LTL-Based Planning in Environments With Probabilistic Observations
Marius Kloetzer and Cristian Mahulea, Member, IEEE

Abstract—This research proposes a centralized method for planning and monitoring the motion of one or a few mobile robots in an environment where regions of interest appear and disappear based on exponential probability density functions. The motion task is given as a linear temporal logic formula over the set of regions of interest. The solution determines robotic trajectories and updates them whenever necessary, such that the task is most likely to be satisfied with respect to probabilistic information on regions. The robots' movement capabilities are abstracted to finite state descriptions, and operations as product automata and graph searches are used in the provided solution. The approach builds up on temporal logic control strategies for static environments by incorporating probabilistic information and by designing an execution monitoring strategy that reacts to actual region observations yielded by robots. Several simulations are included, and a software implementation of the solution is available. The computational complexity of our approach increases exponentially when more robots are considered, and we mention a possible solution to reduce the computational complexity by fusing regions with identical observations.

Note to Practitioners—The paper brings contributions in the area of mobile robots by presenting a procedure for planning a team of robots in a dynamic environment based on high-level requirements. More specifically, we assume a team of identical mobile robots that should automatically accomplish a global mission that includes logical and temporal requirements on attainment or avoidance of some regions from the environment. The regions (areas) of interest alternate between being visible (appeared) or invisible (disappeared) based on probabilistic events. Our method develops an initial movement plan for robots such that the mission has a good chance of being satisfied, and it updates this plan whenever necessary, based on actual observations yielded by the robots. The framework can be used for automatically planning intelligent autonomous systems while possible applications include: robots used for search and rescue missions (where casualties may randomly appear in dangerous areas), automatic transportation systems that should visit some areas where passengers have known rates of appearance, or autonomous vacuum cleaners that allow rich specifications to be input by the owner (such as clean the living room and the dorms, where dust may appear with different rates, and then go to charge in either dorm). As usual for many systems that automatically handle various complex but friendly inputs from the human operator, the proposed method induces a computational overhead when compared to classical methods developed only for specific situations.

Index Terms—Discrete event systems, linear temporal logic (LTL), mobile robots.

I. INTRODUCTION

Recent research on mobile robots proposed different methods for extending the motion tasks to rich and human-like specifications [1]–[4]. Thus, it is now possible for some classes of robots to automatically perform missions as “visit region A or B, and if A is visited go to C, otherwise visit infinitely often B and C while avoiding A” rather than classic problems as “go to A while avoiding obstacles” [5], [6]. An important challenge in enabling rich tasks was the proper choice of a language that allows expressive specifications and intuitive translation from human language to syntactically correct formulas. Solutions were found in a variety of formal languages [7], [8]. After choosing a specification class, another challenge was the development of algorithms that actually control the robot movement such that the desired mission is accomplished. Again, inspiration was drawn from formal analysis [7]–[9], and solutions were developed by appropriately adjusting and extending the available algorithms. Furthermore, another challenge lies in the extension of solutions available for a single robot to multirobot scenarios [1], [4], [10], [11].

In this paper, we formulate robotic missions for a team of cooperating robots by linear temporal logic (LTL) formulas, which is, together with its fragments, e.g., GR(1), among common choices for allowing expressive control objectives [2], [3], [10], [12]. Tasks given in this logic usually imply visits to some regions of interest from the environment in a specific or arbitrary order, avoidance of other regions, and various logical and temporal connectives among regions. The results of the mentioned works are extended in the current research by assuming environments with dynamic observations rather than static ones. Specifically, we consider that the regions of interest have fixed and known locations, but they appear and disappear based on exponential probabilistic density functions with known rates. This framework does not directly link our problem to research areas as probabilistic model checking [13], [14], where the probabilistic nature arises from actions with uncertain outcome.
Similar to multiple approaches for this type of problems, our solution begins with the abstraction of the movement capabilities of each robot into a finite-state description. Different abstractions are possible, as models specific to resource allocation systems based on automata [15] or Petri nets [16], and in this work we consider transition system models [3]. Based on individual robot models, we construct a finite transition system that models the whole team of robots and embeds probabilistic information of observing the regions of interest in specific locations. The problem is solved by developing an execution monitoring strategy for the obtained model, and to this goal we adapt tools used in LTL-based control approaches, such as product automata and graph searches. The monitoring strategy reacts whenever necessary to the regions of interest observed by the robots during the evolution. When choosing or updating the trajectory for the team of robots, our solution solves a multi-criterion optimization problem that maximizes the probability of satisfying the LTL formula while minimizing the number of robots’ movements. The focus of this work is on automatically constructing a monitoring strategy for the robots discrete event driven models, and therefore we make some simplifying assumptions as simple point robots and a centralized architecture that coordinates the team. Similar to the case of many methods developed for related problems, the main drawback of our approach is its computational complexity, a fact that prevents feasible applicability for large teams and/or complex environments and formulas. We mention a method for reducing the number of states of the robotic model in order to alleviate this drawback.

Related works that address the problem of planning mobile robots based on high-level specifications and different probabilistic information include [17]–[21]. All of these works focus on single systems, while the current one deploys a team of cooperating robots. In [17], the specifications are restricted to another formalism, namely probabilistic computation tree logic, while the probabilistic information is induced by uncertainties in robot sensing and control rather than by environment evolution. The works [18]–[20] assume a similar scenario to the one we consider, but the underlying system model and formal methods are entirely different, as the following ways. The robotic model in [18] and [20] is given in the form of a Markov decision process (MDP), where probabilities arise from events in the environment and from possible outcomes of system control actions. We construct a system model in form of a transition system with probabilistic outputs and fixed topology, while a probabilistic output map is modified based on environment events. The system requirements from [18]–[20] belong to a subclass of LTL formulae that include persistent tasks, and the solution takes the form of an a priori computed control policy for MDPs, based on optimization procedures inspired from probabilistic model checking [9], [22], [23] and more complex deterministic Rabin automata. The requirements in this work include any LTL task, and the solution is inspired by model checking techniques that use transition system models and nondeterministic Büchi automata. Rather than obtaining a control policy that includes all possible outcomes from probabilistic information, we obtain a system trajectory that has the maximum probability of satisfying the task and whenever necessary we update this trajectory via an online algorithm. Under this note, in case of a single robot, we consider our approach as an alternative to the MDP-based ones. In some cases, the offline computation of a control policy may be preferable, while in other situations it may be better to iterate an a priori unknown number of online trajectory generations. For multiple robots, the complexity of our centralized method rapidly increases, and future works may search solutions by drawing inspiration from decentralized or task decomposition approaches that use specific MDP or automata models [24]–[27]. Works such as [28]–[30] combine surveillance LTL tasks over nonvarying regions with additional tasks of collecting dynamically changing rewards, thus differentiating in the main goal from our research. We devise a solution to a cooperative team of robots, at the cost of an exponential increase in the model dependent only on the team size. Thus, our work extends the strategy from [21], which was applicable for a single mobile agent, to a team of identical robots. Another difference among mentioned related works and the current research is that our execution monitoring strategy includes a witness for situations in which the environment events lead to the violation of the system task. We provide a downloadable MATLAB implementation for our procedure [31] that start directly with the continuous environment and possible appearing/disappearing events on regions of interest. Also, we report a simplified real-time experiment that mimics the tackled problem.

The remainder of this paper is structured as follows. Section II briefly introduces some necessary preliminaries and states the problem we solve. The solution's steps are presented in Section III, and some conservativeness and complexity aspects are discussed in Section IV. Simulations and a simple experiment are included in Section V, and Section VI formulates some concluding remarks.

II. PROBLEM FORMULATION

Section II-A briefly introduces the formalism that will be used for specifying motion tasks and its connections with a finite transition system. Section II-B outlines the assumptions we make for solving the problem formulated in Section II-C.

A. Preliminaries

1) Linear Temporal Logic: In this work, we consider motion tasks given as formulae of $\text{LTL}_x$, which is a fragment of LTL [7]. With respect to standard LTL, $\text{LTL}_{<x}$ lacks the “next” operator, which is meaningless for continuous trajectories (as are those generated by a moving mobile robot). An $\text{LTL}_{<x}$ formula is recursively defined over a set of atomic propositions $\Pi$, by using the standard Boolean operators ($\neg$—negation, $\lor$—disjunction, $\land$—conjunction, $\rightarrow$—implication, and $\leftrightarrow$—equivalence) and the temporal operators ($\bigcirc$—always, $\diamond$—eventually, and $U$—until). By interconnecting these operators, one can obtain rich specifications, which we will use as motion specifications for mobile robots. Examples include tasks as navigation, surveillance, reachability of more regions in arbitrary or specific order, and convergence into regions. A formal definition of the syntax and semantics of $\text{LTL}_{<x}$ formulae can be found in [7] and [9], while examples will be provided in Section V.


\[ LTL \rightarrow X \] formulas are interpreted over infinite strings with elements from \( 2^\Omega \) (the set of all subsets of \( \Omega \), including the empty set \( \emptyset \)). Any \( LTL \rightarrow X \) formula over set \( \Pi \) can be transformed into a nondeterministic Büchi automaton (see Def. 1) that accepts all and only the input strings satisfying the formula [32]. Available software tools allow such conversions [33], [34].

**Definition 1:** The nondeterministic Büchi automaton corresponding to an \( LTL \rightarrow X \) formula over the set \( \Pi \) has the structure \( B = (S, \Sigma_B, \Sigma_B, \rightarrow, b, F) \), where:

- \( S \) is a finite set of states;
- \( S_0 \subseteq S \) is the set of initial states;
- \( \Sigma_B = 2^\Omega \) is the set of inputs;
- \( \rightarrow \subseteq S \times \Sigma_B \times S \) is the transition relation;
- \( F \subseteq S \) is the set of final states.

The transitions in \( B \) can be nondeterministic, meaning that from a given state there may be multiple outgoing transitions enabled by the same input. Thus, an input sequence can produce more than one sequence of states (called run).

An infinite input word (sequence with elements from \( \Sigma_B \)) is accepted by \( B \) if the word produces at least one run of \( B \) that visits set \( F \) infinitely often.

1) Transition Systems: For a partitioned environment cluttered with possibly overlapping and static regions of interest from set \( \Pi \), the movement capabilities of a mobile robot can be abstracted into a finite state transition system as in Def. 2. Environment partitions can be obtained with cell decomposition algorithms [35], [36], while details on abstractions suitable for different robot dynamics and cell shapes can be found in [37], [38]. The idea is that every cell from the partition corresponds to a single state in the finite state description, and transitions between states correspond to movement capabilities. A satisfaction map shows the regions of interest that are satisfied when the robot is inside a particular cell (under the note that each region is equal with a union of partition cells).

**Definition 2:** A finite state transition system is a tuple \( T = (Q, \delta_T, \Pi^T, \gamma) \), where:

- \( Q \) is the finite number of states (cells from partition);
- \( \delta_T \subseteq Q \times Q \) is the transition relation;
- \( \Pi^T = 2^\Omega \) is the set of possible observations (outputs of states of \( T \) ) yielded by a robot (the power set is used because the regions of interest can overlap);
- \( \gamma : Q \rightarrow \Pi^T \) is a satisfaction map, where \( \gamma(q) \) is the set of all regions from \( \Pi \) that contain cell labeled by \( q \), and \( \gamma(q) = \emptyset \) if cell \( q \) belongs to the left-over space (\( q \) is not included in any region of interest).

Transition systems that we use are deterministic, a fact which implies that any feasible transition in the current state can be chosen by imposing a specific robot control law. A run (or path) of \( T \) is an infinite sequence \( r = q_0 \delta_T q_1 \delta_T \ldots \), with the property that \( \langle q_i, q_{i+1} \rangle \in \delta_T, \forall i > 0 \). A run corresponds to a robot movement through cells from partition, and it induces (through map \( \gamma \) ) an output word, which is the observed sequence of elements from \( \Pi^T \). Since \( \Pi^T = \Sigma_B \), an output word of \( T \) is an input word for \( B \), and thus one can make the connection between robot trajectories and satisfaction of LTL specifications.

Research such as [3] and [39] proposed algorithms that abstract a robot motion into a transition system \( T \) and use model-checking inspired algorithms for controlling \( T \) such that it satisfies a given LTL formula. Unlike such works, in the scenario accounted here, the regions of interest appear and disappear based on probability density functions. These probabilistic information will be handled by modifying the finite state representation \( T \) and the path finding strategies from cases with static observations and by developing an automated monitoring strategy for mobile robots.

**B. Assumptions on Environment and Robots**

We assume a bounded environment where a team of \( n \) identical mobile robots evolve. Some polygonal regions of interest with fixed and known positions are defined in the environment, labeled with elements of set \( \Pi = \{ \pi_1, \ldots, \pi_{|\Pi|} \} \). Each region randomly alternates between being visible (appeared or active) and invisible (disappeared or inactive) on a time basis. Specifically, each region \( \pi_i \in \Pi \) disappears and appears after random delays given by negative exponential probability density functions with rates \( \lambda^a_i \) and \( \lambda^d_i \), respectively. Thus, the probability of having the region \( \pi_i \) visible at a random time moment is denoted by \( p_{\pi_i} \), and it is computed as the steady-state probability of a Markov process as follows:

\[
p_{\pi_i} = \frac{\lambda^a_i}{\lambda^a_i + \lambda^d_i}.
\]

The current state of a region is not known unless one robot visits the region. Regardless of the fact that a region is visible or invisible, its area can be accessed by a robot. However, in the case the region is disappeared and a robot is located over it, the robot does not sense the observation of the corresponding region. We mention that regions’ appearance/disappearance and robot movements are uncorrelated events, so it is possible that a region changes its state while a robot is visiting it.

As plausible real scenarios mimicked by such assumptions, one can imagine that the regions of interest are areas where fires can randomly ignite. Without exterior influence, a burning fire extinguishes after some time, and then it may reignite again. Other scenario may correspond to autonomous taxi agents, where the regions of interest are the pick-up stations and the region appearance/disappearance corresponds to passengers arrival/departure. Alternatively, consider regions of interest as areas where some goods appear, and they disappearance would be induced by becoming expired or by being picked by somebody else. In some scenarios, if the appearance and disappearance rates are not known, one can assume memory-less events and approximate the values \( p_{\pi_i} \forall \pi_i \in \Pi \) based on recordings made by sensors from the environment over time periods.

The environment is partitioned with respect to regions from \( \Pi \), thus obtaining a finite set of convex and polyhedral cells. Each robot is assumed to have a negligible size and a simple kinematic model. Such models include fully-actuated robots (\( x(\tau) = u(\tau) \), where \( x \) is the position and \( u \) the control), affine
or multi-affine dynamics in specific partitioned environments whose evolution can be abstracted to finite state representations [37], [38], [40], [41]. We require that each robot can be kept inside each partition region when needed (by applying a control law under which the region becomes invariant for the system model). Results from [42] can be further used for adapting control laws to real differential-wheel driven robots (as illustrated in the attached video).

The environment partition and the robot assumptions enable us to model the motion of each robot $r_i$, $i = 1, \ldots, n$ by a transition system $T_i = (Q_i, q_{0i}, \delta_i, 2^H_i, \gamma_i)$ as detailed in Def. 2. The relation $\delta_i$ is reflexive due to above robot requirement, and the robots are able to wait by following the self-loop in each state. Since any two robots $r_i$ and $r_j$ are identical, $\delta_i \equiv \delta_j$, and the only difference between models $T_i$ and $T_j$ is given by their initial states. For actually moving a robot between adjacent cells, one has to apply a sequence of control laws specifically designed for the robot model in the corresponding partition. For example, if the robot is fully actuated, one can use the simple strategy from [5]: the current robot position is linked by a line segment to the middle point of the common facet of the current and the next cell; thus, a run of $T_i$ corresponds to a robot trajectory formed by connected line segments, and it can be easily followed by a fully actuated point robot.

A centralized architecture is assumed, in the sense that the whole team of robots is controlled by a central unit that handles the necessary computations and communicates with all robots at any given time. The central unit knows the position of each robot at any given time, but it does not know which regions of interest are appeared and which are disappeared. More specifically, when a robot is inside the region $\pi_i$ it informs the central unit “I observe $\pi_i$” in the case that $\pi_i$ is currently visible, and it informs that “I do not see $\pi_i$” otherwise. The centralized architecture allows synchronization among robot motions, fact that enables the construction of a model for the whole team evolving in the environment with probabilistic information (Section III-A). Thus, the team movement is entirely guided by a control strategy (execution and monitoring) that is implemented on the central unit, with the process feedback corresponding to robot observations. Due to probabilistic regions, we can assign to a team movement a probability of observing a desired sequence of regions by multiplying the observation probabilities along runs. We are interested in computing a team run with maximum probability for observing a sequence of visible regions.

The assumptions presented here induce conservativeness on the developed method. The conservativeness and computational complexity of the solution presented in Section III will be discussed in Section IV.

C. Problem Statement

We aim to provide an algorithmic solution to the following problem.

Problem: Given a team of $n$ robots as assumed in Section II-B, a task as an $LTL_{-\chi}$ formula over the set $\Pi$, and the probabilities $p_{\pi_i}$ for each region from $\Pi$, find a control strategy for the robotic team that yields the maximum probability of satisfying the task.

Note that the $LTL_{-\chi}$ formula expresses a global mission for the whole team, rather than specifying individual tasks for each agent. In order to simplify the further exposition, we consider that the evolution of each robot is modeled by a transition system as in Def. 2, in accordance with the assumptions from Section II-B. This is accomplished by using a triangular decomposition algorithm [36].

The main steps of the solution we provide are presented in Section III. A discrete model for the whole team is constructed and combined with the Büchi automaton corresponding to the $LTL_{-\chi}$ formula, and an optimization problem is formulated and solved on the resulted automaton. The solution is projected to robotic trajectories and a run of the Büchi automaton is used for tracking the correctness of the execution. Based on the actual appearance of the visited regions of interest, an execution monitoring strategy decides whether the robot motion is continued, paused, or the solution is updated.

III. SOLUTION

A. Probabilistic Abstraction of the Robotic Team

Definition 3: The transition system modeling the movement capabilities of the whole team of robots and the probability of satisfying specific regions is $T_T = (Q_T, q_{0T}, \delta_T, \mathcal{O}_T, \gamma_T, \rho_T, \omega^n_T)$, where:

- $Q_T = Q^n$ is the set of states ($Q^n$ is the $n$-times cartesian product of $Q$ with itself);
- $q_{0T} = (q_{01}, q_{02}, \ldots, q_{0n}) \in Q_T$ is the initial state;
- $\delta_T \subseteq Q_T \times Q_T$ is the transition relation, with $\{(q_1, q_2, \ldots, q_n), (q_1', q_2', \ldots, q_n')\} \in \delta_T$ if and only if $(q_i, q_i') \in \delta_i, \forall i = 1, \ldots, n$;
- $\mathcal{O}_T = 2^\Pi$ is the set of possible observations;
- $\gamma_T : Q_T \rightarrow \mathcal{O}_T$ is the satisfaction map, with $\gamma_T(q_1, q_2, \ldots, q_n) = \bigcup^n_{i=1} \gamma(q_i)$;
- $\rho_T : Q_T \times \mathcal{O}_T \rightarrow [0, 1]$ is a probabilistic observation function that measures the probability of observing certain regions at a given state, i.e., $\rho_T(q, \mathcal{O})$ is the probability of observing (satisfying) all and only regions from $\mathcal{O} \in \mathcal{O}_T$ when the current state of $T_T$ is $q \in Q_T$. Computation of $\rho$ is described below (12);
- $\omega^n_T : \delta_T \rightarrow \mathbb{N}$ is a weighting function yielding the number of robots that move (change their cell) during a given transition, i.e., $\forall (q, q') \in \delta_T$, with $q = (q_1, q_2, \ldots, q_n), q' = (q_1', q_2', \ldots, q_n')$, then $\omega^n_T((q, q')) = \sum^n_{i=1} |\{q_i \{ q_i'\}|$.

Informally, $T_T$ captures the possible behaviors of the whole team of $n$ robots. Thus, states of $T_T$ are $n$-tuples in which the $i^{th}$ element shows the location of robot $r_i$, $i = 1, \ldots, n$. Transitions between states correspond to one or more robots changing their current cell from $Q$. When a transition implies that more robots change their cells, these changes are assumed to be synchronous, meaning that the moving robots cross the borders between cells at the same time. Such a behavior can be enforced by waiting modes enabled for robots that arrive faster at the border of the next cell they should visit.

So far, the construction of $T_T$ corresponds to a synchronous product of $n$ transition systems $T_i, i = 1, \ldots, n$, as the one used in [43]. Note that here we do not restrict states of $T_T$ such that at most one robot can be in a cell at a given state, nor we restrict...
transitions such that two robots swap their cells. Such behaviors could lead to collisions in the case of robots with non-negligible size, but in such situations one can assume that local rules are used for avoiding collisions. Otherwise, restricting $Q_T$ and $\delta_T$ such that collisions are avoided during the planning level would add additional conservatism to the solution, since $T_T$ would be more restrictive. In the actual situation of point robots, collisions can be simply ignored for the sake of clarity of the high level solution.

The set of observations obviously contains any possible subset of regions from $\Pi$ that are currently appeared and visited by at least one robot. For a given state $q \in Q_T$, $\gamma_T(q)$ gives the largest set of regions of interest that can be observed when the team is in configuration corresponding to tuple $q$. However, not all regions from $\gamma_T(q)$ may be appeared at a given time instant, and therefore we include map $\rho_T$ for having a probabilistic measure of the actual subset of regions from $\gamma_T(q)$ that may be currently observed. Under these explanations and using the probabilities (1) of regions from $\Pi$ to be visible, $\rho_T$ is computed as follows, for any $q \in Q_T$ and $p \in \mathcal{O}$:

$$\rho_T(q, p) = \begin{cases} 
\prod_{\pi \in p} \Pi_{\pi} & \text{if } p \subseteq \gamma_T(q) \\
0, & \text{otherwise}.
\end{cases}$$

(2)

In other words, the first case of (2) corresponds to the probability of observing exactly the regions from a specific subset $p$ of $\gamma_T(q)$ (including case $p = \emptyset$), and the overall probability is a product between individual probabilities of having some regions visible and others invisible. The last case corresponds to observing one or more regions of interest that are currently not occupied by the robots, hence the probability is zero.

The weighting function $\omega^p_T$ will be used for finding—all possible trajectories that yield the maximum probability of satisfying the $LTL^X$ specification—one with a minimum number of robotic movements. This will be accomplished in Section III-B, and here we mention that the current definition of $\omega^p_T$ can be replaced with different weighting maps, e.g. based on the expected traveled distance when robots move between adjacent cells, or based on energy consumed on a transition of $T_T$.

B. Multicriterion Optimal Solution

This subsection proposes a procedure for finding a run of $T_T$ that optimizes two criteria: 1) it has the greatest chance of producing a word over $2^{|I|}$ satisfying the $LTL^X$ formula and 2) it minimizes the total number of robot movements among all runs yielding the same optimum cost for criterion 1). The chance from criterion 1) of satisfying the specification comes from the fact that observations of $T_T$ are probabilistic. Thus, optimization on criterion 1) means maximizing the product of probabilities of observing certain sets of regions along the run, such that the sequence of these observations satisfies the formula. Since there may be multiple runs of $T_T$ that imply the same optimal value of criterion 1), we are interested in finding one that minimizes the total number of movements (or any other cost that can be embedded in weighting map $\omega^p_T$ from Section III-A).

We begin by converting the imposed $LTL^X$ specification into a Büchi automaton $B = (S, S_0, 2^{|I|}, \rightarrow_B, F, B)$ as in Def. 1, by using an available software tool [34]. Then, we construct a special type of product automaton between $T_T$ and $B$, in which we search for an optimal run.

Let us denote by $P_{(s, p, s')} \subseteq 2^{|I|}$ the set of all inputs of $B$ that enable a transition from $s$ to $s'$, i.e., $P_{(s, p, s')} = \{p \in 2^{|I|} | (s, p, s') \in \rightarrow_B\}$.

Definition 4: The product automaton $A = T_T \times B$ is constructed as follows: $A = (S_A, S_{A0}, \delta_A, \omega^p_A, \omega^m_A, F_A)$, where:
- $S_A = Q_T \times S$ is the set of states;
- $S_{A0} = \{q_0 T\} \times S_0$ is the set of initial states;
- $\delta_A \subseteq S_A \times S_A$ is the transition relation, defined by: $((q, s), (q', s')) \in \delta_A$ if and only if $(q, q') \in \delta_T$ and $s \in \mathcal{O}_T$ such that $\rho_T(q, p) > 0$ and $(s, p, s') \in \rightarrow_B$;
- $\omega^p_A : \delta_A \rightarrow [0, \infty)$ is a probability-based weighting function for transitions of $A$, defined by: $\omega^p_A((q, s), (q', s')) = -\log(\sum_{p \in P_{(s, p, s')}} \rho_T(q, p))$;
- $\omega^m_A : \delta_A \rightarrow \mathbb{N}$ is a movement-based weighting function for transitions of $A$, defined by: $\omega^m_A((q, s), (q', s')) = \omega^m_T((q, q'), \mathcal{V}(q, s), (q', s'))$;
- $F_A = Q_T \times F$ is the set of final states.

Product automaton $A$ extends the model-checking inspired algorithms from [3], [43] by including the available probabilistic information on region visibility in relation $\delta_A$ and map $\omega^p_A$. Construction of automaton $A$ mimics the one from [21], but adds the number of moving robots for each transition of $T_T$ in map $\omega^m_A$.

A transition in $A$ represents a matching condition between a transition of $T_T$ and a transition of $B$ caused by a possible current observation of $T_T$, as shown by relation $\delta_A$\(^1\). Since observations of $T_T$ are probabilistic, transitions of $A$ inherit a probabilistic nature, in the sense that a certain transition of $A$ is feasible (i.e. it can be taken at the current time) if the current state of $T_T$ yields certain observations (see from definition 4). The probability of existence for a transition of $A$ is captured by its cost assigned by map $\omega^p_A$, which represents the chance of being able to follow a transition in $A$ at a certain moment, without priori knowing the actual observation of $T_T$ at that moment. Therefore, the sum of observation probabilities from computation of $\omega^p_A$ encapsulates all observations of $T_T$ that can produce the same transition in $B$. A negation of logarithms of the probability-based weights in $A$ is performed because of the following aspect: for a given start and goal state of $A$, when one finds a path that minimizes the sum of transition costs along it (by standard graph search algorithms, e.g., Dijkstra [44], [45]), basically she finds the path whose product of probabilities is maximum\(^2\). The movement-based weighting function $\omega^m_A$ simply inherits the one from $T_T$, and it can be viewed as a measure of the energy spent by robots when the team follows transitions of $A$.

Cost function $\omega^m_A$ will be used for choosing a path that mini-

\(^1\)An alternative (and probably more intuitive) definition of $\delta_A$ can use the next observation of $T_T$ instead of the current one, while adding an initial dummy state to $T_T$ for accounting observation of $\delta_T$.

\(^2\)Multiplication is equivalent to sum of logarithms, thus minimization of $\sum_{i=1}^n \log(p_i)$ implies the maximization of $\prod_{i=1}^n p_i$, for any values $p_i \in [0, 1]$.\n
izes the total number of movements among all possible paths that optimize the probability-based measure, as next described.

The acceptance condition of $A$ is formulated similar to the one of $B$, in the sense that an infinite run is accepted by $A$ if and only if it visits infinitely often the set of final states $F_A$. If $A$ has at least one accepted run, then it has an accepted run in a prefix-suffix form, consisting of a finite sequence of states (called prefix) followed by infinite repetitions of another finite sequence (called suffix) formed by infinite repetitions of the string $T$.

By using graph searches, we can find in $A$ an accepted path that has a minimum sum of costs given by map $\omega^p_A$ along prefix and suffix. As noted above, this guarantees the maximum probability of having correct observations in $T_T$ (observations that keep feasible the followed transitions of $A$). Note that more than one graph searches are needed for finding an accepted path of $A$. First, an optimum path from each initial state to each final state should be found and stored; second, for each reachable final state an optimum path returning to this state should be found (some final states may not have self-loops); finally, a pair of paths from first and second steps is chosen such that the overall cost on this prefix-suffix run is optimized. Of course, one can simply impose different weights on prefix and suffix, for example by considering that the suffix is infinitely repeated.

Under this optimization guided by costs from map $\omega^p_A$, there may exist multiple paths of $A$ yielding the same overall cost. By projection to states of $T_T$, each such path would correspond to a team movement that maximizes the chance of yielding a sequence of observations satisfying the LTL formula. Therefore, we are interested in choosing a path that minimizes the total number of movements, i.e. that minimizes sum of costs given by $\omega^m_A$ while maintaining the optimal cost given by $\omega^p_A$.

Thus, we are facing the following multi-criterion optimization problem on automaton $A$: “find an accepted run of $A$ that: 1) minimizes the costs given by $\omega^p_A$ and 2) among the set of all paths yielding the same optimum cost from condition 1), it minimizes the costs given by $\omega^m_A$.” We mention that it may not be possible to solve the problem by constructing a combination between $\omega^p_A$ and $\omega^m_A$ and using a single optimization (graph search).

Several possible solutions for solving the above problem are given here.

- Find all paths that satisfy condition 1), and from these choose one that satisfies condition 2).
- Use a graph search to find the optimum cost from condition 1) (without storing an optimal path), and then solve an linear programming (LP) problem equivalent to a graph search, with cost function given by $\omega^m_A$, and with an equality condition imposing that the obtained solution has cost with respect to $\omega^p_A$ equal to the optimum one from condition 1).

Both solutions are untractable due to several reasons: the first optimization of the first solution can yield an infinite number of paths. Although formally correct, the implementation of the second optimization from the second solution generally fails because the mentioned LP equality constraint yields a narrow feasible set (or even empty due to numerical round-off errors) and the LP solvers from [46] usually return errors on overly stringent constraints that prevent finding a starting point, or on stalled residuals or relative error. Moreover, solutions are computationally complex (because of the first optimization from the first solution and the second one from the second solution, respectively), and thus their iteration for finding prefix and suffix of a run of $A$ may be unfeasible.

We developed a solution by modifying the Dijkstra's graph search [45] in Algorithm 1. The main idea is to overload the “smaller” operator from reals to vectors, based on the following rule: for $w_1, w_2 \in \mathbb{R}^2$ with $w_i = [c_i^p, c_i^m]^T$, $i = 1, 2$, where $c_i^p$ and $c_i^m$ are real-valued costs based on probability (map $\omega^p_A$) and number of movements (map $\omega^m_A$), then $w_1 < w_2$ ($w_1$ contains better costs than $w_2$) if either $c_1^p < c_2^p$ or $c_1^p = c_2^p$ and $c_1^m < c_2^m$. Thus, priority is given to optimize cost yielded by the probability-based map $\omega^p_A$, and movement-based costs $\omega^m_A$ are used for deciding the best among intermediate runs with the same probability. Computationally-wise, the algorithm is feasible, since it has a polynomial complexity and it finds in a single run all optimum paths from an initial node to all final nodes (this being a general property of Dijkstra's algorithm). Therefore, for finding an accepted run of $A$ we iterate the algorithm at most $|S_{A0}| + |F_A|$ times ($|S_{A0}|$ for finding optimum prefix from each initial state, $|F_A|$ for a suffix from each final state).

C. Execution Monitoring Strategy

Let us denote by $\text{run}_{A}$ the optimum path of $A$ obtained as described in Section III-B, where the suffix is marked by bold font. Path $\text{run}_{A}$ is projected to the run of $T_T$ denoted by $\text{run}_{T_T}$ = $q_{0T} q_{1T} q_{2T} \cdots q_{pT} s_p$, and the run of $B$
denoted by \( \text{run}_B = s_0s_1\ldots s_ns_{n+1} \ldots \). From construction of automaton \( A \), we have the guarantee that \( \text{run}_B \) is an accepted run of \( B \), and thus the team run \( \text{run}_{T_T} \) can produce a sequence of observations (inputs to \( B \)) that satisfies the \( \text{LTL}_{\neg X} \) formula. As specified in Section III-B, optimization on \( \text{run}_A \) guarantees the maximization of the probability that \( T_T \) produces an output word satisfying the \( \text{LTL}_{\neg X} \) formula, while minimizing the total number of robot movements (transitions between environment regions).

A straightforward projection of \( \text{run}_{T_T} \) to individual runs in transition systems \( T_i \), \( i = 1, \ldots, n \), yields the \( n \) robot trajectories (sequences of states in \( Q \)). As mentioned in Section III-A, the \( n \) robots synchronize when following the produced trajectories, i.e., they change partition regions at the same time. This is accomplished in the centralized architecture by waiting modes enabled at the borders shared by adjacent regions: when a robot reaches such a border, it stops and informs the central unit; when all moving robots (those changing their current region) reach their corresponding borders, the central unit sends a moving signal to all of them. The robots that do not change their region (they wait in the same region for a definite time of indefinite time) do not have to participate to such synchronization; if they have to remain in the same region only for a finite time (finite number of self-loops), they move to the border shared with the next region and wait there; otherwise, they converge inside the current region. Since the projection to runs of \( T_i \) and the mentioned synchronization procedure method are easy enough to understand, we do not introduce more formal notations for their description, and we focus on the synchronized team run \( \text{run}_{T_T} \).

When \( T_T \) follows its run, we have to check the current observation and decide if that produces the desired transition in \( \text{run}_B \). If it does, then \( T_T \) can further follow its path, and if it doesn’t we have to decide if the path should be readjusted or if the formula was violated. The current observation is the non-probabilistic measure (observation) due to the robots visiting the corresponding regions, and it can be viewed as an instance of a random variable generated based on the probability measure on observations of \( T_T \). In the following we give an algorithmic description of the method that tracks \( \text{run}_{T_T} \) and \( \text{run}_B \) and updates them when necessary.

Let us denote by \( h \in \mathcal{G}_T \) the current observation of \( T_T \) (the set of visible regions of interest that contain the positions of mobile robots at the current time instant). During team motion, \( B \) plays the role of a test tool for the satisfaction of \( \text{LTL}_{\neg X} \) formula. For accomplishing this, we store and update a set \( \mathcal{C}^B \subseteq S \) containing the possible current states of \( B \) that can be reached due to the inputs applied so far to automaton \( B \) (these inputs are observations of \( T_T \)). Initially, \( \mathcal{C}^B \) is initialized with \( \mathcal{C}^B = \{ S_0 \} \) and it is updated based on current \( h \) as follows:

\[
\mathcal{C}^B := \{ s \in S | \exists s' \in \mathcal{C}^B \text{ s.t. } (s', h, s) \in \rightarrow_B \}. \tag{3}
\]

During the team movement, whenever \( T_T \) changes its state, and at any change in \( h \), the monitoring strategy updates \( \mathcal{C}^B \) as in (3), checks which of the following cases is true and performs the corresponding action. The execution monitoring strategy begins with \( i = 0 \) (the robots are initially deployed in partition cells corresponding to \( q_{0i} = (q_{0i1}, q_{0i2}, \ldots, q_{0in}) \)).

Case 1) If \( \mathcal{C}^B \cap \{ s_{i+2}, s_{i+3}, \ldots, s_p \} \neq \emptyset \), then new runs in \( A, T_T \) and \( B \) are searched: consider \( q_{0i+1} = q_{0i} \) in \( T_T \), \( S_0 = C^B \) in \( B \), and correspondingly update the set of initial states \( S_{A0} \) of \( A \). Search for a new optimum \( \text{run}_A \), find \( \text{run}_{T_T} \), \( \text{run}_B \) and individual robot runs, then restart the monitoring strategy according to these paths.

Case 2) If \( s_{i+1} \in \mathcal{C}^B \), then \( T_T \) advances to the next state \( q_{i+1} \).

Case 3) If \( s_{i+1} \notin \mathcal{C}^B \), but \( s_i \in \mathcal{C}^B \), then \( T_T \) stays (self-loops) in \( q_{i} \).

Case 4) If \( s_i, s_{i+1} \notin \mathcal{C}^B \), but \( \mathcal{C}^B \neq \emptyset \), then the paths to be followed from now on are modified exactly as in 1).

Case 5) If \( \mathcal{C}^B = \emptyset \), then stop the movement and report that the \( \text{LTL}_{\neg X} \) formula was just violated by the current observation \( h \).

Informally, Case 1) means that the current observation \( h \) produces a faster advancement along \( \text{run}_B \) and thus towards formula accomplishment. To optimize team movement, new runs are computed by considering the current robot positions as initial states in \( T_i \), \( T_T \), and the current states from \( \mathcal{C}^B \) as initial set in \( B \). Case 2) means that until now the desired run of \( B \) is produced by the actual observation of \( T_T \). Case 3) means that the desired run of \( B \) does not advance, but the current observation \( h \) enables a self-loop in its actual state. Thus, robots wait in their current regions for a change in observation of \( T_T \)—such a change will eventually appear, because \( \text{run}_{T_T} \) can be produced by some probabilistic observations along \( \text{run}_{T_T} \). In Case 4), the current observation \( h \) prevented \( \text{run}_{T_T} \) of being followed, but it did not block the entire evolution of \( B \). Therefore, we search for a different continuation of the evolution that satisfies the formula. Case 5) can appear only when any feasible team trajectory can produce (besides desired observations) some observations that are forbidden by the \( \text{LTL}_{\neg X} \) formula (i.e., any path of \( T_T \) that can produce an observation sequence satisfying the formula can also produce an observation sequence that violates it). Without loss of generality, Case 5) considers that automaton \( B \) blocks for unfeasible inputs, rather than including an error sink state.

Examples that illustrate situations in which the above cases appear will be included in Section V. Case 4) may appear only in specific situations, such an exemplification being given in the next paragraph. Informally, replanning from Case 4) is likely to be encountered when the current team deployment has more possible observations, while the formula satisfaction imposes different further evolutions of the system based on particular combinations of those observations.

Occurrence of Case 4). For illustrating the necessity of Case 4) in the monitoring strategy, let us consider an environment with four regions. When appearing, regions \( \pi_1 \) and \( \pi_2 \) overlap, but they have different visibility probabilities. Regions \( \pi_3 \) and \( \pi_4 \) are disjoint. For simplicity, a single robot is considered, with the \( \text{LTL}_{\neg X} \) task \( \langle \pi_1 \lor \pi_2 \rangle \land \neg \pi_1 \cup \pi_3 \lor \pi_2 \cup \pi_4 \rangle \). Thus, the robot should eventually visit \( \pi_1 \) or \( \pi_2 \). The robot should also observe \( \pi_3 \) or \( \pi_4 \), but \( \pi_3 \) should be reached only without observing \( \pi_1 \) until then, while \( \pi_4 \) can be reached if \( \pi_2 \) were not observed. Assume that \( p_{\pi_1} > p_{\pi_2} \) and \( p_{\pi_3} < p_{\pi_4} \). The initial computed trajectory will first drive the robot towards the workspace area.
corresponding to \( \pi_1 \) and \( \pi_2 \) (with the “hope” of observing the region with higher visibility probability, \( \pi_1 \)), and after that towards \( \pi_4 \). If the robot reaches the cell corresponding to \( \pi_1 \) and \( \pi_2 \), but \( \pi_1 \) is invisible and \( \pi_2 \) happens to be appeared, then Case 4) becomes active in the monitoring strategy: \( \beta \) reaches states that are not in the current run of the Büchi automaton. Informally, the planned visit to \( \pi_4 \) is not feasible anymore and new paths of the Büchi automaton are computed (these will drive the robot towards \( \pi_3 \)). Of course, the above formula would be violated (Case 5) if \( \pi_1 \) and \( \pi_2 \) were both visible when the robot reaches their corresponding region. As one can observe from this example, Case 4) can appear only for certain specifications and environments, fact that supports our statement that it is a rarely encountered case in the monitoring strategy.

We note that Cases 1) and 4) are related in the sense that they both modify the expected way of evolving along run of the Büchi automaton, because \( T_T \) outputs an observation that has lower probability that the one(s) expected such that run is properly followed. In such a situation, Case 1) is the effect of a “lucky” observation, while Cases 4), 3), 5) are produced by an “unlucky” one. In Case 4) it is necessary to find new runs, since the actual ones cannot be further followed (see the above example), while in Case 1) one could continue with the actual runs (as a trade off between saving computing time on the central unit and evolving faster towards formula satisfaction). The latter situations means that Case 1) is simply removed from the monitoring strategy, and only the remaining cases are tested. We mention that Case 1) does not appear in the simplified strategy from [21].

The execution strategy cannot yield livelock behaviors for the team motion, because there is a strict advance along run of the Büchi automaton in Cases 1) and 2), while Case 3) is certainly exited in finite time. Only replanning from Case 4) could induce cyclic behaviors, but such cycles are eventually left due to the strictly positive probability of observing the desired regions. Case 5) informs the user about the deadlock in the robot motion.

The probability-based optimization from Section III-B implies that along the evolution Case 2) is most likely to appear, because run is correlated with the most likely observations in team positions along run. Cases 1) or 4) rarely appear, and Case 5) can appear only when there’s no way of satisfying the formula while avoiding its possible violation. The waiting times from Case 3) are also minimized by large probabilities of observing the desired outputs along run. The above cases underline the efficiency of choosing the probability-based criterion as the primary optimization objective in Section III-B, not only with respect to formula satisfaction, but also with respect to computation complexity, as in the frequently occurring Case 2) no additional computations are needed.

IV. CONSERVATISM AND COMPLEXITY

The solution we provide for targeted problem is conservative due to several reasons.

Some sources of conservatism are induced by the assumptions from Section II, where robot collisions are ignored during trajectory construction. One way to ensure collision free movements was mentioned after Def. 3, by removing states and transitions in \( T_T \) such that robots do not swap cells, and at most one robot can be in a cell at any time. However, this would restrict the team motions and may lead to loose solutions. In future work, we intend to incorporate techniques inspired from Resource allocation systems [15] such that the trajectories found as in Section III are correctly followed. For this, inspiration can be taken from [16], where specific Petri net models and monitoring rules are designed for non-synchronized trajectories, such that the maximum number of robots in a cell is limited by a desired capacity and collisions are avoided.

Other conservatism sources result from the solution we provide in Section III, as follows.

- In the finite-state team model from Section III-A, the probabilistic observation map takes into account only the current state. Such memoryless probability measures are generally used in planning approaches based on probabilistic information. Otherwise, the construction of abstractions that have history-dependent probabilities rather than state-dependent ones yields models with more states even for simple scenarios [47], [48].
- While the synchronous movement of the team can be ensured by the centralized architecture, the involved waiting modes induce more communication signals and robot motions with frequent stops. A method for reducing such synchronization moments when a team moves in a static environment was proposed in [49], but such an approach cannot be directly used in the current probabilistic setting.
- The first optimization criterion from Section III-B relies solely on probabilistic data, while the number of traversed states is accounted as the second optimization criterion. Although it is possible to obtain a much longer run with a good probability instead of a shorter run with a slightly worse probability, we use this approach because of the maximum probability of satisfying the formula without the need of recomputing runs during movement. A weighting between probability and movement measures could be used as a single optimization criterion, but finding proper weights may be a challenging problem with a heuristic solution based on the variation ranges of available data.
- As next detailed, the number of states in the team model increases exponentially with the number of robots, fact that prevents the method applicability to larger robot teams.

The computational complexity of our approach mainly arises from the steps presented in Section III. If \( |Q| \) denotes the number of states of a robot model \( T_r \), then the team model \( T_T \) has \( Q^n \) states. The Büchi automaton \( \beta \) corresponding to an LTL formula \( \phi \) has at most \( |\phi| \cdot 2^{|\phi|} \) states, where \( |\phi| \) denotes the size of the formula, given by the number of temporal operators [34]. Thus, the bottleneck of our approach is given by the number of states of the product automaton \( A \), \( S_A < Q^n \cdot |\phi| \cdot 2^{|\phi|} \). For finding an optimal accepted path in this automaton we run the modified Dijkstra’s algorithm \((|S_A| + |F_A|)\)-times and then make \( |S_A| \cdot |F_A| \) comparisons (Section III-B). Each Dijkstra’s run has the order of complexity \( O(|S_A|^2) \) [45].\(^5\) From Def. 4, \( S_{A0} < |F_A| \leq |S_A| \), and thus the complexity order for finding an accepted path is

\(^4\)For usual specifications, the number of states of \( \beta \) is significantly smaller than the given upper-bound.

\(^5\)Optimized Dijkstra implementations use a sorted node list and have a smaller complexity of order \( O(\log(|S_A|)) \).
is $O((|Q|^n \cdot \phi^1 \cdot 2^{|\phi|^1})^3)$. The execution monitoring strategy (Section III-C) may require some reruns of optimization problem from Section III-B, but the number of these updates is reduced due to the maximized probability of satisfying the formula. All of the mentioned computation burden is carried out by the central unit, which generally uses powerful computation resources, while the robots have to execute the received moving/stopping commands and to inform the central unit of their current positions and observed regions of interest.

The computational complexity can be reduced by following two ideas.

- The execution monitoring strategy triggers trajectory updates only in Cases 1) and 4) (Section III-C). As mentioned, the updates from Case 1) are optional, since they imply robot motions for faster accepting a run in $B$, rather than motion pausing and following the current run. Thus, the central unit can begin the computation for Case 1) (robot motion being paused), and during this computation if the robots observe regions that were desired to be visible in the current position, then the central unit interrupts Case 1) and the previous trajectories are followed. Trajectory updates from Case 4) are necessary, since conditions triggering this case are disjoint from those triggering the other cases. However, Case 4) rarely appears for usual specifications and environments.
- The number of states of robot models $T_i$ and of team model $T_T$ can be greatly reduced by collapsing (fusing) into a single state from $T_i$ the adjacent cells from the partition that belong to the same region of interest or that are not included in any region. Informally, in such a case any transition in $T_i$ changes the value of the satisfaction map $\gamma$, but it does not return a sequence of cells to be traversed by the robot. This sequence can be found by a search algorithm on the collapsed cells. Formally, the robot models obtained from such collapsing are simulating quotients of $T_i$ with respect to map $\gamma$, while the correctness of the solution results from the so-called “closeness under stuttering” property of $LTL$ $x$ $[50]$. However, the team trajectory cannot be optimized by considering the number of robot movements, since map $\omega^T$ from $T_T$ becomes irrelevant. This procedure is not formally described in the current work, but its computational effects are mentioned in simulations from Section V.

All presented algorithms were implemented in the MATLAB environment $[31], [46]$, and they include tools from $[34], [36]$ for partitioning the environment and for converting an LTL formula into a Büchi automaton. Simulation examples and involved computation times are included in the next section.

V. EXAMPLES

The simulations presented in this section were created by using our freely-downloadable software package $[31]$. We consider the planar environment from Fig. 1, partitioned in 38 triangular cells denoted by $q_1, \ldots, q_{38}$ and including six regions of interest: $I = \{\pi_3, \ldots, \pi_6\}$. Each region of interest is a union of adjacent triangular cells, as follows.

- $\pi_3$ is composed by cells $q_4, q_5, q_{17}, q_{18}, q_{20}, q_{21}, q_{25}, q_{26}$.
- $\pi_2$ is composed by cells $q_2, q_4, q_5, q_{14}, q_{16}, q_{17}, q_{20}$.

We consider the LTL specification

$$\phi_1 = \square \pi_3 \land \Diamond \left( (\pi_1 \lor \pi_2) \land \pi_4 \land \Diamond (\pi_5 \land \pi_6) \right).$$

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig1.png}
\caption{Environment partitioned in triangular cells, with six regions of interest. Initially, all regions are appeared, and two robots are placed in centroids of cells $q_{39}$ and $q_{37}$. Region $\pi_1$ is colored with red, $\pi_3$ with blue, $\pi_5$ with green, $\pi_2$ with cyan, $\pi_6$ with magenta and $\pi_5$ with gray. When both $\pi_1$ and $\pi_2$ are visible, they overlap in cells $q_4, q_6, q_{17}, q_{20}$.}
\end{figure}
Informally, $\pi_3$ should be always avoided, $\pi_4$ and $\pi_1$ or $\pi_2$ should be eventually occupied and then the robots should reach a position where $\pi_5$ and $\pi_6$ are observed.

The procedures described in Section III-B are performed for finding a team trajectory. The Büchi automaton corresponding to $\phi_1$ has 3 states, while the product automaton $A$ has 4332 states and it was constructed in 6 seconds. The multi-criterion optimization took 17 s and, after projection to a path in $T_2$, the following sequence of cell pairs should be synchronously followed by the two robots:

$$r_{un}r_{T_2} = (q_{20}, q_{37}), (q_{28}, q_{30}), (q_{26}, q_{22}), (q_{14}, q_{25}), (q_{13}, q_{20}), (q_{10}, q_{20}), (q_{6}, q_{8}), (q_{3}, q_{7}), (q_{18}, q_{1}) \ldots (4)$$

The suffix is marked in bold font and means that the robots should stop in states $q_{15}$ and $q_1$, respectively. The projection of $r_{un}A$ to $r_{un}B$ is not given, because of the abstract nature of the states of $B$. Instead, we informally interpret the above robotic path with respect to $r_{un}B$ as follows. First, the robots head towards cells $q_{10}$ and $q_{20}$, respectively, because in these positions there is the best chance of observing $\pi_4$ ($q_{10}$ belongs to region $\pi_4$), and $\pi_1$ or/and $\pi_2$ ($q_{20}$ belongs to both regions $\pi_1$ and $\pi_2$). Then, the robots go to $q_{15}$ and $q_1$, respectively (where regions $\pi_5$ and $\pi_6$ can be observed). As imposed by the formula, region $\pi_3$ should not be observed, even if it was always appeared.

The number of traversed cells is minimized due to the second optimization criterion from Section III-B, e.g., the closer simplex $q_{10}$ is visited instead of farther cells $q_4$, $q_8$ or $q_{17}$, although all of them belong to both $\pi_1$ and $\pi_2$.

An obtained execution is represented in the snapshots from Fig. 2. During this execution, the execution monitoring strategy from Section III-C encounters only Cases 2) and 3) (advancing along the designated path of $T_2$, or pausing the movement until $r_{un}B$ advances due to some desired observation of $T_2$). Thus, Fig. 2(a) corresponds to the situation in which region $\pi_4$ disappeared when the robots are in the second marked position along their trajectories (this event is not known to the central unit, because no robot is currently inside the area corresponding to $\pi_4$). The robots continue to move and they synchronize when entering $q_{28}$ and $q_{30}$, respectively (third marked position along each trajectory). In Fig. 2(b), region $\pi_6$ disappeared and the robots continue to move towards next cells. In Fig. 2(c), the robots reach cells $q_{10}$ and $q_{20}$. In this moment they do not observe $\pi_4$ and ($\pi_1$ or $\pi_2$), and $r_{un}B$ cannot advance. Case 3) from the monitoring strategy is activated, and the robots wait until $[\pi_1 \vee \pi_2] \land \pi_4$ becomes true. In the snapshot from Fig. 2(d), first $\pi_1$ appeared, then $\pi_4$, and now the robots can continue their paths towards $q_{15}$ and $q_1$. In Fig. 2(e), the robots wait for the appearance of $\pi_5$ and $\pi_6$ [the execution monitoring strategy is in Case 3)]. In Fig. 2(f), the formula was satisfied once $\pi_5$ and $\pi_6$ became visible; the robots converge to the centroids of cells $q_{15}$ and $q_1$ and remain there (suffix of $r_{un}r_{T_2}$ has length one).

Example 2: For the scenario depicted in Fig. 1, we now consider the following LTL task:

$$\phi_2 = \Box \neg \pi_3 \land \Diamond (\pi_1 \lor \pi_2 \lor \pi_4) \land \neg (\pi_5 \lor \pi_6) U (\pi_5 \land \pi_6).$$

Informally, $\pi_3$ should be always avoided, either of regions $\pi_1$, $\pi_2$, $\pi_4$ should be eventually visited, and eventually the robots should simultaneously observe regions $\pi_5$ and $\pi_6$ (because the last part of the formula requires that neither $\pi_5$ nor $\pi_6$ are observed until both of them are observed at the same time).

The Büchi automaton corresponding to $\phi_2$ has 4 states and the product automaton $A$ has 5776 states. Optimum run of $A$ was obtained in 33 seconds, and its projection to $T_2$ yields exactly the run from (4). Of course, $r_{un}B$ is different due to the different structure of $B$. An intuitive interpretation of $r_{un}r_{T_2}$ is in accordance with the requirements of $\phi_2$: the robots head towards cells $q_{10}$ and $q_{20}$, respectively, because here they have the largest probability of observing either one of regions $\pi_1$, $\pi_2$, $\pi_4$. Then, they go to $q_{15}$ and $q_1$, respectively and enter at the same time, with the “hope” that both $\pi_5$ and $\pi_6$ are appeared, such that they are simultaneously observed.

Fig. 3 shows a possible execution. In Fig. 3(a), when the robots enter $q_{34}$ and $q_{28}$, region $\pi_1$ is appeared, so it is observed by the second robot. Therefore, $r_{un}B$ is advanced faster and Case 1) from the monitoring strategy is active (intuitively, the robots don’t have to go anymore to $q_{10}$ and $q_{20}$ for observing $\pi_1$, $\pi_2$ or $\pi_4$, because this part of the formula was just satisfied). A new run in $A$ is computed according to the current position of robots and the current state of $B$. As a result, the robots should go directly to $q_{15}$ and $q_1$ on the paths shown in Fig. 3(b). Note that valuable energy for moving the robots may be saved due to activating Case 1); however, if additional computation on the central unit is to be avoided, Case 1) can be simply ignored (as mentioned in Section IV) and the robots continue their initial trajectories. It happened that both $\pi_5$ and $\pi_6$ were appeared when the robots entered cells $q_{15}$ and $q_1$, and therefore the formula is satisfied. Intermediary snapshots when regions appear and disappear are not included, since they did not affect the evolution.

Fig. 4 shows a situation in which the formula $\phi$ is violated. The robots evolve along paths from (4). It happened that $\pi_1$ has not appeared until cells $q_{10}$ and $q_{20}$ were reached, and thus $r_{un}B$ was not advanced faster. In the snapshot from Fig. 4(a), $\pi_4$ is observed by the second robot and the robots continue along the initial run of $T_2$ towards cells $q_{15}$ and $q_1$. When $q_{15}$ and $q_1$ are simultaneously reached [Fig. 4(b)], $\pi_6$ is appeared but $\pi_5$ is disappeared. Case 5) becomes active, and the formula is violated (recall that the last part of $\phi_2$ requires that none of $\pi_5$ and $\pi_6$ are observed until both are observed). Since the robots cannot see the appeared or disappeared regions before entering in the corresponding cells, there is no algorithmic method of adjusting the movement such that the formula is satisfied in such a situation.

For supporting the rapid increase in computational demands mentioned in Section IV, we have performed tests on more robots. We mention that our implementation does not include

94 If the model reduced by cell collapsing were used, the time for constructing $A$ and finding a path is around 1 s.
Fig. 2. Snapshots along execution for Example 1. Positions of the two robots are marked for moments when they synchronize and when a region appears or disappears. The execution monitoring strategy was in Case 2) in snapshots (a), (b), (d), (f) and in Case 3) in snapshots from (c), (e). Explanations are included in the text and briefly in the yellow callouts.
Fig. 3. Snapshots along a possible execution for Example 2. (a) Case 1) from the monitoring strategy becomes active and new trajectories are found in 27 s. (b) When the robots synchronously enter $q_{13}$ and $q_8$, both regions $\pi_r$ and $\pi_g$ were appeared, and the formula is satisfied. Except for (a), only Case 2) from the execution monitoring strategy was active during this execution.

Fig. 4. Possible execution leading to violating the formula from Example 2. (a) Robots reach $q_{16}$ and $q_{23}$. $\pi_d$ is observed, and the movement along initial path is continued. (b) Case 5) from the monitoring strategy becomes active: when the robots synchronously enter $q_{13}$ and $q_1$, only $\pi_g$ is observed, and the formula is violated.

Techniques that could probably optimize the construction or computation on the involved finite state descriptions, and it may not be so efficient as dedicated model checking software tools. Thus, for three robots evolving in the above environment and an LTL specification whose Büchi automaton has three states, the product automaton $A$ has order of $11^{10}$ states and a solution was obtained in more than 10 h. By using the reduced robot models, a solution was obtained in less than 1 min. For a team with four robots and for reduced models, the computation increased to around 3 h. Although the overall computation times are large, we have observed that while size of $A$ increased, less than 25% of the reported times was used for finding a run (this being the part that might be reiterated). These times show that the proposed centralized approach is applicable only for a few robots, and suggest that further research can be conducted for developing decentralized techniques or different abstraction models for similar problems.

A simple real-time experiment performed on the experimental platform from [51] is annexed as a video to this work. The experiment considers one robot and emulates the region disappearances on appearances by externally covering or uncovering them. The task requires that the robot always avoids red regions, first visits the green region, and then visits the blue one. Since we cannot emulate the appearance of a region when the robot is inside it, we use information from an overhead video camera and pause the robot motion in the previous cell.
VI. CONCLUSION

This work presents a method for controlling the motion of a small robotic team based on an LTL formula over a set of regions of interest from a partitioned environment. The regions of interest alternate between appearance and disappearance based on exponential probability density functions with known rates. For accounting this aspect, the robotic team is modeled by a finite transition system with probabilistic observations. We adapt model checking algorithms and graph search procedures for finding a path of the transition system that is most likely to satisfy the formula while optimizing the traveled distance. The robots are moved according to projections of this path. For accounting this aspect, the robotic team is modeled based on exponential probability density functions with known rates. For accounting this aspect, the robotic team is modeled by a finite transition system with probabilistic observations. The method is implemented as a freely downloadable software package [31], and several case studies are included for supporting the developed solution.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers and the Associate Editor for their useful comments and suggestions.

REFERENCES


