LTL-based Planning in Environments with Probabilistic Observations * -draft-

Marius Kloetzer and Cristian Mahulea ‡

January 3, 2017

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.

Published as:
M. Kloetzer and C. Mahulea, "LTL-based Planning in Environments with Probabilistic Observations," IEEE Transactions on Automation Science and Engineering, 12(4): 1407 - 1420, October 2015. DOI: [10.1109/TASE.2015.2454299]

*This work was partially supported by projects DPI2014-57252-R at University of Zaragoza, CEI Iberus Mobility Grants funded by the Ministry of Education of Spain, and PN-II-RU-TE-2014-4-1604 at University of Iasi

†M. Kloetzer is with Department of Automatic Control and Applied Informatics, Technical University of Iasi, Romania email: {kmarius@ac.tuiasi.ro}

‡C. Mahulea is with Aragón Institute of Engineering Research (I3A), University of Zaragoza, Spain, email: {cmahulea@unizar.es}

1 Introduction

Recent researches on mobile robots proposed different methods for extending the motion tasks to rich and human-like specifications [1, 2, 3, 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, 8, 9], and solutions were developed by appropriately adjusting and extending the available algorithms. Furthermore, another challenge lies in the the extension of solutions available for a single robot to multi-robot scenarios [10, 4, 1, 11].

In this paper, we formulate robotic missions for a team of cooperating robots by Linear Temporal Logic (LTL) formulae, which is, together with its fragments, *e.g.*, GR(1), among common choices for allowing expressive control objectives [10, 3, 2, 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, 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, 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, 18, 19, 20, 21]. All 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. Works [18, 19, 20] assume a similar scenario with the one we consider, but the underlying system model and formal methods are entirely different, as follows. The robotic model in [18, 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, 19, 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 notes, 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 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, 25, 26, 27]. Works as [28, 29, 30] combine surveillance LTL tasks over non-varying 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 from 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 the paper is structured as follows. Sec. 2 briefly introduces some necessary preliminaries and states the problem we solve. The solution's steps are presented in Sec. 3, and some conservativeness and complexity aspects are discussed in Sec. 4. Simulations and a simple experiment are included in

Sec. 5, and Sec. 6 formulates some concluding remarks.

2 Problem Formulation

Subsection 2.1 briefly introduces the formalism that will be used for specifying motion tasks and its connections with a finite transition system. Subsection 2.2 outlines the assumptions we make for solving the problem formulated in subsection 2.3.

2.1 Preliminaries

Linear Temporal Logic

In this work, we consider motion tasks given as formulae of LTL_{-X}, which is a fragment of Linear Temporal Logic (LTL) [7]. With respect to standard LTL, LTL_{-X} lacks the "next" operator, which is meaningless for continuous trajectories (as are those generated by a moving mobile robot). An LTL_{-X} formula is recursively defined over a set of atomic propositions Π, by using the standard Boolean operators ($\neg$ - negation, $\vee$ - disjunction, $\wedge$ - conjunction, $\Rightarrow$ - implication, and $\Leftrightarrow$ - equivalence) and the temporal operators ($\Box$ - always, $\diamondsuit$ - eventually, and $\mathcal{U}$ - 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, convergence into regions. A formal definition of the syntax and semantics of LTL_{-X} formulas can be found in [7, 9], while examples will be provided in Sec. 5.

LTL_{-X} formulas are interpreted over infinite strings with elements from 2^{Π} (the set of all subsets of Π, including the empty set $\emptyset$). Any LTL_{-X} formula over set Π 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_{-X} formula over the set Π has the structure $B = (S, S_0, \Sigma_B, \rightarrow_B, F)$, where:*

- *S is a finite set of states;*
- *$S_0 \subseteq S$ is the set of initial states;*
- *$\Sigma_B = 2^{\Pi}$ is the set of inputs;*
- *$\rightarrow_B \subseteq S \times \Sigma_B \times S$ is the transition relation;*
- *$F \subseteq S$ is the set of final states.*

The transitions in B can be non-deterministic, 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 Σ_B) is *accepted* by B if the word produces at least one run of B that visits set F infinitely often.

Transition systems

For a partitioned environment cluttered with possibly overlapping and static regions of interest from set Π, 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* $T = \left(Q, q_0, \delta_T, \Pi^T, \gamma\right)$, *where:*

- Q *is the finite number of states (cells from partition);*
- q_0 *is the initial state (cell containing the initial deployment of robot);*
- $\delta_T \subseteq Q \times Q$ *is the transition relation, where* $(q, q') \in \delta_T$, *with* $q \neq q'$, *if and only if the robot can be steered from any initial condition in* q *to the adjacent cell* q' *without visiting any other neighboring cells in finite time, and* $(q, q) \in \delta_T$ *if the robot can be kept inside cell* q *for an indefinite time;*
- $\Pi^T = 2^{\Pi}$ *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);*
- $\gamma : Q \to \Pi^T$ *is a satisfaction map, where* $\gamma(q)$ *is the set of all regions from* Π *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, 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 $r = q_0\, q_1\, q_2 ...$, with the property that $(q_i, q_{i+1}) \in \delta_T$, $\forall i \geq 0$. A run corresponds to a robot movement through cells from partition, and it induces (through map γ) an output *word*, which is the observed sequence of elements from Π^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.

Researches as [39, 3] 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.

2.2 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 λ_i^d and λ_i^a, respectively. Thus, the *probability* of having the region π_i visible at a random time moment is denoted by p_{π_i} and it is computed as the steady-state probability of a Markov process:

$$p_{\pi_i} = \frac{\lambda_i^a}{\lambda_i^a + \lambda_i^d} \tag{1}$$

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/dissapearance 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 memoryless events and approximate the values p_{π_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 Π, 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 ($\dot{x}(\tau) = u(\tau)$, 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 [40, 37, 38, 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 $T_i = \left(Q, q_{0i}, \delta_i, 2^{\Pi}, \gamma\right)$ as detailed in Def. 2. The relation δ_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 if 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 π_i it informs the central unit "I observe π_i" in the case that π_i is currently visible, and it informs that "I do not see π_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 (Sec. 3.1). 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 Sec. 3 will be discussed in Sec. 4.

2.3 Problem statement

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

Problem: Given a team of n robots as assumed in subsection 2.2, a task as an LTL_{-X} formula over the set Π, and the probabilities p_{π_i} for each region from Π, find a control strategy for the robotic team that yields the maximum probability of satisfying the task.

Note that the LTL_{-X} 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 subsection 2.2. This is accomplished by using a triangular decomposition algorithm [36].

The main steps of the solution we provide are presented in Sec. 3. A discrete model for the whole team is constructed and combined with the Büchi automaton corresponding to the LTL_{-X} 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.

3 Solution

3.1 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* $T_T = (Q_T, q_{0T}, \delta_T, \mathcal{O}_T, \gamma_T, \rho_T, \omega_T^m)$*, 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;*
- $\gamma_T : Q_T \to \mathcal{O}_T$ *is the satisfaction map, with* $\gamma_T((q_1, q_2, \ldots, q_n)) = \bigcup_{i=1}^{n} \gamma(q_i)$*;*
- $\rho_T : Q_T \times \mathcal{O}_T \to [0, 1]$ *is a probabilistic observation function that measures the probability of observing certain regions at a given state, i.e.,* $\rho_T(q, \mathbf{p})$ *is the probability of observing (satisfying) all and only regions from* $\mathbf{p} \in \mathcal{O}_T$ *when the current state of* T_T *is* $q \in Q_T$*. Computation of* ρ *is described below (equation* (2)*);*
- $\omega_T^m : \delta_T \to \mathbb{N}$ *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* $\omega_T^m((q, q')) = \sum_{i=1}^{n} |\{q_i\} \setminus \{q_i'\}|$*.*

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 δ_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 Π 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 ρ_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 Π to be visible, ρ_T is computed as follows, for any $q \in Q_T$ and $\mathbf{p} \in \mathcal{O}_T$:

$$\rho_T(q, \mathbf{p}) = \begin{cases} \left(\prod_{\pi \in \mathbf{p}} p_\pi\right) \cdot \left(\prod_{\pi \in \gamma_T(q) \setminus \mathbf{p}} (1 - p_\pi)\right), \\ \qquad \text{if } \mathbf{p} \subseteq \gamma_T(q) \\ 0, \text{ otherwise.} \end{cases} \tag{2}$$

In words, the first case of (2) corresponds to the probability of observing exactly the regions from a specific subset $\mathbf{p}$ of $\gamma_T(q)$ (including case $\mathbf{p} = \emptyset$), 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 ω_T^m will be used for finding - among 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 subsection 3.2, and here we mention that the current definition of ω_T^m 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.

3.2 Multi-criterion 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^Π 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 ω_T^m from subsection 3.1).

We begin by converting the imposed LTL_{-X} specification into a Büchi automaton $B = \left(S, S_0, 2^\Pi, \rightarrow_B, F\right)$ 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 $P_{(s \rightarrow_B s')} \subseteq 2^\Pi$ the set of *all* inputs of B that enable a transition from s to s', *i.e.* $P_{(s \rightarrow_B s')} = \{\mathbf{p} \in 2^\Pi \,|\, (s, \mathbf{p}, s') \in \rightarrow_B\}$.

Definition 4 *The product automaton $A = T_T \times B$ is constructed as follows: $A = (S_A, S_{A0}, \delta_A, \omega_A^p, \omega_A^m, F_A)$, where:*

- $S_A = Q_T \times S$ *is the set of states;*
- $S_{A0} = \{q_{0T}\} \times S_0$ *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* $\exists \mathbf{p} \in \mathcal{O}_T$ *such that* $\rho_T(q,\mathbf{p}) > 0$ *and* $(s,\mathbf{p},s') \in \rightarrow_B$*;*
- $\omega_A^p : \delta_A \rightarrow [0,\infty)$*, is a* probability-based *weighting function for transitions of A, defined by:* $\omega_A^p((q,s),(q',s')) = -\log\left(\sum_{\mathbf{p} \in P_{(s \rightarrow_B s')}} \rho_T(q,\mathbf{p})\right)$*,* $\forall ((q,s),(q',s')) \in \delta_A$*;*
- $\omega_A^m : \delta_A \rightarrow \mathbb{N}$ *is a* movement-based *weighting function for transitions of A, defined by:* $\omega_A^m((q,s),(q',s')) = \omega_T^m((q,q'))$*,* $\forall ((q,s),(q',s')) \in \delta_A$*;*
- $F_A = Q_T \times F$ *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 δ_A and map ω_A^p. Construction of automaton A mimics the one from [21], but adds the number of moving robots for each transition of T_T in map ω_A^m.

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 δ_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 ($\mathbf{p}$ from definition 4). The probability of existence for a transition of A is captured by its cost assigned by map ω_A^p, 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 ω_A^p 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]), she basically finds the path whose *product* of probabilities is *maximum*[2]. The movement-based weighting function ω_A^m 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 ω_A^m will be used for choosing a path that minimizes 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, than 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) [32, 9].

By using graph searches, we can find in A an accepted path that has a minimum sum of costs given by map ω_A^p along prefix and suffix. As noted

[1] An alternative (and probably more intuitive) definition of δ_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 q_{0T}.

[2] Multiplication is equivalent to sum of logarithms, thus minimization of $\sum_{i=1,2,\ldots} -log(p_i)$ implies the maximization of $\prod_{i=1,2,\ldots} p_i$, for any values $p_i \in (0,1]$.

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 ω_A^p, there may exist multiple paths of A yielding the same optimal 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_{-X} 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 ω_A^m while maintaining the optimal cost given by ω_A^p.

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 ω_A^p and (2) among the set of all path yielding the same optimum cost from condition (1), it minimizes the costs given by ω_A^m". We mention that it may not be possible to solve the problem by constructing a combination between ω_A^p and ω_A^m and using a single optimization (graph search).

Several possible solutions for solving the above problem would be:

(i) Find all paths that satisfy condition (1), and from these choose one that satisfies condition (2);

(ii) 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 ω_A^m, and with an equality condition imposing that the obtained solution has cost with respect to ω_A^p equal to the optimum one from condition (1).

Both solutions are untractable due to several reasons: the first optimization of solution (i) can yield an infinite number of paths. Although formally correct, the implementation of the second optimization from (ii) 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 (i) and second one from (ii), 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 ω_A^p) and number of movements (map ω_A^m), then $w_1 \prec w_2$ (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 ω_A^p, and movement-based costs ω_A^m 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[3]).

Algorithm 1: Optimum paths from an initial state s_0 of A to all final states in F_A

$\forall w_1, w_2 \in \mathbb{R}_+^2$ with $w_i = [c_i^p,\, c_i^m]^T$, $i = 1, 2$:
  $w_1 \prec w_2 \equiv (c_1^p < c_2^p) \vee ((c_1^p = c_2^p) \wedge (c_1^m < c_2^m))$
$cost : S_A \to \mathbb{R}^2$, $cost(s_0) = [0, 0]^T$, $cost(s) = [\infty, \infty]^T$, $\forall s \in S_A \setminus \{s_0\}$
let $prev(s)$ undefined, $\forall s \in S_A$
while $S_A \neq \emptyset$ **do**
  choose $s \in S_A$ s.t. $cost(s) \prec cost(s')$, $\forall s' \in S_A \setminus \{s\}$
  $S_A := S_A \setminus \{s\}$
  for $s' \in S_A$ *s.t.* $(s, s') \in \delta_A$ **do**
    if $\left(cost(s) + [\omega_A^p(s, s'),\, \omega_A^m(s, s')]^T\right) \prec cost(s')$ **then**
      $cost(s') := cost(s) + [\omega_A^p(s, s'),\, \omega_A^m(s, s')]^T$
      $prev(s') := s$
for $s \in F_A$ **do**
  if $cost(s) \neq [\infty, \infty]^T$ **then**
    find path from s_0 to s by tracking back based on map $prev$
    return $cost(s)$ and path from s_0 to s
  else
    final state s not accessible from s_0

3.3 Execution monitoring strategy

Let us denote by $run_A = (q_{0T}, s_0)\ (q_{1T}, s_1)\ (q_{2T}, s_2) \ldots (\mathbf{q_{sT}}, \mathbf{s_s}) \ldots (\mathbf{q_{pT}}, \mathbf{s_p}) \ldots$ the optimum path of A obtained as described in subsection 3.2, where the suffix is marked by bold font. Path run_A is projected to the run of T_T denoted by $run_{T_T} = q_{0T}\, q_{1T} \ldots \mathbf{q_{sT}} \ldots \mathbf{q_{pT}} \ldots$, and the run of B denoted by $run_B = s_0\, s_1 \ldots \mathbf{s_s} \ldots \mathbf{s_p} \ldots$. From construction of automaton A, we have the guarantee that run_B is an accepted run of B, and thus the team run run_{T_T} can produce a sequence of observations (inputs to B) that satisfies the LTL_{-X} formula. As specified in subsection 3.2, optimization on run_A guarantees the maximization of the probability that T_T produces an output word satisfying the LTL_{-X} formula, while minimizing the total number of robot movements (transitions between environment regions).

[3]The suffix-finding procedure suffers another slight modification, since the standard Dijkstra algorithm would yield a path consisting only from the current final state, even if that state does not have a self-loop.

A straightforward projection of 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 subsection 3.1, 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 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 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 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 run_{T_T} and run_B and updates them when necessary.

Let us denote by $h \in \mathcal{O}_T$ 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 LTL_{-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 $q_{0T} = (q_{01}, q_{02}, \ldots, q_{0n})$):

(i) 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 $q_{0T} = q_{i\,T}$ in T_T, $S_0 = \mathcal{C}^B$ in B, and correspondingly update the set of initial states S_{A0} of A. Search for a new optimum run_A, find run_{T_T}, run_B and individual robot runs, then restart the monitoring strategy according to these paths;

(ii) if $s_{i+1} \in \mathcal{C}^B$, then T_T advances to the next state $q_{i+1\,T}$;

(iii) if $s_{i+1} \notin \mathcal{C}^B$, but $s_i \in \mathcal{C}^B$, then T_T stays (self-loops) in $q_{i\,T}$;

(iv) 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 case (i);

(v) if $\mathcal{C}^B = \emptyset$, then stop the movement and report that the LTL_{-X} formula was just violated by the current observation h.

Informally, case (i) means that the current observation h produces a faster advancement along 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 (ii) means that until now the desired run of B is produced by the actual observation of T_T. Case (iii) 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 waits in their current regions for a change in observation of T_T - such a change will eventually appear, because run_B can be produced by some probabilistic observations along run_{T_T}. In case (iv), the current observation h prevented run_B 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 (v) can appear only when any feasible team trajectory can produce (besides desired observations) some observations that are forbidden by the LTL_{-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 (v) 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 Sec. 5. Case (iv) may appear only in specific situations, such an exemplification being given in the next paragraph. Informally, replanning from case (iv) 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 (iv). For illustrating the necessity of case (iv) in the monitoring strategy, let us consider an environment with four regions. When appeared, regions π_1 and π_2 overlap, but they have different visibility probabilities. Regions π_3 and π_4 are disjoint. For simplicity, a single robot is considered, with the LTL_{-X} task $\diamondsuit(\pi_1 \vee \pi_2) \wedge (\neg\pi_1 \mathcal{U} \pi_3 \vee \neg\pi_2 \mathcal{U} \pi_4)$. Thus, the robot should eventually visit π_1 or π_2. The robot should also observe π_3 or π_4, but π_3 should be reached only without observing π_1 until then, while π_4 can be reached if π_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 π_1 and π_2 (with the "hope" of observing the region with higher visibility probability, π_1), and after that towards π_4. If the robot reaches the cell corresponding to π_1 and π_2, but π_1 is invisible and π_2 happens to be appeared, then case (iv) becomes active in the monitoring strategy: B reaches states that are not in the current run_B. Informally, the planned visit to π_4 is not feasible anymore and new paths run_{T_T} and run_B are computed (these will drive the robot towards π_3). Of course, the above formula would be violated (case (v)) if π_1 and π_2 were both visible when the robot reaches their corresponding region. As one can observe from this example, case (iv) 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 (i) and (iv) are related in the sense that they both modify the expected way of evolving along run_B, because T_T outputs an observation that has lower probability that the one(s) expected such that run_B is properly followed. In such a situation, case (i) is the effect of a "lucky" observation, while cases (iv), (iii), (v) are produced by an "unlucky" one. In case (iv) it is necessary to find new runs, since the actual ones cannot be further followed (see the above example), while in case (i) 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 (i) is simply removed from the monitoring strategy, and only the remaining cases are tested. We mention that case (i) 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_B in cases (i) and (ii), while case (iii) is certainly exited in finite time. Only replanning from case (iv) could induce cyclic behaviors, but such cycles are eventually left due to the strictly positive probability of observing the desired regions. Case (v) informs the user about the *deadlock* in the robot motion.

The probability-based optimization from subsection 3.2 implies that along the evolution case (ii) is most likely to appear, because run_B is correlated with the most likely observations in team positions along run_{T_T}. Cases (i) or (iv) rarely appear, and case (v) can appear only when there's no way of satisfying the formula while avoiding its possible violation. The waiting times from case (iii) are also minimized by large probabilities of observing the desired outputs along run_{T_T}. The above cases underline the efficiency of choosing the probability-based criterion as the primary optimization objective in Sec. 3.2, not only with respect to formula satisfaction, but also with respect to computation complexity, as in the frequently occurring case (ii) no additional computations are needed.

4 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 Sec. 2, 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 loosing solutions. In future work, we intend to incorporate techniques inspired from Resource allocation systems [15] such that the trajectories found as in Sec. 3 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 Sec. 3, as follows:

- In the finite-state team model from subsection 3.1, 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 subsection 3.2 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 Sec. 3. If $|Q|$ denotes the number of states of a robot model T_i, then the team model T_T has $|Q|^n$ states. The Büchi automaton B corresponding to an LTL_{-X} formula ϕ has at most $|\phi| \cdot 2^{|\phi|}$ states [4], 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, $S_A \leq |Q|^n \cdot |\phi| \cdot 2^{|\phi|}$. For finding an optimal accepted path in this automaton we run the modified Dijkstra's algorithm $(|S_{A0}| + |F_A|)$-times and then make $|S_{A0}| \cdot |F_A|$ comparisons (subsection 3.2). 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 $O\left(\left(|Q|^n \cdot |\phi| \cdot 2^{|\phi|}\right)^3\right)$. The execution monitoring strategy (subsection 3.3) may require some reruns of optimization problem from subsection 3.2, but the number of these updates is reduced due to the maximized probability of satisfying the formula. All 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:

• First, the execution monitoring strategy triggers trajectory updates only in cases (i) and (iv) (subsection 3.3). As mentioned, the updates from case (i) are

[4] For usual specifications, the number of states of B is significantly smaller than the given upper-bound.

[5] Optimized Dijkstra implementations use a sorted node list and have a smaller complexity of order $O\left(|\delta_A| \cdot log(|S_A|)\right)$.

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 (i) (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 (i) and the previous trajectories are followed. Trajectory updates from case (iv) are necessary, since conditions triggering this case are disjoint from those triggering the other cases. However, case (iv) rarely appears for usual specifications and environments.

• Second, 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 γ, 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 γ, 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 ω_T^m from T_T becomes irrelevant. This procedure is not formally described in the current work, but its computation effects are mentioned in simulations from Sec. 5.

All presented algorithms were implemented in the Matlab environment [46, 31], 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.

5 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: $\Pi = \{\pi_1, \ldots, \pi_6\}$. Each region of interest is a union of adjacent triangular cells, as follows:

- π_1 is composed by cells q_4, q_5, q_{17}, q_{18}, q_{20}, q_{21}, q_{25}, q_{26};
- π_2 is composed by cells q_2, q_4, q_5, q_{14}, q_{16}, q_{17}, q_{20};
- π_3 is composed by cells q_{31} and q_{35};
- π_4 is composed by cells q_8 and q_{10};
- π_5 is composed by cell q_{15};
- π_6 is composed by cell q_1.

The appearance and disappearance rates for each region of interest are given in the following list, together with the probabilities of existence, computed as in equation (1):

- $\lambda_1^a = 0.3$, $\lambda_1^d = 0.7$, $p_{\pi_1} = 0.3$;
- $\lambda_2^a = 0.8$, $\lambda_2^d = 0.3$, $p_{\pi_2} = 0.73$;

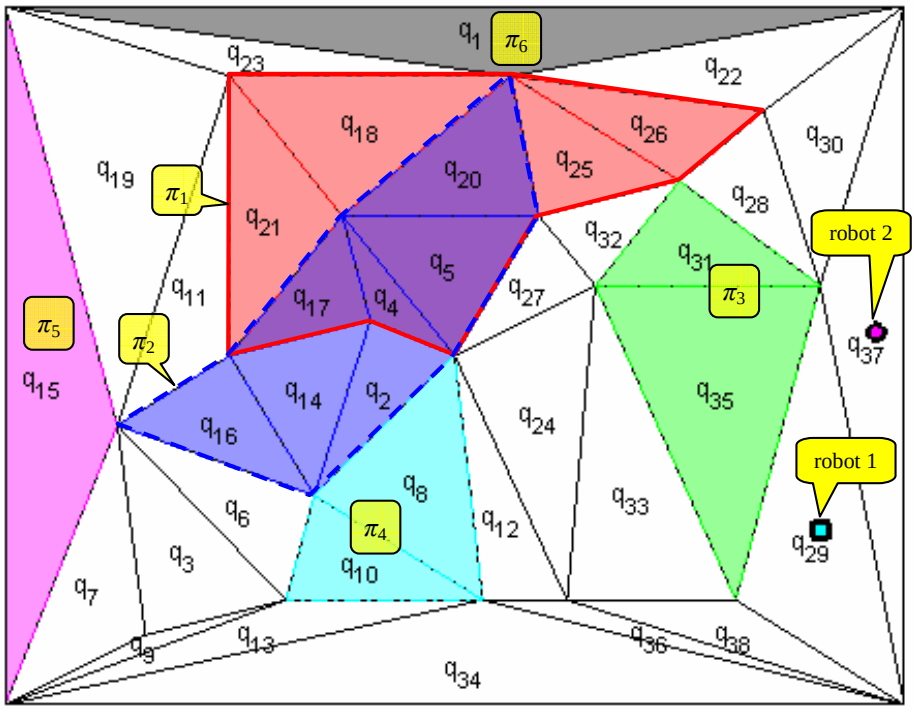


Figure 1: Environment partitioned in triangular cells, with six regions of interest. Initially, all regions are appeared, and two robots are placed in centroids of cells q_{29} and q_{37}. Region π_1 is colored with red, π_2 with blue, π_3 with green, π_4 with cyan, π_5 with magenta and π_6 with grey. When both π_1 and π_2 are visible, they overlap in cells q_4, q_5, q_{17}, q_{20}.

- $\lambda_3^a = 0.9$, $\lambda_3^d = 0.1$, $p_{\pi_3} = 0.9$;
- $\lambda_4^a = 0.5$, $\lambda_4^d = 0.1$, $p_{\pi_4} = 0.83$;
- $\lambda_5^a = 0.5$, $\lambda_5^d = 0.5$, $p_{\pi_5} = 0.5$;
- $\lambda_6^a = 0.5$, $\lambda_6^d = 0.3$, $p_{\pi_6} = 0.63$.

Initially, all regions are assumed to be visible, as in Fig. 1, where they are represented with different colors. Two robots are initially deployed in centroids of cells q_{29} and q_{37}, respectively.

Both T_1 and T_2 have 38 states, the finite state model of the team, T_T, has 1444 states, and these transition systems were constructed in 18 seconds on a medium performance laptop (CoreDuo 2GHz CPU, 4GB RAM). If one constructs models T_i and T_T by collapsing adjacent cells included in the same regions, as mentioned in Sec. 4, than T_1 and T_2 would each have 10 states, T_T 100 states, and all would be constructed in 4 seconds.

On the environment from Fig. 1 we will illustrate various scenarios that can be encountered when planning and moving the robots such that the chance of satisfying an LTL formula over the regions of interest is maximized. For this, we will first consider one formula (ϕ_1) and we will present some possible scenarios

that can occur due to probabilistic nature of regions' appearance and disappearance, and then we will choose another formula (ϕ_2) for depicting additional situations.

Example 1

We consider the LTL specification:

$$\phi_1 = \square\neg\pi_3 \wedge \diamondsuit\left((\pi_1 \vee \pi_2) \wedge \pi_4 \wedge \diamondsuit\left(\pi_5 \wedge \pi_6\right)\right)$$

Informally, π_3 should be always avoided, π_4 and π_1 or π_2 should be eventually occupied and then the robots should reach a position where π_5 and π_6 are observed.

The procedures described in subsection 3.2 are performed for finding a team trajectory. The Büchi automaton corresponding to ϕ_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 seconds [6] and, after projection to a path in T_T, the following sequence of cell pairs should be synchronously followed by the two robots:

$$run_{T_T} = \begin{matrix} \binom{q_{29}}{q_{37}}, \binom{q_{38}}{q_{30}}, \binom{q_{36}}{q_{22}}, \binom{q_{34}}{q_{36}}, \binom{q_{13}}{q_{25}}, \binom{q_{10}}{q_{20}}, \\ \binom{q_6}{q_{20}}, \binom{q_3}{q_{18}}, \binom{q_7}{q_{23}}, \binom{\boldsymbol{q_{15}}}{\boldsymbol{q_1}}, \ldots \end{matrix} \tag{4}$$

The suffix is marked in bold font and it has the meaning that the robots should stop in states q_{15} and q_1, respectively. The projection of run_A to run_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 run_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 π_4 (q_{10} belongs to region π_4), and π_1 or/and π_2 (q_{20} belongs to both regions π_1 and π_2). Then, the robots go to q_{15} and q_1, respectively (where regions π_5 and π_6 can be observed). As imposed by the formula, region π_3 should not be observed, even if it were always appeared.

The number of traversed cells is minimized due to the second optimization criterion from subsection 3.2: *e.g.*, the closer simplex q_{20} is visited instead of farther cells q_4, q_5 or q_{17}, although all of them belong to both π_1 and π_2.

An obtained execution is represented in the snapshots from Fig. 2. During this execution, the execution monitoring strategy from subsection 3.3 encounters only cases (ii) and (iii) (advancing along designated path of T_T, or pausing the movement until run_B advances due to some desired observation of T_T). Thus, Fig. 2(a) corresponds to the situation in which region π_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 π_4). The robots continue to move and they synchronize when entering q_{38} and q_{30}, respectively (third marked position along each trajectory). In Fig. 2(b), region π_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 π_4 and (π_1 or π_2), and run_B cannot advance. Case (iii) from the monitoring strategy is activated, and the robots wait until $(\pi_1 \vee \pi_2) \wedge \pi_4$

[6] If the model reduced by cell collapsing were used, the time for constructing A and finding a path is around 1 second.

becomes true. In the snapshot from Fig. 2(d) first π_1 appeared, then π_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 π_5 and π_6 (the execution monitoring strategy is in case (iii)). In Fig. 2(f) the formula was satisfied once π_5 and π_6 became visible; the robots converge to the centroids of cells q_{15} and q_1 and remain there (suffix of run_{T_T} has length one).

Example 2

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

$$\phi_2 = \square\neg\pi_3 \wedge \diamondsuit(\pi_1 \vee \pi_2 \vee \pi_4) \wedge \neg(\pi_5 \vee \pi_6)\,\mathcal{U}\,(\pi_5 \wedge \pi_6)$$

Informally, π_3 should be always avoided, either of regions π_1, π_2, π_4 should be eventually visited, and eventually the robots should simultaneously observe regions π_5 and π_6 (because the last part of the formula requires that nor π_5 nor π_6 are observed until both of them are observed at the same time).

The Büchi automaton corresponding to ϕ_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_T yields exactly the run from (4). Of course, run_B is different due to the different structure of B. An intuitive interpretation of run_{T_T} is in accordance with the requirements of ϕ_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 π_1, π_2, π_4. Then, they go to q_{15} and q_1, respectively and enter them at the same time, with the "hope" that both π_5 and π_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_{26}, region π_1 is appeared, so it is observed by the second robot. Therefore, run_B is advanced faster and case (i) from the monitoring strategy is active (intuitively, the robots don't have to go anymore to q_{10} and q_{20} for observing π_1, π_2 or π_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 (i); however, if additional computation on the central unit is to be avoided, case (i) can be simply ignored (as mentioned in Sec. 4) and the robots continue their initial trajectories. It happened that both π_5 and π_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 ϕ is violated. The robots evolve along paths from equation (4). It happened that π_1 has not appeared until cells q_{10} and q_{20} were reached, and thus run_B was not advanced faster. In the snapshot from Fig. 4(a), π_4 is observed by the second robot and the robots continue along the initial run of T_T towards cells q_{15} and q_1. When q_{15} and q_1 are simultaneously reached (Fig. 4(b)), π_6 is appeared but π_5 is disappeared. Case (v) becomes active, and the formula is violated (recall that the last part of ϕ_2 requires that none of π_5 and π_6 is 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.

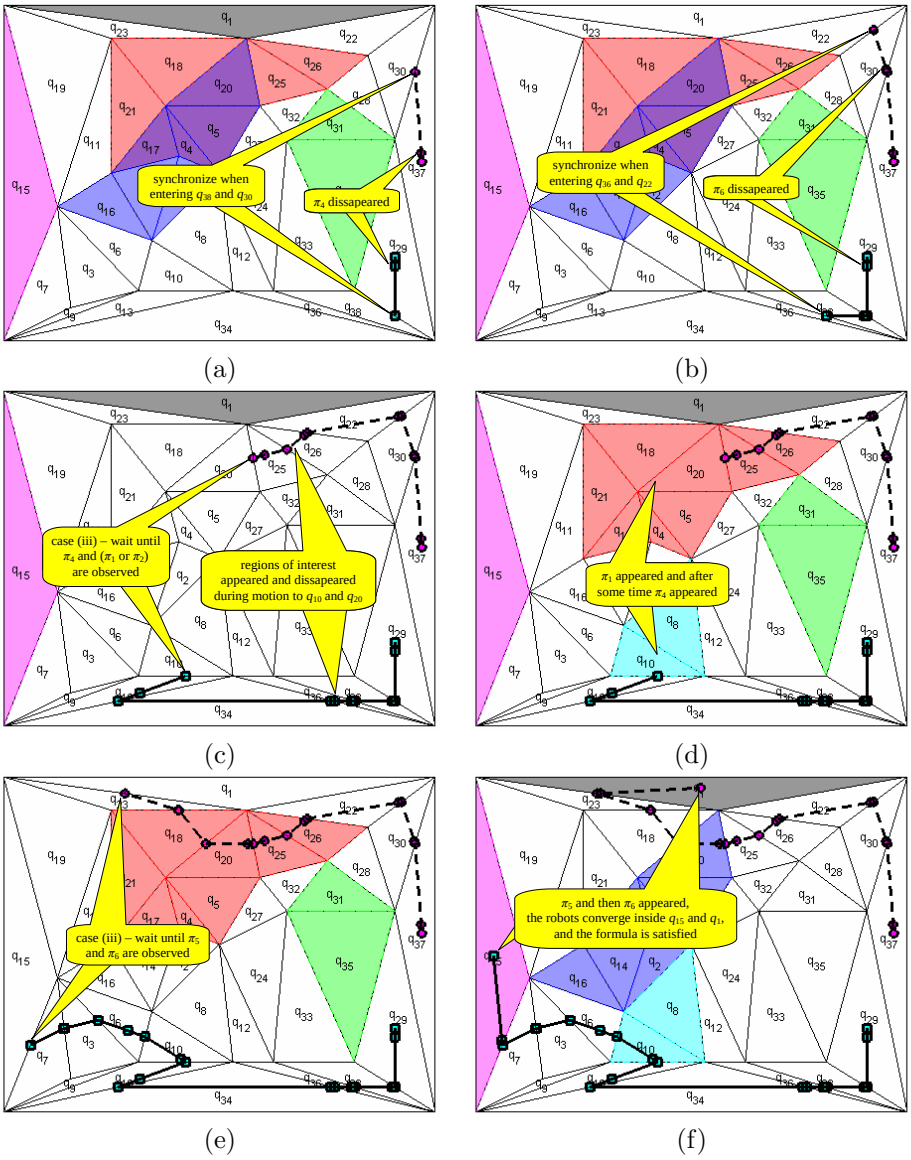
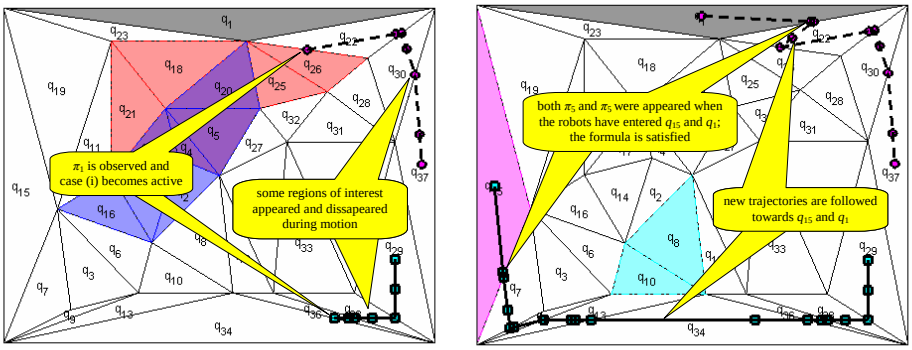
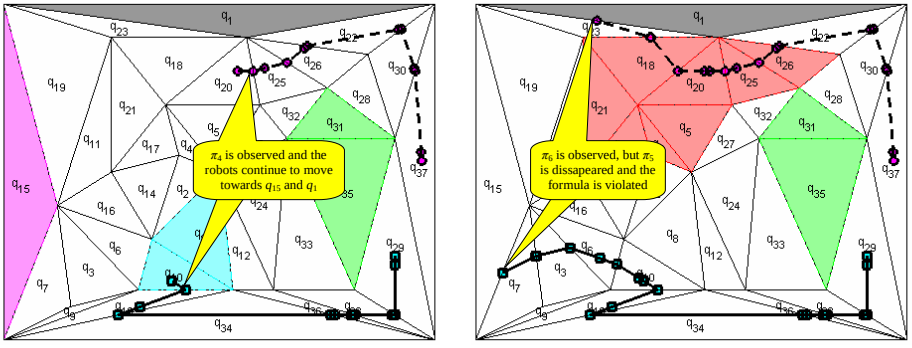


Figure 2: Snapshots along execution for Example 1; positions of the two robots are marked for moments when they synchronize and when a region appears or disappears. The execution monitoring strategy was in case (ii) in snapshots (a), (b), (d), (f) and in case (iii) in snapshots from (c), (e). Explanations are included in the text and briefly in the yellow callouts.



(a) (b)

Figure 3: Snapshots along a possible execution for Example 2. (a) Case (i) from the monitoring strategy becomes active and new trajectories are found in 27 seconds. (b) When the robots synchronously enter q_{15} and q_1 both regions π_5 and π_6 were appeared, and the formula is satisfied. Except for the snapshot (a), only case (ii) from the execution monitoring strategy was active during this execution.



(a) (b)

Figure 4: A possible execution leading to violating the formula from Example 2. (a) Robots reach q_{10} and q_{20}, π_4 is observed, and the movement along initial path is continued. (b) Case (v) from the monitoring strategy becomes active: when the robots synchronously enter q_{15} and q_1, only π_6 is observed, and the formula is violated.

For supporting the rapid increase in computational demands mentioned in Sec. 4, we have performed tests on more robots. We mention that our implementation does not include 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 3 robots evolving in the above environment and an LTL specification whose Büchi automaton has 3 states, the product automaton A has order of 10^5 states and a solution was obtained in more than 10 hours. By using the reduced robot models, a solution was obtained in less than 1 minute. For a team with 4 robots and for reduced models, the computation increased to around 3 hours. 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, then 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.

6 Conclusions

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. Based on the actual appearance states of the visited regions, an execution monitoring strategy alters the motion by pausing it or by updating the path to be further followed. The method is implemented as a freely-downloadable software package [31], and several case studies are included for supporting the developed solution.

References

[1] H. Kress-Gazit, Ed., *IEEE Robotics and Automation Magazine. Special issue on Formal Methods for Robotics and Automation*, 2011, vol. 18, no. 3.

[2] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[3] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[4] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, "Symbolic planning and control of robot motion," *IEEE Robotics and Automation Magazine*, vol. 14, no. 1, pp. 61–71, 2007.

[5] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Boston: MIT Press, 2005.

[6] S. M. LaValle, *Planning Algorithms*. Cambridge, 2006, available at `http://planning.cs.uiuc.edu`.

[7] E. M. M. Clarke, D. Peled, and O. Grumberg, *Model Checking*. MIT Press, 1999.

[8] J. Hopcroft, R. Motwani, and J. Ullman, *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Boston, MA: Addison-Wesley Longman Publishing Co., 2006.

[9] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Boston: MIT Press, 2008.

[10] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multiagent motion tasks based on LTL specifications," in *IEEE Conference on Decision and Control*, vol. 1, 2004, pp. 153–158.

[11] A. Vergnano, C. Thorstensson, B. Lennartson, P. Falkman, M. Pellicciari, F. Leali, and S. Biller, "Modeling and optimization of energy consumption in cooperative multi-robot systems," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 2, pp. 423–428, 2012.

[12] T. Wongpiromsarn, U. Topcu, and R. Murray, "Receding horizon temporal logic planning for dynamical systems," in *IEEE Conference on Decision and Control*, 2009, pp. 5997–6004.

[13] M. Vardi, "Probabilistic linear-time model checking: An overview of the automata-theoretic approach," in *Formal Methods for Real-Time and Probabilistic Systems*, ser. LNCS, 1999, vol. 1601, pp. 265–276.

[14] C. Courcoubetis and M. Yannakakis, "The complexity of probabilistic verification," *Journal of the ACM*, vol. 42, no. 4, pp. 857–907, 1995.

[15] S. Reveliotis and E. Roszkowska, "Conflict Resolution in Free-Ranging Multivehicle Systems: A Resource Allocation Paradigm," *IEEE Transactions on Robotics*, vol. 27, no. 2, pp. 283–296, 2011.

[16] M. Kloetzer, C. Mahulea, and J.-M. Colom, "Petri net approach for deadlock prevention in robot planning," in *IEEE Conference on Emerging Technologies Factory Automation*, 2013, pp. 1–4.

[17] M. Lahijanian, S. B. Andersson, and C. Belta, "Temporal logic motion planning and control with probabilistic satisfaction guarantees," *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 396–409, 2012.

[18] M. Svorenova, I. Cerna, and C. Belta, "Optimal control of MDPs with temporal logic constraints," in *IEEE Conference on Decision and Control*, 2013, pp. 3938–3943.

[19] X. Ding, S. L. Smith, C. Belta, and D. Rus, "LTL control in uncertain environments with probabilistic satisfaction guarantees," in *IFAC World Congress*, 2011.

[20] ——, "Optimal control of Markov decision processes with linear temporal logic constraints," *IEEE Transactions on Automatic Control*, vol. 59, no. 5, pp. 1244–1257, 2014.

[21] M. Kloetzer and C. Mahulea, "LTL planning in dynamic environments," in *IFAC Int. Workshop on Discrete Event Systems*, 2012, pp. 294–300.

[22] A. Abate, M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic model checking of labelled Markov processes via finite approximate bisimulations," in *Horizons of the Mind. A Tribute to Prakash Panangaden*, ser. LNCS, 2014, vol. 8464, pp. 40–58.

[23] L. De Alfaro, "Formal verification of probabilistic systems," Ph.D. dissertation, Stanford University, 1997.

[24] J. Capitan, M. T. Spaan, L. Merino, and A. Ollero, "Decentralized multi-robot cooperation with auctioned POMDPs," *The International Journal of Robotics Research*, vol. 32, no. 6, pp. 650–671, 2013.

[25] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of Markov decision processes," *Mathematics of Operations Research*, vol. 27, no. 4, pp. 819–840, Nov. 2002.

[26] M. Guo, J. Tumova, and D. Dimarogonas, "Cooperative decentralized multi-agent control under local LTL tasks and connectivity constraints," in *IEEE Conference on Decision and Control*, 2014, pp. 75–80.

[27] M. Karimadini and H. Lin, "Guaranteed global performance through local coordinations," *Automatica*, vol. 47, no. 5, pp. 890–898, 2011.

[28] X. Ding, C. Belta, and C. G. Cassandras, "Receding horizon surveillance with temporal logic specifications," in *IEEE Conference on Decision and Control*, 2010, pp. 256–261.

[29] X. Ding, M. Lazar, and C. Belta, "LTL receding horizon control for finite deterministic systems," *Automatica*, vol. 50, no. 2, pp. 399–408, 2014.

[30] M. Svorenova, J. Tumova, J. Barnat, and I. Cerna, "Attraction-based receding horizon path planning with temporal logic constraints," in *IEEE Conference on Decision and Control*, 2012, pp. 6749–6754.

[31] M. Kloetzer and C. Mahulea, "Software tool for LTL-based planning of robotic teams in dynamic environments," http://webdiis.unizar.es/∼cmahulea/research/LTL_prob_obs.zip.

[32] P. Wolper, M. Vardi, and A. Sistla, "Reasoning about infinite computation paths," in *24th IEEE Symposium on Foundations of Computer Science*, E. N. et al., Ed., 1983, pp. 185–194.

[33] G. Holzmann, *The Spin Model Checker, Primer and Reference Manual*. Reading, Massachusetts: Addison-Wesley, 2004.

[34] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *Conf. on Computer Aided Verification*, ser. LNCS, vol. 2102, 2001, pp. 53–65.

[35] M. D. Berg, O. Cheong, and M. van Kreveld, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer, 2008.

[36] M. Kloetzer and N. Ghita, "Software tool for constructing cell decompositions," in *IEEE Conference on Automation Science and Engineering*, 2011, pp. 507–512.

[37] L. C. G. J. M. Habets, P. J. Collins, and J. H. van Schuppen, "Reachability and control synthesis for piecewise-affine hybrid systems on simplices," *IEEE Transactions on Automatic Control*, vol. 51, pp. 938–948, 2006.

[38] C. Belta and L. Habets, "Controlling a class of nonlinear systems on rectangles," *IEEE Transactions on Automatic Control*, vol. 51, no. 11, pp. 1749–1759, 2006.

[39] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *IEEE Conference on Decision and Control*, 2005, pp. 4885–4890.

[40] L. C. G. J. M. Habets and J. H. van Schuppen, "A control problem for affine dynamical systems on a full-dimensional polytope," *Automatica*, vol. 40, pp. 21–35, 2004.

[41] M. Kloetzer, C. Mahulea, C. Belta, and M. Silva, "An automated framework for formal verification of timed continuous petri nets," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 3, pp. 460–471, 2010.

[42] J. Desai, J. Ostrowski, and V. Kumar, "Controlling formations of multiple mobile robots," in *IEEE International Conference on Robotics and Automation*, 1998, pp. 2864–2869.

[43] M. Kloetzer and C. Belta, "Automatic deployment of distributed teams of robots from temporal logic motion specifications," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 48–61, 2010.

[44] E. Dijkstra, "A note on two problems in connexion with graphs," in *Numerische Mathematik*. Mathematisch Centrum, Amsterdam, The Netherlands, 1959, vol. 1, pp. 269–271.

[45] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, Massachusetts and New York: The MIT Press and McGraw-Hill Book Company, 2001.

[46] The MathWorks, "MATLAB® 2010b," Natick, MA.

[47] R. Cowlagi and P. Tsiotras, "Hierarchical motion planning with dynamical feasibility guarantees for mobile robotic vehicles," *IEEE Transactions on Robotics*, vol. 28, no. 2, pp. 379–395, 2012.

[48] M. Kloetzer and C. Mahulea, "A Petri net based approach for multi-robot path planning," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 24, no. 4, pp. 417–445, 2014.

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

[50] D. Paun and M. Chechik, "On closure under stuttering," *Formal Aspects of Computing*, vol. 14, no. 4, pp. 342–368, 2003.

[51] M. Kloetzer, S. Magdici, and A. Burlacu, "Experimental platform and matlab toolbox for planning mobile robots," in *Int. Conf. on System Theory, Control and Computing*, 2012, pp. 1–6.