Wu, Ngo Anh Vien, and Manuel Lopes. Relational activity processes for modeling concurrent cooperation. In *International Conference on Robotics and Automation*, pages 5505–5511. IEEE, 2016.

[39] John N Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine learning*, 16(3):185–202, 1994.

[40] Jana Tumova and Dimos V Dimarogonas. Multi-agent planning under local LTL specifications and event-based synchronization. *Automatica*, 70:239–248, 2016.

[41] Alphan Ulusoy, Stephen L Smith, Xu Chu Ding, Calin Belta, and Daniela Rus. Optimality and robustness in multi-robot path planning with temporal logic constraints. *The International Journal of Robotics Research*, 32(8):889–911, 2013.

[42] Robert Zlot and Anthony Stentz. Complex task allocation for multiple robots. In *IEEE International Conference on Robotics and Automation*, pages 1515–1522. IEEE, 2005.