The International Journal of Robotics Research
XX(X):1–28
©The Author(s) 2019
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Probabilistic planning with formal performance guarantees for mobile service robots

Bruno Lacerda[1], Fatma Faruq[2], David Parker[2] and Nick Hawes[1]

Abstract

We present a framework for mobile service robot task planning and execution, based on the use of probabilistic verification techniques for the generation of optimal policies with attached formal performance guarantees. Our approach is based on a Markov decision process model of the robot in its environment, encompassing a topological map where nodes represent relevant locations in the environment, and a range of tasks that can be executed in different locations. The navigation in the topological map is modelled stochastically for a specific time of day. This is done by using spatio-temporal models that provide, for a given time of day, the probability of successfully navigating between two topological nodes, and the expected time to do so. We then present a methodology to generate cost optimal policies for tasks specified in co-safe linear temporal logic. Our key contribution is to address scenarios in which the task may not be achievable with probability one. We introduce a task progression function and present an approach to generate policies that are formally guaranteed to, in decreasing order of priority: maximise the probability of finishing the task; maximise progress towards completion, if this is not possible; and minimise the expected time or cost required. We illustrate and evaluate our approach with a scalability evaluation in a simulated scenario, and reporting on its implementation in a robot performing service tasks in an office environment for long periods of time.

Keywords

Mobile Service Robots, Planning under Uncertainty, Markov Decision Processes, Linear Temporal Logic

1 Introduction

In recent years, we have witnessed substantial developments in the field of mobile service robots, to a point where more and more of these platforms are deployed amongst humans (Hawes et al. 2017). In order to have these robots behave autonomously and face the inherent uncertainty of human populated environments, we need to equip them with deliberation capabilities that allow for automatic generation and execution of plans that achieve a set of goals. Further, by continuously running in such environments, these robots have the opportunity to *learn* about the dynamics of the environment. For example, a robot can learn about the probability of successfully navigating between two locations at a given time of day, the probability of a doorway being open, or the expected wait time until someone opens a door and holds it for the robot to go through. Thus, when taking decisions about acting in such an environment, the robot should explicitly take these quantities into account, in order to achieve better performance.

Markov decision processes (MDPs) are a widely used formalism to model sequential decision-making problems for scenarios like the above, where there is inherent uncertainty about the system's evolution. In general, planning for MDPs is performed by defining a reward structure over the model, and using dynamic programming techniques such as value or policy iteration (Puterman 1994) to generate a policy that maximises the cumulative discounted reward over an infinite horizon. This approach has the drawback of requiring the designer to map the planning goal into a reward structure over the MDP, a process which can be cumbersome and error-prone. Furthermore, the use of a discount factor can render the value of the resulting optimisation meaningless in real life (e.g., discounting a cost representing execution time). An alternative approach that has been followed is to define the goal as reaching a set of states, and then minimise cost to do so. This approach has the limitation

[1]Oxford Robotics Institute, University of Oxford, UK.
[2]School of Computer Science, University of Birmingham, UK.

Corresponding author:
Bruno Lacerda, Oxford Robotics Institute, University of Oxford, UK.
Email: bruno@robots.ox.ac.uk

of providing an inexpressive specification language, where important concepts such as goal sequencing or safety cannot be explicitly represented. Thus, in recent years, there has been increasing interest in the use of higher-level formalisms to specify planning problems, in order to allow the designer to tackle intricate goal behaviours more naturally. In particular, linear temporal logic (LTL) (Pnueli 1981) has been proposed as a suitable specification language, due to it being a powerful and intuitive formalism to unambiguously specify a variety of tasks. Furthermore, algorithms and tools exist to generate provably correct policies from an MDP model of the system and an LTL task specification (Kwiatkowska et al. 2011; Ding et al. 2014b). These approaches build upon techniques from the field of formal verification, in particular, *probabilistic model checking*, which provides tools to reason not just about the probability of satisfying an LTL specification, but also about other quantitative metrics such as cost structures. These might represent, for example, expected execution time of an action in the MDP. Minimising the expected cumulative value for such a cost structure represents calculating a policy that achieves the goal as quickly as possible.

This paper provides contributions on both modelling and planning for mobile service tasks, with the overall goal of developing a planning framework for mobile service robot tasks that is able to utilise formal verification techniques over an accurate environmental model, such that the obtained formal guarantees have real world meaning.

On the modelling side, we present a general modelling approach for mobile service robot tasks, based on a topological map of the environment, encompassing relevant locations for action execution. In order to tackle the inherent uncertainty of environments populated by humans, we model the robot navigation in the topological map, along with other actions that can be executed, as an MDP. This MDP is such that the probabilistic outcomes and expected duration of actions are obtained from the predictions of external spatio-temporal models of the environment for a given time of day.

On the policy generation side, we present a methodology for generating policies for arbitrary MDPs that minimises the expected time to achieve a task specified in the *co-safe* fragment of LTL (i.e., a task that can be completed in a finite horizon). In particular, our approach allows for the generation of cost-optimal policies for tasks that cannot be achieved with probability one in the model (i.e., there is some probability that part of the co-safe LTL specification cannot be achieved). Typically, methods for generation of cost optimal policies assume the existence of a *proper* policy, i.e., a policy that achieves the task with probability one. This assumption, along with requiring that policies that do not reach the goal have infinite cost, provides a convergence guarantee of MDP solution algorithms to the optimal policy. However, in many real life scenarios, the probability of successfully completing the task is not one, and explicitly reasoning about ways to maximise this probability is of importance. Thus, we remove the requirement of probability one for task satisfaction by introducing the notion of *partial satisfiability*, and then tackle the question of what to do when the task becomes unsatisfiable during execution. This question is especially relevant in the mobile robots domain (e.g., part of a task can not be achieved because it requires navigation to an inaccessible area of the environment), and is motivated by our own experience with the deployment of these techniques in mobile robots, where we want the robot to do as much of the task as possible.

Example 1. *Consider Figure 1, depicting an environment where a robot needs to navigate to offices in a building to perform security checks (further details on the figure will be described throughout the paper).*

During the execution of this task, some doors might be closed, making offices inaccessible. This yields the overall task unsatisfiable, yet we still want the robot to check as many offices as it can.

To tackle the issue of partial task satisfaction, and performing “as much as possible” of a given task, we define a *task progression function* for co-safe LTL formulas, which can be encoded as a reward function on the MDP. Using this, we show that the problems of (i) maximising the probability of satisfying a co-safe LTL formula; (ii) maximising the task progression reward (i.e., fulfilling as much of the formula as possible); and (iii) minimising a cost function while achieving (i) and (ii) can be solved by a lexicographic version of value iteration on a *pruned product MDP*. We also describe how to analyse the resulting policy to calculate for example, the distribution of locations the robot might be in after finishing the task, or conditional expectations such as expected cost to succeed or expected cost to failure. The MDP solution technique we describe is based on the work presented in Lacerda et al. (2015b,a). Here, we contextualise the approach in the domain of mobile service robots; improve and extend its formalisation, providing formal proofs of its correctness; and provide a more extensive evaluation and comparison with a state-of-the-art approach for a similar problem.

After a literature review in Section 2, we introduce the formalisms used and fix notation in Section 3; in Section 4 we describe our MDP-based modelling of mobile service robot planning problems; in Section 5 we show how to solve the MDP for partially satisfiable co-safe LTL specifications. In Section 6 we discuss extra guarantees that can be obtained from the policy, which can be used to obtain more fine-grained guarantees of the expected behaviour of the

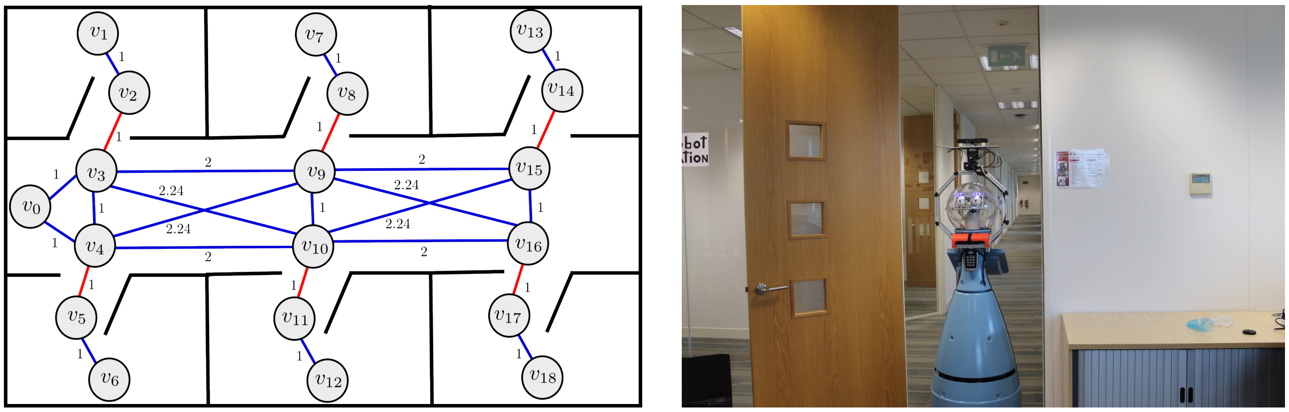


Figure 1. A mock-up office environment with 6 rooms used for illustration and evaluation purposes throughout the paper (left), and a mobile service robot deployed in an office scenario using the techniques presented here (right).

robot. Then, in Section 7 we illustrate and evaluate our approach on a simulated example, and then report on two real robot applications. We finish with a discussion on the approach and its possible future enhancements in Section 8.

2 Related Work

2.1 Planning and Robotics

In order to be autonomous in real world applications, it is crucial for robots to *act deliberately* in their environment (Ingrand and Ghallab 2014). Acting deliberately here is understood as "performing actions that are motivated by some intended objectives and that are justifiable by sound reasoning with respect to these objectives." In Ingrand and Ghallab (2014), five deliberation functions are identified: planning, acting, monitoring, observing, and learning. Our work is mainly focused on (task) planning, but our modelling approach allows for the inclusion of learning to define the probabilistic transition function and cost models of the MDP; supports acting and monitoring through our ROS implementation; and can also allow for planning to observe by modelling observation state variables appropriately. Efforts to integrate task planning with execution in robots include Cashmore et al. (2015); McGann et al. (2008). Cashmore et al. (2015) provides a software package that allows for the use of PDDL planners for planning and execution using the ROS middleware. McGann et al. (2008) provides a full deliberation framework for AUVs, using the EUROPA temporal planner (Frank and Jónsson 2003). Other examples of mobile robots using planning and scheduling to provide autonomy are Veloso et al. (2015); Hanheide et al. (2015); Stock et al. (2015); Mudrová et al. (2015).

One main distinction between the work above and ours is our explicit (and quantitative) modelling of uncertainty, and our introduction of probabilistic performance guarantees for robot behaviour. Reasoning explicitly about uncertainty is fundamental for robust deployment of robot systems, something we have shown through successful long-term mobile service robot deployments in real world scenarios (Hawes et al. 2017). Furthermore, the ability to provide guarantees can can be used to inform other system processes, and for execution monitoring. It can also be important for legal certification of robot systems to be deployed in real-life. In our opinion, frameworks simply based on execution and replanning on unexpected events, while more scalable, lack the performance guarantees required for safe and robust robot deployment. As stated in Ingrand and Ghallab (2014), "Although MDPs are often used in robotics at the sensory-motor level (...), [such] techniques are not as widely disseminated at the deliberative planning and acting level". Our work presents an effort in this direction.

2.2 Markov Decision Processes and Linear Temporal Logic

Finding cost-optimal policies for MDPs using LTL specifications has been tackled in several ways. Svoreňová et al. (2013); Ding et al. (2014b) study the problem of maximising the probability of satisfying an LTL specification, while minimising the long-term average cost to pass between two states that satisfy an "optimising" atomic proposition, which must be visited infinitely often. In Lacerda et al. (2014), cost-optimal policies for co-safe LTL are generated, and a mechanism for the dynamic addition of tasks during execution is presented. In these cases, the cost minimisation is done *only over states where the probability of satisfying the specification is one*. Conversely, Ulusoy et al. (2012); Cizelj and Belta (2014); Wolff et al. (2013); Lahijanian et al. (2012) deal with maximising the probability of satisfying temporal logic specifications but cost-optimality is not taken into account.

Other work on policy generation for MDPs with temporal logic specifications has focused on using the more traditional approach of finding policies that maximise *discounted cumulative rewards over an infinite horizon*, while constraining such policies to the class that satisfies a set of "until" specifications (Teichteil-Königsbuch 2012a; Sprauel et al. 2014). In these works, the temporal logic constraints are disconnected from the reward structure. This means that, in general, no optimal policies exist for such a problem. However, the authors prove that one can always build *randomised policies* that achieve a notion of ϵ-optimality. In our work, the main focus is the LTL specification, and the generation of cost-optimal policies that maximise its probability of satisfaction. This makes our class of optimal policies simpler: given that we prioritise maximisation of the probability of satisfaction over minimisation of the expected cost, the class of deterministic finite-history policies suffices to solve our problem.

There has also been work on generation of cost-optimal policies for specifications where the probability of success is not one. Teichteil-Königsbuch (2012b); Kolobov et al. (2012) present approaches to generate cost-optimal policies to reach a target state, in the presence of *unavoidable dead ends*, i.e., states which cannot be avoided with probability one from the initial state, and for which the probability of reaching the target is zero. Our work extends these approaches by focusing on co-safe LTL, instead of simpler single-state reachability. This fact requires the construction of a *product MDP* from the original model and the co-safe LTL specification, and also introduces the notion of *partial satisfiability*, which is not present when the goal is reaching a single state. Ding et al. (2013) present an approach, based on *constrained* MDPs, to generate cost-optimal policies for co-safe LTL, where the probability of satisfying the specification is kept above a threshold. Contrary to our approach, this requires the threshold for probability of satisfaction to be provided by the designer beforehand, and does not include the notion of partial task specification.

2.3 Partial Satisfiability

In recent years, there has been interest in approaches that deal with temporal logic specifications that cannot be *fully satisfied*. Tumova et al. (2013); Castro et al. (2013); Vasile et al. (2017) deal with the generation of *minimium-violation* controllers, i..e, controllers that only violate sets of safety rules for the shortest possible amount of time. Maly et al. (2013) also defines a function for task progression for co-safe LTL, and generates plans that satisfy as much of the specification as possible. This is further extended in Lahijanian et al. (2016), where three levels of partial satisfiability are defined. Lahijanian et al. (2015) introduces the notion of *weighted skipping*, where one can "simulate" the occurrence of certain atomic propositions that have not occurred at a certain point of the execution, with a user-defined cost being associated to the skipping of each atomic proposition. This allows the weighted skipping approach to encode the notion of progression, and we will compare our approach to an adaptation of this work to MDPs. Contrary to our work, the focus of all works above is on controlling a hybrid model of the system using a deterministic discretisation of the continuous dynamics of the robot, with the works that tackle uncertainty doing so by replanning.

Finally, the work in Lahijanian and Kwiatkowska (2016) adapts the work in Lahijanian et al. (2015) to MDPs. It replaces the notion of skipping cost with a substitution cost that explicitly represents the cost to ignore an occurrence of a certain atomic proposition, and substitute it by the occurrence of another atomic proposition. The substitution cost can allow for a reduction of the size of the model to be solved when compared to weighted skipping, as one just need to enumerate the possible substitutions instead of all the possible skippings. However, encoding progression using a substitution cost adds an extra design burden. It requires the designer to know beforehand what occurrences of atomic propositions need to be substituted (and what that substitution should be) in order for the robot to progress towards the goal. Our work further expands on the works above by providing a fully (ROS) integrated framework for task planning and execution, and implementing it on a real robot that has been deployed for long periods of time.

An alternative way of modelling partial satisfiability is the use of real-valued logics, such as signal temporal logic (STL) or metric temporal logic (MTL), where one can represent the *level of satisfaction* of the specification, instead of simply considering boolean satisfaction such as with LTL. Using such logics, there has been research on building *robust* controllers that keep the level of satisfaction of the temporal logic specification as high as possible during execution (Fainekos and Pappas 2009; Donzé and Maler 2010; Raman et al. 2014). Compared to our work, these works do not explicitly model uncertainty thus cannot provide probabilistic guarantees of performance. Sadigh and Kapoor (2016) propose a probabilistic version of STL. In their work however, uncertainty is included on the logic side, with the underlying model considered to be a hybrid dynamical system. The extra expressibility of STL and MTL entail an increase in complexity when compared to the use of LTL. This can hinder the scalability of these approaches. Furthermore, these works consider lower level control of systems with continuous dynamics, in contrast with our focus on higher level task planning.

3 Preliminaries

We start by providing some notation and the notions of MDPs and LTL needed for the remainder of the paper.

3.1 Notation

With X denoting a set, we define: (i) 2^X as the set containing all subsets of X; (ii) X^* as the set containing all the finite sequences of elements of X; (iii) X^ω as the set containing all the infinite sequences of elements of X; and (iv) $Dist(X)$ as the set of probability distributions over X. For discrete sets X, we denote elements of $Dist(X)$ as $p_1 : x_1 + ... + p_n : x_n$ where $p_1, ..., p_n \in (0,1]$ are such that $p_1 + ... + p_n = 1$, and $x_1, ..., x_n$ are elements of X. Furthermore, given $\delta \in Dist(X)$, we write $supp(\delta)$ to denote the support of δ, i.e., the largest set $X' \subseteq X$ such that $\delta(x) > 0$ for all $x \in X'$. Let $\rho = \rho_0 \dots \rho_{n-1}, \rho' = \rho'_0 \dots \rho'_{m-1} \in X^*$ and $\sigma = \sigma_0\sigma_1 \dots \in X^\omega$. We define (i) the length of ρ as $|\rho| = n$; (ii) the concatenation of ρ and ρ' as $\rho \cdot \rho' = \rho_0 \dots \rho_{m-1}\rho'_0 \dots \rho'_{n-1} \in X^*$; and (iii) the concatenation of ρ and σ as $\rho \cdot \sigma = \rho_0 \dots \rho_{m-1}\sigma_0\sigma_1 \dots \in X^\omega$.

3.2 Markov Decision Processes

We will model the environment for our mobile service robot as a *Markov decision process* (MDP) with atomic propositions labelling states. In Section 4 we will describe how this is done. Here, we introduce MDPs, fixing notation and present the results we build our methodology upon.

Definition 1. Flat MDP. *A (flat) MDP is a tuple $\mathcal{M} = \langle S, \overline{s}, A, \delta_\mathcal{M}, AP, Lab\rangle$, where:*

- *S is a finite set of states;*
- *$\overline{s} \in S$ is the initial state;*
- *A is a finite set of actions;*
- *$\delta_\mathcal{M} : S \times A \times S \to [0,1]$ is a probabilistic transition function, where $\sum_{s' \in S} \delta_\mathcal{M}(s, a, s') \in \{0,1\}$ for all $s \in S$, $a \in A$;*
- *AP is a set of atomic propositions;*
- *$Lab : S \to 2^{AP}$ is a labelling function, such that $p \in Lab(s)$ iff p is true in $s \in S$.*

Definition 2. Enabled Actions. *Let $\mathcal{M} = \langle S, \overline{s}, A, \delta_\mathcal{M}, AP, Lab\rangle$ be an MDP, and $s \in S$. We define the set of enabled actions in s as:*

$$A_s = \{a \in A \mid \delta_\mathcal{M}(s, a, s') > 0 \text{ for some } s' \in S\} \quad (1)$$

An MDP model represents possible (probabilistic) evolutions of the state of a system: in each state s, any of the enabled actions $a \in A_s$ can be selected and the probability of evolving to a successor state s' is then $\delta_\mathcal{M}(s, a, s')$.

Definition 3. Paths. *An infinite* path *through $\mathcal{M} = \langle S, \overline{s}, A, \delta_\mathcal{M}, AP, Lab\rangle$ is a sequence $\sigma = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots$ where $\delta_\mathcal{M}(s_i, a_i, s_{i+1}) > 0$ for all $i \in \mathbb{N}$. A finite path $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots \xrightarrow{a_{n-1}} s_n$ is a prefix of an infinite path. We denote the sets of all finite and infinite paths of $\mathcal{M}$ starting from state s by $FPath_{\mathcal{M},s}$ and $IPath_{\mathcal{M},s}$, respectively.*

The choice of action to take at each step of the execution of an MDP $\mathcal{M}$ is made by a *policy*, which can base its decision on the history of $\mathcal{M}$ up to the current state.

Definition 4. Policy. *Let $\mathcal{M} = \langle S, \overline{s}, A, \delta_\mathcal{M}, AP, Lab\rangle$ be an MDP. A (deterministic) policy over $\mathcal{M}$ is a function $\pi : FPath_{\mathcal{M},\overline{s}} \to A$ such that, for any finite path σ ending in state s_n, $\pi(\sigma) \in A_{s_n}$. The set of all policies over $\mathcal{M}$ is denoted by $\Pi_\mathcal{M}$.*

In this work, we restrict ourselves to deterministic policies, as opposed to randomised policies where the choice of action is defined as a distribution over enabled actions in s_n. As we will see, for the problem tackled here, there always exists an optimal deterministic policy. Important classes of policy include those that are *memoryless* (which only base their choice on the current state, i.e., defined over S, also known as stationary or Markovian) and *finite-memory* (which need to track only a finite set of "modes", i.e., defined over $S \times \{1, ..., m\}$, $m \in \mathbb{N}$).

Under a particular policy π for $\mathcal{M}$, all nondeterminism is resolved and the behaviour of $\mathcal{M}$ is fully probabilistic. Formally, we can represent this using an (infinite) *induced discrete-time Markov chain*, whose states are finite paths of $\mathcal{M}$. This leads us, using a standard construction (Kemeny et al. 1976), to the definition of a probability measure $Pr^\pi_{\mathcal{M},s}$ over the set of infinite paths $IPath_{\mathcal{M},s}$.

Definition 5. Probability Measures and Expected Values for a Policy. *Let $P \subseteq IPath_{\mathcal{M},s}$. We write $Pr^\pi_{\mathcal{M},s}(P)$ to denote the probability of an infinite run of $\mathcal{M}$ under π yielding an infinite path in P. Analogously, let $f : IPath_{\mathcal{M},s} \to \mathbb{R}$. We write $E^\pi_{\mathcal{M},s}(f)$ to denote the* expected value *of f with respect to the probability measure $Pr^\pi_{\mathcal{M},s}$. We can then consider the maximum probabilities or expected values over all policies:*

$$Pr^{\max}_{\mathcal{M},s}(P) = \sup_\pi Pr^\pi_{\mathcal{M},s}(P) \quad (2)$$

$$E^{\max}_{\mathcal{M},s}(f) = \sup_\pi E^\pi_{\mathcal{M},s}(f) \quad (3)$$

Minimum values $Pr^{\min}_{\mathcal{M},s}(P)$ or $E^{\min}_{\mathcal{M},s}(f)$ are defined analogously by replacing sup *with* inf.

Using the definition above, we can now formalise the main problems we are interested on.

Problem 1. Probabilistic Reachability. *Let $\mathcal{M} = \langle S, \overline{s}, A, \delta_{\mathcal{M}}, AP, Lab\rangle$ be an MDP, and $G \subseteq S$. We define $reach_G \subseteq IPath_{\mathcal{M},s}$ as:*

$$reach_G = \{(s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots) \mid s_i \in G \text{ for some } i \in \mathbb{N}\} \quad (4)$$

Calculate the maximum probability of reaching a state in G, along with the corresponding optimal policy π^*, i.e., find:

$$Pr^{\max}_{\mathcal{M},s}(reach_G) \quad (5)$$

$$\pi^* = \arg\max_{\pi} Pr^{\pi}_{\mathcal{M},s}(reach_G) \quad (6)$$

Problem 2. Expected Cumulative Reward. *Let $\mathcal{M} = \langle S, \overline{s}, A, \delta_{\mathcal{M}}, AP, Lab\rangle$ be an MDP, and $r : S \times A \to \mathbb{R}_{\geq 0}$ be a* reward structure *over $\mathcal{M}$. We define $cumul_r : IPath_{\mathcal{M},s} \to \mathbb{R}_{\geq 0}$ such that:*

$$cumul_r(s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots) = \sum_{i=0}^{\infty} r(s_i, a_i) \quad (7)$$

Calculate the maximum expected value for $cumul_r$, along with the corresponding optimal policy π^*, i.e., find:

$$E^{\max}_{\mathcal{M},s}(cumul_r) \quad (8)$$

$$\pi^* = \arg\max_{\pi} E^{\pi}_{\mathcal{M},s}(cumul_r) \quad (9)$$

The *expected cumulative cost* problem is defined analogously for a *cost structure* $c : S \times A \to \mathbb{R}_{\geq 0}$ and replacing $\max$ with $\min$ in equations 8 and 9.

The problems above can be solved using standard MDP algorithms such as value or policy iteration (Puterman 1994) (which for these problems yield memoryless policies). Note that for the cumulative rewards and costs problems, the expected value might not converge. Thus, as part of a preprocessing step, one needs to perform graph analysis to identify states for which the reward is infinite (Forejt et al. 2011). Also, in order to guarantee convergence of solution methods such as value iteration, non-accepting zero-cost strongly connected components are removed as part of the preprocessing step (de Alfaro 1997).

3.3 Factored MDP Representation

The above representation of an MDP is known as a *flat* representation. While presenting most of our results, we will use such representation. However, in modelling terms, enumerating all the possible states and the transition function can be very cumbersome. Thus, we will use a *factored* representation for our models, where the state is decomposed into relevant *state features*, and the transition function is encoded over such features. We use a probabilistic STRIPS-like representation of factored MDPs, based on the PPDDL (Younes and Littman 2004) and PRISM (Kwiatkowska et al. 2011) modelling languages.

Definition 6. State Features. *A set of state features is a set $X = \{X_1, \dots, X_n\}$, where each X_i can take values in a finite domain $\mathtt{dom}(X_i)$. We write $\mathtt{val}(X)$ to denote the set of (partial) assignments over state features in X, i.e., $\mathtt{val}(X) = (\mathtt{dom}(X_1) \cup \{\top\}) \times \dots \times (\mathtt{dom}(X_n) \cup \{\top\})$ where $\top$ is used to denote that there is no assignment to a specific state feature. Given an assignment $v \in \mathtt{val}(X)$, we write $v(X_i)$ to denote the assignment of state feature X_i by v.*

We will use assignments over state features to represent factored states, action preconditions, and action effects. Furthermore, to simplify the presentation, in some cases we will represent assignments in the form $v = \langle (X_i = x_i), \dots, (X_j = x_j)\rangle$, representing that $v(X_i) = x_i$, and $v(X_j) = x_j$, and omitting the state features X_k such that $v(X_k) = \top$.

Definition 7. Factored MDP. *A (factored) MDP is a tuple $\mathcal{M} = \langle X, \overline{x}, A\rangle$ where:*

- *$X = \{X_1, \dots, X_n\}$ is a set of state features;*
- *$\overline{x} \in \mathtt{val}(X)$ is the initial state, represented as a total assignment of values to the state features, i.e., $\overline{x}(X_i) \neq \top$ for all $X_i \in X$;*
- *$A = \{a_1, \dots, a_m\}$ is a finite set of actions. Each $a \in A$ is a tuple of the form $a = \langle pre_a, \mathit{eff}_a\rangle$, where:*
  - *$pre_a \in \mathtt{val}(X)$ is a set of preconditions over the state features that must hold for a to be applicable. $pre_a(X_i) = \top$ means that the value of X_i is irrelevant for the applicability of a;*
  - *$\mathit{eff}_a \in Dist(\mathtt{val}(X))$ is the distribution of possible outcomes (effects) of a, i.e., $\mathit{eff}_a = \sum_{j=1}^{m} p_j : \mathit{eff}_{a,j}$, $\mathit{eff}_{a,j} \in \mathtt{val}(X)$. In this case, $\mathit{eff}_{a,j}(X_i) = \top$ means that the j-th possible outcome of a does not change the value of state feature X_i;*

Definition 8. Enabled Actions. *Let $\mathcal{M} = \langle X, \overline{x}, A, AP, Lab\rangle$ be an MDP and $x \in \mathtt{dom}(X_1) \times \dots \times \mathtt{dom}(X_n)$ a factored state. The set of enabled actions in x is defined as:*

$$A_x = \{a \in A \mid pre_a(X_i) = \top \text{ or } pre_a(X_i) = x(X_i), \text{ for all } i \in \{0, \dots, n\}\} \quad (10)$$

Definition 9. Action Outcome. *Let $x \in \mathtt{val}(X)$ be a factored state, and $a = \langle pre_a, \sum_{j=1}^{m} p_j : \mathit{eff}_{a,j}\rangle \in A_x$. Outcome j of action a has probability p_j, and yields state x_a^j defined as:*

$$x_a^j(X_i) = \begin{cases} x(X_i) & \text{if } \mathit{eff}_j(X_i) = \top \\ \mathit{eff}_j(X_i) & \text{otherwise} \end{cases} \quad (11)$$

The flat and factored MDP representations are equivalent in terms of modelling power: a flat MDP can be represented as a factored MDP with one state factor, and we can also build a flat MDP from a factored MDP.

Definition 10. Equivalent Flat MDP. *Let $\mathcal{M}^{fact} = \langle X, \overline{x}, A\rangle$ be a factored state representation of an MDP. The equivalent flat state MDP representation is given by tuple $\mathcal{M}^{flat} = \langle S, \overline{x}, A, \delta_{\mathcal{M}}, AP, Lab\rangle$ where:*

- $S = \mathtt{dom}(X_1) \times \ldots \times \mathtt{dom}(X_n)$;
- *For $s, s' \in S$ and $a = \langle pre_a, \mathit{eff}_a\rangle \in A_x$ the transition function is defined as:*

$$\delta_{\mathcal{M}}(s,a,s') = \begin{cases} p_j & \text{if } a \in A_s \text{ and } s' = s_a^j \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

- $AP = \bigcup_{i=1}^{n} \bigcup_{v \in \mathtt{dom}(X_i)} (X_i = v)$;
- *Lab maps to each state the values of its corresponding state features.*

Example 2. *Consider a robot that can move between two locations v_1 and v_2. A water bottle is placed in location v_1 and the robot can pick it up and place it back in both locations. We assume a probability 0.8 of successfully picking up the object without breaking it, a probability 0.9 of successfully placing the object without breaking it, and that the robot can navigate between locations v_1 and v_2 without failing. We can model this example as a factored MDP, with $X = \{robot\_loc, obj\_state\}$, where:*

$$\mathtt{dom}(robot\_loc) = \{v_1, v_2\} \tag{13}$$

$$\mathtt{dom}(obj\_state) = \{at\_v_1, at\_v_2, with\_rob, broken\} \tag{14}$$

We assume both the robot and the object start at location v_1, hence:

$$\overline{x} = \langle (robot\_loc = v_1), (obj\_state = at\_v_1)\rangle \tag{15}$$

The action set is formed of navigation actions between v_1, and v_2, and the ability to pick up and place the object in v_1 and v_2. We exemplify the action definitions for location v_1, with the actions for location v_2 being analogous[*]

$$\begin{aligned} move\_to\_v_2 = \quad & pre = \langle (robot\_loc = v_1)\rangle \\ & \mathit{eff} = 1.0 : \langle (robot\_loc = v_2)\rangle \end{aligned} \tag{16}$$

$$\begin{aligned} pick\_at\_v_1 = \quad & pre = \langle (robot\_loc = v_1), \\ & \qquad (obj\_state = at\_v_1)\rangle \\ & \mathit{eff} = 0.8 : \langle (obj\_state = with\_rob)\rangle + \\ & \qquad 0.2 : \langle (obj\_state = broken)\rangle \end{aligned} \tag{17}$$

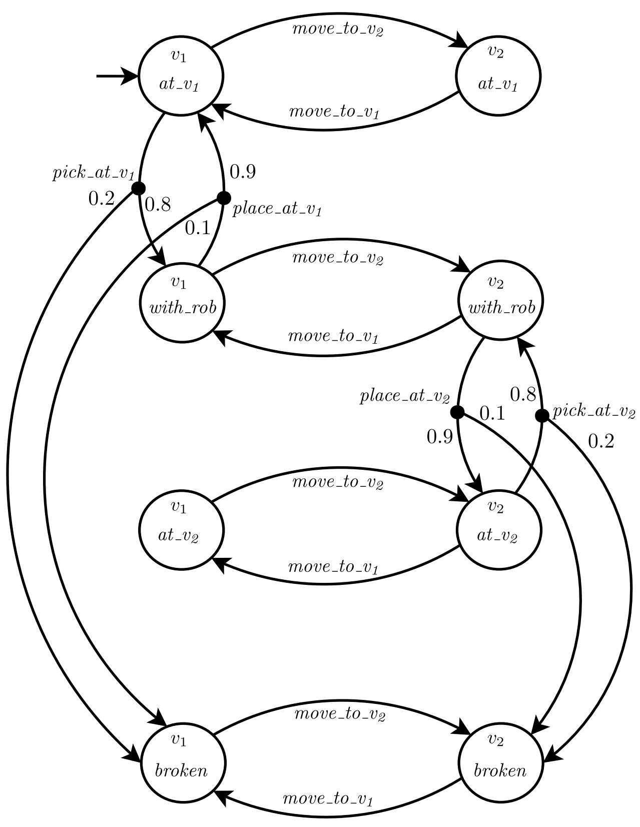


Figure 2. Transition system representation of the flat MDP described in Example 2.

$$\begin{aligned} place\_at\_v_1 = \quad & pre = \langle (robot\_loc = v_1), \\ & \qquad (obj\_state = with\_rob)\rangle \\ & \mathit{eff} = 0.9 : \langle (obj\_state = at\_v_1)\rangle + \\ & \qquad 0.1 : \langle (obj\_state = broken)\rangle \end{aligned} \tag{18}$$

One can "flatten" the representation, resulting on the flat MDP depicted in Figure 2 as a transition system.

Note that the factored representation is much more compact than the flat one. In fact, the size of the equivalent flat MDP representation is exponential in the number of state features in the factored MDP representation. The traditional approaches for solving MDPs, such as value iteration, are based on full (flat) state enumeration. There are other search techniques that try to take advantage of the compactness of the flat representation, but in this work we use an adaptation of value iteration to solve our model, and leave adapting more efficient search techniques to our problem as future work.

[*]For simplicity, in this work we only use *grounded* action definitions, i.e., we do not allow the use of variable symbols in the action definitions.

Finally, another representation of factored MDPs is based on representing the action outcomes as dynamic Bayesian networks (DBNs) (Boutilier et al. 1999). A DBN representation of the transition function can be more compact than the representation used in our work, when a given action has an exponential number of outcomes. We choose the STRIPS-like factored representation because it is much more intuitive for non-specialists, hence it makes modelling problems much easier from the user point-of-view. Furthermore, in our mobile robot application domains, actions do not typically have an exponential number of outcomes, thus the use of DBNs does not bring any advantage in terms of model compactness. In fact, modelling the navigation actions for the mobile robot using DBNs is quite cumbersome.

3.4 Linear Temporal Logic

3.4.1 Syntax and Semantics

Linear temporal logic (LTL) is an extension of propositional logic which allows reasoning about infinite sequences of states. It was developed as a means for formal reasoning about concurrent systems (Pnueli 1981), and provides a convenient, flexible, and powerful way to formally specify a variety of qualitative properties. Due to this fact, its use as a specification language for robot tasks is becoming more widespread (Belta et al. 2007; Kress-Gazit et al. 2009).

Definition 11. Syntax. *Let $p \in AP$. LTL formulas φ over atomic propositions AP are defined using the following grammar:*

$$\varphi ::= true \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathtt{X}\,\varphi \mid \varphi\,\mathtt{U}\,\varphi \tag{19}$$

The propositional connectives have the usual meaning. 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 second argument will eventually become true in some state, and the first argument will be continuously true until this point.

Definition 12. Semantics. *Let $\sigma = \sigma_0\sigma_1 \ldots \in (2^{AP})^\omega$. The notion of satisfaction, $\models$, is defined as follows:*

- $\sigma \models true$;
- $\sigma \models p$ *if and only if* $p \in \sigma_0$;
- $\sigma \models (\neg\varphi)$ *if and only if* $\sigma \not\models \varphi$;
- $\sigma \models (\varphi \wedge \psi)$ *if and only if* $\sigma \models \varphi$ *and* $\sigma \models \psi$;
- $\sigma \models (\mathtt{X}\,\varphi)$ *if and only if* $\sigma_1\sigma_2 \ldots \models \varphi$;
- $\sigma \models (\varphi\,\mathtt{U}\,\psi)$ *if and only if there exists* $t' \geq 0$ *such that* $\sigma_{t'}\sigma_{t'+1} \ldots \models \psi$ *and for all* $t < t'$, $\sigma_t\sigma_{t+1} \ldots \models \varphi$.

We also define the ω-language of all infinite sequences that satisfy φ as:

$$\mathcal{L}(\varphi) = \{\sigma \in (2^{AP})^\omega \mid \sigma \models \varphi\} \tag{20}$$

The other propositional connectives can be derived from the ones above in the usual way. Other useful LTL operators can be derived from the ones above. Of particular interest for our work is the "eventually" operator $\mathtt{F}\,\varphi$, which requires that φ is satisfied in some future state:

$$\mathtt{F}\,\varphi \equiv true\,\mathtt{U}\,\varphi \tag{21}$$

One can straightforwardly adapt the notion of satisfaction to infinite paths over an MDP.

Definition 13. Evaluation of MDP Paths. *For an MDP $\mathcal{M}$ and $\sigma = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \in IPath_{\mathcal{M},s}$, we say that σ satisfies an LTL formula φ if $Lab(s_0)Lab(s1)\ldots \models \varphi$. We also write $\sigma \models \varphi$ in this case.*

Thus, the notion of satisfaction can be seen as a property evaluated over infinite paths of $\mathcal{M}$, and we can define the problem of maximising the probability of satisfaction of an LTL formula.

Problem 3. Probabilistic Satisfaction. *Let $\mathcal{M} = \langle S, \overline{s}, A, \delta_{\mathcal{M}}, AP, Lab\rangle$ be an MDP, and φ an LTL formula over AP. Calculate the maximum probability of satisfying φ, along with the corresponding optimal policy π^*, i.e., find:*

$$Pr^{\max}_{\mathcal{M},s}(\varphi) = Pr^{\max}_{\mathcal{M},s}(\{\sigma \in IPath_{\mathcal{M},s} \mid \sigma \models \varphi\}) \tag{22}$$

$$\pi^* = \arg\max_{\pi} Pr^{\pi}_{\mathcal{M},s}(\varphi) \tag{23}$$

It is known that this problem can be reduced to a probabilistic reachability problem (Problem 1) on an MDP obtained by the *product* of $\mathcal{M}$ with a *deterministic Rabin automaton* obtained from φ (Vardi 1985). In this work we are interested in the co-safe fragment of LTL, as defined below, where deterministic finite automata (DFA) suffice. Hence, we refer the reader to Vardi (1985) for more details about Problem 3 for LTL formulas outside the co-safe fragment, and focus on co-safe LTL.

3.4.2 Co-safe LTL and Deterministic Finite Automata

The semantics of LTL is defined over infinite sequences of atomic propositions. However, in this work, we are interested in minimising expected time to task completion. Thus, we will use the fragment of LTL that can be unambiguously satisfied by finite sequences. This fragment is named co-safe LTL, and is composed of the LTL formulas that always have a finite *good prefix* (Kupferman and Vardi 2001).

Definition 14. Good Prefix. *Let φ be an LTL formula and $\sigma = \sigma_0\sigma_1 \ldots \in (2^{AP})^\omega$ such that $\sigma \models \varphi$. We say that σ has a good prefix for φ if there exists $n \in \mathbb{N}$ for which the truncated finite sequence $\sigma|_n = \sigma_0\sigma_1 \ldots \sigma_n$ is such that for every $\sigma' \in (2^{AP})^\omega$ the concatenation $\sigma|_n \cdot \sigma' \models \varphi$.*

Definition 15. Co-safe LTL. *Let φ be an LTL formula. We say that φ is a co-safe LTL formula if for all $\sigma \in \mathcal{L}(\varphi)$, σ has a good prefix for φ.*

Remark 1. Syntactic Restriction. *In this work, we will only use the syntactically co-safe class of LTL formulas, i.e., formulas where negation ($\neg$) is only applied to atomic propositions, and only the* X*,* U*, and* F *operators occur. While there are co-safe LTL formulas that do not satisfy this syntactic restriction, these happen mostly due to redundancy in the specification, thus, from a specification point of view, the language of interest is the syntactically co-safe fragment.*

As stated above, any co-safe LTL formula φ can be translated to a DFA that accepts exactly the set of good prefixes for φ, i.e., the set of finite sequences that will satisfy φ regardless of how they are "completed" (for all suffixes of infinite length).

Definition 16. DFA. *A deterministic finite automaton (DFA) is a tuple $\mathcal{A} = \langle Q, \overline{q}, Q_F, \Sigma, \delta_{\mathcal{A}} \rangle$, where:*

- *Q is a finite set of states;*
- *$\overline{q} \in Q$ is the initial state;*
- *$Q_F \subseteq Q$ is the set of accepting states;*
- *Σ is the alphabet;*
- *$\delta_{\mathcal{A}_\varphi} : Q \times \Sigma \to Q$ is a transition function.*

The transition function $\delta_{\mathcal{A}}$ is extended to a function over finite sequences $\delta^+_{\mathcal{A}} : Q \times \Sigma^* \to Q$ by applying it sequentially to the finite sequence of visited states. The language accepted by $\mathcal{A}$ is then defined as:

$$\mathcal{L}(\mathcal{A}) = \{\rho \in \Sigma^* \mid \delta^+_{\mathcal{A}}(\overline{q}, \rho) \in Q_F\} \qquad (24)$$

Proposition 1. Kupferman and Vardi (2001). *Let φ be a co-safe LTL formula. There exists a DFA $\mathcal{A}_\varphi = \langle Q, \overline{q}, Q_F, 2^{AP}, \delta_{\mathcal{A}_\varphi} \rangle$ such that:*

$$\mathcal{L}(\mathcal{A}_\varphi) = \{\rho \in (2^{AP})^* \mid \rho \text{ is a good prefix for } \varphi\} \qquad (25)$$

Given that a good prefix satisfies φ regardless of how it is "completed', once a run reaches an accepting state, it never leaves Q_F from then on.

Proposition 2. *Let φ be a co-safe LTL formula, and $\mathcal{A}_\varphi = \langle Q, \overline{q}, Q_F, 2^{AP}, \delta_{\mathcal{A}_\varphi} \rangle$:*

$$\text{For all } q_F \in Q_F \text{ and } \alpha \in 2^{AP},\ \delta_{\mathcal{A}_\varphi}(q_F, \alpha) \in Q_F \qquad (26)$$

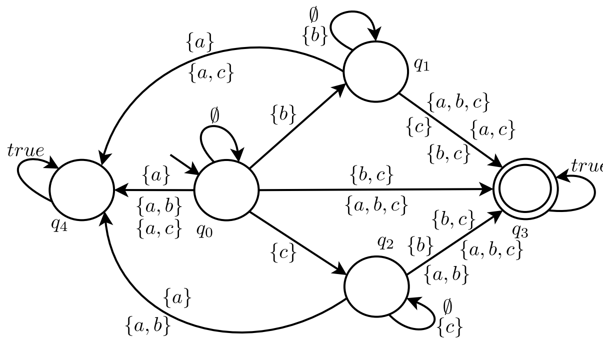


Figure 3. A DFA $\mathcal{A}_\varphi$ for $\varphi = ((\neg a)\,\mathtt{U}\,b) \wedge ((\neg a)\,\mathtt{U}\,c)$, with the transition labels depicted.

Proof. Let $q_F \in Q_F$ and $\alpha \in 2^{AP}$. Given that, by Proposition 1, all $\rho \in (2^{AP})^*$ such that $\delta^+_{\mathcal{A}}(\overline{q}, \rho) = q_F$ is a good prefix for φ, then $\rho.\alpha \in (2^{AP})^*$ is also a good prefix for φ. Thus, $\delta^+_{\mathcal{A}}(\overline{q}, \rho.\alpha) = \delta_{\mathcal{A}_\varphi}(q_F, \alpha) \in Q_F$ because $\mathcal{A}_\varphi$ accepts *exactly* the good prefixes for α.

Given Proposition 2, we will assume that the DFA is simplified such that Q_F is a singleton $\{q_F\}$ such that $\delta_{\mathcal{A}_\varphi}(q_F, \alpha) = q_F$ for all $\alpha \in 2^{AP}$.

Example 3. *Consider the syntactically co-safe LTL specification $\varphi = ((\neg a)\,\mathtt{U}\,b) \wedge ((\neg a)\,\mathtt{U}\,c)$. In Figure 3, we depict the corresponding DFA $\mathcal{A}_\varphi$. Note that the transitions are labelled by subsets of AP, depicted next to each transition. Each subset of AP labelling a transition can trigger it, evolving the DFA state from the transition source to its target.*

3.4.3 Product MDP and Policies for Co-Safe LTL

The restriction of Problem 3 to co-safe LTL specifications can be reduced to an instance of Problem 1, by building a product MDP.

Definition 17. Product MDP. *Let $\mathcal{M} = \langle S, \overline{s}, A, \delta_{\mathcal{M}}, AP, Lab \rangle$ be an MDP, and $\mathcal{A}_\varphi = \langle Q, \overline{q}, \{q_F\}, 2^{AP}, \delta_{\mathcal{A}_\varphi} \rangle$ be a DFA obtained from co-safe LTL formula φ. We define the* product MDP *$\mathcal{M}_\varphi = \mathcal{M} \otimes \mathcal{A}_\varphi = \langle S_\varphi, \overline{s_\varphi}, A, \delta_{\mathcal{M}_\varphi}, AP, Lab_\varphi \rangle$, where:*

- *$S_\varphi = S \times Q$*
- *$\overline{s_\varphi} = (\overline{s}, \delta_{\mathcal{A}_\varphi}(\overline{q}, Lab(\overline{s})))$*
- *$\delta_{\mathcal{M}_\varphi}((s, q), a, (s', q')) = \begin{cases} \delta_{\mathcal{M}}(s, a, s') & \text{if } q' = \delta_{\mathcal{A}_\varphi}(q, Lab(s')) \\ 0 & \text{otherwise} \end{cases}$*
- *$Lab_\varphi((s, q)) = Lab(s)$*

We also define the set of accepting states $acc_\varphi \in S_\varphi$ as:

$$acc_\varphi = \{(s, q_F) \in S_\varphi\} \qquad (27)$$

Note that, since we know the initial state $\overline{s}$ of the original MDP, we can build the state space of the product MDP in a forward reachability fashion, thus only considering the set of states that are reachable from the initial state. The product MDP $\mathcal{M}_\varphi$ behaves like the original MDP $\mathcal{M}$, but is augmented with information about the satisfaction of φ. Once a path of $\mathcal{M}_\varphi$ reaches an *accepting state* (i.e., a state of the form (s, q_F)), it is a good prefix for φ, and we are sure that φ is satisfied. Furthermore, $\mathcal{M}_\varphi$ preserves the probabilities of paths from $\mathcal{M}$ (see, e.g., Baier and Katoen (2008)). Thus:

Proposition 3. *Let $\mathcal{M} = \langle S, \overline{s}, A, \delta_\mathcal{M}, AP, Lab\rangle$ be an MDP and φ be a co-safe LTL formula. Then:*

$$Pr^{\max}_{\mathcal{M},s}(\varphi) = Pr^{\max}_{\mathcal{M}_\varphi,s_\varphi}(reach_{acc_\varphi}) \tag{28}$$

Thus we can reduce Problem 3 in $\mathcal{M}$ to an instance of Problem 1 in $\mathcal{M}_\varphi$.

Remark 2. Policy Class. *As already mentioned, finding optimal solutions for Problem 1 can be found using standard MDP techniques such as value or policy iteration, and there always exists an optimal* memoryless *policy. Thus, when solving for $\mathcal{M}_\varphi$, we obtain a policy $\pi^* : S \times Q \to A$. This policy can be seen as a* finite-memory *policy for $\mathcal{M}$, where the states of Q represent the different policy modes. Intuitively, the elements q of states (s, q) in $\mathcal{M}_\varphi$ represent the modes of the finite-memory policy, which keep track of the relevant (for the satisfaction of φ) parts of the path executed by the MDP so far, and $\pi^*(s, q)$ gives the action to take in state s for mode q. Precise details of a similar policy construction (for probabilistic safety properties) can be found in Forejt et al. (2011). Thus, finite-memory policies suffice to solve Problem 3.*

4 MDP Model of Mobile Service Robot Tasks

In this section, we describe how we model a mobile service robot working in indoor environments as an MDP. This model has been developed in the context of the STRANDS project (Hawes et al. 2017), with the aim of deploying mobile robots in everyday environments during long periods of time. It has been used for robot deployments in the project's later years, with a total combined time of more than 8 months of robot execution. In order to achieve this robustness, we have focused on a model where probabilistic guarantees of robot performance can be obtained, while also considering scalability of the state space, in order to provide the ability for online policy generation. This allows for the system to be easily integrated with other components used for higher level task management and scheduling (Mudrová et al. 2015).

4.1 Primitive Skills

In this work, we assume the robot has a set of implemented *primitive skills*. Examples of such skills are navigating between locations in the environment, checking whether a door is open, waiting for it to open, or searching for an object at a certain location. We also assume that the primitive skill implementations are in charge of deciding whether to retry a given behaviour or not. When a skill outputs failure, it means that it cannot be executed successfully on the current instance of task execution. This is to avoid situations where the robot keeps trying the same action until it is successful, allowing for the generation of policies that reason about what to do when a certain action fails. This allows the robot to try other ways of achieving the goal. In our implementation, the set of skills is assumed to be implemented as *actionlib* action servers within the ROS middleware.

4.2 Topological Map

Topological map representations typically consist of a set of nodes that represent physical locations in the environment and a set of edges representing the robot's ability to move between these locations. The nodes are associated with descriptions of the locations (e.g. kitchen, workstation, etc), and the edges represent a connection between two of these locations. In our representation, each edge is associated with a particular continuous navigation action (e.g, laser based navigation, door crossing, docking to charging station, ...) that moves the robots between the associated nodes. This representation allows one to implement a unified system that can tackle different challenges in terms of continuous navigation.

Definition 18. Topological Map. *A topological map is a tuple $\mathcal{T} = \langle V, E, N, nav\rangle$, where:*

- *$V = \{v_1, ..., v_n\}$ is a set of relevant locations in the environment, encoded as robot poses on a global frame, i.e., each v_i is of the form (x, y, θ);*
- *$E \subseteq V \times V$ represents the ability to navigate between two environment locations without visiting another location;*
- *N is a finite set of possible continuous navigation action implementations that the robot can execute, part of its primitive skills;*
- *$nav : E \to N$ is a function that maps each edge $e = (v_i, v_j)$ to the continuous navigation action to be executed to drive the robot from v_i to v_j.*

Example 4. *The graph structure presented in Figure 1 represents a topological map, where (bi-directional) edges represent possible navigation actions between nodes,* *overlayed over the 2D metric representation of the office environment. We define $N = \{laser, door\}$, with laser representing standard laser based navigation, and door representing a special behaviour for traversing narrow doors. Red edges (v_r, v'_r) in the figure are such that $nav(v_r, v'_r) = door$, and blue edges (v_b, v'_b) are such that $nav(v_b, v'_b) = laser$.*

4.3 Navigation Edge Partition

We also distinguish between two types of edges. We split the set of edges between edges the robot can try to navigate through normally, and edges for which a *guard fulfilling* action must be executed. This guard fulfilling action can be for example a checking action (e.g., waiting for a closed door to be opened by a human, checking if a narrow corridor is clear and navigation through it is possible), or a manipulation action (if the robot has an arm, opening the door), and is assumed to be implemented as part of the robot's primitive skills.

Definition 19. Navigation Edge Partition. *We partition the set of edges E of a topological map $\mathcal{T}$ as $E = E_u \cup E_g$, where:*

- *E_u is a set of unguarded edges that the robot can try to navigate through without any guard fulfilling action;*
- *E_g is a set of guarded edges where a guard fulfilling action is required to be executed before navigation.*

For $e \in E_g$, we write $guard_e$ to represent the guard fulfilling action associated to e.

Example 5. *For the topological map depicted in Figure 1, we define E_g to be the set of edges depicted in red, as these are representing navigation through a door that might be closed. Hence, the robot can only navigate on these edges after checking if the corresponding door is open. We assume the robot has a $check\_door$ behaviour implemented that returns true if and only if the door is open. Thus, for all $e \in E_g$, $guard_e = check\_door$.*

4.4 Probabilistic Edge Models

We assume that, while moving in its environment, the robot gathers data on the failure or success of navigation through an edge, and the time taken to do so. One can use this data to build probabilistic edge models, for example using the technique presented in Pulido Fentanes et al. (2015).

Definition 20. Edge Model. *Let $E = E_g \cup E_u$ be edges of a topological graph, and $e = (v_i, v_j) \in E$. We define the outcome probabilities when navigating from v_i to v_j at time $t \in \mathbb{R}_{\geq 0}$ as $p^t_e \in Dist(V \cup \{\perp\})$, where $p^t_e(v)$ represents the probability of visiting v immediately after v_i when trying to navigate between v_i and v_j, and $p^t_e(\perp)$ represents the probability of failing to navigate between v_i and v_j, i.e., getting into as situation from which the robot cannot recover. We also define the expected time for the navigation attempt from v_i to v_j at time t as τ^t_e.*

Similarly, we assume the existence of models for guard fulfilling actions.

Definition 21. Guard Fulfilling Action Model. *Let $e = (v_i, v_j) \in E_g$. For time $t \in \mathbb{R}_{\geq 0}$, we define the probability distribution of the guard fulfilling action $guard_e$ being successful as $p^t_{guard_e} \in Dist(\{0, 1\})$ and the expected time for its execution as $\tau^t_{guard_e}$.*

Example 6. *Consider edge (v_3, v_4) from the topological map depicted in Figure 1. A possible edge model for a given time t (e.g., tomorrow at 9am) can be $p^t_{v_3,v_4} = 0.7 : v_4 + 0.2 : v_0 + 0.1 : \perp$. This distribution means that when trying to navigate between v_3 and v_4 at time t, the robot can successfully reach v_4 with probability 0.7 The laser based navigation is able to avoid obstacles, such as humans. While doing so, the robot sometimes ends up in v_0 with probability 0.2. Finally, when the area is too crowded, the robot is not able to perform the navigation action successfully and gets stuck, requiring human help. This is represented by $\perp$ in our model. The prediction for the probability of such occurrence in this case is 0.1. Note that, for a different time of day, these predictions might be different. For example, assume v_3 and v_4 are located around the staff kitchen, hence the area is more crowded at lunch time. Thus, for a time t' around lunch time the edge traversal model can be, for example, $p^{t'}_{v_3,v_4} = 0.6 : v_4 + 0.4 : \perp$, i.e., with probability 0.4 the action will fail. Regarding expected time, one can assume, for example, $\tau^t_{v_3,v_4} = 8.5$, and $\tau^{t'}_{v_3,v_4} = 12$, meaning that at time t, the expected time to attempt to navigate between v_3 and v_4 is 8.5 seconds, and at time t', due to the larger crowd, is 12 seconds.*

We also illustrate models for guard fulfilling actions. For example, for $e = (v_3, v_2)$, $p^t_{guard_e} = 0.9 : 1 + 0.1 : 0$, meaning that at time t there is a probability of 0.9 of the door being open. Checking if the door is open can be done very quickly and in almost constant time, hence one can assume that $\tau^t_{guard_e} = 0.01$ for all time points t. Finally, traversing the door can be easily done by the specialised door passing behaviour, hence $p^t_{v_4,v_5} = 0.99 : v_5 + 0.01 : \perp$, and $\tau^t_{v_4,v_5} = 3$.

4.5 Navigation MDP

With the models presented in the previous subsections, we are now able to present the underlying probabilistic high-level navigation model we propose for a mobile robot. This is a *navigation MDP*, which is instantiated to a specific time t using the edge models described above, creating a "snapshot" of the world model at that given time instance.

The definition of *Navigation MDP* is presented using the factored MDP representation presented in Subsection 3.3.

Definition 22. Navigation MDP. *Let $\mathcal{T} = \langle V, E, N, nav\rangle$ be a topological map, with $E = E_u \cup E_g$, $\overline{v} \in V$ be the initial location of the robot, and $t \in \mathbb{R}$ be a point in time. We define the navigation MDP as $\mathcal{M}^t_{\mathcal{T}} = \langle X_{\mathcal{T}}, \overline{x^t_{\mathcal{T}}}, A^t_{\mathcal{T}}\rangle$ where:*

- $X_{\mathcal{T}} = \{loc\} \cup \bigcup_{e \in E_g} trav_e$, *where* $\mathtt{dom}(loc) = V \cup \{\bot\}$ *and* $\mathtt{dom}(trav_e) = \{-1, 0, 1\}$, *i.e., the state space is composed of a feature indicating the location of the robot, along with whether it failed to navigate, and, for each $e \in E_g$, a feature indicating whether e is traversable: -1 indicates that traversability is still unknown, i.e., the guard fulfilling action has not been executed, 0 indicates e is not traversable, i.e., the guard fulfilling action failed, and 1 indicates that the edge is traversable, i.e., the guard fulfilling action was successful;*
- $\overline{x^t_{\mathcal{T}}} = \overline{v} \times \bigtimes_{e \in E_g} -1$, *where $\overline{v}$ is the location of the robot at time t (which can be observed at model building time). The robot starts in its initial location, and, for edges requiring a guard fulfilling action, all edge traversability features are initialised as unknown;*
- $A^t_{\mathcal{T}} = \bigcup_{e \in E} nav_e \cup \bigcup_{e \in E_g} guard_e$ *where, for* $e = (v, v')$:
  - *If $e \in E_u$, then:*

$$\begin{aligned} nav_e = \quad & pre = \langle (loc = v)\rangle, \\ & eff = \sum_{v' \in supp(p^t_e)} p^t_e(v') : \langle (loc = v')\rangle \end{aligned} \tag{29}$$

  - *If $e \in E_g$, then:*

$$\begin{aligned} nav_e = \quad & pre = \langle (loc = v), (trav_e = 1)\rangle, \\ & eff = \sum_{v' \in supp(p^t_e)} p^t_e(v') : \langle (loc = v')\rangle \end{aligned} \tag{30}$$

$$\begin{aligned} guard_e = \quad & pre = \langle (loc = v), (trav_e = -1)\rangle, \\ & eff = p^t_{guard_e}(0) : \langle (trav_e = 0)\rangle + \\ & \quad p^t_{guard_e}(1) : \langle (trav_e = 1)\rangle \end{aligned} \tag{31}$$

Example 7. *Consider the topological map depicted in Fig 1, and the edge models described in Example 6. Assume the robot starts in node v_3. The navigation MDP at time t is such that:*

- $X_{\mathcal{T}} = \{loc\} \cup \{trav_e \mid e \in E_r\}$, *where E_r is the set of edges depicted in red, and :*

$$\mathtt{dom}(loc) = \{v_0, ..., v_{15}, \bot\} \tag{32}$$

$$\mathtt{dom}(trav_e) = \{-1, 0, 1\} \textit{ for all } e \in E_r \tag{33}$$

- $\overline{x^t_{\mathcal{T}}} = (v_3, -1, -1, -1, -1, -1, -1)$
- *We present the definition of some of the available actions:*

$$\begin{aligned} nav_{v_3,v_4} = \quad & pre = \langle (loc = v_3)\rangle, \\ & eff = 0.7 : \langle (loc = v_4)\rangle + \\ & \quad 0.2 : \langle (loc = v_0)\rangle + 0.1 : \langle (loc = \bot)\rangle \end{aligned} \tag{34}$$

$$\begin{aligned} nav_{v_3,v_2} = \quad & pre = \langle (loc = v_3), (trav_{v_3,v_2} = 1)\rangle, \\ & eff = 0.99 : \langle (loc = v_2)\rangle + \\ & \quad 0.01 : \langle (loc = \bot)\rangle \end{aligned} \tag{35}$$

$$\begin{aligned} guard_{v_3,v_2} = \quad & pre = \langle (loc = v_3), \\ & \quad (trav_{v_3,v_2} = -1)\rangle, \\ & eff = 0.9 : \langle (trav_{v_3,v_2} = 1)\rangle + \\ & \quad 0.1 : \langle (trav_{v_3,v_2} = 0)\rangle \end{aligned} \tag{36}$$

Other action definitions are analogous.

Remark 3. Resetting Guard Fulfilling Actions Result. *In Definition 22, we assume that the outcome of a guard fulfilling action is* persistent, *in the sense that, once a given guard fulfilling action is executed, the value of the associated state variable does not change any more. This approach assumes that the environment is not changed by external agents, for example, after the robot successfully opens a door, the door is not closed any more. In many cases, this assumption is not valid, as the robot is deployed alongside humans that also change the state of the environment (e.g., a human closes a door). Hence, one can also take the approach where the successful guard fulfilling action outcomes are forgotten immediately after the action is executed successfully. To do so, one can replace all actions a that have as precondition the robot being in the origin node of the guarded edge by copies a^{-1}, a^0 and a^1, each for a possible value of the guard fulfilling action outcome. One of the copies resets the value of the outcome to -1 when it is 1, while the other copies keep the other two possible values unchanged.*

4.6 Modelling General Actions

In the previous subsections, we have discussed how to model the robot's navigation actions and their outcomes as an MDP. In order to allow for more involved behaviours than simple navigation, we also model the execution of actions in certain locations of the environment. We model these actions using a set of state features that are used to define the preconditions and effects of the action, and temporal models for probabilities of action effects and expected time for action execution, in the same vein as the probabilistic edge models described in Subsection 4.4. As with our definition of factored MDP, the following is based on the PPDDL (Younes and Littman 2004) and PRISM (Kwiatkowska et al. 2011) modelling languages.

Definition 23. General Action. *A general action is defined as a tuple* $\alpha = \langle X_\alpha, pre_\alpha, p^t_\alpha, \tau^t_\alpha, \rangle$ *where:*

- $X_\alpha = \{X_{\alpha,1}, ..., X_{\alpha_n}\}$ *is the set of state features relevant for the execution of* α*, i.e., the state features used to define* α*'s preconditions and effects;*
- $pre_\alpha \in \mathtt{val}(X_\alpha)$ *is the set of preconditions that must hold for* α *to be executable;*
- $p^t_\alpha \in Dist(X_\alpha)$ *is a temporal probabilistic model that represents the distribution of possible effects of* α*, at time* $t \in \mathbb{R}_{\geq 0}$*;*
- τ^t_α *represents the expected time for the execution of* α *at time* $t \in \mathbb{R}_{\geq 0}$.

Action α *is also associated to a primitive skill implementing the action behaviour on the mobile robot.*

Example 8. Searching for an Object. *Consider the topological map in Figure 1, and assume the robot can search for an object that might be located in* v_1*. We can model such a search action as* $search\_at\_v_1 = \langle X_{search\_at\_v_1}, pre_{search\_at\_v_1}, p^t_{search\_at\_v_1}, \tau^t_{search\_at\_v_1} \rangle$*, where:*

- $X_{search\_at\_v_1} = \{loc, object\_at\_v_1\}$*, with* $\mathtt{dom}(loc) = V \cup \{\perp\}$ *representing the position of the robot in the topological map, and* $\mathtt{dom}(object\_at\_v_1) = \{-1, 0, 1\}$*. For* $object\_at\_v_1$*,* -1 *represents that the robot does not know whether the object is at* v_1*,* 0 *represents that the robot has searched for the object at* v_1 *and has not found it, and* 1 *represents that the robot found the object at* v_1*;*
- $pre_{search\_at\_v_1} = \langle (loc = v_1), (object\_at\_v_1 = -1) \rangle$*;*
- $p^t_{search\_at\_v_1} \in Dist(\{0,1\})$ *is obtained form a probabilistic model of object location over time. For example, for a given* t *(e.g., tomorrow, at 10am), we can have* $p^t_{search\_at\_v_1} = 0.7 : (object\_at\_v_1 = 0) + 0.3 : (object\_at\_v_1 = 1)$*, and for* t' *(e.g., tomorrow, 5pm) we can have* $p^{t'}_{search\_at\_v_1} = 0.4 : (object\_at\_v_1 = 0) + 0.6 : (object\_at\_v_1 = 1)$[†]*;*
- *Similarly to the model of location over time, one can have* $\tau^t_{search\_at\_v_1} = 55$ *and* $\tau^{t'}_{search\_at\_v_1} = 33$*. The different expected times can be explained by the implementation of the primitive skill associated to* $search\_at\_v_1$*. If one assumes a timeout is implemented such that the primitive skill outputs "object not found" after a period of time, when the probability of the object being at* v_1 *is higher, the expected time for execution of the action is lower, as the action will finish before the timeout is reached more often.*

We require the action definitions to be *consistent* between each other, and in relation to the navigation MDP, in the sense that shared state features have the same domain and the same initial assignment.

Definition 24. Consistent Action Definitions. *Let* $\alpha_1 = \langle X_{\alpha,1}, pre_{\alpha,1}, p^t_{\alpha,1}, \tau^t_{\alpha,1} \rangle$ *and* $\alpha_2 = \langle X_{\alpha,2}, pre_{\alpha,2}, p^t_{\alpha,2}, \tau^t_{\alpha,2} \rangle$ *be action definitions, and* $\mathtt{dom}_1$ *and* $\mathtt{dom}_2$ *the domains of the state features in* $X_{\alpha,1}$ *and* $X_{\alpha,2}$*, respectively. We say that* α_1 *and* α_2 *are consistent with one another if for all* $X \in X_{\alpha,1} \cap X_{\alpha,2}$*:*

$$dom_1(X) = dom_2(X) \text{ and } \overline{x_1}(X) = \overline{x_2}(X) \tag{37}$$

With the (consistent) general action models presented above, we can build a *general actions* MDP, that represents the initial assignment of the state features relevant to the general actions, and the effects of those actions at a particular time t.

Definition 25. General Actions MDP. *Let* $\Lambda = \{\alpha_1, ..., \alpha_m\}$ *be a set of general actions, and* $t \in \mathbb{R}_{\geq 0}$ *a point in time. The General Actions MDP is defined as* $\mathcal{M}^t_\Lambda = \langle X_\Lambda, \overline{x^t_\Lambda}, A^t_\Lambda \rangle$*, where:*

- $X_\Lambda = \{X_1, \ldots, X_n\}$ *is the union of all* n *distinct state features relevant for the actions in* Λ*, i.e.:*

$$X_\Lambda = \bigcup_{\alpha \in \Lambda} X_\alpha \tag{38}$$

Note that certain actions can have state features in common, i.e., it is possible that $X_{\alpha_i} \cap X_{\alpha_j} \neq \emptyset$ *for some* i, j*;*

- $\overline{x^t_\Lambda} \in \mathtt{dom}(X_1) \times ... \times \mathtt{dom}(X_n)$ *is the assignment of values to the state variables at time* t*;*

[†]As with the spatio-temporal edge models, details of how such model can be learnt are outside the scope of this work. Here, we assume the model is an outside component that the robot can query.

- $A^t_\Lambda = \{a^t_{\alpha_1}, ..., a^t_{\alpha_m}\}$, *where each* $a^t_{\alpha_i}$ *is obtained directly from the general action definition of* $\alpha_i = \langle X_{\alpha_i}, pre_{\alpha_i}, p^t_{\alpha_i}, \tau^t_{\alpha_i}\rangle$, *i.e.:*

$$a^t_{\alpha_i} = \begin{array}{l} pre = \langle pre_{\alpha_i}\rangle \\ eff = \langle p^t_{\alpha_i}\rangle \end{array} \tag{39}$$

Example 9. Searching for an Object. *Assume that it is unknown whether the object is at* v_1 *and the robot is at* v_3. *The General Actions MDP for the search action defined in Example 8 at time* t' *is defined as* $\mathcal{M}^{t'}_{search\_at\_v_1} = \langle X_{search\_at\_v_1}, \overline{x}_{search\_at\_v_1}, \{a^{t'}_{search\_at\_v_1}\}\rangle$, *where:*

$$\overline{x}_{search\_at\_v_1} = (v_3, -1) \tag{40}$$

$$a^{t'}_{search\_at\_v_1} = \begin{array}{l} pre = \langle (loc = v_1), (object\_at\_v_1 = -1)\rangle, \\ eff = 0.4 : \langle (object\_at\_v_1 = 0)\rangle + \\ \quad 0.6 : (object\_at\_v_1 = 1)\rangle \end{array} \tag{41}$$

Adding more action definitions to the General Actions MDP would be done by adding their state features to the MDP, and their action preconditions and effects at time t' *to the set of actions of the MDP, in a similar manner to the one described above.*

The General Actions MDP is used to *extend* the navigation MDP model in order to incorporate general action execution. This is done by joining the state features and action definitions together.

Definition 26. Navigation and General Actions MDP. *Let* $t \in \mathbb{R}$ *be a point in time,* $\mathcal{M}^t_\mathcal{T} = \langle X_\mathcal{T}, \overline{x^t_\mathcal{T}}, A^t_\mathcal{T}\rangle$ *be a navigation MDP for time t and* $\mathcal{M}^t_\Lambda = \langle X_\Lambda, \overline{x^t_\Lambda}, A^t_\Lambda\rangle$ *be a General Actions MDP for time t. We define the Navigation and General Actions MDP* $\mathcal{M}^t_{\mathcal{T},\Lambda} = \langle X_{\mathcal{T},\Lambda}, \overline{x^t_{\mathcal{T},\Lambda}}, A^t_{\mathcal{T},\Lambda}\rangle$ *where:*

- $X_{\mathcal{T},\Lambda} = X_\mathcal{T} \cup X_\Lambda$
- $\overline{x_{\mathcal{T},\Lambda}}(X) = \begin{cases} \overline{x_\mathcal{T}}(X) & \text{if } X \in X_\mathcal{T} \\ \overline{x_\Lambda}(X) & \text{if } X \in X_\Lambda \end{cases}$
- $A_{\mathcal{T},\Lambda} = A^t_\mathcal{T} \cup A^t_\Lambda$

4.7 Expected Time Cost Structure

We finish the presentation of our modelling approach by defining a cost function over the (flat representation of the) Navigation and General Actions MDP. This represents the expected time to execute an action in $\mathcal{M}^t_\mathcal{T}$, and is obtained directly from the probabilistic models of action execution time.

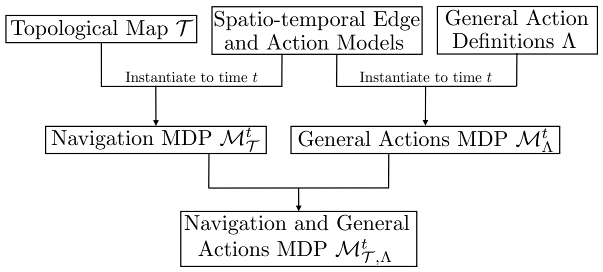

Topological Map $\mathcal{T}$
Spatio-temporal Edge and Action Models
General Action Definitions Λ
Instantiate to time t
Instantiate to time t
Navigation MDP $\mathcal{M}^t_\mathcal{T}$
General Actions MDP $\mathcal{M}^t_\Lambda$
Navigation and General Actions MDP $\mathcal{M}^t_{\mathcal{T},\Lambda}$

Figure 4. Diagram of the modelling approach.

Definition 27. Expected Time Cost Function. *Let* $\mathcal{M}^t_{\mathcal{T},\Lambda} = \langle S, \overline{s}, A, \delta_\mathcal{M}, AP, Lab\rangle$ *be the flat state representation of a Navigation and General Action MDP. We define the expected time cost function* $c^t_\tau : S \times A \to \mathbb{R}_{\geq 0}$ *as:*

$$c^t_\tau(s,a) = \begin{cases} \tau^t_e & \text{if } a = nav_e \text{ for some } e \in E \\ \tau^t_{guard_e} & \text{if } a = guard_e \text{ for some } e \in E_g \\ \tau^t_\alpha & \text{if } a = a_\alpha \text{ for some } \alpha \in \Lambda \end{cases} \tag{42}$$

We define the expected time cost function over $S \times A$ as is standard with MDPs. This way, we can apply known results, such as the solution of Problem 2 directly. Furthermore, the technique presented in the next section also assumes a cost structure over state-action pairs.

4.8 Summary and Discussion

In Figure 4, we depict the overall approach presented in this section. The designer provides a topological map and general action definitions. These are instantiated into MDPs for a specific point in time t using the spatio-temporal models. Finally, the two MDPs are put together into a Navigation and General Action MDP.

We note the relevance of the spatio-temporal models for longer autonomy applications: by having a dynamic model we can more accurately capture the environment. In turn, this allows us to provide more accurate formal performance guarantees using the techniques presented in the next section. Note that our approach of taking a "snapshot" of the dynamic model and using it for planning also allows for much better scalability, as each planning instance keeps Markovian action outcomes. We argue that this approach provides a good trade-off between model accuracy and planning scalability. Finally, note that these models are optional, and for applications where spatio-temporal models are not present or needed, one can use a stationary model instead, for example a simple model based on the distance between locations and assuming a constant speed. Our implementation provides general interfaces to plug in different environmental models to the MDP construction.

5 Policy Generation for Partially Satisfiable Task Specifications

In this section, we present a general approach for minimising the expected cost to satisfy a co-safe LTL specification φ, where the probability of satisfying φ can be less than one. Furthermore, the approach allows for one to do "as much of the task as possible", generating policies that get as close to the co-safe LTL goal as possible, given the environmental restrictions (e.g., a closed door that the robot cannot pass through). This is done by maximising a notion of progression defined over the DFA $\mathcal{A}_\varphi$ obtained from the co-safe LTL specification φ.

Broadly speaking, we tackle the following problem (we provide a more exact description of the problem in Subsection 5.2).

Problem 4. *Given an MDP $\mathcal{M}$, a cost structure $c : S \times A \to \mathbb{R}_{\geq 0}$, and a co-safe LTL specification φ such that $Pr^{\max}_{\mathcal{M},\overline{s}}(\varphi) > 0$, find a policy that fulfils the following objectives, in decreasing order of priority:*

- o_1 *Maximises the probability of satisfying φ. Problem 5 provides a more exact description of this objective;*
- o_2 *When φ becomes unsatisfiable (i.e., when we reach a state s such that $Pr^{\max}_{\mathcal{M},s}(\varphi) = 0$), gets as close as possible to satisfying φ. Problem 6 provides a more exact description of this objective; and*
- o_3 *Has a minimal expected cost to achieve o_1 and o_2. Problem 7 provides a more exact description of this objective.*

Note that the approach presented in this section is applicable to arbitrary MDPs, non-negative cost structures and co-safe LTL specifications, not only the MDP models of mobile robot tasks $\mathcal{M}^t_{\mathcal{T},\Lambda}$ and the cost structure $c^t_{\mathcal{T}}$ presented in the previous section. It was, however, developed with planning for mobile robots in mind, with the theoretical notion of partial satisfiability being motivated directly from issues we encountered while deploying the techniques introduced in Lacerda et al. (2014) in the real world. The approach presented here was first introduced in Lacerda et al. (2015b,a). Here, we present a modified notion of progression, by introducing a normalisation factor. By doing so, the values for the distance to acceptance function are now more intuitive. Furthermore, we also provide a more thorough formalisation and illustration of the different steps of the approach.

5.1 Task Progression Function

We start by defining what we mean by satisfying a co-safe LTL specification as much as possible. We propose a notion of *task progression*, defined from a distance function $d_\varphi : Q \to \mathbb{R}_{\geq 0}$ that maps each state of $\mathcal{A}_\varphi$ to a value representing how close we are to reaching an accepting state, in terms of how many sets of atomic propositions still need to be satisfied in order to reach an accepting state. In the following, let φ be a co-safe LTL formula, and $\mathcal{A}_\varphi = \langle Q, \overline{q}, \{q_F\}, 2^{AP}, \delta_{\mathcal{A}_\varphi}\rangle$. We write $q \to^* q'$ if there is a path in $\mathcal{A}_\varphi$ that leads it from state q to state q'.

Definition 28. *Let $Suc_q \subseteq Q$ be the set of successors of state q, and $|\delta_{q,q'}| \in \{0, \ldots, 2^{|AP|}\}$ be the number of transitions from q to q'. We define the distance to acceptance function $d_\varphi : Q \to \mathbb{R}_{\geq 0}$ as:*

$$d_\varphi(q) = \begin{cases} 0 & \text{if } q = q_F \\ \min\limits_{q' \in Suc_q} d_\varphi(q') + h(q,q') & \text{if } q \neq q_F \text{ and} \\ & q \to^* q_F \\ |AP||Q| & \text{otherwise} \end{cases} \tag{43}$$

The function $h : Q \times Q \to \mathbb{R}_{\geq 0}$ represents the difficulty of moving from q to q' in the DFA, and is defined as:

$$h(q,q') = \log_2\left(\left\{\frac{2^{|AP|}}{|\delta_{q,q'}|}\right\}\right) \tag{44}$$

This function represents the number of states that need to be visited, starting in q, to reach the accepting state q_F, balanced by the "difficulty" of transitioning between these states. This balancing is done because we assume that the more transitions there are between two DFA states, the easier it should be for the system to evolve to a state that satisfies one of the transition labels. Given that the maximum number of possible transition labels between two states is exponential in the total number of atomic proposition, and the fact that typically the number of transition labels between two states is exponential in the atomic propositions that are required to hold (or required to not hold) for the transition to be valid, we use log_2 to "linearise" the difficulty function. This provides values for the distance function that better align with the intuition of how a distance function should perform. However, it provides the same ordering of states in terms of their distance to the goal, yielding a qualitatively equivalent representation. Note that the maximum transition difficulty is $|AP|$, when $|\delta_{q,q'}| = 1$, and the minimum transition difficulty is 0, when $|\delta_{q,q'}| = 2^{|AP|}$. Hence, if there is no path from q to q_F, we set $d_\varphi(q)$ to the maximum value $|AP||Q|$.

Remark 4. Calculating d_φ. *The values d_φ can be calculated by finding the fixed-point of the following recursive calculation:*

$$d^0_\varphi(q) = \begin{cases} 0 & \text{if } q \in Q_F \\ |AP||Q| & \text{if } q \notin Q_F \end{cases} \tag{45}$$

$$d^{k+1}_\varphi(q) = \min\left\{d^k_\varphi(q), \min_{q' \in Suc_q} d_\varphi(q') + h(q,q')\right\} \tag{46}$$

Finally, we note that all non-accepting states have a distance to satisfaction strictly greater than 0.

Proposition 4. *For all $q \in Q \setminus \{q_F\}$, $d_\varphi(q) > 0$.*

Proof. To have distance to q_F equal to 0, by definition of d_φ, there would have to exist a state $q \in Q \setminus \{q_F\}$ such that $|\delta_{q,q_F}| = 2^{|AP|}$. But then any σ yielding a run of $\mathcal{A}_\varphi$ reaching q would also be a good prefix for φ, because $\delta(q, \alpha) = q_F$ for all $\alpha \in 2^{AP}$. This leads to a contradiction because $\mathcal{A}_\varphi$ accepts *exactly* the set of good prefixes for φ.

Using this distance function, we define the notion of *progression towards the accepting state* as a measure of how much the distance to an accepting state is reduced by moving from q to q'.

Definition 29. Progression Function. *The progression $p_\varphi : Q \times Q \to \mathbb{R}_{\geq 0}$ between two states of $\mathcal{A}_\varphi$ is defined as:*

$$p_\varphi(q, q') = \begin{cases} \max\{0, d_\varphi(q) - d_\varphi(q')\} & \text{if } q' \in Suc_q \text{ and } q' \not\to^* q \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

We require $q' \not\to^* q$ in the first condition to guarantee that there are no cycles in the DFA with non-zero values for p_φ. This is needed in order to guarantee convergence of infinite sums of values of p_φ. Moreover, we only take positive progression values into account.

Proposition 5. *Let $\rho = q_0 q_1 q_2 \ldots \in Q^\omega$ be an infinite sequence of states visited by an infinite run of $\mathcal{A}_\varphi$. The following holds:*

$$\sum_{i=0}^{\infty} p_\varphi(q_i, q_{i+1}) \text{ converges.} \tag{48}$$

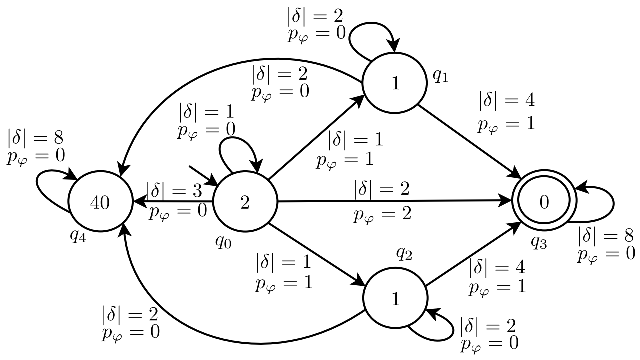
Proof. Note that, by definition, $p_\varphi(q, q') \geq 0$ for all $q, q' \in Q$. Thus, we only need to prove that there exists only a finite number of consecutive states q, q' in ρ such that $p_\varphi(q, q') > 0$. Any pair q, q' of consecutive states occurring infinitely often in ρ need to be such that $q' \to^* q$. Thus, by definition of p_φ, $p_\varphi(q, q') = 0$. In other words, q, q' can only occur consecutively in ρ an infinite number of times if $p_\varphi(q, q') = 0$. Hence the sum converges.

We can check the existence of paths between q' and q by computing the strongly connected components of $\mathcal{A}_\varphi$.

Finally, all state evolutions to an accepting state entail progression strictly greater than 0.

Proposition 6. *$p_\varphi(q, q_F) > 0$ for all $q \in Q \setminus \{q_F\}$ such that $q_F \in Suc_q$.*

Proof. Direct consequence of Proposition 4, and the fact that $\delta_{\mathcal{A}_\varphi}(q_F, \alpha) = q_F$ for all $\alpha \in 2^{AP}$.



$|\delta| = 2$ $p_\varphi = 0$
$|\delta| = 2$ $p_\varphi = 0$
$|\delta| = 1$ $p_\varphi = 0$
$|\delta| = 8$ $p_\varphi = 0$
$|\delta| = 3$ $p_\varphi = 0$
$|\delta| = 1$ $p_\varphi = 1$
$|\delta| = 4$ $p_\varphi = 1$
$|\delta| = 2$ $p_\varphi = 2$
$|\delta| = 8$ $p_\varphi = 0$
$|\delta| = 1$ $p_\varphi = 1$
$|\delta| = 4$ $p_\varphi = 1$
$|\delta| = 2$ $p_\varphi = 0$
$|\delta| = 2$ $p_\varphi = 0$
40
2
1
1
0
q_0
q_1
q_2
q_3
q_4

Figure 5. The DFA from Figure 3, with states labelled with the distance function d_φ, and transition labelled with number of transitions $|\delta|$, and progression p_φ.

Example 10. *In Figure 5, we show the progression function calculation for the DFA depicted in Figure 3. We depict the value $d_\varphi(q_i)$ inside each state, and we label each edge with the number of transitions corresponding to that edge $|\delta|$, and the value of p_φ for the states connected by that edge. Note that $d_\varphi(q_4) = 40$ because there is no path from q_4 to the accepting state.*

Remark 5. Usage of minimal DFA. *It is known that any DFA can be* minimised *into a DFA with a minimum number of states that accepts the same language as the original DFA (Hopcroft et al. 2006). Different DFA will generate slightly different distance to acceptance functions. However, since the DFA we use are representing a specific co-safe LTL formula, distance to acceptance maps to a measure of how close the formula is from being satisfied regardless of the DFA being minimal or not. Thus, our approach makes no assumption on the DFA structure. We note that, in spite of the above, the most sensible approach is to use the minimal form, as it reduces the size of the product MDP that has to be solved.*

Remark 6. Other notions of "distance to acceptance". *Other works (Bhatia et al. 2010, 2011; Ding et al. 2014a) use a notion of "distance to acceptance" similar to ours. However, in those works the notion of distance to acceptance is not used for partial satisfaction. Instead, in Ding et al. (2014a) the distance function is used as an "energy function" to be minimised by a receding horizon control procedure. Given this fact, the distance is defined over the product model, instead of over the DFA model. This does not allow for the encoding of the notion of doing as much as possible, even when the probability of reaching the accepting state in the product becomes zero, because for such states the distance to acceptance in the product is infinity. In order to address such cases, our approach is based on a notion of distance over the DFA, which allows* *us to reason about all the possible ways to satisfy the specification, regardless of the limitations imposed by the environment model. In Bhatia et al. (2010, 2011) there is a notion of distance to acceptance in the DFA, but it is mixed with other quantities obtained from a lower level planner in order to guide a sampling procedure. In this case, the authors to not use the number of labels in each transition to normalise the distance to acceptance and simply find the shortest path. This means that for the DFA in Fig. 5 states q_0, q_1 and q_2 will have the same distance to acceptance, which does not match the notion of doing as much as possible we want to encode in the distance function. We will further explain the decision to define the distance function over the DFA further when addressing Objective o_2 next.*

5.2 Individual Problem Formulation and Solution

We are now in a position to formalise each objective stated in Problem 4 in isolation. We will discuss how to prioritise them in the next section; here we focus on building a common model where the three objectives can be optimised using standard techniques, such as value iteration. In the following, let $\mathcal{M}$ be an MDP, c a cost structure over $\mathcal{M}$, and φ a co-safe LTL formula.

5.2.1 Maximise Probability of Success. Objective o_1 is formalised in the following way.

Problem 5. *Calculate $Pr^{\max}_{\mathcal{M},\overline{s}}(\varphi)$, and find the corresponding policy:*

$$\pi_1 = \arg\max_\pi Pr^{\pi}_{\mathcal{M},\overline{s}}(\varphi) \tag{49}$$

This is the standard problem of maximising the probability of satisfying a co-safe LTL specification (Problem 3), and can be solved by value iteration on the product MDP $\mathcal{M}_\varphi$ (see Definition 17), as stated in Proposition 3.

5.2.2 Maximise Progression. Before we formally define Objective o_2, we introduce the notion of accumulated progression of a run of an MDP.

Definition 30. Accumulated Progression. *Let $\sigma = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \in IPath_{\mathcal{M},\overline{s}}$, and $q^\sigma_i = \delta^+_{\mathcal{A}_\varphi}(\overline{q}, Lab(s_0)Lab(s_1)\ldots Lab(s_i))$. We define the accumulated progression of σ as:*

$$prog(\sigma) = \sum_{i=0}^{\infty} p_\varphi(q^\sigma_i, q^\sigma_{i+1}) \tag{50}$$

Objective o_2, is formalised in the following way.

Problem 6. *Calculate $E^{\max}_{\mathcal{M},\overline{s}}(prog)$, and find the corresponding policy:*

$$\pi_2 = \arg\max_\pi E^{\pi}_{\mathcal{M},\overline{s}}(prog) \tag{51}$$

Problem 6 can also be solved using value iteration on the product MDP, by defining a progression reward on the product by lifting the progression function from the DFA to the product MDP.

Definition 31. Progression Reward. *The progression reward over the product MDP $\mathcal{M}_\varphi$ is a function $r_\varphi : (S \times Q) \times A \to \mathbb{R}_{\geq 0}$ where:*

$$r_\varphi((s,q),a) = \sum_{(s',q') \in S\times Q} \delta_{\mathcal{M}_\varphi}((s,q),a,(s',q'))p_\varphi(q,q') \tag{52}$$

The progression reward is simply the expected progression (taking into account the transition function of the MDP) of executing action a in state $(s,q) \in S_\varphi = S \times Q$. Note that the expected cumulative progression reward is greater than zero for all states (s,q) where we can reduce the distance to an accepting state (in the DFA), even if the probability of reaching an accepting state in the product MDP for (s,q) is already 0. Such states are easy to identify at the DFA level, but impossible to identify at the product MDP level as some of the DFA structure can be lost due to the restrictions imposed by the original MDP model. This is the main reason why we define the distance to acceptance function over the DFA, without taking into account the restrictions imposed by the original MDP model.

Proposition 7. *The value of $E^{\max}_{\mathcal{M},\overline{s}}(prog)$ converges, and the following equality holds:*

$$E^{\max}_{\mathcal{M},\overline{s}}(prog) = E^{\max}_{\mathcal{M}_\varphi,\overline{s_\varphi}}(cumul_{r_\varphi}) \tag{53}$$

Proof. The fact that $E^{\max}_{\mathcal{M},\overline{s}}(prog)$ converges is a direct consequence of Proposition 5 and the fact that the expected value of a distribution over finite values is finite. The equality holds because $\mathcal{M}_\varphi$ preserves the probabilities of paths from $\mathcal{M}$, and $r_\varphi((s,q),a)$ represents the expected value for $p_\varphi(q,q')$ when executing action a in state $(s,q) \in S_\varphi = S \times Q$, evolving the MDP state to s' according to the probabilistic transition function of $\mathcal{M}$, and the DFA state to q', according to the the transition function of $\mathcal{A}_\varphi$ and the value of $Lab(s')$.

Thus, Problem 6 can be reduced to an expected cumulative reward maximisation problem (Problem 2) on $\mathcal{M}_\varphi$, which can be solved by standard techniques such as value iteration.

Remark 7. Maximise Probability vs. Maximise Progression. *Given that one gathers progression as the DFA gets closer to its accepting state, there is some relation between maximising the probability of satisfying the specification, and maximising cumulative progression. However, note that the two objectives can be conflicting. For example,* *maximising progression might lead to policies that get close (according to the distance function) to an accepting state with high probability but have probability zero of reaching it, instead of policies that reach an accepting state with some probability. We will deal with this issue by prioritising probability of success over progression. However, the approach presented in this paper can be easily changed so that progression is prioritised over probability of success, i.e., the designer can choose the priority order between o_1 and o_2.*

5.2.3 Minimise Expected Cost. Before we formally define Objective o_3, we introduce the notion of final progression point of an infinite run, and accumulated cost until a final progression point is reached. In the following, let $\sigma = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \in IPath_{\mathcal{M},\bar{s}}$.

Definition 32. Final Progression Point. *We define the final progression point $k^{\sigma}_{p_\varphi}$ as:*

$$k^{\sigma}_{p_\varphi} = \min\{k \in \mathbb{N} \mid E^{\max}_{\mathcal{M},s_k}(prog) = 0\} \tag{54}$$

Note that, according to Proposition 5, the value of $prog(\sigma)$ always converges. Thus, we are guaranteed that the final progression point always exists.

Definition 33. Cumulative Cost until Final Progression Point. *We define the cumulative cost until no more progression p_φ can be accumulated as:*

$$cumul_c^{k^{\sigma}_{p_\varphi}}(\sigma) = \sum_{i=0}^{k^{\sigma}_{p_\varphi}} c(s_i, a_i) \tag{55}$$

Finally, Objective o_3 can be formalised in the following way.

Problem 7. *Calculate $E^{\min}_{\mathcal{M},\bar{s}}(cumul_c^{k^{\sigma}_{p_\varphi}})$, and find the corresponding policy:*

$$\pi_3 = \arg\min_\pi E^{\pi}_{\mathcal{M},\bar{s}}(cumul_c^{k^{\sigma}_{p_\varphi}})$$

By lifting the cost structure to the product, we can pose the problem above on the product MDP.

Proposition 8. *Let $c_\varphi : (S \times Q) \times A \to \mathbb{R}$ such that:*

$$c_\varphi((s,q),a) = c(s,a) \tag{56}$$

The following equality holds:

$$E^{\min}_{\mathcal{M},\bar{s}}(cumul_c^{k^{\sigma}_{p_\varphi}}) = E^{\min}_{\mathcal{M}_\varphi,\overline{s_\varphi}}(cumul_{c_\varphi}^{k^{\sigma}_{p_\varphi}}) \tag{57}$$

Proof. The result follows given that $\mathcal{M}_\varphi$ preserves the probabilities of paths from $\mathcal{M}$ and the cost function c_φ is directly mirroring c.

Note that solely minimising expected cost until no more progression can be accumulated can lead to bad policies. For example, when the cost of reaching a state from where φ is not satisfiable is lower than the cost of reaching an accepting state, the solution for Problem 7 will be a policy that tries to make φ not satisfiable as soon as possible (as the shortest path to a final progression point might be through making φ unsatisfiable). Our approach of giving least priority to this objective ensures this behaviour does not occur.

Furthermore, in contrast with the previous problems, Problem 7 cannot be solved directly on the product MDP $\mathcal{M}_\varphi$ because the value $k^{\sigma}_{p_\varphi}$ is not constant, being dependent on the path σ. In the next subsection, we tackle this issue by introducing a pruning operation over $\mathcal{M}_\varphi$. This operation preserves the values for Objectives o_1 and o_2, and provides a way of stopping the accumulation of cost when no more progression can be attained while still using a standard solution method such as value iteration.

5.3 Pruned Product MDP

We provide an approach to *prune* the product MDP based on the progression reward, removing states from $\mathcal{M}_\varphi$ for which it is not possible to accumulate more r_φ, and removing all the transitions (along with the corresponding costs) from all such states.

Definition 34. Pruned Product MDP. *Let $\mathcal{M}_\varphi = \mathcal{M} \otimes \mathcal{A}_\varphi = \langle S_\varphi, \overline{s_\varphi}, A, \delta_{\mathcal{M}_\varphi}, AP, Lab_\varphi\rangle$ be a product MDP. We define the pruned product MDP as $\mathcal{M}^{prune}_\varphi = \langle S^{prune}_\varphi, \overline{s_\varphi}, A, \delta_{\mathcal{M}^{prune}_\varphi}, AP, Lab_\varphi\rangle$, where:*

- *$S^{prune}_\varphi = S_{prog} \cup Suc_{S_{prog}}$, where $S_{prog} \subseteq S_\varphi$ is the set of product states from which more progression can be accumulated, and $Suc_{S_{prog}}$ is the set of their successors:*

$$S_{prog} = \{(s,q) \in S_\varphi \mid E^{\max}_{\mathcal{M},s}(prog) > 0\} \tag{58}$$

$$\begin{aligned} Suc_{S_{prog}} = \{&(s,q) \in S_\varphi \setminus S_{prog} \mid \\ &\exists s'_\varphi \in S_{prog}, a \in A \text{ s. t. } \delta_{\mathcal{M}_\varphi}(s'_\varphi, a, s_\varphi) > 0\}\} \end{aligned} \tag{59}$$

- *The transition function is defined as:*

$$\delta_{\mathcal{M}^{prune}_\varphi}((s,q),a,(s',q')) = \begin{cases} \delta_{\mathcal{M}_\varphi}((s,q),a,(s',q')) & \text{if } (s,q) \in S_{prog} \\ 0 & \text{otherwise} \end{cases} \tag{60}$$

We also add a zero-cost self-loop, labelled by a dummy action $\top$, to every state in $Suc_{S_{prog}}$, so that the pruned product MDP can generate infinite sequences.

Remark 8. Constructing $\mathcal{M}^{prune}_\varphi$. *The main challenge when constructing $\mathcal{M}^{prune}_\varphi$ is to find S_{prog}. That can be* *done by finding the fixed-point of the following recursive calculation:*

$$S^0_{prog} = \{s_\varphi \in S \times Q \mid \exists a \in A \text{ s. t. } r_\varphi(s_\varphi, a) > 0\} \quad (61)$$

$$S^{k+1}_{prog} = S^k_{prog} \cup \{s_\varphi \in S \times Q \mid \exists s'_\varphi \in S^k_{prog}, a \in A \text{ s.t. } \delta_{\mathcal{M}_\varphi}(s_\varphi, a, s'_\varphi) > 0\} \quad (62)$$

The calculation above starts with the states S^0_{prog} for which there is an immediate progression reward and then incrementally adds states for which there is some probability of reaching a state in S^0_{prog} in k steps, according to the transition function $\delta_{\mathcal{M}_\varphi}$.

We now present the relevant relation between the product MDP, the final progression point, and the pruned product MDP.

Proposition 9. *All paths in $IPath_{\mathcal{M}_\varphi, \overline{s_\varphi}}$ are of the form:*

$$\sigma = \overline{s_\varphi} \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots \xrightarrow{a_{k^\sigma_{p_\varphi}-1}} s_{k^\sigma_{p_\varphi}} \xrightarrow{a_{k^\sigma_{p_\varphi}}} \dots \quad (63)$$

Furthermore, state $s_{k^\sigma_{p_\varphi}} \in S_\varphi$ is such that the following properties hold:

1. $E^{\max}_{\mathcal{M}_\varphi, s_{k^\sigma_{p_\varphi}}}(cumul_{r_\varphi}) = 0$;
2. $s_{k^\sigma_{p_\varphi}} \in Suc_{S_{prog}}$;
3. $p_{\min} = Pr^{\min}_{\mathcal{M}_\varphi, s_{k^\sigma_{p_\varphi}}}(reach_{acc_\varphi}) = 1$ *or* $p_{\max} = Pr^{\max}_{\mathcal{M}_\varphi, s_{k^\sigma_{p_\varphi}}}(reach_{acc_\varphi}) = 0$.

Proof. Equation 63 holds because the final progression point $k^\sigma_{p_\varphi}$ is always a finite value. This follows directly from Proposition 5.

Property 1 is a direct consequence of Proposition 7 and the definition of final progression point for σ.

Property 2 is a direct consequence of the definition of $Suc_{S_{prog}}$: it is the set of states s such that $E^{\max}_{\mathcal{M},s}(prog) = 0$ and s is an immediate successor of a state s' such that $E^{\max}_{\mathcal{M},s}(prog) > 0$.

For Property 3, we start by showing that $p_{\max} \in \{0, 1\}$ Assume $0 < p_{\max} < 1$. Then there is some probability of reaching an accepting state, i.e, a state of the form (s, q_F). From Proposition 6 and the definition of r_φ (Definition 31), this means that $E^{\max}_{\mathcal{M}_\varphi, s_{k^\sigma_{p_\varphi}}}(cumul_{r_\varphi}) > 0$, which is a contradiction with Property 1.

Now, following the same argument, one can see that if $p_{\max} = 1$, then $s_{k^\sigma_{p_\varphi}}$ is an accepting state, i.e., it is of the form (s, q_F): if this is not the case, $E^{\max}_{\mathcal{M}_\varphi, s_{k^\sigma_{p_\varphi}}}(cumul_{r_\varphi}) > 0$ and there is a contradiction. Thus, $p_{\min} = 1$ because $s_{k^\sigma_{p_\varphi}} \in acc_\varphi$.

This proposition means that the pruned product MDP $\mathcal{M}^{prune}_\varphi$ preserves the probabilities of finite paths relevant to the satisfaction of φ (o_1) and the maximisation of r_φ (o_2). This is because $Suc_{S_{prog}}$ is the set of terminal states of the $\mathcal{M}^{prune}_\varphi$, and the pruned states are not required for the calculation of o_1 and o_2, because we do not need to take into account the evolution of $\mathcal{M}_\varphi$ after the final progression point of each sequence in $IPath_{\mathcal{M}_\varphi, \overline{s_\varphi}}$ has been reached. Furthermore, given that states in $Suc_{S_{prog}}$ only have zero-cost self-loops, we can reduce the indefinite horizon cost minimisation of Objective o_3 to an infinite horizon cost minimisation that is guaranteed to converge.

Proposition 10. *The following equalities hold:*

$$Pr^{\max}_{\mathcal{M},\overline{s}}(\varphi) = Pr^{\max}_{\mathcal{M}^{prune}_\varphi, \overline{s_\varphi}}(reach_{acc_\varphi}) \quad (64)$$

$$E^{\max}_{\mathcal{M},\overline{s}}(prog) = E^{\max}_{\mathcal{M}^{prune}_\varphi, \overline{s_\varphi}}(cumul_{r_\varphi}) \quad (65)$$

$$E^{\min}_{\mathcal{M},\overline{s}}(cumul_c^{k^\sigma_{p_\varphi}}) = E^{\min}_{\mathcal{M}^{prune}_\varphi, \overline{s_\varphi}}(cumul_{c_\varphi}) \quad (66)$$

Proof. Given Proposition 3, to prove Equality (64) it suffices to prove that:

$$Pr^{\max}_{\mathcal{M}_\varphi, \overline{s_\varphi}}(reach_{acc_\varphi}) = Pr^{\max}_{\mathcal{M}^{prune}_\varphi, \overline{s_\varphi}}(reach_{acc_\varphi}) \quad (67)$$

By construction of $\mathcal{M}^{prune}_\varphi$, its behaviour mirrors the behaviour of $\mathcal{M}_\varphi$ until the final progression point $s_{k^\sigma_{p_\varphi}}$ is reached. Furthermore, by Property 3 of Proposition 9, after reaching $s_{k^\sigma_{p_\varphi}}$, acc_φ will either (i) be reached regardless of the policy, or (ii) never be reached regardless of the policy. Thus, the behaviour of $\mathcal{M}^{prune}_\varphi$ after reaching $s_{k^\sigma_{p_\varphi}}$ is irrelevant for the probability of reaching acc and the result follows.

Given Proposition 7, to prove Equality (65) it suffices to prove that:

$$E^{\max}_{\mathcal{M}_\varphi, \overline{s_\varphi}}(cumul_{r_\varphi}) = E^{\max}_{\mathcal{M}^{prune}_\varphi, \overline{s_\varphi}}(cumul_{r_\varphi}) \quad (68)$$

This can be done with a similar argument as the one used above, but now using Property 1 of Proposition 9 instead of Property 3.

Finally, given Proposition 8, to prove Equality (66) it suffices to prove that:

$$E^{\min}_{\mathcal{M}_\varphi, \overline{s_\varphi}}(cumul_{c_\varphi}^{k^\sigma_{p_\varphi}})) = E^{\min}_{\mathcal{M}^{prune}_\varphi, \overline{s_\varphi}}(cumul_{c_\varphi}) \quad (69)$$

The above follow using a similar argument as the one used above, but now noting that, according to Property 2 of Proposition 9, $s_{k^\sigma_{p_\varphi}} \in Suc_{S_{prog}}$ and by construction of $\mathcal{M}^{prune}_\varphi$ the only action available from states in $Suc_{S_{prog}}$ is a zero-cost dummy action. Thus the indefinite horizon sum associated to $cumul_{c_\varphi}^{k^\sigma_{p_\varphi}}$ in $\mathcal{M}_\varphi$ is equal to the infinite horizon sum associated to $cumul_{c_\varphi}$ in $\mathcal{M}^{prune}_\varphi$.

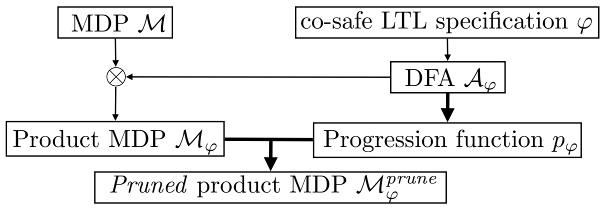

MDP $\mathcal{M}$
co-safe LTL specification φ
DFA $\mathcal{A}_\varphi$
Product MDP $\mathcal{M}_\varphi$
Progression function p_φ
Pruned product MDP $\mathcal{M}_\varphi^{prune}$

Figure 6. Diagram of the approach for the problem reduction, with our contributions in bold.

Thus, Problem 5 can be reduced to solving a reachability problem, and Problems 6 and 7 can be reduced to expected cumulative reward/cost problems, where convergence is guaranteed. These problems can be solved on the same underlying model $\mathcal{M}_\varphi^{prune}$ using standard techniques, such as value iteration. The construction of the product MDP is depicted in Figure 6.

5.4 Nested Value Iteration

In order to optimise o_1, o_2 and o_3 in decreasing order of priority, we introduce a "nested" version of value iteration in Algorithm 1.

Algorithm 1 keeps track of a value table per objective, and uses the lower level priority objectives to break ties, i.e., using them to decide which action to execute only when the value for a pair of actions is the same for the higher-priority objective(s). Given that all objectives are guaranteed to converge in $\mathcal{M}_\varphi^{prune}$, as proved in Proposition 10, the procedure is guaranteed to converge to the optimal policy for Objectives o_1, o_2 and o_3. This approach has similarities to the work in Teichteil-Königsbuch (2012b); Kolobov et al. (2012). However, they just take into account *single state reachability*, while we use co-safe LTL goals, which are more general, and introduce the notion of task progression. It can also be seen as a simplified version of *lexicographic value iteration*, as introduced in Wray et al. (2015), where we do not introduce the notion of slack between objectives, and have a fixed number of objectives. Extending the algorithm to use slack is straightforward, but for our case where the three objectives are closely related, will not yield significantly different results.

6 Verification of Policy Guarantees

While we optimise our policy for probability of satisfaction and expected cost, there are other relevant metrics that we can obtain by performing verification over the obtained policy. In this section, we provide two relevant properties that can be analysed, one specific to the mobile robot model presented in Section 4, and a general one that can be applied to any MDP. This is by no means an extensive description of possible guarantees that can be obtained from the policy, it is just an example of some particularly relevant properties.

Algorithm 1 NESTED VALUE ITERATION

Input: Pruned product $\mathcal{M}_\varphi^{prune}$, cost c_φ, progression reward r_φ
Output: Optimal policy $\pi^* : S \times Q \to A$

```
1: for all $s_\varphi \in S \times Q$ do
2:     $V_p(s_\varphi) \leftarrow \begin{cases} 1 & \text{if } s_\varphi = (s, q_F), \text{ where } q_F \in Q_F \\ 0 & \text{otherwise} \end{cases}$
3:     $V_r(s_\varphi) \leftarrow 0$
4:     $V_c(s_\varphi) \leftarrow \begin{cases} 0 & \text{if } s_\varphi \text{ is a terminal state} \\ \infty & \text{otherwise} \end{cases}$
5: end for
6: while $V_p$ or $V_r$ or $V_c$ have not converged do
7:     for all $s_\varphi \in S \times Q$ which are not terminal do
8:         for all $a \in A(s_\varphi)$ do
9:             $v_p \leftarrow \sum_{s'_\varphi \in S \times Q} \delta_{\mathcal{M}_\varphi^{r_\varphi}}(s_\varphi, a, s'_\varphi) V_p(s'_\varphi)$
10:            $v_r \leftarrow r_\varphi(s_\varphi, a) + \sum_{s'_\varphi \in S \times Q} \delta_{\mathcal{M}_\varphi^{r_\varphi}}(s_\varphi, a, s'_\varphi) V_r(s'_\varphi)$
11:            $v_c \leftarrow c_\varphi(s_\varphi, a) + \sum_{s'_\varphi \in S \times Q} \delta_{\mathcal{M}_\varphi^{r_\varphi}}(s_\varphi, a, s'_\varphi) V_c(s'_\varphi)$
12:            if $v_p > V_p(s_\varphi)$ then
13:                $V_p(s_\varphi) \leftarrow v_p$
14:                $V_r(s_\varphi) \leftarrow v_r$
15:                $V_c(s_\varphi) \leftarrow v_c$
16:                $\pi(s_\varphi) \leftarrow a$
17:            else if $v_p = V_p(s_\varphi) \wedge v_r > V_r(s_\varphi)$ then
18:                $V_r(s_\varphi) \leftarrow v_r$
19:                $V_c(s_\varphi) \leftarrow v_c$
20:                $\pi(s_\varphi) \leftarrow a$
21:            else if $v_p = V_p(s_\varphi) \wedge v_r = V_r(s_\varphi) \wedge v_c < V_c(s_\varphi)$ then
22:                $V_c(s_\varphi) \leftarrow v_c$
23:                $\pi(s_\varphi) \leftarrow a$
24:            end if
25:        end for
26:    end for
27: end while
28: return $\pi^* \leftarrow \pi$
```

Central to the verification of properties of the obtained policies is the notion of *discrete-time Markov chain (DTMC)* induced by a policy.

Definition 35. Induced Discrete-Time Markov Chain. *Let $\mathcal{M} = \langle S, \overline{s}, A, \delta_\mathcal{M}, AP, Lab\rangle$ be an MDP, and $\pi : S \to A$ a memoryless policy over $\mathcal{M}$. The DTMC induced by π is a tuple $C = \langle S, \overline{s}, \delta_C, AP, Lab\rangle$, where the probabilistic transition function $\delta_C : S \times S \to [0,1]$ is such that:*

$$\delta_C(s, s') = \delta_\mathcal{M}(s, \pi(s), s') \tag{70}$$

Note that, even though the potential state space of C is the same as the state space of $\mathcal{M}$, the fact that the actions available at each state are fixed as defined by the policy means that the set of *reachable states* of C is typically much smaller than the set of reachable states of $\mathcal{M}$. This means that in general policy verification (i.e., model checking of C) can be done much quicker than policy generation (i.e., model checking of $\mathcal{M}$).

In our case, we will use DTMC $C_\varphi = \langle S_\varphi^{prune}, \overline{s_\varphi}, \delta_{C_\varphi}, AP, Lab_\varphi\rangle$, which is the DTMC induced by π^* (calculated using Algorithm 1) for $\mathcal{M}_\varphi^{prune}$. We remind the reader that while π^* is a finite-memory policy for $\mathcal{M}$, it is a memoryless policy for $\mathcal{M}_\varphi^{prune}$.

6.1 Calculation of Conditional Expectations

We start with a property of interest for policies generated using our methodology, for arbitrary models (in contrast with policies generated for the mobile robot model presented in Section 4). Our approach allows us to generate a policy π^* where the optimised expected cumulative cost is the expectation to reach a state from which it is not possible to gather more cumulative progression reward. This value can have a very large variance because the expected costs to reach each of these states can be very different. For example, consider the environment on Figure 1, and assume that the task is to visit location v_1 and visit location v_6. The expected time for the execution of π^* when the doors are closed is very different to the expected time for execution of π^* when the doors are open. Furthermore, it is relevant to ask whether it is possible to achieve the whole co-safe LTL specification, what is the expected time to do so. Thus, one can be interested in "splitting" the expected cumulative cost value, into expected cost *to success*, and expected cost *to failure*. These conditional expectations are more informative and can be used for execution monitoring, or for a higher level task scheduler. More concretely, we are interested in the following problem.

Problem 8. *Find the expected cost of π^* until success and until failure, i.e., find:*

$$E^{\pi^*}_{\mathcal{M}_\varphi^{prune},\overline{s_\varphi}}(cumul_{c_\varphi} \mid acc_\varphi) = E_{C_\varphi,\overline{s_\varphi}}(cumul_{c_\varphi} \mid acc_\varphi) \tag{71}$$

$$E^{\pi^*}_{\mathcal{M}_\varphi^{prune},\overline{s_\varphi}}(cumul_{c_\varphi} \mid \neg acc_\varphi) = E_{C_\varphi,\overline{s_\varphi}}(cumul_{c_\varphi} \mid \neg acc_\varphi) \tag{72}$$

In order to calculate the conditional expectation $E_{C_\varphi,\overline{s_\varphi}}(cumul_{c_\varphi} \mid acc_\varphi)$ efficiently, one can prune C_φ in order to keep only the paths that lead to satisfaction of φ.

Definition 36. *Let $C_\varphi = \langle S, \overline{s}, \delta_C, AP, Lab\rangle$ be the induced DTMC. The DTMC representing the* successful *runs of C_φ is defined as $C_\varphi^\vDash = \langle S^\vDash, \overline{s}, \delta_{C_\varphi^\vDash}, AP, Lab\rangle$, where:*

- $S^\vDash = S_\varphi^{prune} \setminus \{s \in S_\varphi^{prune} \mid Pr_{C_\varphi,s}(reach_{acc_\varphi} = 0)\}$
- *The probabilistic transition function $\delta_{C_\varphi^\vDash} : S^\vDash \times S^\vDash \to [0,1]$ is such that:*

$$\delta_{C_\varphi^\vDash}(s,s') = \delta_{C_\varphi}(s,s')\frac{Pr_{C_\varphi,s'}(reach_{acc_\varphi})}{Pr_{C_\varphi,s}(reach_{acc_\varphi})} \tag{73}$$

So, to build $C_\vDash$ we simply remove all states for which the probability of satisfying the specification is zero, and normalise the transition function accordingly.

Proposition 11. Baier et al. (2014). *The following equality holds:*

$$E_{C_\varphi,\overline{s_\varphi}}(cumul_{c_\varphi} \mid acc_\varphi) = E_{C_\varphi^\vDash,\overline{s_\varphi}}(cumul_{c_\varphi}) \tag{74}$$

Thus, we reduced the problem of calculating a conditional expectation to an expected cumulative cost calculation (i.e., an instance of Problem 2 for DTMCs). This can be calculated efficiently using standard techniques such as value iteration. Furthermore, during the calculation of π^*, the value $Pr_{C_\varphi,\overline{s_\varphi}}(reach_{acc_\varphi}) = Pr_{\mathcal{M}_\varphi^{prune},\overline{s_\varphi}}(reach_{acc_\varphi})$ is calculated for all states of C_φ. Hence those values do not be to be recalculated in order to build $C_\varphi^\vDash$. We refer the interested reader to Baier et al. (2014), where a a proof of Proposition 11 is presented, along with a more general discussion on the calculation of conditional expectations for Markovian models.

Finally, we can directly calculate the expected cost to failure $E_{C_\varphi,\overline{s_\varphi}}(cumul_{c_\varphi} \mid \neg reach_{acc_\varphi})$, since we know $E_{C_\varphi,\overline{s_\varphi}}(cumul_{c_\varphi})$, $E_{C_\varphi,\overline{s_\varphi}}(cumul_{c_\varphi} \mid reach_{acc_\varphi})$ and $Pr_{C_\varphi,\overline{s_\varphi}}(reach_{acc_\varphi})$

6.2 Probabilities of Robot Location after Task Completion

For the model of a mobile service robot presented in Section 4, one can also be interested in the distributions over possible final locations of the robot, as this information can be used for future planning.

Problem 9. *Let $term_v$ be the set of terminal states where location is $v \in V$, i.e.:*

$$term_v = \{s \in S_\varphi^{prune} \mid s \in Suc_{S_{prog}} \text{ and } s(loc) = v\} \tag{75}$$

Calculate the probability of reaching one such state under π^*, i.e.:

$$Pr^{\pi}_{\mathcal{M}_\varphi^{prune},\overline{s_\varphi}}(reach_{term_v}) = Pr_{C_\varphi,\overline{s_\varphi}}(reach_{term_v}) \tag{76}$$

The problem above is an instance of Problem 1 over C_φ. Note that we can calculate the probability above for different locations, or even sets of locations. Furthermore, after calculating $Pr_{C_\varphi,\overline{s_\varphi}}(reach_{term_v})$, we can calculate the expected cost (remember that for the mobile robot model this represents expected time) given that the robot finishes in a certain location by a straightforward adaptation of the conditional expectations calculation presented in the previous subsection – one just needs to replace $Pr_{C_\varphi,\overline{s_\varphi}}(reach_{acc_\varphi})$ with $Pr_{C_\varphi,\overline{s_\varphi}}(reach_{term_v})$ when pruning the DTMC.

7 Implementation and Evaluation

In this section, we empirically evaluate our approach. We start by evaluating the scalability of the approach, comparing it with a version of weighted skipping, an alternative approach to deal with partially satisfiable co-safe LTL specifications. Then, we report on an implementation which has been used for long-term deployments of mobile service robots in real office environments Hawes et al. (2017). To illustrate the use of our approach, we provide details on our implementation of a fetch-and-carry task.

7.1 Evaluation of Policy Generation

In this subsection, we evaluate the approach presented in Section 5 in terms of scalability, and compare it to an alternative approach based on an adaptation of the weighted skipping approach (Lahijanian et al. 2015) to MDPs.

The weighted skipping approach is based on defining an extra cost structure which encodes the penalty associated with ignoring the occurrence (or non-occurrence) of a certain atomic proposition. In broad terms, this defines a notion of preference over the atomic propositions. Like our approach, weighted skipping can deal with specifications that cannot be satisfied with probability one in the model and allows for the generation of policies that “do as much as possible”. Furthermore, it also allows for reasoning about ignoring a part of the specification with lower associated skipping cost in order to achieve a part of the specification with higher skipping cost, something our approach does not allow. For example, with weighted skipping, one can choose to visit a region that should be avoided in order to reach a goal region that could not be reached otherwise. However, this extra flexibility requires the construction of an extended product MDP $\mathcal{M}_{\varphi}^{ws}$, which encodes all possible skipping options at each state. This yields a large increase in the size of the model that needs to be solved, and substantially affects the scalability of the method, as we will show below. In our adaptation of the approach, we use a version of nested value iteration to first minimise the expected skipping cost, and then minimise the expected cost of the cost structure given as input to the problem.

We implemented both our approach and the adaptation of the weighted skipping approach in the widely used PRISM model checker Kwiatkowska et al. (2011), which already has support for solving MDPs against properties in LTL, in particular allowing for the maximisation of the probability of satisfying an LTL formula, i.e., a solution to Problem 3. We will refer to this approach as *MaxProb*, and will also compare our approach to it, in order to report on the overhead of maximising progression towards the goal and minimising expected cost, in conjunction with the probability maximisation.

We evaluate the approaches on the environment depicted in Figure 1. We assume that the robot starts in v_3 and the probability of each door being open is 0.9, i.e., the *check_door* guard fulfilling action is defined as in Equation 36. Furthermore, we assume that all navigation actions are successful with probability 1, except from v_3 to v_4, which will successfully make the robot reach v_4 with probability 0.8, but might finish in v_0, with probability 0.2. A cost structure, representing the expected time taken to move between each pair of nodes is defined using a coarse distance metric and assuming constant speed. We depict this as weights on the graph in Figure 1. Finally, we assume that the *check_door* actions take 0.01 seconds.

Assume that v_0 should always be avoided (e.g., to prevent the robot blocking a fire exit), and that our task is to visit nodes v_1, v_6 and v_{18} (i.e., visit three rooms). We represent the atomic propositions $loc = v$, with $v \in V$ as at_v This specification can be written in co-safe LTL as:

$$\varphi = \bigwedge_{v \in \{v_1, v_6, v_{18}\}} ((\neg at_{v_0}) \, \mathtt{U} \, at_v) \tag{77}$$

The policy obtained for φ using our approach tries to visit v_1, v_6 and v_{18} in the order which minimises expected cost (i.e., expected time). However, since there is a chance the robot will end up in v_0 when navigating directly between v_3 and v_4, the policy takes a “detour” through v_{10}. This is because our approach prioritises robustness – in the form of maximisation of the probability of success – over optimality of expected time. If a closed door is encountered during execution, the policy simply tries to go to the next unvisited closest node, a behaviour that comes from maximising the progression reward. Based on the values in Figure 1 the probability of the policy satisfying the specification is 0.729, with an expected time of 17.27 seconds. The values for the expected time of success and failure are 18.27 and 14.58, respectively. The lower time for failure is explained by the fact that the more of the specification the robot fails to satisfy (due to closed doors), the less it has to do.

The policy obtained using weighted skipping is the same as the one described above, and the one using MaxProb differs when the robot finds a closed door. Given that this makes the overall specification unsatisfiable, the policy is not defined for further states and the robot stops executing the task.

In Figures 7 and 8, we present the model sizes (defined as the number of states plus the number of transitions) and time for solving the model, for the MaxProb approach, for our approach, and for the weighted skipping approach. On the left hand side of the figures, we show the size of the original (flat) model or of the specification DFA (depending on whether we are analysing scalability in terms of model size or specification size), and the sizes of the product MDP (used to maximise probability of success), of the pruned product MDP (used by our approach), and of the weighted skipping product MDP. On the right hand side of the

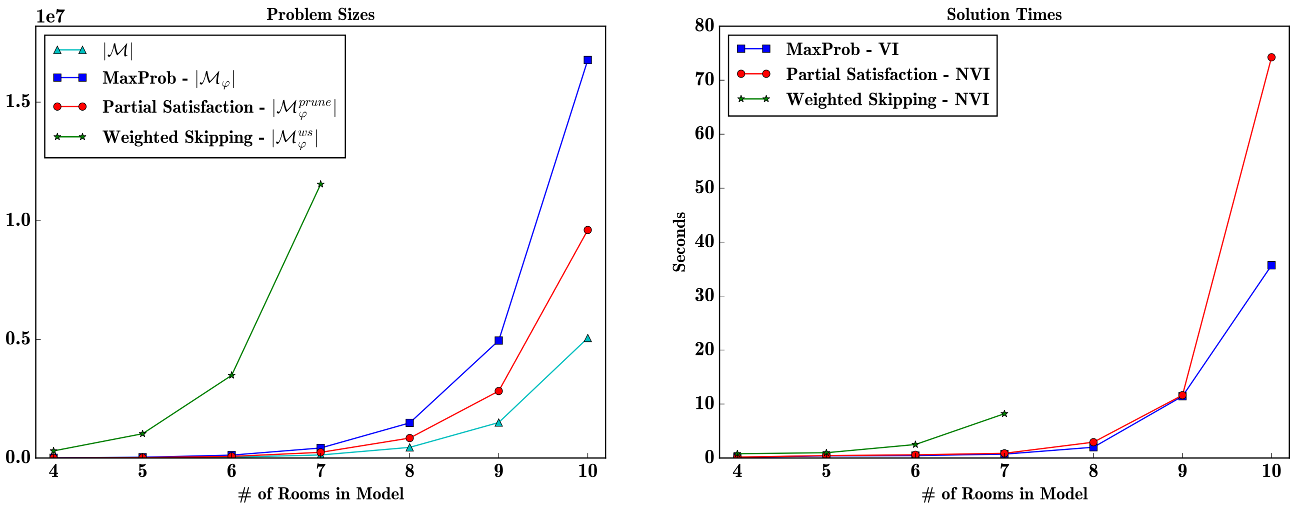


Figure 7. Model sizes and solution times for a specification where the robot must visit 3 rooms, while varying the total number of rooms modelled.

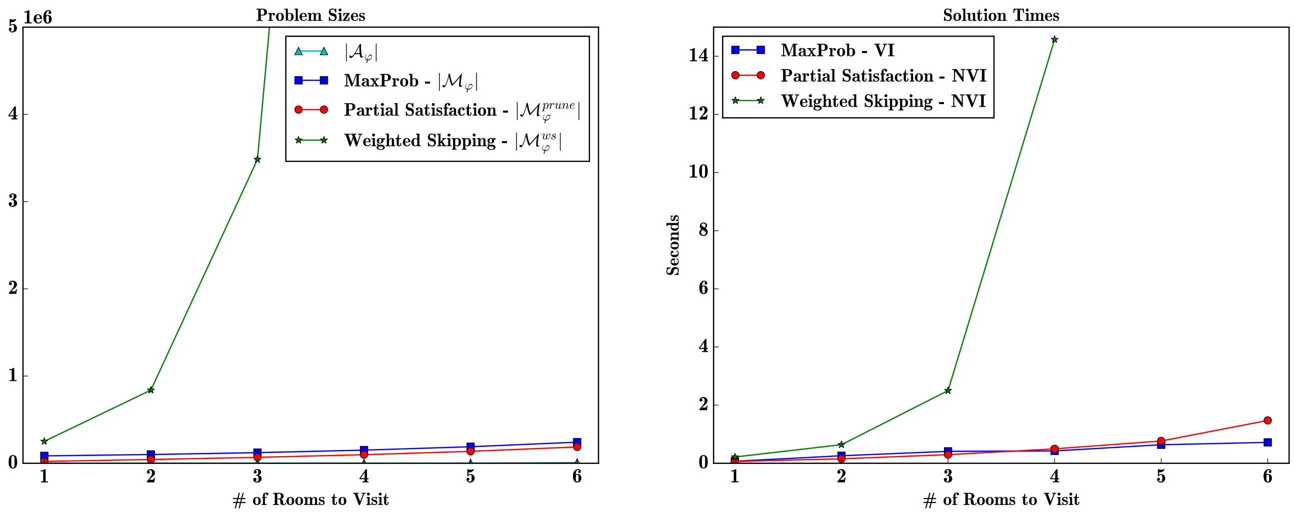


Figure 8. Model sizes and solution times for a model with 6 rooms, while varying the total number of rooms to be visited.

figures, we give the time taken to solve the models (on an Intel® Core™i7 at 2.70GHz x 8 processor, 16GB of RAM), using "vanilla" value iteration on $\mathcal{M}_\varphi$ for probability maximisation, nested value iteration on $\mathcal{M}_\varphi^{prune}$ for our approach, and nested value iteration (adapted to optimise two objectives, as explained above) on $\mathcal{M}_\varphi^{ws}$ for weighted skipping. Note that we are only comparing times to solve the models, in order to provide a fair comparison where optimisations of the procedures to build the product models are not taken into account. In particular, our implementation of the construction of $\mathcal{M}_\varphi^{ws}$ is particularly slow when compared with the pruning operation required to build $\mathcal{M}_\varphi^{prune}$. By only comparing times to solve the model, we are comparing exactly the same solution algorithm (nested value iteration). We also note that the time for the pruning operation is less than for solving the model afterwards, even though it can take some seconds for the larger models we analyse.

In Figure 7, we report on scalability in terms of model size, by varying the number of rooms in the model. The specification is to visit the two leftmost rooms and the rightmost room in the model, while avoiding v_0 (for example, Equation 77 is the specification used for 6 rooms). One can see that the weighted skipping approach scales poorly compared to the other approaches, with our implementation running out of memory when trying to build $\mathcal{M}_{\varphi}^{ws}$ for 8 rooms. On the other hand, the sizes of $\mathcal{M}_{\varphi}$ and $\mathcal{M}_{\varphi}^{prune}$ are of the same order of magnitude as the size of the original model $\mathcal{M}$. Furthermore, the pruning operation does not reduce the state space substantially, as progression towards the goal is achievable in most states of the model. In terms of solution times, one can see that the overhead of optimising three objectives using nested value iteration only stops being negligible for the larger model of 10 rooms. In fact, the main contributor for the increase of solution times is the size of the original model $\mathcal{M}$, which grows exponentially with the number of rooms, as each room added requires adding an extra state feature representing the state of the corresponding door.

In Figure 8, we report on scalability in terms of specification size, by varying the number of rooms to visit, for a model with 6 rooms. One can see that the size of $\mathcal{A}_{\varphi}$ is negligible (it varies from 15 to 8385) when compared to the size of the product structures, with it not even being visible in the plot. This is a common trend in this type of problems, where typically the model size is much larger than the specification size. In this case, weighted skipping scales worse because, as we add more atomic propositions to the specification, more possible combination of possible "skippings" need to be added to $\mathcal{M}_{\varphi}^{ws}$. For 4 rooms, the size of $\mathcal{M}_{\varphi}^{ws}$ is 1.70×10^7, and our procedure to build $\mathcal{M}_{\varphi}^{ws}$ ran out of memory for 5 rooms. The other approaches scale much better as the number of rooms to visit increases, with the overhead of nested iteration only being noticeable for the largest specification.

We finish by noting that the specification we tested is computationally challenging, featuring the combinatorial explosion of choosing the best order to visit a set of locations. Furthermore, when compared to a simpler approach of only maximising the probability of task satisfaction, our approach does not incur a substantial overhead. However, it is able to generate policies that not only maximise the probability of task satisfaction, but are also defined for states where the probability of satisfying the full task is zero but there still is some possible progression towards the goal, and minimise the expected cumulative cost while doing so.

7.2 Robot Implementation

In this subsection, we report on the use of our approach in a real mobile service robot, deployed for long periods of time in everyday environments, in the context of the STRANDS project[‡].

In Figure 9, we provide a high level depiction of the architecture for the implementation in the real robot. The implementation runs on ROS and connects the policies generated by the PRISM implementation of our approach with an executable policy on the robot platform. The framework assumes the implementation of spatio-temporal edge and action outcome and duration models, a set of primitive skills implemented as ROS actionlib actions, and a set of continuous navigation actions implemented in ROS.

The designer defines the topological map (according to Definition 18), overlaying it on the 2D metric map of the environment, using an extension of the RVIZ tool. They also represent the action definitions (according to Definition 23). The action definitions also include a matching between possible outcomes of the execution of primitive skills on the robot with each possible successor state in the MDP. Finally, the specification is defined as a co-safe LTL task and an execution time t.

The *MDP constructor* then takes as input the topological map, the action definitions, and the spatio-temporal model prediction at time t to build a navigation and general actions MDP $\mathcal{M}_{\mathcal{T},\Lambda}^{t}$ (according to Definition 26). The *PRISM language translator* then takes the representation of $\mathcal{M}_{\mathcal{T},\Lambda}^{t}$ and the co-safe language specification and generates an input problem in the PRISM modelling language. This problem is solved using our PRISM implementation of the approach presented in Section 5, obtaining a PRISM representation of the optimal policy π^*. The *policy parser* then uses this policy and the representation of $\mathcal{M}_{\mathcal{T},\Lambda}^{t}$ to generate an executable representation of the policy in ROS. This representation is then executed by the *policy executor*. It appropriately calls the navigation actions and primitive skills available, according to π^*, and matches the outcomes from the execution of these behaviours on the robot with the corresponding MDP successor state. This matching is done using the action definitions, and is required in order to evolve the current state of the system and find the next action associated to π^*, which is then triggered by the executor.

The implementation is open source and available online[§]. It has been deployed on the robot depicted in Figure 1 (however, note that the approach is platform independent), which has been used in several human-populated environments in the context of the STRANDS project, for periods of up to four months. Such environments include public spaces at the project's research partners premises; Haus der Barmherzigkeit, an elder-care facility in Vienna, Austria[¶]; office spaces of G4S Technology in Tewkesbury, United Kingdom[‖]; and the Transport Systems Catapult (TSC) Innovation Centre, in Milton Keynes, United Kingdom[**]. We refer the interested

[‡] http://strands-project.eu
[§] https://github.com/strands-project/strands_executive/tree/kinetic-devel/mdp_plan_exec
[¶] https://www.hausderbarmherzigkeit.at/en
[‖] http://www.g4s.com/
[**] https://ts.catapult.org.uk/

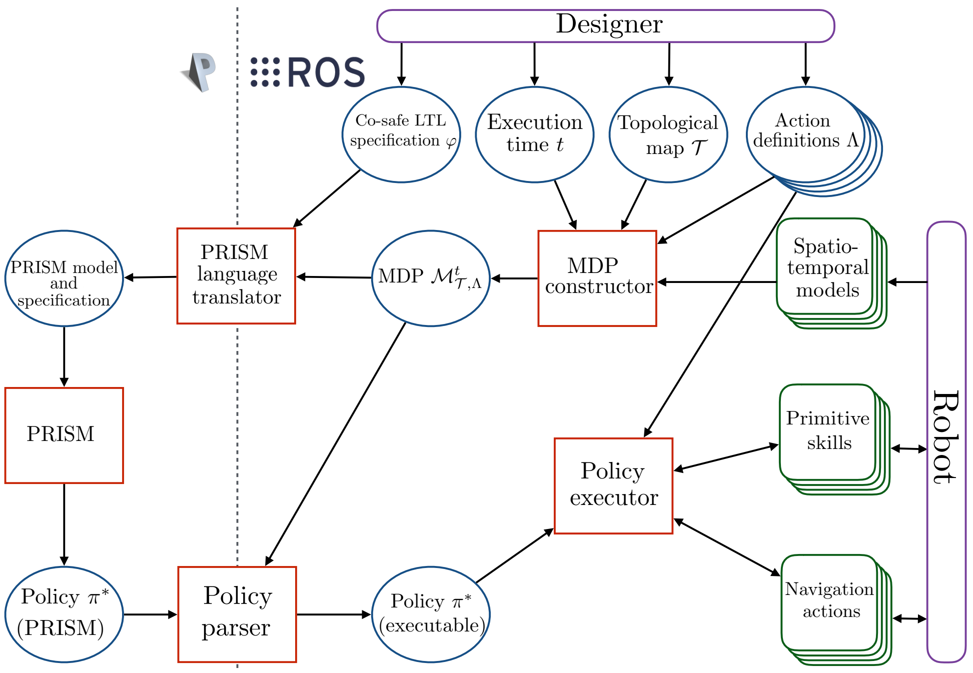

Designer
ROS
Co-safe LTL specification φ
Execution time t
Topological map $\mathcal{T}$
Action definitions Λ
PRISM model and specification
PRISM language translator
MDP $\mathcal{M}^t_{\mathcal{T},\Lambda}$
MDP constructor
Spatio-temporal models
Robot
PRISM
Policy executor
Primitive skills
Policy π^* (PRISM)
Policy parser
Policy π^* (executable)
Navigation actions

Figure 9. High level view of the architecture for the real robot implementation. Blue circles represent the different structures used on the implementation, red squares represented the methods implemented for policy execution and in order to connect between the ROS code and the PRISM software, and green rounded squares represent external pieces of code that can be plugged in to the framework.

reader to Hawes et al. (2017) for details on these deployments.

Here, we report on a particularly interesting task on the largest environment in which the robot was deployed: a delivery task at the TSC Innovation Centre. The overall area of the environment was more than $2000m^2$, and the topological map had 94 nodes. There were also 4 doors where the robot needed to wait for someone to hold the door before it could go through. This was modelled using guarded navigation actions for each door.

The robot is fitted with a basket in which users at the site can put and retrieve objects. The task was for the robot to navigate to the sender and wait for the object to be placed in its basket. Then, navigate to the location of the receiver, and wait for the object to be retrieved. If the receiver does not retrieve the object, the robot needs to navigate back to the sender to give the object back. After this, the robot needs to navigate to one of a pre-defined set of locations $final$. This is needed to avoid the awkward behaviour where the robot just stands next to the human after the interaction is finished.

Using a *speak-and-wait* primitive skill, where the robot speaks and then waits for a button press from the user or for a timeout, we created action $retrieve$ for retrieving the object, action $deliver$ for delivering the object, and action $return$ for returning the object to the sender. These actions correspond to calls to the *speak-and-wait* primitive skill with different speeches to be performed (e.g., for the action of retrieving the object, the speech argument for the primitive skill is *"Hi, please place the object to be delivered in my basket and press the button on my screen when you are done."*). We also defined three state features $retrieved$, $delivered$ and $returned$, all with domain $\{-1, 0, 1\}$. For example, $retrieved = -1$ represents that the robot has not tried to retrieve the object for delivery yet, $retrieved = 0$ means that the *speak and wait* primitive skill timed-out when trying to retrieve the object (i.e., the sender was not there to send the object), and $retrieved = 1$ means that the sender has pressed the button on the robot screen (and we assume that they have placed the object in the basket). The values for $delivered$ and $returned$ have similar meanings. We also only allow each action to be executed only once, i.e., they have the corresponding state feature equal to -1 as one of their preconditions. This is because the robot has other tasks to execute, and cannot wait indefinitely for a user to place or retrieve an object.

The specification is to reach a state where the $retrieve$ action has timed-out (i.e., the sender did not place the object in the basket), or the $deliver$ action was successful (i.e., the receiver retrieved the object from the basket), or the $return$ action was successful (i.e., the robot successfully returned the object back to the sender, because the receiver did not retrieve it). After that, the robot must navigate to one of the locations in *final*. This can be written as:

$$\begin{aligned}\varphi = \mathtt{F}(((&retrieved = 0) \vee (delivered = 1)\vee \\ &(returned = 1))\wedge \\ &(\mathtt{F} \textstyle\bigvee_{v\in final}(loc = v)))\end{aligned} \qquad (78)$$

The model for this problem has 10,976 states and 31,624 transitions. The product MDP has 16,672 states and 48,040 transitions, and the full pipeline described in Figure 9, i.e., building the MDP representation, solving it in PRISM against φ using the approach presented in Section 5, and parsing the policy back into an executable structure in ROS is done under 7 seconds. This means the approach allows for online integration on the real robot, as the procedure is done in a relatively short amount of time, even for a large environment and an involved task specification. A video showing our mobile robot performing this task is available online at `https://youtu.be/MFS-bOdCW6Q`.

8 Conclusions

In this paper, we presented an automated framework for deployment of policies for mobile service robots, based on probabilistic model checking techniques. The modelling approach is based on three main ingredients, (i) using Markov decision processes to represent the uncertainty associated with the robot's navigation and other action executions; (ii) using co-safe linear temporal logic, a flexible, unambiguous, and well-tested specification language, to encode goals; and (iii) allowing for the plugging of accurate spatio-temporal environmental models, learned from experience, for filling the probabilistic outcomes of the MDP, and defining a cost function representing expected time for execution. We then present a general method for the generation of cost-optimal policies for co-safe LTL specifications over MDP models, which can be applied to specifications that are not satisfiable with probability one. This policy generation approach also encodes a notion of satisfying as much of the specification as possible, by introducing a function that represents a notion of progression towards the goal and also maximising the expected value of this function.

By applying our policy generation approach to the MDP models of mobile robot tasks we propose, we are able to provide meaningful formal performance guarantees on the robot's behaviour, such as probability of task satisfaction and expected time for task completion. Finally, our framework has been implemented on a real mobile service robot, and used for its long term deployment in real office environments, thus proving its suitability for real world use. It can also be easily ported to other platforms, as it is implemented in the ROS middleware as a set of robot-independent packages.

There are several avenues for further work. Of particular relevance are the use of probabilistic planning techniques such as heuristic search or sampling-based approaches to improve the scalability of our methodology while still being able to provide formal performance guarantees; the introduction of more fine-grained timing guarantees such as time-bounded guarantees; and the extension of these techniques to multi-robot systems, allowing for the deployment of robot teams with probabilistic guarantees both at global (team) and local (individual robot) levels.

Funding

The research leading to these results has received funding from the EU Seventh Framework Programme (FP7/2007-2013) under grant agreement No 600623, STRANDS; UK Research and Innovation and EPSRC through the Robotics and Artificial Intelligence for Nuclear (RAIN) research hub [EP/R026084/1]; and the PRINCESS project, funded by the DARPA BRASS programme.

References

Baier C and Katoen JP (2008) *Principles of Model Checking*. MIT Press.

Baier C, Klein J, Klüppelholz S and Märcker S (2014) Computing conditional probabilities in Markovian models efficiently. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer.

Belta C, Bicchi A, Egerstedt M, Frazzoli E, Klavins E and Pappas G (2007) Symbolic planning and control of robot motion [grand challenges of robotics]. *IEEE Robotics & Automation Magazine* 14(1): 61–70.

Bhatia A, Kavraki LE and Vardi MY (2010) Sampling-based motion planning with temporal goals. In: *Proc. of the 2010 IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Bhatia A, Maly MR, Kavraki LE and Vardi MY (2011) Motion planning with complex goals. *IEEE Robotics & Automation Magazine (RAM)* 18(3).

Boutilier C, Dean T and Hanks S (1999) Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11(1).

Cashmore M, Fox M, Long D, Magazzeni D, Ridder B, Carrera A, Palomeras N, Hurtós N and Carreras M (2015) Rosplan: Planning in the robot operating system. In: *Proc. of the 25th Int. Conf. on Planning and Scheduling (ICAPS)*.

Castro L, Chaudhari P, Tumova J, Karaman S, Frazzoli E and Rus D (2013) Incremental sampling-based algorithm for minimum-violation motion planning. In: *Proc. of 52nd IEEE Conf. on Decision and Control (CDC)*.

Cizelj I and Belta C (2014) Control of noisy differential-drive vehicles from time-bounded temporal logic specifications. *Int. Journal of Robotics Research* 33(8).

de Alfaro L (1997) *Formal Verification of Probabilistic Systems*. PhD Thesis, Stanford University.

Ding X, Lazar M and Belta C (2014a) LTL receding horizon control for finite deterministic systems. *Automatica* 50(2).

Ding X, Pinto A and Surana A (2013) Strategic planning under uncertainties via constrained Markov decision processes. In: *Proc. of the 2013 IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Ding X, Smith S, Belta C and Rus D (2014b) Optimal control of Markov decision processes with linear temporal logic constraints. *IEEE Trans. on Automatic Control* 59(5).

Donzé A and Maler O (2010) Robust satisfaction of temporal logic over real-valued signals. In: *Proc. of the Int. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS)*.

Fainekos GE and Pappas GJ (2009) Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* 410(42).

Forejt V, Kwiatkowska M, Norman G and Parker D (2011) Automated verification techniques for probabilistic systems. In: *Formal Methods for Eternal Networked Software Systems (SFM)*, *LNCS*, volume 6659. Springer.

Frank J and Jónsson A (2003) Constraint-based attribute and interval planning. *Constraints* 8(4).

Hanheide M, Göbelbecker M, Horn GS, Pronobis A, Sjöö K, Aydemir A, Jensfelt P, Gretton C, Dearden R, Janicek M et al. (2015) Robot task planning and explanation in open and uncertain worlds. *Artificial Intelligence* .

Hawes N, Burbridge C, Jovan F, Kunze L, Lacerda B, Mudrová L, Young J, Wyatt JL, Hebesberger D, Körtner T, Ambrus R, Bore N, Folkesson J, Jensfelt P, Beyer L, Hermans A, Leibe B, Aldoma A, Faulhammer T, Zillich M, Vincze M, Al-Omari M, Chinellato E, Duckworth P, Gatsoulis Y, Hogg DC, Cohn AG, Dondrup C, Fentanes JP, Krajník T, Santos JM, Duckett T and Hanheide M (2017) The STRANDS project: Long-term autonomy in everyday environments. *IEEE Robotics and Automation Magazine* 24(3): 146–156.

Hopcroft JE, Motwani R and Ullman JD (2006) *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co. Inc.

Ingrand F and Ghallab M (2014) Deliberation for autonomous robots: A survey. *Artificial Intelligence* .

Kemeny J, Snell J and Knapp A (1976) *Denumerable Markov chains*. 2nd edition. Springer-Verlag.

Kolobov A, Mausam and Weld D (2012) A theory of goal-oriented MDPs with dead ends. In: *Proc. of 28th Conf. on Uncertainty in Artificial Intelligence (UAI)*.

Kress-Gazit H, Fainekos GE and Pappas GJ (2009) Temporal logic-based reactive mission and motion planning. *IEEE Trans. on Robotics* 25(6): 1370–1381.

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

Kwiatkowska M, Norman G and Parker D (2011) PRISM 4.0: Verification of probabilistic real-time systems. In: *Computer Aided Verification (CAV)*, *LNCS*, volume 6806. Springer.

Lacerda B, Parker D and Hawes N (2014) Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications. In: *Proc. of 2014 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*.

Lacerda B, Parker D and Hawes N (2015a) Nested value iteration for partially satisfiable co-safe LTL specifications. In: *Proc. of AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents (SDMIA)*.

Lacerda B, Parker D and Hawes N (2015b) Optimal policy generation for partially satisfiable co-safe LTL specifications. In: *Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

Lahijanian M, Almagor S, Fried D, Kavraki LE and Vardi MY (2015) This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction. In: *Proc. of the 29th AAAI Conf. on Artificial Intelligence (AAAI)*. pp. 3664–3671.

Lahijanian M, Andersson S and Belta C (2012) Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Trans. on Robotics* 28(2).

Lahijanian M and Kwiatkowska M (2016) Specification revision for Markov decision processes with optimal trade-off. In: *Proc. of the 55th Conf. on Decision and Control (CDC 2016)*. IEEE.

Lahijanian M, Maly MR, Fried D, Kavraki LE, Kress-Gazit H and Vardi MY (2016) Iterative temporal planning in uncertain environments with partial satisfaction guarantees. *IEEE Transactions on Robotics* 32(3).

Maly M, Lahijanian M, Kavraki L, Kress-Gazit H and Vardi M (2013) Iterative temporal motion planning for hybrid systems in partially unknown environments. In: *Proc. of 16th Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*.

McGann C, Py F, Rajan K, Thomas H, Henthorn R and McEwen R (2008) A deliberative architecture for auv control. In: *Proc. of the 2008 IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Mudrová L, Lacerda B and Hawes N (2015) An integrated control framework for long-term autonomy in mobile service robots. In: *Proc. of the 7th European Conf. on Mobile Robotics (ECMR)*. Lincoln, United Kingdom.

Pnueli A (1981) The temporal semantics of concurrent programs. *Theoretical Computer Science* 13.

Pulido Fentanes J, Lacerda B, Krajník T, Hawes N and Hanheide M (2015) Now or later? predicting and maximising success of navigation actions from long-term experience. In: *Proc. of the 2015 IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Puterman M (1994) *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

Raman V, Donzé A, Maasoumy M, Murray RM, Sangiovanni-Vincentelli A and Seshia SA (2014) Model predictive control with signal temporal logic specifications. In: *Proc. of the 53rd Conf. on Decision and Control (CDC)*.

Sadigh D and Kapoor A (2016) Safe control under uncertainty with probabilistic signal temporal logic. In: *Proc. of Robotics: Science and Systems (RSS)*.

Sprauel J, Teichteil-Königsbuch F and Kolobov A (2014) Saturated path-constrained MDP: Planning under uncertainty and deterministic model-checking constraints. In: *Proc. of 28th AAAI Conf. on Artificial Intelligence (AAAI)*.

Stock S, Mansouri M, Pecora F and Hertzberg J (2015) Hierarchical hybrid planning in a mobile service robot. In: *Proc. of the 38th Joint German/Austrian Conf. on Artificial Intelligence (KI)*.

Svoreňová M, Černá I and Belta C (2013) Optimal control of MDPs with temporal logic constraints. In: *Proc. of 52nd IEEE Conf. on Decision and Control (CDC)*.

Teichteil-Königsbuch F (2012a) Path-constrained Markov decision processes: bridging the gap between probabilistic model-checking and decision-theoretic planning. In: *Proc. of 2012 European Conf. on Artificial Intelligence (ECAI)*.

Teichteil-Königsbuch F (2012b) Stochastic safest and shortest path problems. In: *Proc. of the 26th AAAI Conf. on Artificial Intelligence (AAAI)*.

Tumova J, Hall G, Karaman S, Frazzoli E and Rus D (2013) Least-violating control strategy synthesis with safety rules. In: *Proc. of 16th Int. Conf. on Hybrid Systems: Computation and Control (HSCC)*.

Ulusoy A, Wongpiromsarn T and Belta C (2012) Incremental control synthesis in probabilistic environments with temporal logic constraints. In: *Proc. of 51st IEEE Conf. on Decision and Control (CDC)*.

Vardi M (1985) Automatic verification of probabilistic concurrent finite state programs. In: *Proc. of 26th IEEE Annual Symp. on Foundations of Comp. Sci. (FOCS)*.

Vasile CI, Tumova J, Karaman S, Belta C and Rus D (2017) Minimum-violation scLTL motion planning for mobility-on-demand. In: *Proc. of the 2017 IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Veloso MM, Biswas J, Coltin B and Rosenthal S (2015) CoBots: Robust symbiotic autonomous mobile service robots. In: *Proc. of the 24th Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

Wolff E, Topcu U and Murray R (2013) Efficient reactive controller synthesis for a fragment of linear temporal logic. In: *Proc. of 2013 IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Wray KH, Zilberstein S and Mouaddib AI (2015) Multi-objective MDPs with conditional lexicographic reward preferences. In: *Proc. of the 29th AAAI Conf. on Artificial Intelligence (AAAI)*. pp. 3418–3424.

Younes HLS and Littman ML (2004) PPDDL 1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical Report CMU-CS-04-162.