Multi-Objective Planning with Multiple High Level Task Specifications

Seyedshams Feyzabadi Stefano Carpin

***Abstract*— We present an algorithm to solve a sequential stochastic decision making problem whereby a robot is subject to multiple objective functions and is asked to complete a number of subgoals specified using a subset of linear temporal logic. Each subgoal is associated with a desired satisfaction probability that will be met in expectation by the policy produced by the algorithm. Our method relies on the theory of constrained Markov Decision Processes and on methods coming from the realm of formal verification. The key idea is the definition of a product operation that can recursively incorporate more and more subgoals into the underlying planner. Ultimately, a policy is computed solving a linear program and we outline conditions for the existence and correctness of the solution. Our findings are validated in various simulation scenarios.**

I. Introduction

Multi-objective planning continues to be one of the problems of major interest in robotics. As robots become more and more capable, they are tasked with complex missions including multiple subgoals, and the robot is expected to autonomously determine a plan achieving multiple goals at once. Recently, we have used Constrained Markov Decision Processes (CMDPs) to solve problems where multiple objective functions are considered [4], [10], [11]. Rather than combining them into a single objective function whose meaning is hard to interpret, with CMDPs one optimizes performance with respect to one objective function while imposing constraints (i.e., bounds on the expected value) on the others. In this way, the problem is setup using parameters that are easy to understand for the end-user, rather than introducing artificial functions that have no practical meaning.

At the same time, there is increasing interest in developing methods to express complex missions in a simple way. To this end, linear temporal logic (LTL) has emerged as a promising tool, also because of its connection to formal verification methods. When defining a complex mission with multiple subgoals, however, it may be impossible to achieve all of them at the same time. Therefore it is of interest being able to formulate problems where different subgoals are associated with a desired probability of success, and let the planner determine a policy meeting these objectives in expectation.

S. Feyzabadi and S. Carpin are with the School of Engineering, University of California, Merced, CA, USA.

This work is supported by the National Institute of Standards and Technology under cooperative agreement 70NANB12H143. Any opinions, findings, and conclusions or recommendations expressed in these materials are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the funding agencies of the U.S. Government.

In this paper we propose a technique to tackle all of these objectives at once, i.e., we consider a method to determine a policy that optimizes one objective function (e.g., travel time) while at the same time obeying to constraints on other objective functions (e.g., consumed fuel, length of traversed path, etc.) and satisfying a set of subgoals with a desired level of probability (e.g., visit a certain area with a certain probability and eventually stop in another region with a different probability). Our method builds upon the theory of Markov Decision Processes, i.e., it assumes that action outcomes are stochastic, but the state is observable. The main contributions of this paper are the following

- We formulate a multi-objective planning problem featuring multiple objective functions and multiple subgoals to be probabilistically satisfied.
- We define a *product* operation between CMDPs and finite automata expressing mission goals allowing to precisely determine the probability that the system will complete any of the assigned subgoals.
- We provide conditions for the existence of an optimal solution and formulate an algorithm to determine it.
- We demonstrate the algorithm in three different planning scenarios.

The remainder of this paper is organized as following. State of the art is discussed in Section II whereas in Section III we provide relevant background about CMDPs and LTLs. The problem we solve is formulated in Section IV and our solution is discussed in Section V. Section VI illustrates how the method performs in three different tasks, and finally conclusions are offered in Section VII.

II. State of the Art

MDPs have been widely used to solve sequential stochastic decision making problems where the state of the system is observable [12]. Most of these approaches optimize over one objective function only, although some consider multiple objectives [5], [6], [10], [11] through the use of CMDPs. LTL is being increasingly used to formulate task specifications for robotics and automation [16] and there have been some efforts to solve MDPs with LTL constraints. Lacerda et al. [13] solve a planning problem with LTL specification in a cost optimal form. Ding et al. [8] focused on long term surveillance problems. Their work maximizes the probability of satisfying task specifications expressed as LTL formulas. The earlier work of Ding [7] uses LTL properties in motion planning. Wongpiromsarn et al. [16] used LTL properties on multi-agent planning and concentrate on maximizing the probability of satisfying just one LTL formula. This work was extended in [15]. Their application is in the area of

autonomous vehicles operating in an environment where humans are present (e.g., autonomous forklifts on the factory floor.). Etessami et al. [9] used MDPs with multiple LTL properties. They use approximation methods to satisfy LTL properties with different probabilities. They limit their work to very simple LTL properties that can be extracted from reachability graphs. Therefore, the approach is not easily extensible to a broader class of LTL formulas.

Considering the mentioned related work, we see that most of these approaches consider just a single objective. For instance, some approaches maximize the probability of satisfying an LTL formula and do not control the total cost of accomplishing the task. Others emphasize the cost of completing the task but then the probability of satisfying the LTL specification can be arbitrarily low. Our goal is to find a policy that satisfies a set of LTL specifications with pre-defined probabilities while the total cost of completing the task is minimized and bounds are met for the extra costs.

III. Background

A. Finite State Automata

We recall some classical definitions, but we nevertheless assume that the reader is familiar with basic terms like word, language accepted by an automaton, and other concepts in automata theory (see [14] for an introduction to the topic).

Definition 1: A nondeterministic finite automata (NFA) is a 5-tuple $\mathcal{N} = (Q_{\mathcal{N}}, q_0, \delta_{\mathcal{N}}, F, \Sigma)$ where:

- $Q_{\mathcal{N}}$ is a finite set of states.
- $q_0 \in Q_{\mathcal{N}}$ is the initial state.
- $\delta_{\mathcal{N}} : Q_{\mathcal{N}} \times \Sigma \to 2^{Q_{\mathcal{N}}}$ is the transition function.
- $F \subseteq Q_{\mathcal{N}}$ is the set of accepting states.
- Σ is a finite set of symbols called alphabet.

Note that according to the definition it can be that $\delta_{\mathcal{N}}(q, a) = \emptyset$ for some couple (q, a). In such case the transition is not defined. The set of words over Σ is partitioned into the set of words accepted by $\mathcal{N}$ (also called the language accepted by $\mathcal{N}$) and the set of words rejected by $\mathcal{N}$. A deterministic finite automaton is a special case of an NFA where the successor state is uniquely determined, as implied by the following definition.

Definition 2: A deterministic finite automaton (DFA) $\mathcal{D} = (Q_{\mathcal{D}}, q_0, \delta_{\mathcal{D}}, F, \Sigma)$ is an NFA in which $|\delta_{\mathcal{D}}(q, a)| \leq 1$ for each $q \in Q_{\mathcal{D}}$ and $a \in \Sigma$.

It is well known that every NFA $\mathcal{N}$ can be converted into an equivalent DFA $\mathcal{D}$, i.e., a DFA accepting all and only the words accepted by the NFA. Hence in the following we will simply consider DFAs. For a DFA, we define $\delta^*_{\mathcal{D}}(q_0, w)$ to be the state reached by recursively applying the transition function $\delta_{\mathcal{D}}$ to all symbols in w. For a DFA it is still allowed that for some state and symbols $\delta_{\mathcal{D}}(q, a) = \emptyset$. If while computing $\delta^*_{\mathcal{D}}(q_0, w)$ the transition function is not defined for one of the symbols in w, then the automaton stops, and the word is rejected. A total DFA is a special type of DFA where there is one and only one transition at any state for every input symbol.

Definition 3: A total deterministic finite automaton (total-DFA) is a DFA in which $|\delta(q, a)| = 1$ for each $q \in Q$ and $a \in \Sigma$.

Every DFA $\mathcal{D} = (Q_{\mathcal{D}}, q_0, \delta_{\mathcal{D}}, F, \Sigma)$, can be converted into a total-DFA $\mathcal{T} = (Q_{\mathcal{T}}, q_0, \delta_{\mathcal{T}}, F, \Sigma)$ accepting the same language. The conversion is obtained modifying the state set and the transition function as follows (other elements do not change)

- $Q_{\mathcal{T}} = Q_{\mathcal{D}} \cup \{q_s\}$, where q_s is a new state not in $Q_{\mathcal{D}}$.
- $\delta_{\mathcal{T}} : Q_{\mathcal{T}} \times \Sigma \to Q_{\mathcal{T}}$ is the transition function defined as follows:

$$\delta_{\mathcal{T}}(q, a) = \begin{cases} \delta_{\mathcal{D}}(q, a) & \text{if } q \in Q_{\mathcal{D}} \wedge \delta_{\mathcal{D}}(q, a) \neq \emptyset \\ q_s & \text{otherwise.} \end{cases}$$

As per the above definition, if $\delta_{\mathcal{D}}(q, a)$ is not defined in $\mathcal{D}$, then the state enters the newly added state q_s (called *sink state*) and remains there, as per the second case in the definition of the transition function $\delta_{\mathcal{T}}$. Starting from a total-DFA, we introduce a new type of automata, the extended-total-DFA, whose purpose will become justified in the sequel.

Definition 4: Let $\mathcal{T} = (Q_{\mathcal{T}}, q_0, \delta_{\mathcal{T}}, F, \Sigma)$ be a total-DFA and let $q_s \in Q$ be its sink state. The extended-total-DFA $\mathcal{E} = (Q_{\mathcal{E}}, q_0, \delta_{\mathcal{E}}, F, \Sigma)$ induced by $\mathcal{T}$ is defined as follows:

- $Q_{\mathcal{E}} = Q_{\mathcal{T}} \cup \{q_a\}$ where q_a is a new state not in $Q_{\mathcal{T}}$.
- $\delta_{\mathcal{E}} : Q_{\mathcal{E}} \times \Sigma \to Q_{\mathcal{E}}$ is the transition function defined as follows:

$$\delta_{\mathcal{E}}(q, a) = \begin{cases} \delta_{\mathcal{T}}(q, a) & \text{if } q \neq q_s \wedge q \neq q_a \\ q_a & \text{otherwise.} \end{cases}$$

The extended-total-DFA extends the total-DFA by introducing an extra state q_a. We say that q_a is the *absorbing state* of the extended-total-DFA. As per the transition definition $\delta_{\mathcal{E}}$, if while processing a word w the state enters q_s, then at the next processed input symbol the state will move to q_a and remain there. Based on these definitions, we can define when a word w is accepted or rejected for these types of automata. Starting from a DFA $\mathcal{D}$ it is then possible to sequentially induce a total-DFA $\mathcal{T}$ and an extended-total-DFA $\mathcal{E}$. It is easy to see that a word w with finite length is accepted by the extended-total-DFA $\mathcal{E}$ if and only if it as accepted by the DFA $\mathcal{D}$. For a word w not accepted by $\mathcal{T}$ we can have that $\delta^*_{\mathcal{E}}(q_0, w)$ is q_s, q_a or any other non final state. Note that indeed the final state can be q_s and not q_a when q_s is entered after processing the last symbol of w. To eliminate this variability in $\delta^*_{\mathcal{E}}(q_0, w)$ when the string is not accepted, we add a new symbol $\mathscr{G}$ to Σ that appears twice at the end of each word processed by the automata, and we alter the transition function to ensure that $\delta^*(q_0, w)$ is either a final state or q_a.

Under the assumption that every word w ends with the character $\mathscr{G}$, we define a new extended-total-DFA $\mathcal{E} = (Q_{\mathcal{E}}, q_0, \delta'_{\mathcal{E}}, F, \Sigma')$ where

- $\Sigma' = \Sigma \cup \{\mathscr{G}\}$
-

$$\delta'_{\mathcal{E}}(q, \mathscr{G}) = \begin{cases} q & \text{if } q \in F \\ q_a & \text{if } q = q_s \vee q = q_a \\ q_s & \text{otherwise.} \end{cases}$$

Hence for each word ending with two $\mathscr{G}$ symbols we have $\delta'^{*}_{\mathcal{E}}(q_0, w)$ is either a state in F or q_a. Note that after appending $\mathscr{G}$ twice to every word w processed by the extended-total-DFA, it is ensured that state q_s is visited at most once. This fact will be important to exactly establish the probability that a LTL property is satisfied.

B. Linear Temporal Logic

LTL has been extensively used to specify desired behaviors of a variety of reactive systems, and even a cursory introduction to the topic is beyond the scope of this paper. The reader is referred to [2] for more details. LTL formulas are used to specify properties of reactive systems, and the terms formula and property are then used interchangeably in the following. We consider a subset of LTL formulas leading to the so called syntactically co-safe LTL (sc-LTL) properties [6]. Starting from a set of atomic propositions Π, a sc-LTL formula is built using the standard operators and ($\wedge$), or ($\vee$), not ($\neg$), and the temporal operators *eventually* ($\Diamond$), *next* ($\bigcirc$), and *until* (U). Furthermore, the operator $\neg$ can only be used in front of atomic propositions. When comparing sc-LTL formulas with full LTL formulas, the reader will note that the set of temporal operators misses the *always* operator. As pointed out in [6], since robots operate over missions with finite duration, the *always* operator is mostly irrelevant when specifying mission objectives and can therefore be omitted. sc-LTL properties are verified in a finite amount of time, as opposed to safety properties that are violated in a finite amount of time. An sc-LTL (or LTL) formula ϕ splits the set of infinite length strings over 2^{Π} into two subsets, i.e., the subset of strings satisfying the property and those not satisfying it. It is well known that given a sc-LTL formula ϕ, there exists a DFA accepting all and only the strings satisfying ϕ [2]. An sc-LTL property is satisfied by all words starting with a set of *good* prefixes. When the DFA processes a word with a good prefix, after having processed such prefix it enters a final state and remains there, thus accepting the word irrespectively of what comes afterwards.

C. Labeled CMDP

In this section we extend the classic definition of CMDPs with a set of atomic propositions to track which properties are verified during a trajectory. The reader is referred to [1] for a comprehensive introduction to CMDPs.

Definition 5: A finite CMDP is defined by the tuple $\mathcal{C} = (S, \beta, A, c_i, \Pr)$ where

- S is a finite set of states.
- β is a mass distribution over S giving the initial distribution over S, i.e., $\beta(s_j)$ is the probability that the initial state of the CMDP is s_j.
- $A = \cup_{s \in S} A(s)$ is a finite set of actions, where $A(s)$ is the set of actions executable in state s. Based on these definitions let $K = \{(s, a) \in S \times A \mid s \in S \wedge a \in A(s)\}$.
- c_i: $K \to \mathbb{R}_{\geq 0}$, $i = 0, \ldots, n$ are $n+1$ cost functions. When action a is executed in state s, each of the costs $c_i(s, a)$ is incurred.
- $\Pr$: $K \times S \to [0, 1]$ is the transition probability function where $\Pr(s_1, a, s_2)$ is the probability of reaching state s_2 from s_1 after applying action a.

A labeled constrained Markov decision process (LCMDP) is obtained from a CMDP by adding a set of atomic propositions and a labeling function.

Definition 6: A LCMDP is defined by the tuple $\mathcal{L} = (S, \beta, A, c_i, \Pr, AP, L)$ where the first five parameters are as in the CMDP definition and:

- AP is a finite set of binary atomic propositions.
- L: $S \to 2^{AP}$ is a labeling function assigning to each state the set of atomic propositions that are true in the state.

MDPs and CMPDs are special cases of the more general LCMDP just defined. A *policy* π defines which action should be taken in every state. In the following we are only concerned with so-called Markovian policies, i.e., policies in which the decision is only a function of the current state and does not depend on the previous history. It is known [1] that when considering CMDPs one in general needs randomized policies to optimally solve the problem. Randomized policies have the form $\pi : S \to \mathbb{P}(A)$ where $\mathbb{P}(A)$ is the set of mass distributions over A. For a given state s, $\pi(s) \in \mathbb{P}(A(s))$, i.e., the action is chosen from the set $A(s)$ according to the mass distribution $\mathbb{P}(A(s))$. Given a start state $s_0 \in S$ and a policy π, a stochastic process $\omega = s_0, a_0, s_1, a_1, \ldots$ is defined, with $a_i \in A(s_i)$. Such sequence is also called *trajectory*, and records all states and actions taken. In an LCMDP every trajectory induces a sequence of atomic propositions $\mathcal{L}(\omega) = L(s_0)L(s_1)L(s_2)\ldots$ as per the labeling function L. Given a policy π and an initial mass distribution β, the probability of every trajectory can be determined, and it is a function of both π and β. Moreover, we indicate with S_t the random variable for the state at time t and let A_t be the random variable for the action at time t. In the sequel, we rely on the concept of *absorbing* LCMDP that is formalized in the following.

Definition 7: An LCMDP is absorbing if its state set S can be partitioned into two subsets S' (transient states) and M (absorbing states) so that for each policy π:

1) for each $s \in S'$, $\sum_{t=0}^{+\infty} \Pr_{\beta}^{\pi}[S_t = s] < +\infty$ where $\Pr_{\beta}^{\pi}$ is the probability distribution induced by β and π.
2) for each $s \in S'$, $s_m \in M$ and $a \in A(s_m)$ we have $\Pr(s_m, a, s) = 0$.
3) $c_i(s, a) = 0$ for each $s \in M$ and each $0 \leq i \leq n$.

Informally stated, an LCMDP is absorbing if for every policy the state will eventually reach the set of absorbing states M (first condition), from which it cannot escape (second condition), and where no more costs are accrued (third condition). Without loss of generality, in the following we assume that the absorbing set M consists of a single state s_a and that $A(s_a) = \{a_a\}$. Note that the third condition implies $\Pr(s_a, a_a, s_a) = 1$. From now onwards, unless otherwise specified, we just consider absorbing LCMDPs.

For an LCMDP we can define $n+1$ total cost functions

of a policy π and initial distribution β

$$c_i(\pi, \beta) = \mathbb{E}\left[\sum_{t=0}^{+\infty} c_i(s_t, a_t)\right]$$

where the expectation is taken with respect to the probability distribution over trajectories induced by π and β. Note that this expectation exists because of the assumption that the LCMDP is absorbing, i.e., it will enter for sure the absorbing state from which no more costs will be added. This cost function is different from the finite horizon cost function or discounted infinite horizon cost function often considered in the theory of MDPs. However, we maintain that in autonomous robotic applications total cost is the most appropriate cost model because tasks have a finite duration, but the duration is not necessarily known.

The connection between LCMDP and sc-LTL formulas is as follows. A policy π over an LCMDP generates a trajectory $\omega = s_0, a_0, s_1, s_2, a_2, \ldots$, and this is associated with the infinite string $\mathcal{L}(\omega) = L(s_0)L(s_1)L(s_2)\ldots$. We say that ω satisfies an sc-LTL formula ϕ if and only if $\mathcal{L}(\omega)$ satisfies ϕ. Since sc-LTL properties consider words with finite length only, they accept or reject infinite words with their finite prefix. This means that after a certain time, $t > 0$, the total-DFA associated with ϕ will accept or reject the string $\mathcal{L}(\omega)$.

IV. Problem formulation

Building upon the definitions we introduced in the previous section, we can now formulate the problem we tackle in this paper.

Multi-objective, multi-property MDP – Given:

- an LCMDP $\mathcal{M} = (S, \beta, A, c_i, \Pr, AP, L)$ with $n+1$ costs functions $c_0, c_1, \ldots, c_n$;
- m sc-LTL formulas $\phi_1, \ldots, \phi_m$ over AP;
- n cost bounds $B_1, \ldots, B_n$;
- m probability bounds $P_{\phi_1}, \ldots, P_{\phi_m}$;

determine a policy π for $\mathcal{M}$ that:

- minimizes in expectation the cost $c_0(\pi, \beta)$;
- for each cost c_i, $(1 \leq i \leq n)$, $c_i(\pi, \beta) \leq B_i$;
- for every trajectory ω, each of the m formulas ϕ_i is satisfied with at least probability P_{ϕ_i}.

V. Proposed solution

Given an sc-LTL formula ϕ_i there exists a DFA $\mathcal{D}_i$ accepting all and only the words satisfying ϕ_i. Given the equivalence between DFAs and extended-total-DFAs, there is then an extended-total-DFA $\mathcal{E}_i$ accepting the language satisfying ϕ_i. Starting from the LCMDP $\mathcal{M}$ and the m extended-total-DFAs associated with the m formulas ϕ_i, our solution builds upon two steps. First, we define a product operation between an LCMDP and a DFA that returns a new LCMDP, and we then sequentially compute the product between $\mathcal{M}$ and the every $\mathcal{E}_i$. Since the product generates a new LCMDP with a larger state set, at every step we introduce a pruning procedure removing states that cannot be reached while following any policy. In the second step we solve the final LCMDP using a linear program formulation.

A. Product definition

Our definition of product is given in definition 8. Products between transition systems or CMDPs were already considered in literature [2], [6], but our approach is different. In particular, the product between an LCDMP and an extended-total-DFA gives a new LCMDP, and this will allow us to define an iterated product between an LCMDP and multiple extended-total-DFAs. Consistently with the fact that the result is an LCMDP, the product does not include accepting states, because these are not part of the definition of an LCMDP.

Definition 8: Let $\mathcal{M} = (S, \beta, A, c_i, \Pr, AP, L)$ be an LCMDP with absorbing state s_a, and let $\mathcal{E} = (Q, q_0, \delta, F, \Sigma)$ be an extended total-DFA with $\Sigma = 2^{AP}$. The product between $\mathcal{M}$ and $\mathcal{E}$ is an absorbing LCDMP $\mathcal{M} \otimes \mathcal{E} = (S_p, \beta_p, A_p, c_{p_i}, \Pr_p, AP, L_p)$ where:

- $S_p = S \times Q$.
- $\beta_p(s, q) = \begin{cases} \beta(s) & \text{if } q = q_0 \\ 0 & \text{otherwise} \end{cases}$
- $A_p = A$.
- $c_{p_i}((s, q), a) = c_i(s, a)$ for $0 \leq i \leq n$.
- $$\Pr_p((s, q), a, (s', q')) = \begin{cases} \Pr(s, a, s') & \text{if } q' = \delta(q, L(s)) \\ 0 & \text{otherwise} \end{cases}$$
- $L_p(s, q) = L(s)$.

States of the type (s, q_s) where q_s is the sink state in the extended-total-DFA are called *sink states* for the product. It is easy to verify that $\mathcal{M} \otimes \mathcal{E}$ is indeed absorbing and that its absorbing states are (s_a, q) where $q \in F \cup \{q_a\}$ and let $Q_{ai} = F_i \cup \{q_{ai}\}$ be the set of absorbing states in $\mathcal{E}_i$. When considering the product between an LCMDP $\mathcal{M}$ and multiple extended-total-DFAs $\mathcal{E}_1, \ldots, \mathcal{E}_m$ the product $\mathcal{M} \otimes \mathcal{E}_1 \otimes \ldots \mathcal{E}_m$ is also absorbing. Let its states have the form $(s, q_1, \ldots, q_m)$. Its set of absorbing states is then

$$S_a = \{(s, q_1, \ldots, q_m) \in S_p \mid s = s_a \wedge \forall i \; q_i \in Q_{ai}\},$$

i.e., the first component is the absorbing state of $\mathcal{M}$ and all of the m other state components are in the absorbing states of the extended-total-DFAs. Accordingly, the set $S'_p = S_p \setminus S_a$ is the set of transient states in the product. Figure 1 shows an example of product between an LCMDP and two total-extended-DFAs. Note that there are three absorbing states at the very right and that without loss of generality one could combine them into a single absorbing state. However in the figure we do not combine them to better illustrate the process leading to the product.

B. Reducing Size of State Space

The product as per definition 8 creates LCMDP $\mathcal{M} \otimes \mathcal{E} = (S_p, \beta_p, A_p, c_{p_i}, \Pr_p, AP, L_p)$ where $S_p = S \times Q$. However, in practice many states in S_p will be *unreachable*, i.e., they will not be visited under any policy because of the way the new transition probability function $\Pr_p$ is defined. Therefore, rather than creating S_p as per the definition, we replace it with a set of states $\mathcal{R} \subset S \times Q$ that are reachable, i.e.,

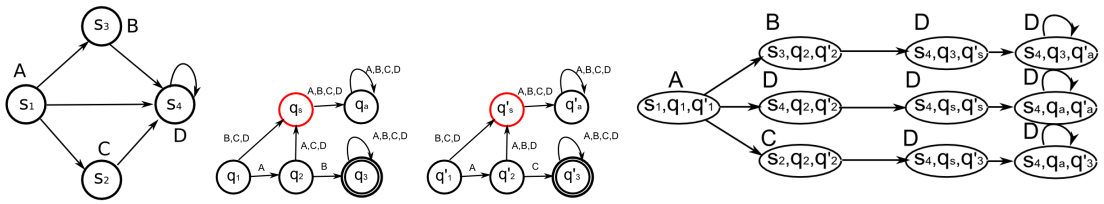

Fig. 1. Left picture shows the original LCMDP with $AP = \{A, B, C, D\}$. Atomic propositions are placed near the vertices in which they are verified, as per the labeling function. The two middle pictures are two extended total DFAs $\mathcal{E}_1$ and $\mathcal{E}_2$. The right picture shows the pruned product model $\mathcal{M} \otimes \mathcal{E}_1 \otimes \mathcal{E}_2$.

states that can be reached. The key observation is that if $\Pr_p((s,q),a,(s',q')) = 0$ for each $(s,q) \in S \times Q$ and $a \in A$, then (s',q') can be omitted from S_p. These states can be easily identified based on the transition function for the extended-total DFA, as per the definition of $\Pr_p$.

Algorithm 1 takes as input an LCMDP $\mathcal{M}$ and an extended total-DFA $\mathcal{E}$, and returns the set $\mathcal{R} \subset S \times Q$ of states reachable from the starting states following the transition function $\Pr_p$. The algorithm calls an internal function Reachable-States-From, shown in Algorithm 2 that returns the set of states that are reachable in one transition from a state on product LCMDP. The algorithm uses a function Post(s) that takes a state in $\mathcal{M}$ and returns the set of states that can be reached from state s in one step.

Data: LCMDP, $\mathcal{M} = (S, \beta, A, c_i, \Pr, AP, L)$,
extended-total-DFA $\mathcal{E} = (Q, q_0, \delta, F, \Sigma)$
Result: Set of reachable states in the product
$(\mathcal{R} \subset S \times Q)$
$\mathcal{R} = \{(s,q) \mid s \in S \wedge q = q_0 \wedge \beta(s) > 0\}$
$\mathcal{R}_{seen} = \emptyset$
while $\mathcal{R} \neq \mathcal{R}_{seen}$ **do**
  select $(s,q) \in \mathcal{R} \setminus \mathcal{R}_{seen}$;
  $Q' \leftarrow$ Reachable-States-From$((s,q), \mathcal{M}, \mathcal{E})$;
  $\mathcal{R} \leftarrow \mathcal{R} \cup Q'$;
  $\mathcal{R}_{seen} \leftarrow \mathcal{R}_{seen} \cup \{(s,q)\}$;
end

Algorithm 1: Selection of reachable states in product LCMDP

Data: $(s,q) \in S \times Q$, $\mathcal{M} = (S, \beta, A, c_i, \Pr, AP, L)$,
$\mathcal{E} = (Q, q_0, \delta, F, \Sigma)$
Result: A set of reachable states from (s,q)
$\Delta \leftarrow L(s)$;
$C \leftarrow \emptyset$;
$S' \leftarrow$Post(s);
for $a \in \Delta$ **do**
  $q' \leftarrow \delta(q,a)$;
  for $s' \in S'$ **do**
    $C = C \cup (s',q')$;
  end
end
return C

Algorithm 2: Reachable-States-From

After computing $\mathcal{R}$, we can then generate the product LCMDP $\mathcal{M} \otimes \mathcal{E} = (S_p, \beta_p, A_p, c_{p_i}, \Pr_p, AP, L_p)$ where $S_p = \mathcal{R}$. Note that, by construction, states excluded from $\mathcal{R}$ are irrelevant in the following computations, because they cannot be reached under any policy.

C. Solving the Optimization Problem

The reason for introducing the product LCMDP is to convert the problem of satisfying an sc-LTL formula into a reachability problem. Similar constructions were used in the past (e.g., [6]), although our construction is different because of the way we defined the product. The relationship between satisfiability and reachability is established by the following theorem whose simple proof is omitted in the interest of space.

Theorem 1: Let $\mathcal{M}$ be an LCMDP and $\mathcal{E}$ be an extended total DFA associated with the sc-LTL formula ϕ. Let $\omega = (s_1,q_1),\ldots,(s_n,q_n),\ldots$ be a trajectory of $\mathcal{M} \otimes \mathcal{E}$ and $\mathcal{L}(\omega)$ be the corresponding string. $\mathcal{L}(\omega)$ satisfies ϕ if and only if ω does not include any state $(s,q) \in S_p$ where $q = q_s$.

According to the theorem, if a trajectory ω of $\mathcal{M} \otimes \mathcal{E}$ reaches a sink state, then the corresponding sc-LTL property ϕ is not satisfied by $\mathcal{L}(\omega)$. Otherwise, it does. Therefore, we are looking for traces starting from an initial state, without passing through any of sink states and ending in the absorbing state. These trajectories satisfy ϕ by construction and as we will show in the following we can compute the probability of satisfying ϕ.

Theorem 2: Let $\mathcal{M}_p = \mathcal{M} \otimes \mathcal{E}$ be a product LCMDP where $\mathcal{E}$ is the extended-total-DFA associated with an sc-LTL property ϕ. If a policy π generates trajectories satisfying ϕ with probability P_ϕ, then with probability $1 - P_\phi$ the policy will generate trajectories passing sink states $(s,q) \in S_p$ where $q = q_s$.

Proof: Because of the definition of transition function for an extended-total-DFA and the definition of product $\mathcal{M} \otimes \mathcal{E}$, every state of $\mathcal{M}_p$ of the type (s,q_a) is entered at most once. Moreover a state of the type (s,q_a) is entered if and only if the property ϕ is not satisfied, so if the policy π satisfies ϕ with probability P_ϕ, the trajectory enters states (s,q_a) with probability $1 - P_\phi$. ■

Trajectories are induced by policies, and the theory of occupation measures presented in the following can be used to compute the probability that a policy reaches a state, as well as to estimate the expected costs (the reader is referred to [1] for a thorough introduction to occupation measures). Given an absorbing LCMDP, let S' be its set of transient

states. We define $K = \{(s,a) \mid s \in S' \wedge a = A(s)\}$ as its *state-action* space. For a policy π we define the occupation measures $\rho(s,a)$ for each element in K as

$$\rho(s,a) = \sum_{t=0}^{+\infty} \Pr_\beta^\pi[S_t = s, A_t = a]$$

where $\Pr_\beta^\pi[S_t = s, A_t = a]$ is the probability induced by the policy π and the initial distribution β. Note that in general the occupation measure is a sum of probabilities but not a probability itself. An occupation measure $\rho(s,a)$, however, is a probability when one ensures that state s is visited only once. If state s is visited only once at time t, then all terms in the definition of $\rho(s,a)$ are 0, except one, and then this is indeed a probability by definition. To ensure that a state is visited just once, one has to either formulate an appropriate policy π or impose specific properties on the transition probabilities of the underlying state space. Given an LCMDP $\mathcal{M}$ and m extended-total-DFAs $\mathcal{E}_1, \ldots, \mathcal{E}_m$ associated with formulas $\phi_1, \ldots, \phi_m$, let

$$\mathcal{M}_p = \mathcal{M} \otimes \mathcal{E}_1 \otimes \mathcal{E}_2 \cdots \otimes \mathcal{E}_m.$$

For each property ϕ_i, we define the set

$$\mathscr{S}_i = \{(s, q_1, \ldots, q_i, \ldots, q_m) \in S_p \mid q_i = q_{s_i}\}$$

i.e., the set of states in $\mathcal{M}_p$ including the sink state for the extended-total-DFA associated with ϕ_i. To ease the notation, from now onwards we write the tuple $((s, q_1, q_2, \ldots, q_n), a)$ as (x,a) with the understanding that x is a state in $\mathcal{M}_p$ and $a \in A(x)$ is an action for x as recursively defined by the product. At this point we have all the elements to formulate the main theorem providing the solution for the problem we defined in Section IV.

Theorem 3: Let $\mathcal{M} = (S, \beta, A, c_i, \Pr, AP, L)$, $\phi_1, \ldots, \phi_m$, $B_1, \ldots, B_n$ and $P_{\phi_1} \ldots P_{\phi_m}$ be as in Section IV. Let $\mathcal{E}_1, \ldots, \mathcal{E}_m$ be m extended-total-DFAs associated with $\phi_1, \ldots, \phi_m$, $\mathcal{M}_p = \mathcal{M} \otimes \mathcal{E}_1 \otimes \mathcal{E}_2 \cdots \otimes \mathcal{E}_m$, and K be the state action space associated with the set S'_p of transient states in $\mathcal{M}_p$. The multi-objective, multi-property MDP problem admits a solution if and only if the following linear program is feasible:

$$\min_{\rho(x,a) \in K} \sum_{x \in S'_p} \sum_{a \in A(x)} c_0(x,a)\rho(x,a) \tag{1}$$

subject to

$$\sum_{x \in S'_p} \sum_{a \in A(x)} c_i(x,a)\rho(x,a) \leq B_i, i = 1, ..., n \tag{2}$$

$$\sum_{a \in A(\mathscr{S}_i)} \rho(\mathscr{S}_i, a) \leq 1 - P_{\phi_i}, i = 1, ..., m \tag{3}$$

$$\sum_{x' \in S'_p} \sum_{a \in A(x')} \rho(x,a)[\delta_x(x') - \Pr_p(x', a, x)] = \\ = \beta(x), \forall x \in S'_p \tag{4}$$

$$\rho(x,a) \geq 0, \forall x \in S'_p \tag{5}$$

where $\delta_x(x')$ equals to one, if $x = x'$ and zero otherwise.

Proof: The proof follows from the theory of CMDPs and our definition of product. First consider the linear program excluding constraints in (3). This is precisely the linear program to solve a CMDP with an occupancy measure approach (see, e.g., [1]). Next, consider the additional m constraints in (3), and let us consider one specific formula ϕ_i. Because of our definition of product, each sink state in $\mathscr{S}_i$ is entered at most once. More precisely, if one state $\mathscr{S}_i$ is entered once, then no other state in $\mathscr{S}_i$ is entered. Therefore $\sum_{a \in A(\mathscr{S}_i)} \rho(\mathscr{S}_i, a)$ is indeed a probability because at most one of its component is different from 0. Combining this fact with Theorem 2, it follows that each of the m constraints in (3) bounds the probability that formula ϕ_i is not satisfied, and this concludes the proof. ■

At this point it should be clear why the sink states q_s and q_a were introduced when defining the extended-total-DFAs. q_s is separated from q_a in order to make sure q_s can be hit at most one time for every policy realization, and our definition of product allows to track multiple formulas in parallel. If the linear program is solvable, its solution provides a randomized, stationary, Markovian policy as follows (see [1]):

$$\pi(x,a) = \frac{\rho(x,a)}{\sum_{a \in A(x)} \rho(x,a)}, x \in S'_p, a \in A(x)$$

where $\pi(x,a)$ is the probability of taking action a when in state x.

VI. Experiments and Results

In this section we show how the proposed system can be used to compute policies based on complex task specifications accounting for multiple objective functions and properties to be specified with different confidence levels. In all cases the robot moves in a grid-like environment, with four-connectivity, i.e., at each location the robot can go up/down/left/right, unless the motion is forbidden by the environment (e.g., when the robot is at the boundary or when there are obstacles.)

A. Outdoor navigation

In the first scenario we consider a robot moving in an outdoor domain without obstacles. Its environment is modeled with the elevation map displayed in Figure 2. A risk map is generated based on the terrain map where risk is a function of altitude. The corresponding discretized risk map is shown in the middle picture of the same figure where warmer colors represent riskier areas. That is to say that while completing its mission the robot has to avoid, if possible, to move through risky areas. At each location the outcome of actions is non-deterministic. To be specific, an action attempting to move the robot towards a location not higher than its current location succeeds with probability 0.9 and fails with probability 0.1. If the action fails the robot remains at its own location or moves to one of its neighbors with lower height (excluding the intended target location).

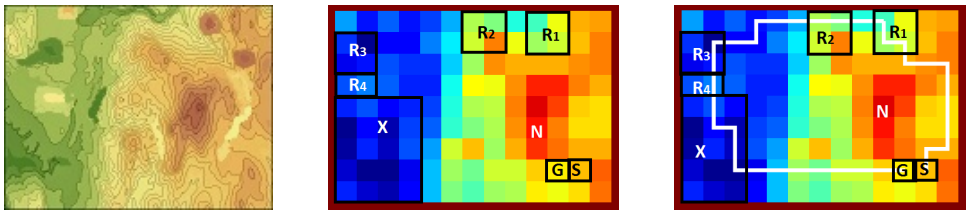

Fig. 2. Navigation experiment: Left : Terrain map retrieved from Internet. Middle : Risk map and labels. Right : A sample path satisfying all tasks.

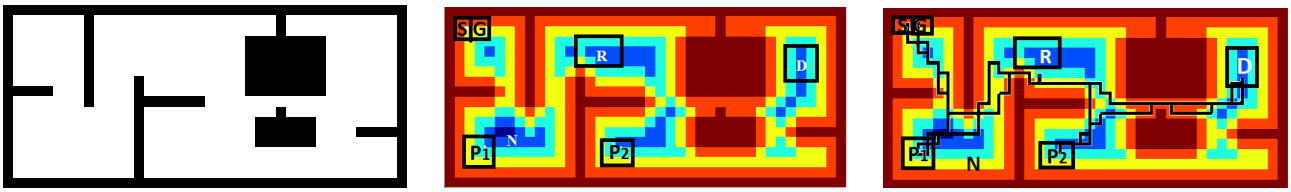

Fig. 3. Factory experiment: Left : factory floor map. Middle : Risk map of the factory and its labels. Right : A sample path satisfying all properties.

Each of these locations is selected with equal probability. When the robot selects an action trying to move to a higher location, the action succeeds with probability of 0.3, and fails in similar way. The same picture in Figure 2 labels some regions in the map. The objective is to find a path from *S* (start) to *G* (goal) that has the lowest possible cumulative risk along the path and whose length is bounded by a given threshold, while the robot has to perform three additional tasks specified in sc-LTL as follows.

1) Visit region R_1 then R_2 in order to read some sensory data, i.e., $\phi_1 = \Diamond R_1 \bigcirc \Diamond R_2$.
2) Visit region R_3 and Leave region R_3 from its border to region R_4, i.e., $\phi_2 = \Diamond R_3 \bigcirc (R_3 U R_4)$.
3) Visit at least two consecutive states in region X in order to explore that region, i.e., $\phi_3 = \Diamond(X \bigcirc X)$.

Consistently with our problem definition, we will setup different probability bounds for each of the properties specifying the three tasks. The right picture in Figure 2 shows a valid path that satisfies all the properties.

B. Factory environment

In the second scenario we consider a factory-like environment where the robot is tasked with picukp-delivery tasks in an environment with multiple obstacles. This is similar to our former work considered in [10], [11]. Actions are non-deterministic and succeed with probability 0.8. When failure occurs, the robot ends up in one of the free adjacent free locations (excluding the target location). The factory map is shown in the left picture of Figure 3. A risk map is defined as proximity to obstacles ranging as seen in the middle panel of the same figure. Regions in the map are defined according to the labels shown in the same picture which will be used in order to define tasks. The robot starts from *S* and ends at *G*. As in the previous task, the robot has to minimize the overall cumulative task while obeying to a bound on the traveled length. The following additional tasks are defined:

1) Go to P_1 to pick up an object, then deliver it at D location, i.e., $\phi_1 = \Diamond P_1 \bigcirc \Diamond D$.
2) Go to P_2 to pick up an object, then deliver it at D location, i.e., $\phi_2 = \Diamond P_2 \bigcirc \Diamond D$.
3) Stay away from region R, i.e., $\phi_3 = (\neg R) U G$. Since there is no *always* ($\Box$) operator for sc-LTL properties, it needs to be defined requiring that the robot ends in G.

The right panel in Figure 3 shows a sample path.

C. Rapid Deployment

Finally we consider a rapid deployment problem, similar to what we did in our recent work [3], [4]. In a rapid deployment scenario the robot is tasked with visiting a number of locations with the objective of maximizing the probability of success while obeying to a bound on the time to complete the task. Figure 4 shows a map where four regions are labeled as *A*, *B*, *C* and *D* and the goal area is marked with *G*. The objective is to visit each region with a certain probability. Four tasks are defined as:

1) Visit region *A*, i.e., $\phi_1 = \Diamond A$
2) Visit region *B*, i.e., $\phi_2 = \Diamond B$
3) Visit region *C*, i.e., $\phi_3 = \Diamond C$
4) Visit region *D*, i.e., $\phi_4 = \Diamond D$

Since the tasks are defined independent from each other, assuming the probability of visiting *A* is *P(A)* and so on, the probability of successfully completing the mission is then $P(A)P(B)P(C)P(D)$. Therefore, given a target probability of success for the whole mission, one can accordingly set the probabilities to satisfy each of the four tasks.

D. Results

For the outdoor navigation scenario we set the satisfaction probabilities of ϕ_1, ϕ_2 and ϕ_3 to be 0.7, 0.8 and 0.5 respectively. We also set the upper bound of the path length to be 30. For the factory scenario, the satisfaction rate of the tasks were 0.7, 0.7 and 0.8 while the upper bound of the path length was set to 150. For the rapid deployment scenario the probabilities are set to 0.6, 0.8, 0.5 and 0.9 respectively.

After determining the policy π from the linear program we simulated 1000 executions for each of the problems. Table

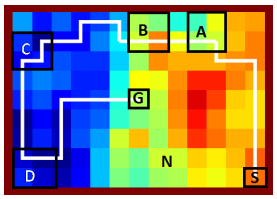

Fig. 4. Rapid deployment experiment: A sample path satisfying all tasks.

Experiment	ϕ_1	ϕ_2	ϕ_3	ϕ_4	Optimization
Navigation	36.6	244.6	2427	N/A	91.6
Factory	157.6	1012.4	8928.4	N/A	158.5
Deployment	11.4	20.4	72.9	274.5	47.1

TABLE I

TIME SPENT FOR EVERY LTL PRODUCT AND THE TIME FOR SOLVING LINEAR PROGRAMMING PROBLEMS (IN SECONDS.)

I shows the time taken to compute the various products as well as the time to solve the linear program. All tests were run on a Linux machine with quad core i7 CPU at 2.5 GHz with 12 MB of RAM and coded in Matlab.

Table II shows the number of states for all experiments. The last column (NP – no pruning) in the table displays how many states would have been generated if the product was calculated without pruning unreachable states. A tenfold reduction is obtained in average. Table III illustrates the number of times, in 1000 simulations, that the sink state of every extended-total-DFAs associated with each sc-LTL properties are reached. To interpret the results it is important to recall that the sink state is traversed when the property is not verified, i.e., for a given probability P_ϕ then in N trials we would expect that the sink state is traversed $(1 - P_\phi)N$ times. The table confirms that the algorithm produces policies consistent with the objectives (small deviations from the expected number are due to the randomized nature of the experiments.) Total risk along the path for experiments I, II and III are 106.5, 1049.9 and 128.1 respectively. Average path length for experiments I, II and III is 24.55, 142.7 and 34.5 in order.

Experiment	Original	ϕ_1	ϕ_2	ϕ_3	ϕ_4	NP
Navigation	117	600	1526	5024	N/A	25272
Factory	608	1550	3136	9400	N/A	87552
Deployment	117	356	474	950	1894	29952

TABLE II

ORIGINAL NUMBER OF STATES, AND THE NUMBER OF STATES AFTER EACH PRODUCT OPERATION. NP STANDS FOR NOT PRUNED.

Experiment	ϕ_1	ϕ_2	ϕ_3	ϕ_4
Navigation	311	195	455	N/A
Factory	259	305	194	N/A
Deployment	393	188	495	102

TABLE III

NUMBER OF TIMES EVERY SINK STATE OF EVERY EXTENDED-TOTAL-DFA IS REACHED.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we tackled the problem of multi-objective planning in non-deterministic environments while two types of constraints have to be satisfied. The first category concerns total cost values where one cost is minimized and the remaining are bounded in expectation. The second category is related to high level tasks. High level tasks are defined as sc-LTL properties, and converted to extended-total-DFAs. By calculating the product between the original CMDP and DFAs, and taking advantage of occupation measures, we are able to extract proper policies that achieve the goals matching a given probability of success (if feasible). A pruning algorithm is also used in order to remove a set of useless states. Our approach can be extended in different ways. The order in which the DFAs are multiplied with LCMDP has an impact on the size of the product and could be optimized. Another extension we will consider is risk aversion, i.e., producing policies that not only minimize expectations, but bound variance from the expectation.

REFERENCES

[1] E. Altman. *Constrained Markov Decision Processes*. Stochastic modeling. Chapman & Hall/CRC, 1999.

[2] C. Baier and J.P Katoen. *Principles of model checking*. MIT Press, 2008.

[3] S. Carpin, M. Pavone, and B.M. Sadler. Rapid multirobot deployment with time constraints. In *Proc. of IROS*, pages 1147–1154, 2014.

[4] Y-L. Chow, M. Pavone, B.M. Sadler, and S. Carpin. Trading safety versus performance: Rapid deployment of robotic swarms with robust performance constraints. *ASME Journal of Dynamical Systems, Measurements and Control*, 137(3):031005, 2015.

[5] X. Ding, B. Englot, A. Pinto, A. Speranzon, and A. Surana. Hierarchical multi-objective planning: From mission specifications to contingency management. In *Proc. of ICRA*, pages 3735–3742, 2014.

[6] X. Ding, A. Pinto, and A. Surana. Strategic planning under uncertainties via constrained markov decision processes. In *Proc. of ICRA*, pages 4568–4575, 2013.

[7] X. Ding, S. L. Smith, C. Belta, and D. Rus. Ltl control in uncertain environments with probabilistic satisfaction guarantees. In *World Congress*, volume 18, pages 3515–3520, 2011.

[8] X. Ding, S. L. Smith, C. Belta, and D. Rus. Optimal control of markov decision processes with linear temporal logic constraints. *IEEE Trans. Automat. Contr.*, 59(5):1244–1257, 2014.

[9] K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 50–65. Springer, 2007.

[10] S. Feyzabadi and S. Carpin. Risk-aware path planning using hierarchical constrained markov decision processes. In *Proc. of CASE*, pages 297–303, 2014.

[11] S. Feyzabadi and S. Carpin. HCMDP: a hierarchical solution to constrained markov decision processes. In *Proc. of ICRA*, pages 3971–3978, 2015.

[12] A. Kolobov. Planning with Markov decision processes: An AI perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–210, 2012.

[13] B. Lacerda, D. Parker, and N. Hawes. Optimal and dynamic planning for markov decision processes with co-safe ltl specifications. In *Proc. of IROS*, pages 1511–1516, 2014.

[14] M. Sipser. *Introduction to the theory of computation*. Course technology CENGAGE learning, 2006.

[15] A. Ulusoy, T. Wongpiromsarn, and C. Belta. Incremental controller synthesis in probabilistic environments with temporal logic constraints. *IJRR*, 33(8):1130–1144, 2014.

[16] T. Wongpiromsarn, A. Ulusoy, C. Belta, E. Frazzoli, and D. Rus. Incremental synthesis of control policies for heterogeneous multi-agent systems with linear temporal logic specifications. In *Proc. of ICRA*, pages 5011–5018, 2013.