Optimal Motion Planning for Markov Decision Processes with Co-Safe Linear Temporal Logic Specifications

Bruno Lacerda, David Parker and Nick Hawes
School of Computer Science, University of Birmingham, United Kingdom

Abstract

We present preliminary work on the application of probabilistic model checking to motion planning for robot systems, using specifications in co-safe linear temporal logic. We describe our approach, implemented with the probabilistic model checker PRISM, illustrate it with a simple simulated example and discuss further extensions and improvements.

1 Introduction

In this paper, we describe our first steps on using probabilistic model checking to create robot controllers that satisfy a set of (probabilistic) temporal logic specifications in a given Markov decision process (MDP) model of the robot's behaviour. Probabilistic model checking allows one to analyse probabilistic properties of a model such as an MDP. These properties are generally specified in a temporal logic formalism, which allows reasoning over sequences of states of the model. In this work, we use the probabilistic model checker PRISM (Kwiatkowska, Norman, and Parker 2011) to solve a specific problem in the mobile robotics domain: optimal motion planning with linear temporal logic (LTL) goals.

To be more specific, we will minimize the expected time required to satisfy formulas in the co-safe fragment of LTL. This fragment represents LTL-specified "tasks" that can be "completed" in a finite time. We will show that finding strategies[1] that minimize the expected cost to satisfy such formulas in a given MDP model can be reduced to finding strategies that minimize the cost to reach a set of states in the MDP obtained by composing the MDP model with a Rabin automaton that represents the LTL formula. We will then show how one can create a "navigation" MDP from data gathered from a robot platform, and provide an example of finding optimal strategies for such a model. These strategies represent time-optimal path plans for the robot, where the goal is not simply reaching a given state, but can be *temporally extended* goals that require, for example, a set of states to be visited in a given order. An example of such a task is a mail delivery robot that needs to deliver mail to different rooms in a building and minimize the time spent in delivery so it can be available to do other tasks as soon as possible.

[1] Strategies are usually known as *policies* in the MDP literature.

In recent years, there has been growing interest in the use of LTL as a task specification language for robot systems (Ding et al. 2011; Guo, Johansson, and Dimarogonas 2013; Jing and Kress-Gazit 2013; Lacerda and Lima 2011; Svoreňová, Černá, and Belta 2013; Ulusoy, Wongpiromsarn, and Belta 2012; Wolff, Topcu, and Murray 2012; 2013). This interest stems from two properties of LTL that make it a desirable specification language in the robotics domain: (i) LTL is a close-to-natural-language formalism and (ii) one can build correct-by-construction controllers from a model of the system and the LTL specification. Thus, LTL provides the possibility to bridge the gap between the specification a designer has for a (robot) system, and an implementation of a controller that leads the system's behaviour to that specification.

The work in (Ding et al. 2011; Svoreňová, Černá, and Belta 2013) tackles the problem of maximizing the probability of satisfying a given LTL formula while minimizing the long-term average cost between two states that satisfy a given "optimizing" atomic proposition pre-defined by the designer. It is the work most closely related to ours. The main difference is that, broadly speaking, the goal there is to find the strategy that reaches the steady-state of the system that minimizes a given infinite horizon cost function. In our work, the goal is more related to transient analysis, where we want to find the strategy that minimizes the accumulated cost on a finite horizon. Also, the optimizing atomic proposition used for the long-term optimization is somewhat artificial, while in our case we can define the problem in a cleaner way. In terms of application, the work in (Ding et al. 2011; Svoreňová, Černá, and Belta 2013) is especially suited for the execution of static and persistent tasks that do not have an exact notion of being "completed", e.g., patrolling a building "forever". In our case, we only allow for finite horizon tasks. This enables the possibility of moving on to more dynamic task allocation and on-line (re-)planning for minimizing the mixing of different incoming tasks defined by the end-user at different times.

In Section II, we present MDPs and co-safe LTL, the formalisms used in this work. In Section III our strategy generation approach using PRISM is described. We build upon the general approach presented in (Kwiatkowska and Parker 2013), and show how it can be extended to handle the generation of strategies that minimize the cost of satisfying a co-

safe LTL formula. Section IV describes how one can build an MDP model of a motion planning problem from data learned by a robot platform, and Section V illustrates the approach with a simulated application example. Finally, in Section VI, we finish with a discussion about the current approach and on our plans for future work.

2 Preliminaries

2.1 Markov Decision Processes

We start by defining Markov decision processes with atomic propositions labelling the states and a cost function associated with state-action pairs.

Definition 1 (Markov Decision Process with Cost Function)**.** A Markov decision process (MDP) is a tuple $\mathcal{M} = \langle S, \overline{s}, A, \delta_{\mathcal{M}}, AP, Lab, c\rangle$, where:

- S is a finite set of states.
- $\overline{s} \in S$ is the initial state.
- A is a finite set of actions.
- $\delta_{\mathcal{M}} : S \times A \to Dist(S)$ is a (partial) probabilistic transition function, mapping state-action pairs to probability distributions over S. To simplify the notation, we also define the set of *enabled* actions in s as $A(s) = \{a \in A \mid \delta_{\mathcal{M}}(s,a) \text{ is defined}\}$.
- AP is a set of atomic propositions.
- $Lab : S \to 2^{AP}$ is a labelling function, such that $p \in Lab(s)$ if and only if the atomic proposition p is true in state s.
- $c : S \times A \to \mathbb{R}_{\geq 0}$ is the cost function, associating each state-action pair with a non-negative value.

An MDP model represents all the possible evolutions of the state of the system, by nondeterministically applying an enabled action in each state, and transitioning to a successor state according to the probabilities specified by the transition function. This behaviour is explicitly represented by the sets of finite paths ($FPath_{\mathcal{M}}$) and infinite paths ($IPath_{\mathcal{M}}$) generated by $\mathcal{M}$:

$$FPath_{\mathcal{M}} = \{s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \xrightarrow{a_{n-1}} s_n \mid n \in \mathbb{N},\ s_0 = \overline{s},\ s_i \in S \text{ for all } i \in \{0,...,n\},\ a_i \in A(s_i) \text{ and } \delta_{\mathcal{M}}(s_i,a_i)(s_{i+1}) > 0 \text{ for all } i \in \{0,...,n-1\}\} \quad (1)$$

$$IPath_{\mathcal{M}} = \{s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \mid s_0 = \overline{s},\ s_i \in S \text{ for all } i \in \mathbb{N},\ a_i \in A(s_i) \text{ and } \delta_{\mathcal{M}}(s_i,a_i)(s_{i+1}) > 0 \text{ for all } i \in \mathbb{N}\} \quad (2)$$

We will be defining strategies over finite paths of $\mathcal{M}$ and evaluating the temporal logic formulas over infinite paths of $\mathcal{M}$. Note that, because of the nondeterminism between actions choices in an MDP, we cannot directly define probability measures over $FPath_{\mathcal{M}}$ or $IPath_{\mathcal{M}}$, i.e., the notion of the probability that the system generates a given sequence is not well-defined. This nondeterminism is resolved by adding the notion of strategy.

Definition 2 (Strategy)**.** Let $\mathcal{M}$ be an MDP. A strategy for $\mathcal{M}$ is a function $\sigma : FPath_{\mathcal{M}} \to A$ such that, for all $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \xrightarrow{a_{n-1}} s_n \in FPath_{\mathcal{M}}$, $\sigma(\pi) \in A(s_n)$.

A strategy chooses, at each step of the execution, an enabled action to be executed next, taking into account the finite history that occurred before the current step[2].

Given an MDP $\mathcal{M}$ and a strategy σ for it, we have a fully probabilistic system, for which one can check various quantitative properties. Note that the system is still probabilistic because the outcome of executing an action in a given state is a distribution over states. We will be particularly interested in the following property:

- Let $p \in AP$. What is the expected value of the accumulated cost to reach a state $s \in S$ such that $p \in Lab(s)$? This is denoted $E^{\sigma}_{\mathcal{M}}(acc(c, Fp))$, where $acc(c, Fp) : IPath \to \mathbb{R}_{\geq} \cup \infty$ is defined as:

$$acc(c,Fp)(s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ...) = \begin{cases} \infty & \text{if for all } j \in \mathbb{N},\ p \notin Lab(s_j) \\ \sum_{j=0}^{\min\{k \in \mathbb{N} \mid p \in Lab(s_k)\}} c(s_j, a_j) & \text{otherwise} \end{cases} \quad (3)$$

We will use this quantity to generate strategies that minimize the cost of satisfying a given atomic proposition. This is supported by the PRISM software and can be solved by using "classic" MDP algorithms like value or policy iteration, given that we are solving a (cost optimizing) probabilistic reachability problems. In Section III we will extend the specification language to allow the inclusion of the co-safe fragment of LTL formulas.

2.2 Linear Temporal Logic

LTL is an extension of propositional logic which allows reasoning over an infinite sequence of states. It was developed as a means for formal reasoning about concurrent systems (Pnueli 1981), and provides a convenient and powerful way to formally specify a variety of qualitative properties of a system.

Definition 3 (Syntax)**.** An LTL formula over a set AP of atomic propositions has the following syntax:

- $true$, $false$ and $p \in AP$ are LTL formulas;
- If φ and ψ are LTL formulas then $(\neg\varphi)$, $(\varphi \vee \psi)$, $(\varphi \wedge \psi)$, $(X\varphi)$, $(\varphi U \psi)$ and $(\varphi R \psi)$ are also LTL formulas.

The temporal operators can be understood as follows:

- The X operator is read "next", meaning that the formula it precedes will be true in the next state.
- The U operator is read "until", meaning that its first argument will be true until its second argument becomes true (and the second argument must become true in some state, i.e., a sequence where φ is always satisfied but ψ is never satisfied does not satisfy $\varphi U \psi$).

[2] One can also define randomized strategies, but that is outside of the scope of this document.

- The R operator is read "release", meaning that its second argument can only become false if its first argument became true before (i.e., the occurrence of the first argument releases the need for the second argument to be true). For this operator, if the first argument never becomes true, the second argument must remain true for all times.

We will be evaluating LTL formulas over infinite sequences in $IPath_{\mathcal{M}}$.

Definition 4 (Semantics)**.** Let $\mathcal{M}$ be an MDP, $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \in IPath_{\mathcal{M}}$ and $p \in AP$. The notion of local satisfaction, $\vDash$, is defined as follows:

- $\pi \vDash true$ and $\pi \nvDash false$;
- $\pi \vDash p$ if and only if $p \in Lab(s_0)$;
- $\pi \vDash (\neg\varphi)$ if and only if $\pi \nvDash \varphi$;
- $\pi \vDash (\varphi \vee \psi)$ if and only if $\pi \vDash \varphi$ or $\pi \vDash \psi$;
- $\pi \vDash (\varphi \wedge \psi)$ if and only if $\pi \vDash \varphi$ and $\pi \vDash \psi$;
- $\pi \vDash (X\varphi)$ if and only if $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} ... \vDash \varphi$;
- $\pi \vDash (\varphi U \psi)$ if and only if there exists $t \geq 0$ such that $s_t \xrightarrow{a_t} s_{t+1} \xrightarrow{a_{t+1}} ... \vDash \psi$ and for all $t' \in [0, t)$ $s_{t'} \xrightarrow{a_{t'}} s_{t'+1} \xrightarrow{a_{t'+1}} ... \vDash \varphi$;
- $\pi \vDash (\varphi R \psi)$ if and only if for all $t \geq 0$, if $s_t \xrightarrow{a_t} s_{t+1} \xrightarrow{a_{t+1}} ... \vDash \psi$ then there exists $t' \in [0, t)$ such that $s_{t'} \xrightarrow{a_{t'}} s_{t'+1} \xrightarrow{a_{t'+1}} ... \vDash \varphi$;

We denote by $IPath^{\varphi}$ the set of elements of $IPath_{\mathcal{M}}$ that satisfy φ.

We will also use other temporal operators, which we will define by abbreviation: (i) $(F\varphi) \equiv (trueU\varphi)$ and (ii) $(G\varphi) \equiv (falseR\varphi) = (\neg F(\neg\varphi))$. The F operator is read "eventually" and requires the existence of a future state where the formula it precedes is true. The G operator is read "always" and requires the formula it precedes to be true in all future states.

We now introduce the class of LTL formulas which we will use to write our goals. We are interested in minimizing the accumulated cost of reaching some goal. However, this problem is not well defined for arbitrary LTL formulas, since the accumulated value of the cost will be infinity. Thus, we need to restrict ourselves to LTL formulas that can be "satisfied" in a finite-horizon. This is a well-defined class, called co-safe formulas. These are formulas for which the satisfying infinite sequences always have a finite good prefix.

Definition 5 (Good Prefix)**.** Let φ be an LTL formula and $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ...$ such that $\pi \vDash \varphi$. π has a good prefix if there exists $n \in \mathbb{N}$ such that the truncated finite sequence $\pi|_n = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \xrightarrow{a_{n-1}} s_n$ is such that $\pi|_n.\pi' \vDash \varphi$ for any infinite sequence π', where $\pi|_n.\pi'$ is obtained by concatenating the finite sequence $\pi|_n$ with the infinite sequence π'. We write $\pi|_n \vDash^{fin} \varphi$ when $\pi|_n$ is a good prefix for π.

Definition 6 (Co-Safe LTL)**.** The LTL formula φ is said to be co-safe if all the infinite sequences π such that $\pi \vDash \varphi$ have a good prefix.

It is known that an LTL formula in the positive normal form[3] where only the the temporal operators X, F and U are used is co-safe (Kupferman and Vardi 2001). We will keep our formulas within this syntactic restriction.

3 Optimal Strategy Generation for Co-Safe LTL

Our goal is to use PRISM to synthesise strategies that minimize the expected value of the accumulated cost to satisfy a given co-safe LTL formula. Thus, we are interested in, given a co-safe LTL formula φ, finding a strategy that minimizes the expected value of the accumulated cost to generate a good prefix that satisfies φ:

$$\sigma^{min}_{\mathcal{M}}(c, \varphi) := \underset{\sigma: FPath_{\mathcal{M}} \to A}{\arg\min} E^{\sigma}_{\mathcal{M}}(acc(c, \varphi)) \tag{4}$$

Defining $k_{\varphi} = \min\{k \in \mathbb{N} \mid s_0 \xrightarrow{a_0} ... \xrightarrow{a_{k-1}} s_k \vDash^{fin} \varphi\}$, the acc function is now defined as:

$$acc(c, \varphi)(s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ...) = \begin{cases} \infty & \text{if } s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} ... \nvDash \varphi \\ \sum_{j=0}^{k_{\varphi}} c(s_j, a_j) & \text{otherwise} \end{cases} \tag{5}$$

To find such a strategy, the automata-theoretical approach to LTL model checking is used. It is known that any LTL formula can be translated into a deterministic Rabin automaton A_{φ} that accepts exactly the infinite sequences that accept φ. One can compose this automaton with the MDP, obtaining the product MDP $\mathcal{M}_{\varphi} = \mathcal{M} \otimes A_{\varphi}$ which represents the synchronous execution of $\mathcal{M}$ and A_{φ}. Thus, the state space of $\mathcal{M}_{\varphi}$ is $S \times Q$, where Q is the states of A_{φ}, i.e., states are pairs (s, q), with s representing the current state of $\mathcal{M}$'s execution and q the current state of A_{φ}'s execution. Furthermore, since we assume that φ is a co-safe LTL formula, A_{φ} has the following property: once an accepting state is reached, the automaton will stay in an accepting state regardless of what occurs afterwards. Thus, we just need to minimize the cost to reach such a state. If we label all states (s, q) of $\mathcal{M}_{\varphi}$ for which q is an accepting state of A_{φ} with a new atomic proposition acc, the following holds:

$$E^{min}_{\mathcal{M}}(c, \varphi) = E^{min}_{\mathcal{M}_{\varphi}}(c, Facc) \tag{6}$$

Thus, by applying value or policy iteration on $\mathcal{M}_{\varphi}$, as described before, we can find a memoryless strategy $\sigma : S \times Q \to A$ that minimizes the cost of reaching an accepting state in $\mathcal{M}_{\varphi}$. Using this strategy and $\mathcal{M}_{\varphi}$, we can implement a finite memory strategy for $\mathcal{M}$, by keeping track of the current state of $\mathcal{M}_{\varphi}$ and executing the memoryless strategy. Thus, the part of the state pair in the product MDP corresponding to the current state of A_{φ} can be seen as a way to keep some memory about the path executed by the MDP until a given moment.

[3] An LTL formula is said to be in the positive normal form if only atomic propositions are negated.

4 Instantiation to a Motion Planning Problem

In this section, we will discuss how the optimal strategy generation we presented can be used in a specific robot application: motion planning. We assume that a navigation graph is provided, and that some data was gathered about the time taken between connected nodes in the graph or possibilities of failure. We will call this a "learned" navigation graph[4]. This navigation graph will then be translated into an MDP. We choose to present the navigation graph first, and then show how to build a navigation MDP from a learned navigation graph in order to simplify the presentation. The learned navigation graph can be seen as a stepping stone to the final navigation MDP that we will use for planning.

Definition 7 (Learned Navigation Graph)**.** A learned navigation graph is a tuple $nav = \langle V, E, SR, F, T \rangle$, where:

- $V = \{v_0, ..., v_n\}$, where $v_i = (x_i, y_i, \theta_i)$ is a 2D robot pose.
- $E \subseteq V \times V$ is a set of edges, specifying navigation connections between different robot poses.
- $SR : E \to [0, 1]$ is a function that maps its edge (v_i, v_j) to the probability that the robot will successfully navigate from v_i to v_j without passing through any other navigation node, using its continuous navigation planner.
- $F : E \to Dist(V)$ is a function that is defined for all (v_i, v_j) such that $SR(v_i, v_j) < 1$ and maps each edge (v_i, v_j) to a distribution over nodes.
  - $F(v_i, v_j)(v_k)$ represents the probability of either (i) the continuous navigation failing while trying to reach v_j and the closest point that the robot can navigate to after failure is node v_k or (ii) the robot passing through node v_k before reaching node v_j.
- $T : (E \times \{suc, fail\}) \to \mathbb{R}_{\geq 0}$ is the expected time function:
  - $T((v_i, v_j), suc)$ represents the expected time for the robot to travel between nodes v_i and v_j, given that that navigation is successful.
  - $T((v_i, v_j), fail)$ is defined for edges (v_i, v_j) such that $SR(v_i, v_j) < 1$ and represents the expected time between the robot starting to move from v_i to v_j, and a failure occurring. A failure is either continuous navigation failing or passing through a node that is not v_j while navigating to v_j. We also include in this average the time the robot takes to get to the closest navigable point, when the navigation fails but is recoverable.

From this learned navigation graph, one can create a navigation MDP.

Definition 8 (Navigation MDP)**.** Let $nav = \langle V, E, SR, F, T \rangle$, with $V = \{v_0, ..., v_n\}$ be a learned navigation graph. The navigation MDP associated to nav is the MDP $\mathcal{M}_{nav} = \langle S_{nav}, \overline{s}_{nav}, A_{nav}, \delta_{nav}, AP_{nav}, Lab_{nav}, c_{nav} \rangle$ where:

[4]Our ongoing work also includes the development of an algorithm to learn such a graph.

- $S = S_{normal} \cup S_{fail} \cup S_{rec}$, where:

$$S_{normal} = \{s_1, ..., s_n\} \tag{7}$$

$$S_{fail} = \bigcup_{i=0}^{n} \{s_{ij}^f \mid (v_i, v_j) \in E\} \tag{8}$$

$$S_{rec} = \bigcup_{i=0}^{n} \{s_{ij}^r \mid (v_i, v_j) \in E\} \tag{9}$$

  The states of S_{normal} represent the navigation nodes when no failure occurred. The states S_{fail} represents failures in a given edge. The states S_{rec} represent the situations where the robot navigation failed, but the robot recovered and navigated back to the original node where it started the navigation from. These states are used so that the robot does not try to go through a blocked path indefinitely, by adding a "one-step memory state" and disallowing the robot from repeatedly trying to execute the same navigation action and failing (in the transition function, state s_{ij}^r will not have the action to move to s_j enabled).
- $\overline{s}_{nav} \in S_{normal}$ is the initial state of the robot in its environment
- $A_{nav} = A_{mov} \cup \{recover\}$, where:

$$A_{mov} = \bigcup_{i=0}^{n} \{goto_{ij} \mid (v_i, v_j) \in E\} \tag{10}$$

- The transition function just mirrors the edges in the learned navigation graph, with the addition of the recover states, as explained above. We will show the cases for which the transition function is not undefined:
  - For $s_i \in S_{normal}$ and $a = goto_{ij}$ for some j:

$$\delta_{nav}(s_i, goto_{ij})(s') = \begin{cases} SR(v_i, v_j) & \text{if } s' = s_j \\ 1 - SR(v_i, v_j) & \text{if } s' = s_{ij}^f \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

  - For $s_{ij}^f \in S_{fail}$ and $a = recover$,

$$\delta_{nav}(s_{ij}^f, recover)(s') = \begin{cases} F(v_i, v_j)(v_k) & \text{if } (s' = s_k,\ k \neq i) \\ F(v_i, v_j)(v_i) & \text{if } s' = s_{ij}^r \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

  - For $s_{ij}^r \in S_{recover}$ and $a = goto_{ik}$, with $k \neq j$, $\delta_{nav}(s_{ij}^r, goto_{ik}) = \delta_{nav}(s_i, goto_{ik})$
- The set of atomic propositions is the nodes of the navigation graph plus a failure proposition (we can add more useful labels to the states if we want to, but for basic navigation this suffices and makes the exposition simpler):

$$AP_{nav} = V \cup \{failure\} \tag{13}$$

- The labelling function maps its state to the node in the graph it represents, and also the information if it is a state where a failure occurred.

$$Lab_{nav}(s) = \begin{cases} \{v_i\} & \text{if } s = s_i \\ \{v_i, failure\} & \text{if } s = s_{ij}^r \text{ or} \\ & s = s_{ij}^f, \text{ for some } j \end{cases} \tag{14}$$

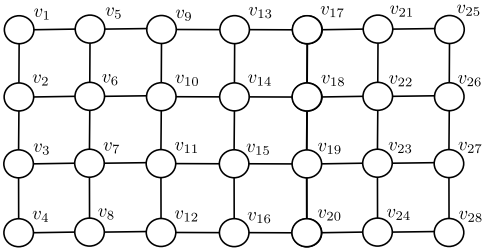

Figure 1: A navigation graph.

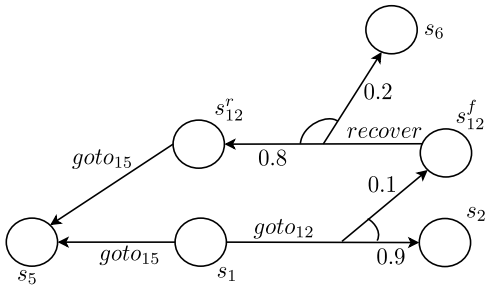

Figure 2: The navigation MDP obtained from a fragment of the navigation graph in Fig. 1.

- The cost function mirrors the T function of the learned navigation graph:

$$c_{nav}(s,a) = \begin{cases} T((v_i,v_j),suc) & \text{if } (s=s_i \text{ or } s=s_{ij}^r) \text{ and } a=goto_{ij} \\ \max\{T(v_i,v_j),fail) - T((v_i,v_j),suc),0\} & \text{if } s=s_{ij}^f \text{ and } a=recover \\ 0 & \text{otherwise} \end{cases} \quad (15)$$

  Note that for the recovery, we subtract the expected value for the cost for succeeding, because $T((v_i,v_j),fail)$ counts the time since the robot starts the navigation until it fails.

Example 1. In Figure 1, we depict a possible navigation graph nav, where all edges are bidirectional. For edges (v_1,v_2) and (v_1,v_5), we have the following values:

- $SR(v_1,v_2) = 0.9$.
- $F(v_1,v_2)(v_1) = 0.8$ and $F(v_1,v_2)(v_6) = 0.2$.
- $T((v_1,v_2),suc) = 2$ and $T((v_1,v_2),fail) = 3$.
- $SR(v_1,v_5) = 1$ and $T((v_1,v_5),suc) = 1$.

In Figure 2, we depict the navigation MDP built from the fragment of nav for which we defined SR, F and T. For the costs of the state-action pairs in the MDP, we have:

- $c_{nav}(s_1,goto_{12}) = T((v_1,v_2),suc) = 2$.
- $c_{nav}(s_{12}^f,recover) = \max\{T((v_1,v_2),fail) - T((v_1,v_2),suc),0\} = 1$.
- $c_{nav}(s_1,goto_{15}) = c_{nav}(s_{12}^f,goto_{15}) = T((v_1,v_5),suc) = 1$.

5 Application Examples

In this section, we show some examples of motion plans obtained for the navigation graph in Figure 1. We start by assuming that all the actions have zero probability of failure, and that the cost of each edge is one. With this simplified example, we will show the motion plans obtained for different LTL formulas. Then, we will see the impact of different costs and action outcomes on the plans obtained for a simple LTL specification.

We start with an example where we want the robot to find the shortest path to visit nodes v_4 and v_{28}. This can be specified by:

$$\sigma_{\mathcal{M}}^{min}(c, Fv_4 \wedge Fv_{28}) \quad (16)$$

We kept the number of nodes to visit small so we are able to illustrate the example. However, note that we can generalize this type of formula, and create strategies that solve the travelling salesman problem in a probabilistic domain. In Figure 3, we illustrate the obtained strategy. Given that this strategy is a finite memory strategy, we split the figure, showing the actions to take in each node when (a) neither v_4 nor v_{28} have been visited, (b) v_4 has been visited but v_{28} has not and (c) v_{28} has been visited but v_4 has not.

We can see that the obtained strategy has some memory about which goal states were already visited. As mentioned previously, this is due to the fact that we are generating memoryless strategies for the product MDP, for which the space state is the Cartesian product of the states of the navigation MDP and the states of the Rabin automaton obtained for the LTL formula (which, informally, provide memory for the strategy obtained for the original MDP). Also, note that an optimal action is generated for all the states of the product automaton, thus we have an optimal action for each state, in each situation.

We now impose an ordering on how the nodes should be visited. Node 15 should be visited after node 28. This can be specified by the following co-safe LTL formula:

$$\sigma_{\mathcal{M}}^{min}(c, F(v_{28} \wedge Fv_{15})) \quad (17)$$

Note that, in this case, an optimal strategy only needs to consider two cases: (i) we already visited v_{28} and need to go to v_{15} or (ii) we did not visit v_{28} yet, and need to visit it. In Figure 4 (a) and (b), we depict the strategies generated from PRISM for cases (i) and (ii), respectively.

Also, we note that this is a "soft" ordering, in the sense that we not disallow v_{15} from being visited before v_{28}, we only require it to be visited after v_{15} is visited. If we want to impose a "hard" ordering where the robot can only visit v_{15} after he visits v_{28}, we can use the following strategy:

$$\sigma_{\mathcal{M}}^{min}(c, (\neg v_{15} U v_{28}) \wedge (Fv_{15})) \quad (18)$$

The strategy for (18) is depicted in Figure 5. Note that, when going to v_{28}, the strategy avoids v_{15}. Also, there is no

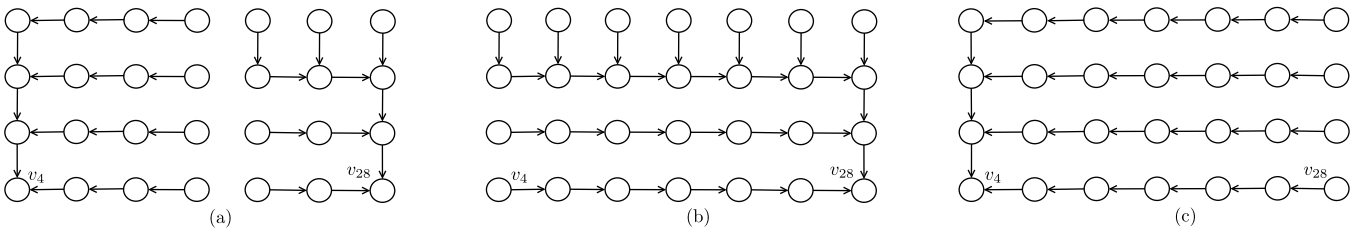

Figure 3: Depiction of the strategy obtained from equation (16). The arrows depict the action to be executed at each node. (a) neither v_4 nor v_{28} have been visited. (b) v_4 has been visited but v_{28} has not. (c) v_{28} has been visited but v_4 has not.

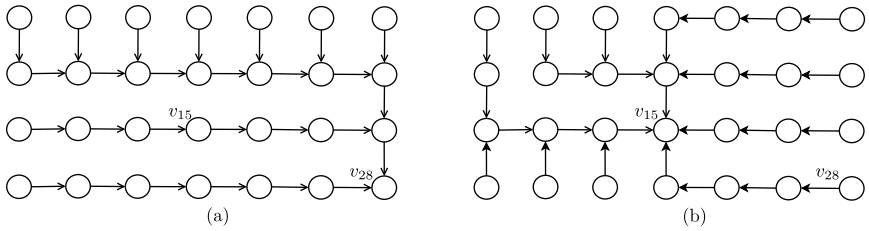

Figure 4: Depiction of the strategy obtained from equation (17). The arrows depict the action to be executed at each node. (a) v_{28} has not been visited yet. (b) v_{28} has been visited but v_{15} has not.

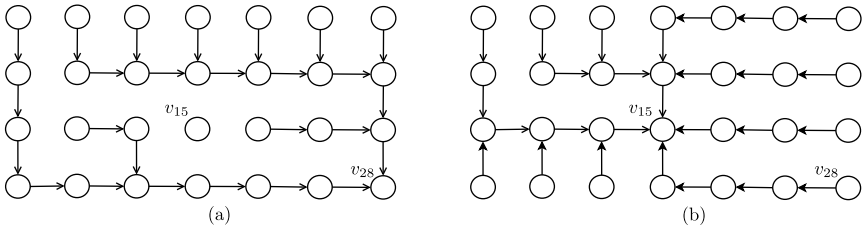

Figure 5: Depiction of the strategy obtained from equation (18). The arrows depict the action to be executed at each node. (a) v_{28} has not been visited yet. (b) v_{28} has been visited but v_{15} has not.

action defined for v_{15} in that situation because if v_{15} is visited before v_{28}, the LTL formula is not satisfied, thus there is no notion of optimal action in that state of the MDP.

Until now, we have ignored both the cost and the probabilistic nature of the transitions, in order to more clearly illustrate the different strategies generated from different LTL formulas. However, our choice for using MDP models has mainly to do with the fact that they elegantly handle the uncertainty inherent in robot applications. We now illustrate the impact of uncertainty on the outcome of actions and the different costs in the motion plans generated by PRISM. We will focus our attention on the specification:

$$\sigma_{\mathcal{M}}^{min}(c, Fv_{25} \vee Fv_{28}) \tag{19}$$

For this specification, the robot needs to decide which node, v_{25} or v_{28}, to visit. In Figure 6 (a), we depict the strategy obtained when all the weights are one and the probability of action failures is zero. As expected, in each state, the robot moves in direction to the closest goal, thus it will visit the node closer to its initial state.

We now assume that edges connecting the first and second row of our navigation graph[5] have a cost of 10 (i.e., 10 times more than other edges). We also assume that there is a 10% probability of navigation failure when going from v_{13} to v_{17}. We assume that when this happens, we can always recover to v_{18} but we pay an extra cost of 10 to navigate from the failure position to v_{18}. In this case, we want to avoid navigating through the high cost edges. Thus, the decision on which node to go visit in each state changes. For example, now when the robot is on v_{26}, it goes in the direction of v_{28} in order to avoid passing through the costly edge. This can be seen in Figure 6 (b). We also added a grey arrow from v_{13} to v_{18} to represent the failure possibility between v_{13} and v_{17}. Note that when that failure occurs, the robot stops going towards v_{25} and starts going towards v_{28} instead, because from v_{18} the less costly approach is to visit v_{28}. We stress that this change in goal is encompassed in our model, and no re-planning was needed to cope with the move failure. Finally, if we increase the probability of navigation failure when going from v_{13} to v_{17}, and/or increase the cost of recovering to v_{18}, the strategy stops going through that "bad" edge. We depict this in Figure 6 (c).

6 Discussion

We have presented an approach that uses the probabilistic model checker PRISM to generate cost-optimal motion plans for goals given as co-safe LTL formulas. This is still preliminary work, and as such, there are many possible directions for future work. These include:

- *Develop a learning algorithm for the navigation graph.* We are developing an algorithm that, in an initial stage, learns a navigation graph from the definition of the edges, and when it is executing the navigation tasks given in LTL also updates the navigation graph according to his experience. Our final goal is to have these navigation graphs dependent on time of day, so we can explore them in long-term autonomy scenarios. Work on learning this kind of graphs is, to the best of our knowledge, scarce, with (Haigh and Veloso 1999) being one of the few related works known by us.
- *Allow for addition of new tasks during execution of previous ones, minimizing the expected time of both tasks simultaneously.* Given that the product of a navigation MDP and a Rabin automaton is still an MDP, when the robot is executing a plan for a given task and a new one arrives, we can simply do a new product with the current product MDP being executed. After this new product is done - and given that we are using co-safe LTL - it is easy to distinguish nodes that are accepting for each task. We can choose to minimize the time to complete both tasks, or use some new approach that allows us to, for example, minimize the expected time for a given task, while keeping the expected time of the other under a given threshold. To build strategies for this type of specification, we will use the multi-objective strategy generation of PRISM. When a given task is fulfilled, we can also "invert" the product and shrink the product MDP by ignoring the Rabin automaton obtained for that task. Thus, we will have a dynamic structure that represents the navigation MDP and the tasks being executed at a given moment.
- *Integrate with a scheduler with notion of real time.* In the approach presented in this paper, there is not an exact notion of real time, i.e., there is no notion of executing a task at a given time of day. What we presented is an algorithm that plans to execute tasks as soon as possible. One can investigate how to use a real time scheduler that uses PRISM as an oracle, asking it what is the expected time of executing a given task at a given time of day, and then places that task in the best time for it to be executed in both a timely and as less conflicting with other tasks as possible manner.
- *Create a web-based interface where non-specialist users can schedule tasks.* Building upon the previous points, one can think of building a system where a non-specialist user can define tasks using some grammar (e.g. structured english as defined in (Kress-Gazit, Fainekos, and Pappas 2007)), which is then translated into LTL formulas and time of day to be executed in. This would then go to the integrated scheduler/motion planner which would both schedule the task and create the optimal motion plan to fulfil it.
- *Expand tasks that can be specified:*
  - *Add reactivity to the models.* The approach we presented does not include reactivity to sensor readings of the robot nor actions which are not motion based. Adding this possibility would greatly increase the expressibility of the specification language, by allowing, for example, the specification of a task where the robot needs to find an object and then take it to a given point in the environment. To handle this reactivity, we will investigate the extension of the methodology presented here to the domain of stochastic games. Also, we will

[5] Edges (v_1, v_2), (v_5, v_6), (v_9, v_{10}), (v_{13}, v_{14}), (v_{17}, v_{18}), (v_{21}, v_{22}) and (v_{25}, v_{26}).

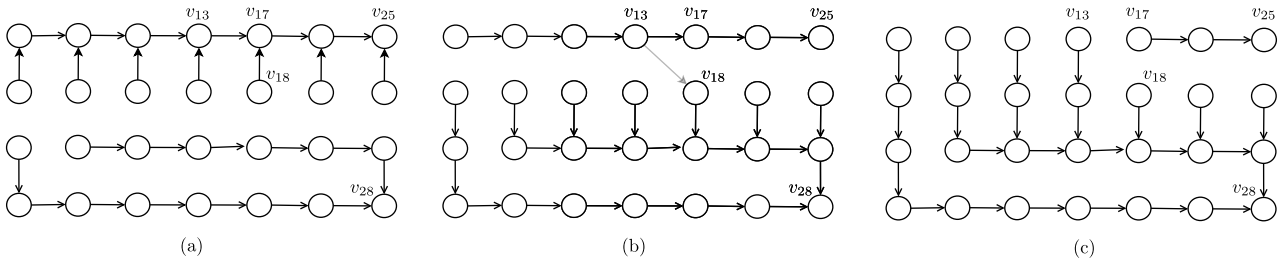

Figure 6: Depiction of the strategy obtained from equation (19), when neither v_{25} nor v_{28} have been visited. The arrows depict the action to be executed at each node. (a) All costs equal to 1, and no uncertainty on action outcome. (b) Edges between first and second row with cost 10, and 10% probability of failing navigation between v_{13} and v_{17}, and then recovering to v_{18} with a cost of 10. (c) Edges between first and second row with cost 10, and 50% probability of failing navigation between v_{13} and v_{17}, and then recovering to v_{18} with a cost of 55.

investigate how supervisory control theory (Cassandras and Lafortune 2006), and notions such as uncontrollable events and least-permissive supervisors can be used in this domain.

- *Investigate the use of other classes of LTL.* In the work presented here, we are only using co-safe LTL. One can think of extending it to larger classes of LTL. One particularly interesting example is the GR(1) class, for which a translation from the structured english mentioned above is defined. This class also elegantly includes the notion of reactivity we just described. Also, how one can create optimal plans for LTL in general without the notion of "optimizing proposition" used in (Ding et al. 2011; Svoreňová, Černá, and Belta 2013), which is somewhat artificial, and does not allow the minimization of expected time for finite-horizon tasks, which, as illustrated in this work, are also interesting.
- *Explore PRISM's multi-objective capabilities.* PRISM allows generation of strategies for multi-objective specifications (Forejt et al. 2011). With this approach, one can specify tasks like "minimize expected time while maximizing the probability of keeping yourself inside a safe region". One first approach for this can be using co-safe LTL for minimizing the expected time to fulfill a task, and minimize or maximize the probability of satisfying safe LTL formulas (broadly speaking, formulas that enforce that something "bad" will never happen) for maximization or minimization of probabilities.

Note that these approaches to increase the expressibility of the specification language are complementary and can all be integrated into a single framework.

The large amount of future work routes shows the potential of this work to be used in a wide array of applications.

7 Acknowledgements

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No 600623, STRANDS, and the EPSRC grant EP/K014293/1.

References

Cassandras, C. G., and Lafortune, S. 2006. *Introduction to Discrete Event Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Ding, X. C.; Smith, S. L.; Belta, C.; and Rus, D. 2011. MDP optimal control under temporal logic constraints. In *Proceedings of CDC-ECC'11: The 2011 50th IEEE Conference on Decision and Control and European Control Conference*, 532–538.

Forejt, V.; Kwiatkowska, M.; Norman, G.; Parker, D.; and Qu, H. 2011. Quantitative multi-objective verification for probabilistic systems. In *Proc. 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11)*, volume 6605 of *LNCS*, 112–127. Springer.

Guo, M.; Johansson, K. H.; and Dimarogonas, D. V. 2013. Motion and action planning under LTL specications using navigation functions and action description language. In *Proceedings of IROS '13: The IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Haigh, K. Z., and Veloso, M. M. 1999. Learning situation-dependent costs: Improving planning from probabilistic robot execution. *Robotics and Autonomous Systems* 29(2):145–174.

Jing, G., and Kress-Gazit, H. 2013. Improving the continuous execution of reactive LTL-based controllers. In *Proceedings of ICRA '13: The 2013 IEEE International Conference on Robotics and Automation*, 5439–5445.

Kress-Gazit, H.; Fainekos, G. E.; and Pappas, G. J. 2007. From structured english to robot motion. In *Proceedings of IROS '07: IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2717 – 2722.

Kupferman, O., and Vardi, M. 2001. Model checking of safety properties. *Formal Methods in System Design* 19(3):291–314.

Kwiatkowska, M., and Parker, D. 2013. Automated verification and strategy synthesis for probabilistic systems. In *Proc. 11th International Symposium on Automated Technology for Verification and Analysis (ATVA'13)*, volume 8172 of *LNCS*, 5–22. Springer.

Kwiatkowska, M.; Norman, G.; and Parker, D. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of CAV'11: The 23rd International Conference on Computer Aided Verification*, volume 6806 of *LNCS*, 585–591. Springer.

Lacerda, B., and Lima, P. U. 2011. Designing Petri net supervisors from LTL specifications. In *Proceedings of RSS VII – The 2011 Robotics: Science and Systems Conference*.

Pnueli, A. 1981. The temporal semantics of concurrent programs. *Theoretical Computer Science* 13:45–60.

Svoreňová, M.; Černá, I.; and Belta, C. 2013. Optimal control of MDPs with temporal logic constraints. In *Proceedings of CDC'13: The 52nd IEEE Conference on Decision and Control*.

Ulusoy, A.; Wongpiromsarn, T.; and Belta, C. 2012. Incremental control synthesis in probabilistic environments with temporal logic constraints. In *Proceedings of CDC: The 2012 IEEE Conference on Decision and Control*.

Wolff, E. M.; Topcu, U.; and Murray, R. M. 2012. Optimal control with weighted average costs and temporal logic specifications. In *Proceedings of RSS VIII – The 2012 Robotics: Science and Systems Conference*.

Wolff, E. M.; Topcu, U.; and Murray, R. M. 2013. Efficient reactive controller synthesis for a fragment of linear temporal logic. In *Proceedings of ICRA '13: The 2013 IEEE International Conference on Robotics and Automation*.