2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Long-Run Multi-Robot Planning under Uncertain Action Durations for Persistent Tasks

Carlos Azevedo[1], Bruno Lacerda[2], Nick Hawes[2] and Pedro Lima[1]

Abstract— **This paper presents an approach for multi-robot long-term planning under uncertainty over the duration of actions. The proposed methodology takes advantage of generalized stochastic Petri nets with rewards (GSPNR) to model multi-robot problems. A GSPNR allows for unified modeling of action selection, uncertainty on the duration of action execution, and for goal specification through the use of transition rewards and rewards per time unit. Our approach relies on the interpretation of the GSPNR model as an equivalent embedded Markov reward automaton (MRA). We then build on a state-of-the-art method to compute the long-run average reward over MRAs, extending it to enable the extraction of the optimal policy. We provide an empirical evaluation of the proposed approach on a simulated multi-robot monitoring problem, evaluating its performance and scalability. The results show that the synthesized policy outperforms a policy obtained from an infinite horizon discounted reward formulation as well as a carefully hand-crafted policy.**

I. Introduction

In recent years, there has been a growing interest on the use of multi-robot systems for disaster prevention applications, such as flooding and forest fire detection [1]. These applications require the robots to repeat tasks efficiently over long periods of time. Robust and efficient multi-robot coordination strategies are crucial to achieve such a goal. Robustness requires approaches that take into account the uncertainty inherent to teams of robots executing actions in real environments, while efficiency requires considering long horizons during planning.

Example 1: Consider a scenario where a team of homogeneous robots can move between a set of locations. The robots can also perform sensing actions, and the task consists of persistently monitoring some of these locations in order to anticipate a wildfire spreading. However, each robot has limited battery autonomy and can only execute actions for a certain amount of time before needing to recharge. Both the time taken to execute each action, and the time taken to discharge or recharge are not deterministic; they are influenced by many uncontrollable factors, such as congestion in navigation or battery temperature. The goal is to find a policy that optimally controls the multi-robot system, selecting, among multiple choices, how the system should behave and which actions should be executed by the robots. This policy must ensure that the higher priority locations are always monitored whilst the ones with lower priority are only monitored when there are enough robots available, without letting the higher priority locations go unmonitored.

In this paper, we propose an approach that addresses the challenges related to Example 1. We capture the uncertainty in action duration by modeling the problem as a *generalized stochastic Petri net with rewards* (GSPNR) [2]. These rewards are an intuitive instrument to define the team objective, for example, allowing to specify a prioritization over key locations. Then, we adapt a state-of-the-art model checking algorithm [3] to synthesize policies that optimize the long-run average (LRA) reward properties of the multi-robot task.

Our main contributions are: (i) the use of GSPNRs to model multi-robot persistent monitoring tasks, exploiting their translation into *embedded Markov reward automata* (MRAs) to synthesize policies that optimize team-level objectives; (ii) the extension of state-of-the-art methods to model check LRA properties over an MRA, in order to extract the corresponding optimal policy; and (iii) the application of LRA properties to provide an automated approach for solving multi-robot persistent surveillance and monitoring problems under action duration uncertainty.

II. Related Work

The use of Petri nets (PNs) for modeling and analyzing robot tasks has been advocated in several works. Relevant examples are [4], where PNs are used for modeling and execution of single-robot plans and [5], where an extra focus is given to the modeling of the stochasticity of the environment, through the use of GSPNs. These works assume a plan for the robots has already been created by some other means.

PNs have also been used for behavior synthesis. [6] uses Boolean specifications to define path planning goals over a PN model of the team, and an integer linear program formulation to synthesize a team controller, whilst [7] uses ideas from supervisory control to enforce a set of coordination rules specified in safe linear temporal logic. The work in [8] uses a PN to model an instance of a similar problem to the persistent monitoring under battery constraints example we consider in our work, and proposes a genetic algorithm to find approximate solutions. The three works above assume a deterministic model, not tackling uncertainty. In contrast, [9] uses a GSPN model similar to the one we use, thus tackling uncertainty on action duration. We extend it to include more general rewards and tackle LRA properties.

Other works that tackle uncertainty on action duration use generalized semi-Markov decision processes (GSMDPs) to model muti-robot systems [10], [11]. GSMDPs are able to handle arbitrary distributions over action duration but at the

[1]Institute For Systems and Robotics, Instituto Superior Técnico, University of Lisbon, Portugal {cguerraazevedo, pedro.lima}@tecnico.ulisboa.pt
[2]Oxford Robotics Institute, University of Oxford, UK {bruno, nickh}@oxfordrobotics.institute

cost of very high complexity, due to both the modeling of arbitrary distributions, and the requirement of a full joint state space. This requirement is relaxed in [12], which uses single robot planning models, for single robot goals, that consider the stochastic influence of other robots in the duration of navigation actions. Asynchronous execution has also been enabled in a decentralized, partially observable, setup through the use of macro actions [13], [14], where synchronization happens only at the level of the macro action controllers, rather than at each timestep. However, these methodologies do not scale well enough to solve infinite-horizon problems, and restrict the problem formulation to a finite-horizon view.

Finally, [15] has looked at synthesis for steady state properties over MDPs, a class of properties similar to LRA rewards. They, however, focus on MDP models, which cannot encode continuous action duration.

III. Preliminaries

A. Generalized Stochastic Petri Nets

A generalized stochastic Petri net (GSPN) is tuple $G = \langle P, T, W^+, W^-, F, m_0 \rangle$, where: P is a non-empty and finite set of places; T is a finite set of transitions; $W^- : P \times T \to \mathbb{N}$ and $W^+ : T \times P \to \mathbb{N}$ are input and output arc weight functions, respectively; $F : T_E \to \mathbb{R}_{\geq 0}$ is a function that associates a rate to each exponential transition; and $m_0 : P \to \mathbb{N}$ is the initial marking.

A marking, $m : P \to \mathbb{N}$, assigns to each place a non-negative integer number of tokens and represents the state of the system. The set of input and output places of transition $t \in T$ is given by $IN(t) = \{p \in P \mid W^-(p,t) > 0\}$ and $OUT(t) = \{p \in P \mid W^+(t,p) > 0\}$, respectively. A transition is said to be enabled if each input place $p \in IN(T)$ is marked with at least $W^-(p,t)$ tokens. Enabled transitions can *fire*, updating the marking according to the arc weight functions. When t fires, $W^-(p_{in}, t)$ defines the number of tokens removed from p_{in}, for each place $p_{in} \in IN(T)$; and $W^+(t, p_{out})$ the number of tokens added to p_{out}, for each place $p_{out} \in OUT(T)$. Transitioning from marking m to marking m' due to the firing of t is denoted as $m[t\rangle m'$. We denote the set that contains all markings reachable from the initial marking m_0, under the firing rule, as $R(G)$.

The set T is partitioned into the subset of immediate transitions T_I and the subset of exponential transitions T_E, with $T = T_I \cup T_E$. The exponential transitions, once enabled, fire only after an exponentially distributed duration elapses. Every exponential distribution t_e has a rate given by $F(t_e)$ while immediate transitions take zero time to fire.

In marking m, the set of enabled immediate transitions is defined by $\Lambda(m) = \{t \in T_I \mid m(p) \geq W^-(p,t) \forall p \in IN(t)\}$ and the set of enabled exponential transitions by $\Gamma(m) = \{t \in T_E \mid m(p) \geq W^-(p,t) \forall p \in IN(t)\}$. A race condition occurs when there is more than one exponential transition enabled in the same marking. When this happens, the probability of transition $t_e \in T_E$ firing first is given by $P(m, t_e) = F(t_e) / \sum_{t_i \in \Gamma(m)} F(t_i)$. The set of all vanishing markings is defined as $R_V(G) = \{m \in R(GR) \mid \Lambda(m) \neq \emptyset\}$ and the set of all tangible markings is defined as $R_T(G) = \{m \in R(GR) \mid \Gamma(m) \neq \emptyset\}$. Note that with this definition, $R_V(G) \cap R_T(G) \neq \emptyset$, which is non-standard. This is due to our inpterpretation of GSPNs as models for multi-robot decision making, and will be further discussed in Section V-C.

IV. Problem Formulation

A. GSPNR Models for Persistent Monitoring

Recall the monitoring task introduced in Example 1 and consider that each decision is made instantaneously while actions take some stochastic time to execute. In order to capture this problem, we take advantage of *GSPNs with rewards* (GSPNR), which maintain the same formal elements of GSPNs and add place and transition rewards.

A GSPNR is defined by the tuple $G^r = \langle P, T, W^+, W^-, F, m_0, r_P, r_T \rangle$ and when modeling multi-robot persistent monitoring tasks can be interpreted as follows. The set of places P models local robot states. These represent action execution or action selection states for subsets of the multi-robot system. $T = T_I \cup T_E$ where T_I represents the set of all available controllable actions, and T_E is the set of timed events that are triggered by environmental changes. These events correspond, for example, to a robot finishing executing an action, or the battery level of a robot becoming low. Exponential transitions capture the duration uncertainty associated with them. W^- defines which timed events can trigger or what are the available actions, in a local state. W^+ dictates the local state update of such timed events or action selection. The initial number of robots executing each action or in each decision state is represented by m_0. The place reward $r_P : P \to \mathbb{R}_{\geq 0}$ gives a reward of $r_P(p)$ per time unit in markings where there is at least one token in p. The transition reward $r_T : T_I \to \mathbb{R}_{\geq 0}$ gives a reward every time the action $t \in T_I$ is selected.

Crucially, in this representation, each robot is viewed as a token, providing an intuitive and compact model of the multi-robot system. Therefore, the global state of the system in each moment is represented by its marking. Tangible markings model states where all robots are executing a time consuming action, while vanishing markings model states where a subset of robots has to make a decision. When compared with standard modelling approaches, such as multi-agent Markov decision processes (MMDPs), this model provides: better scalability and fully asynchronous execution [9], ensuring a smaller state-space and a more fluid policy execution.

B. Long Run Average Objective

We start by defining a stationary deterministic policy over a GSPNR as a function $\pi : R(G^r) \to T_I \cup \{\textit{wait}\}$ that maps markings to immediate transitions, representing the triggering of an action; or to a *wait* action, representing the system waiting for the occurrence of the next environmental event before deciding whether to trigger an action. We denote the set of all such policies over G^r as Π_{G^r}. We can now define our objective as finding the policy that maximizes the expected LRA reward over G^r:

$$LRA_{G^r} = \max_{\pi \in \Pi_{G^r}} E\left[\lim_{\tau \to \infty} \frac{\text{rew}(\pi, \tau)}{\tau}\right],$$

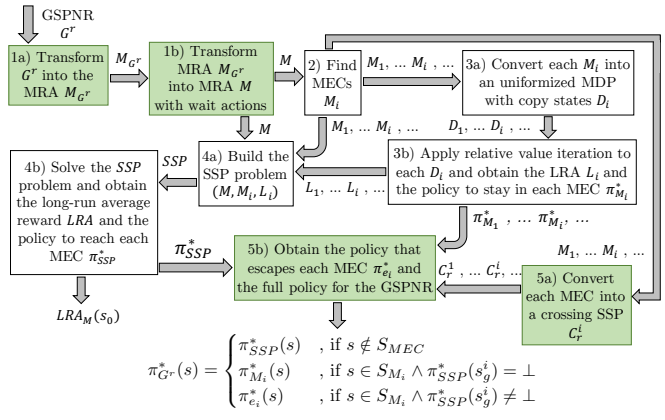

Fig. 1. Diagram summarizing the steps of the proposed solution method. The blocks highlighted in green represent the proposed extensions.

where E is the expected value and $rew(\pi, \tau)$ is the accumulated sum of rewards r_P and r_T obtained until time τ, starting in the initial marking m_0 and following policy π.

V. Solution Method

A. Solution Overview

The proposed model is able to capture multi-robot persistent surveillance and monitoring problems, with homogeneous robot teams and uncertainty in action durations. This section details a method to synthesize policies for GSPNRs that optimize the average reward in the long-run. This solution is summarized in Fig. 1 and can be split into five steps: 1) transform the GSPNR model to the *embedded MRA* with wait actions M; 2) determine the maximal end components (MECs) $\{M_1, ..., M_i\}$ of the MRA M; 3) compute the LRA reward for each MEC M_i and obtain the optimal policy that stays indefinitely in each MEC $\pi^*_{M_i}$; 4) obtain the LRA reward of M and the optimal policy to reach each MEC π^*_{SSP}, by reducing M to a stochastic shortest path problem (SSP) on a Markov decision process (MDP), where each MEC is a pair of states of the SSP; 5) convert each MEC, along the optimal policy path, to an SSP on an MDP and obtain the optimal policy that escapes each MEC $\pi^*_{e_i}$.

In the following sections, we focus on steps 1 and 5, since the remaining steps are methods already covered in the literature [3] [16]. Subsections V-B and V-C, details how to perform the transformation in step 1 and Subsection V-E explains how to address step 5. A brief explanation of steps 2, 3 and 4 can be found on Subsection V-D.

B. From GSPNR to MRA

We interpret the marking process of the GSPNR model as an MRA. By doing so, we can adapt methods for model checking MRAs in order to obtain policies for the GSPNR model. We extend the conversion presented in [17], by adding rewards to the models. An illustrative example of this transformation is shown in Fig. 2.

An MRA is an MDP extension that considers both immediate and continuous-time transitions. The *embedded MRA*

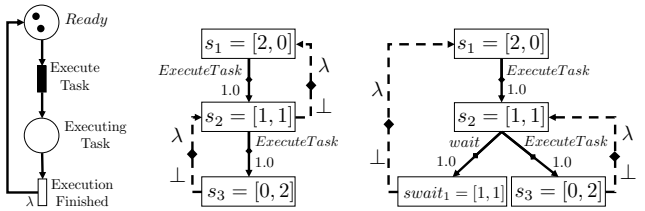

Fig. 2. (left) Example GSPNR G^r, where circles, solid rectangles and white rectangles represent places, immediate transitions and exponential transitions, respectively. (middle) The embedded MRA M_{G^r} obtained from G^r. The dashed lines correspond to exponential transitions and the solid lines to immediate transitions. (right) The MRA M with wait actions.

of the GSPNR G^r is a tuple $M_{G^r} = \langle S, s_0, Act, \rightarrow, \Rightarrow, \rho, \sigma \rangle$. The state space of M_{G^r} corresponds to the reachable markings of G^r, i.e. $S = R(G^r)$. The initial state is defined as $s_0 = m_0$. The set of actions is given by $Act = T_I \cup \{\bot\}$, where $\bot$ represents all uncontrollable actions. $\rightarrow \subseteq S \times Act \times Dist(S)$ is the set of immediate transitions, where $Dist(S)$ is the set of probability distributions over S. In the embedded MRA, $\rightarrow = \{(m, t, \delta_{m'}) \mid m \in R(G^r),\ t \in T_I \text{ and } m[t\rangle m'\}$, where $\delta_{m'}$ represents the Dirac delta distribution, $\delta_{m'}(m') = 1$. In words, immediate transitions in G^r correspond to deterministic transitions in M_{G^r}. $\Rightarrow \subseteq\ S \times \mathbb{R}_{>0} \times S$ is the set of exponential transitions. In the embedded MRA, $\Rightarrow = \{(m, F(t), m') \mid m \in R(G^r),\ t \in T_E \text{ and } m[t\rangle m'\}$. In words, for each exponential transition t enabled in a marking in $R(G^r)$, there is a transition in the MRA, labeled by $\bot$, with rate equal to $F(t)$. The state reward function is given by $\rho(m) = \sum_{p_i \in P_{\geq 1}} r_p(p_i)$, where $P_{\geq 1}$ is the set of places with at least one token, in the marking $m \in R(G^r)$. This means that, when in state $s \in S$, the system earns a reward per time unit equal to the sum of the rewards of each place p_i where there is at least one robot. The transition reward function is defined as $\sigma(a) = r_T(t)$ with $a = t$ and $t \in T_I$. Intuitively, a reward of $r_T(t)$ is awarded every time selecting action a corresponds to firing the immediate transition t.

The race conditions, vanishing and tangible markings have analogous definitions in MRAs by replacing markings with states. The sets of vanishing and tangible states are denoted by S_V and S_T, respectively. Moreover, we define the total rate between two states as $Fr(s, s') = \sum_{(s,\lambda,s') \in \Rightarrow} \lambda$ and the exit rate of a state as $Ex(s) = \sum_{s' \in S} Fr(s, s')$.

The algorithmic procedure to construct the MRA starts in the initial marking, generating the reachability graph by firing all the enabled transitions until all markings reachable from the initial marking have been explored.

C. Adding the Wait Action

Approaches for LRA assume the set of *hybrid* (i.e. both vanishing and tangible) states $S_H = S_V \cap S_T$ to be empty. This is typically achieved using the *urgency assumption*, which gives priority to the execution of immediate transitions, by removing exponential transitions from states that also have immediate transitions. However, this does not match our semantics of allowing policies to *wait* for the occurrence of an environmental event.

We now describe how, given an MRA M with $S_H \neq \emptyset$, we can build an MRA M^{wait}. M^{wait} has no hybrid states, i.e. it satisfies $S_V^{wait} \cap S_T^{wait} = \emptyset$, whilst maintaining our semantics of allowing the choice to wait for the occurrence of an environmental event. To build M^{wait}, for each $s_H \in S_H$, we (i) add a new state s_H^{new} to M; (ii) move all exponential transitions starting on s_H to start on s_H^{new} (this makes s_H only a vanishing state); and (iii) add a new immediate transition of the form $(s_H, wait, \delta_{s_H^{new}})$, to M (i.e., we add a new immediate transition to s_H that represents the system deciding to move to a state where it waits for the occurrence of an environmental event). Thus, $S_V^{wait} = S_V$, $S_T^{wait} = (S_T \setminus S_H) \cup S_H^{new}$ and $S_V^{wait} \cap S_T^{wait} = \emptyset$. Here S_H^{new} represents the set that contains all the newly created states s_H^{new}. We show an example of the construction of M^{wait} on Fig. 2. In the remainder of the paper, we will assume we are working with M^{wait}, but will write M to keep notation simple.

D. Computing the Long-Run Average Reward

We now give an overview of the LRA computation algorithm proposed in [16] and [3]. This relates to steps 2–4 of the diagram presented in Fig. 1.

In step 2 of the LRA computation algorithm, we find the MECs of the MRA. An MEC is a sub-MRA that is not contained in any other sub-MRA and whose underlying graph is strongly connected. A sub-MRA is a non-empty fraction of an MRA. We denote the state space of MEC M_i as S_{M_i} and define $S_{MEC} = \cup_{i=1}^{N_{MEC}} S_{M_i}$, where $N_{MEC}(M)$ refers to the total number of MECs in the MRA M.

It is known that once the system enters a MEC, there exists a policy that keeps it in that MEC indefinitely, whilst visiting all its states infinitely often; furthermore, for any policy, the MRA will end in one of the MECs and stay there forever [3]. Thus, to compute the LRA properties from initial state s_0, one must compute the LRA for each MEC (along with the corresponding policy) and then find a policy that drives the MRA from s_0 to the MEC with better ratio between the probability of reaching it and the LRA in that MEC. The MECs can be computed efficiently using graph-based approaches [18].

Step 3, computes the LRA reward L_i of each MEC M_i by posing the problem as a total expected reward optimization on a uniformized MDP, after starting in a tangible state and finishing upon encountering a tangible state for the second time. This can be computed using relative value iteration [3] and yields the optimal policy to stay indefinitely in each MEC, $\pi^*_{M_i}(s)$ for all $s \in S_{M_i}$.

Finally, step 4 builds an SSP for the MRA M, where all states in each MEC M_i are replaced by two states s_g^i and s_u^i, and a cost of LRA_{M_j} is assigned to each $(s_g^i, \perp)$ pair. Intuitively, s_g^i represents the gate to and from M_i, where we decide whether to (i) stay in M_i and apply policy $\pi^*_{M_i}(s)$; or (ii) leave M_i towards another MEC M_j with better LRA_{M_j}. If we decide the former, we move to s_u^i, which corresponds to applying policy $\pi^*_{M_i}(s)$. This yields the policy $\pi^*_{SSP}(s)$ for all $s \in S_{M_{G^r}} \setminus S_{MEC}$.

E. Extracting the Optimal Policy

The procedure described in the previous subsection yields a policy π^*_{SSP} that defines which final MEC M_f the system will try to reach, and the policy $\pi^*_{M_f}$ that determines how to behave within M_f in order to never leave it. However, in order to reach M_f, the system may have to cross other intermediary MECs. The policies obtained so far do not fully define which actions should be taken in order to cross these MECs. Instead, policy π^*_{SSP} only determines the optimal state-action pair (s_g^i, α_g^i), where α_g^i is the last action that should be selected inside M_i in order to leave it. For this reason, the optimal policy is undefined for all other states of each intermediary MEC. Thus, to build a complete policy π^*_{GR}, we must find the optimal actions that lead to s_g^i from any state in an intermediary MEC. This policy must be obtained for all MEC M_i that must be crossed, according to π^*_{SSP}.

Step 5 creates this policy denoted as $\pi^*_{e_i}$. To do this, we transform each MEC M_i to be crossed into a *crossing SSP* C_r^i over an MDP. Let $M_i = \langle S_i, s_0^i, Act_i, \rightarrow_i, \Rightarrow_i, \rho_i, \sigma_i \rangle$ be a MEC of the MRA M. The *crossing SSP* is defined as $C_r^i = \langle S, s_0, A, P, c \rangle$, where $S = S_i \cup \{tr\}$, i.e. a terminal state tr with a self-loop is added and the action α_g^i is redirected to tr with probability 1; $s_0 = s_0^i$; $A = Act_i$; the probabilistic transition function $P : S \times A \times S \rightarrow [0, 1]$ is defined as:

$$P(s, \alpha, s') = \begin{cases} Fr(s, s')/Ex(s) & \text{if } s \in S_T^{M_i},\ \alpha = \perp \\ & \text{and } s' \neq s \\ \mu(s') & \text{if } s \in S_V^{M_i}, \\ & \alpha \in Act_i \setminus \{\perp\} \\ & \text{and } s \xrightarrow{\alpha} \mu(s') \\ 1 & \text{if } s = s_g^i,\ a = \alpha_g^i \\ & \text{and } s' = tr \\ 1 & \text{if } s, s' = tr,\ \alpha = \perp; \end{cases}$$

and the cost function $c : S \times A \rightarrow \mathbb{R}_{\geq 0}$ is defined as:

$$c(s, \alpha') = \begin{cases} 1/Ex(s) & \text{if } s \in S_T^{M_i} \\ 0 & \text{otherwise.} \end{cases}$$

Obtaining the optimal policy $\pi^*_{e_i}$ that escapes the MEC M_i is thus reduced to the minimization of the total expected cost on the *crossing SSP* C_r^i. The cost function ensures that the obtained policy minimizes the expected amount of time spent in the intermediary MECs and the system reaches its steady-state as quickly as possible.

The full optimal policy is defined, for all $i \in N_{MEC}(M)$:

$$\pi^*_{G^r}(s) = \begin{cases} \pi^*_{SSP}(s) & \text{, if } s \notin S_{MEC} \\ \pi^*_{M_i}(s) & \text{, if } s \in S_{M_i} \wedge \pi^*_{SSP}(s_g^i) = \perp \\ \pi^*_{e_i}(s) & \text{, if } s \in S_{M_i} \wedge \pi^*_{SSP}(s_g^i) \neq \perp \end{cases}$$

Example 2: Consider the MRA depicted in Fig. 3 (left) and suppose that the optimal policy π^*_{SSP} takes the system to the MEC M_3 and stays there according to policy $\pi^*_{M_3}$. Since the starting state is s_0, to reach M_3 it has to go through M_1. However, π^*_{SSP} only defines that to reach M_3, the system has to select action a in state s_0 and then select action c in state

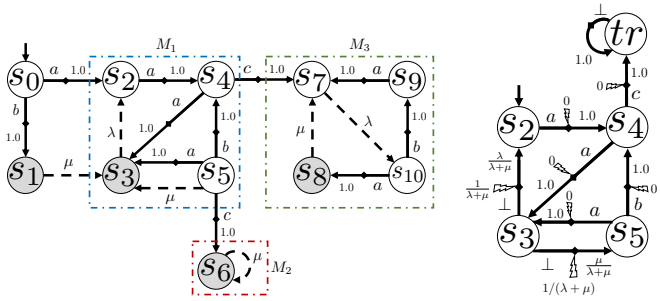

Fig. 3. (left) Example of an arbitrary MRA with three MECs: M_1, M_2 and M_3. The tangible states and the vanishing states are depicted in grey and white, respectively. (right) The *crossing SSP* C_r^1 obtained from M_1. The bolt icon represents the cost of that state-action-state.

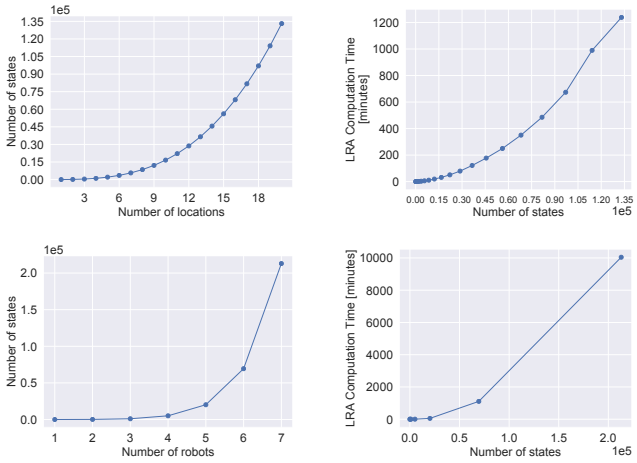

Fig. 4. Performance and scalability analysis obtained by: (top) increasing number of locations for a fixed number of 3 robots; (bottom) increasing number of robots for a fixed number of 4 locations.

s_4. Thus, the optimal policy for states s_2, s_3 and s_5, such that M_1 is crossed, is not defined. To compute it, M_1 is reduced into the *crossing SSP* C_r^1 shown in Fig. 3 (right). Then leaving the state s_3 is assigned with a cost $c(s_3, \perp) = \frac{1}{\lambda+\mu}$. Finally, the policy $\pi^*_{e_1}$ can be obtained through value iteration, where: $\pi^*_{e_1}(s_2) = a$, $\pi^*_{e_1}(s_3) = \perp$ and $\pi^*_{e_1}(s_5) = b$.

VI. Experiments

In this section we evaluate the performance and scalability of our approach, comparing it with 3 baselines in simulation. All experiments were run on a machine with a CPU@3.7GHz, and 32GB of RAM. Example 1 serves as the application scenario for the experiments. We created several variations of this task. All these variations model the problem using the same building blocks that are depicted in Fig. 5 (left). The robots are able to monitor, navigate and recharge. The execution of these actions is captured by the places labeled $Monitoring$, $Navigating$ and $Charging$, respectively. The uncertainty in the duration of these actions is captured by the exponential transitions labeled $Finished$, $Arrived$ and $Charged$, respectively. The exponential transitions labeled as $Discharged$ capture the battery consumption stochasticity. Moreover, the places labeled $Ready$ represent local decision states where the robots can select monitoring the same location or moving to another location, by choosing between the immediate transitions $Repeat$ and $GoFromTo$, respectively.

A. Performance and Scalability

In the first scenario we model a team of 3 robots, where the number of locations was increased from 1 to 20. In order to assign the same monitoring priority to each location and have a constant maximum total reward of 1, a place reward of $r_P = 1/n_l$ is awarded to each $Monitoring$ place, where n_l is the number of locations to monitor. The second scenario considers a fixed number of 4 locations to monitor, with the number of robots being increased from 1 to 7. An equal reward of 1 is assigned to each of the $Monitoring$ places.

The plots in Fig. 4 (top) and (bottom) show the results obtained for the first and second scenario, respectively. Fig. 4 (top left) and (bottom left) show that the number of states grows super linearly as the number of locations and the number of robots increases, respectively. Nevertheless, it increases much slower than other multi-agent models, such as MMDPs, where state representation requires the exact position of each robot rather than counting the number of robots in each local state. On the other hand, Fig. 4 (top right) and (bottom right) show that the long-run average computation requires a considerable amount of time. However, our method provides a robust policy that can be run as long as the parameters of the system remain the same. Therefore, it is a pre-deployment computation that then yields optimal rewards in the long run.

B. Simulation

We simulated a 4 robot setup taking advantage of ROS and the Stage simulator. The map contains 6 locations plus a charging station as it can be seen in Fig. 5 (right). The simulation was kept as realistic as possible, by adding noise to the measurements and by using standard ROS packages for localization and navigation. The multi-robot team is initialized with random battery levels and all the robots start in the charging station. Locations *L4*, *L5* and *L6* were assigned with a monitoring priority of 1, 3 and 5 respectively. The rates specified for each exponential transition were obtained taking into account the distance between locations and the maximum speed of each robot.

The policy obtained using the presented method was compared against a policy obtained through the maximization of the total expected discounted reward, that we shall refer as *discounted policy* for simplicity. It was also compared against a hand coded policy and a random policy. The *discounted policy* was obtained for a discount factor of 0.9999999. The hand coded policy sends the robots to locations *L6*, *L5* and *L4* with decreasing order of priority. When all three priority locations are already being monitored the hand coded policy sends the fourth robot to location *L2*, since it is the one closest to location *L6*. The random policy selects a location for each robot according to a uniform distribution.

Fig. 6 depicts the accumulated reward over time for 100 runs of each policy. These results demonstrate that the policy obtained through our method outperforms all the

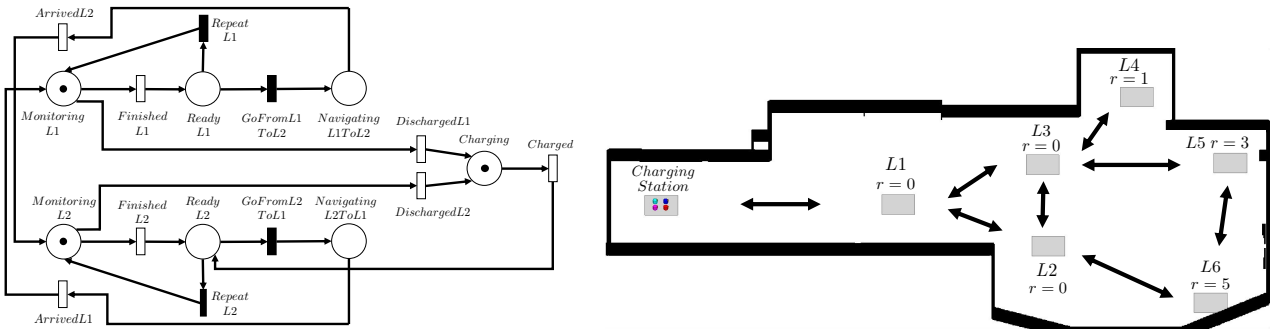

Fig. 5. (left) GSPNR model of a simple monitoring example with 3 robots and 2 locations. (right) Map of the simulated environment in Stage ,where the grey rectangles represent locations and the arrows the edges connecting them. The colored discs represent the robots.

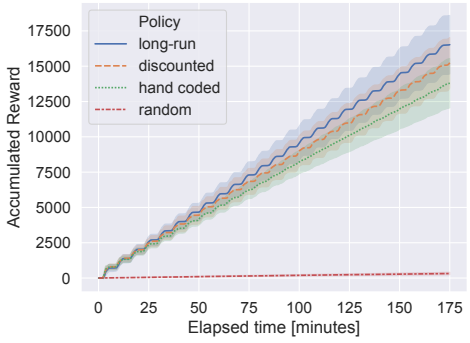

Fig. 6. Accumulated reward following each policy in the Stage simulation. The lines represent the mean and the shadows the standard deviation.

others baselines even before 3 hours of runtime. Generally speaking, we expect real missions to be longer than 3 hours, and therefore after 43200 hours we expect to outperform the discounted baseline by 3.1×10^5 (based on a linear extrapolation of the results). The reason is that our method optimizes the reward earned taking into account the cyclic behavior of the system on an infinite horizon, while the discounted approaches take a myopic view even with discount factors close to 1. Henceforth, for many applications, this gain in performance is worth the one-time cost discussed in the previous section. As far as we are aware, all existing methods for planning under uncertain action duration in the multi-robot community obtain a solution considering a finite horizon or a discounted infinite horizon.

VII. Future Work

In this paper, we have presented a multi-robot planning approach for long-term scenarios, that provides policies for persistent monitoring scenarios where uncertainty over the duration of tasks are taken into consideration.

In the future, we intend to scale the method for larger teams, exploiting sub-optimal policies, through the use of heuristics. Additionally, we will investigate the extension of the current method to handle other classes of specifications, such as linear temporal logic missions or multi-objective problems.

Acknowledgment

This work was supported by the Portuguese Fundação para a Ciência e Tecnologia (FCT) under the grant SFRH/BD/135014/2017 and the LARSyS - FCT Project UIDB/50009/2020, and by UK Research and Innovation and EPSRC through the Robotics and Artificial Intelligence for Nuclear (RAIN) research hub [EP/R026084/1].

References

[1] R. R. Murphy, S. Tadokoro, and A. Kleiner, "Disaster robotics," in *Springer Handbook of Robotics*, 2016.
[2] C. Azevedo, B. Lacerda, N. Hawes, and P. Lima, "Long-run multi-robot planning with uncertain task durations," in *AAMAS*, 2020.
[3] Y. Butkova, R. Wimmer, and H. Hermanns, "Long-run rewards for Markov automata," in *TACAS*, 2017.
[4] V. A. Ziparo, L. Iocchi, P. U. Lima, D. Nardi, and P. F. Palamara, "Petri net plans – A framework for collaboration and coordination in multi-robot systems," *AAMAS*, 2011.
[5] H. Costelha and P. Lima, "Robot task plan representation by Petri nets; Modelling, identification, analysis and execution," *Aut. Robots*, 2012.
[6] C. Mahulea and M. Kloetzer, "Robot planning based on Boolean specifications using Petri net models," *IEEE-TAC*, 2018.
[7] B. Lacerda and P. Lima, "Petri net based multi-robot task coordination from temporal logic specifications," *Rob. and Aut. Sys.*, 2019.
[8] H. Park and J. R. Morrison, "System design and resource analysis for persistent robotic presence with multiple refueling stations," in *ICUAS*, 2019.
[9] M. Mansouri, B. Lacerda, N. Hawes, and F. Pecora, "Multi-robot planning under uncertain travel times and safety constraints," in *IJCAI*, 2019.
[10] H. L. S. Younes and R. G. Simmons, "Solving Generalized Semi-Markov Decision Processes using continuous phase-type distributions," in *AAAI*, 2004.
[11] J. V. Messias, M. T. J. Spaan, and P. U. Lima, "GSMDPs for multi-robot sequential decision-making," in *AAAI*, 2013.
[12] C. Street, B. Lacerda, M. Mühlig, and N. Hawes, "Multi-robot planning under uncertainty with congestion-aware models," in *AAMAS*, 2020.
[13] S. Omidshafiei, A.-A. Agha-Mohammadi, C. Amato, S.-Y. Liu, J. P. How, and J. Vian, "Decentralized control of multi-robot partially observable Markov decision processes using belief space macro-actions," *IJRR*, 2017.
[14] C. Amato, G. Konidaris, L. P. Kaelbling, and J. P. How, "Modeling and planning with macro-actions in decentralized POMDPs," *JAIR*, 2019.
[15] A. Velasquez, "Steady-state policy synthesis for verifiable control," in *IJCAI*, 2019.
[16] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer, "Analysis of timed and long-run objectives for markov automata," *LMCS*, 2014.
[17] C. Eisentraut, H. Hermanns, J.-P. Katoen, and L. Zhang, "A semantics for every GSPN," in *Petri Nets*, 2013.
[18] K. Chatterjee and M. Henzinger, "Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification," in *SODA*, 2011.