Rewarding Behaviors

Fahiem Bacchus
Dept. Computer Science
University of Waterloo
Waterloo, Ontario
Canada, N2L 3G1
fbacchus@logos.uwaterloo.ca

Craig Boutilier
Dept. Computer Science
University of British Columbia
Vancouver, B.C.
Canada, V6T 1Z4
cebly@cs.ubc.cs

Adam Grove
NEC Research Institute
4 Independence Way
Princeton NJ 08540, USA
grove@research.nj.nec.com

Abstract

Markov decision processes (MDPs) are a very popular tool for decision theoretic planning (DTP), partly because of the well-developed, expressive theory that includes effective solution techniques. But the Markov assumption—that dynamics and rewards depend on the current state only, and not on history—is often inappropriate. This is especially true of rewards: we frequently wish to associate rewards with behaviors that extend over time. Of course, such reward processes can be encoded in an MDP should we have a rich enough state space (where states encode enough history). However it is often difficult to "hand craft" suitable state spaces that encode an appropriate amount of history.

We consider this problem in the case where non-Markovian rewards are encoded by assigning values to formulas of a temporal logic. These formulas characterize the value of temporally extended behaviors. We argue that this allows a natural representation of many commonly encountered non-Markovian rewards. The main result is an algorithm which, given a decision process with non-Markovian rewards expressed in this manner, *automatically* constructs an equivalent MDP (with Markovian reward structure), allowing optimal policy construction using standard techniques.

1 Introduction

Recent years have seen a tremendous interest in extending the classical planning paradigm to deal with domains involving uncertain information, actions with uncertain effects, and problems with competing objectives. Much work in *decision theoretic planning* (DTP), generally aimed at addressing these issues, has adopted the theory of *Markov decision processes* (MDPs) as the underlying conceptual and computational model [DKKN93, TR94, BD94, BDG95]. MDPs allow one to formulate problems in which an agent is involved in an on-going, *process-oriented* interaction with the environment and receives rewards at various system states. This generalizes the classical *goal-oriented* view of planning [BP95]. Instead of classical *plans*, one considers the more flexible concept of a *policy*, namely a mapping from each state to the action that should be executed in that state. Effective optimization methods exist for computing policies such that an agent executing the policy will maximize its accumulated reward over time [Put94].

The fundamental assumption underling the formulation of a planning problem as an MDP is that the system dynamics and rewards are Markovian. That is, the manner in which the system behaves when an action is executed, and the rewards received, depend only on the system's current state, not on states previously visited. For example, if we wish to control a robot it is usually not difficult to find a state space in which the robot's actions can be described as Markovian (stochastic) state transitions. In fact, this is often the most natural way to represent the effects of actions. Assigning natural Markovian rewards can be more problematic.

Although it is sometimes easy to associate rewards with individual states (e.g., in a navigation problem where rewards are associated with locations), often a reward is most naturally assigned to some behavior that occurs over an extended period. In such cases, it can be difficult to encode the reward as a function of state. For instance, we may reward an agent in states where coffee has just been delivered, but *only* if this state was preceded by a state (perhaps within k steps) where a coffee request was issued, withholding reward for spurious delivery. This reward is properly a function of the system *trajectory* or *history*, and not of the state alone. Typical forms of desirable temporally extended behaviors include response to requests, bounded response, lack of response, maintaining safety constraints, and so on. Temporally extended goals of this nature have been examined to some extent in the literature [HH92, Dru89, Kab90, GK91], but not in the context of generating effective policies.

The key difficulty with non-Markovian rewards is that standard optimization techniques, most based on Bellman's [Bel57] dynamic programming principle, cannot be used. One way of dealing with this predicament is to formulate an equivalent decision problem in which the rewards are Markovian. In particular, one can augment the state space of the underlying system by adding variables that keep track of the *history* relevant to the reward function. For instance, Boutilier and Puterman [BP95] suggest straightforward ways of encoding reward functions that involve simple requests. This approach has the advantage that existing optimization methods for MDPs can be used.

Unfortunately, in general, finding a good way to augment the state space requires considerable cleverness—especially if we are concerned with minimizing the size of the resulting augmented space for computational reasons. In this paper, we examine the problem of rewarding temporally extended behaviors. We provide a natural, and quite expressive, means for specifying rewards attached to behaviors extended over time. Furthermore, we solve the problem of computing policies in the face of these non-Markovian rewards by developing an algorithm that *automatically* constructs a Markovian reward process and associated MDP. Our algorithm automates the process of generating an appropriate augmentation of the state space, and, when coupled with traditional policy construction techniques, provides a way of computing policies for a much richer range of reward functions.

In Section 2 we introduce NMRDPs, essentially MDPs with non-Markovian reward. System dynamics are specified as with MDPs, but rewards are associated with formulas in a suitable temporal logic. We define *temporally-extended reward functions* (TERFs) by requiring that the reward associated with a formula be given at any state in which the formula is satisfied. We note that the decision to reward an agent in a given state should depend only on past states, not on future states. For this reason, it will be more natural to encode our reward formulas using a *past* or backward-looking temporal logic rather than the usual future or forward logics like LTL, CTL [Eme90] or MTL [AH90]. In Section 3, we describe a number of interesting and useful classes of target behaviors and show how they can be encoded by TERFs.

In Section 4, we consider the problem of constructing optimal policies for NMRDPs. As mentioned, dynamic programming cannot be used to construct policies in this setting. Nominally, this requires one to resort to optimization over a policy space that maps *histories* (rather than states) into actions, a process that would incur great computational expense. We present a procedure that, instead, expands the original state space by attaching a temporal formula to each state. This formula keeps track of an appropriate amount of relevant history. By constructing a state-based (Markovian) reward function for the extended state space, we convert the NMRDP into an equivalent MDP; in particular, optimal policies for this MDP determine optimal policies for the original NMRDP in a natural way. In this way, we obtain a compact representation of the required history-dependent policy by considering only relevant history, and can produce this policy using computationally-effective MDP algorithms.

2 Non-Markovian Rewards

2.1 Markov Decision Processes

Much recent work in DTP considers planning problems that can be modeled by *completely observable Markov Decision Processes* [How60, Put94]. In this model, we assume that there is a finite set of system states S, a set of actions A, and a reward function R. The effects of actions cannot be predicted with certainty; hence we write $\Pr(s_1, a, s_2) = p$ (or $s_1 \xrightarrow{a,p} s_2$) to denote that s_2 is reached with probability p when action a is performed in state s_1. Complete observability entails that the agent always knows what state it is in. We assume that the state space is characterized by a set of features, or logical propositions. This allows actions to be described compactly using probabilistic STRIPS rules [KHW94, BD94], Bayes nets [DK89, BDG95] or other action representations.

A real-valued *reward function* R reflects the objectives, tasks and goals to be accomplished by the agent, with $R(s)$ denoting the (immediate) utility of being in state s. For our purposes, then, an MDP consists of S, A, R and the set of transition distributions $\{\Pr(\cdot, a, \cdot) : a \in A\}$.

A *stationary Markovian policy* is a mapping $\pi : S \to A$, where $\pi(s)$ denotes the action an agent should perform whenever it is in state s. One might think of such policies as reactive or universal plans [Sch87]. Given an MDP, an agent ought to adopt a policy that maximizes the expected *value* over its (potentially infinite) trajectory through the state space. The most common value criterion in DTP for infinite-horizon problems is *discounted total reward*: the current value of future rewards is discounted by some factor β $(0 < \beta < 1)$, and we maximize the expected accumulated discounted rewards over an infinite time period. The expected value of a fixed policy π at any given state s can be shown to satisfy [How60]:

$$V_\pi(s) = R(s) + \beta \sum_{t \in S} \Pr(s, \pi(s), t) \cdot V_\pi(t)$$

The value of π at any initial state s can be computed by solving this system of linear equations. A policy π is *optimal* if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in S$ and policies π'.

Techniques for constructing optimal policies in the case of discounted rewards have been well-studied, and include algorithms such as *value iteration* [Bel57] and *policy iteration* [How60]. It should be noted that each of these algorithms exploits the Markovian nature of the reward process. We refer to [Put94] for an excellent treatment of MDPs and associated computational methods.

2.2 A Temporal Logic of the Past

To reward agents for (temporally extended) *behaviors*, as opposed to simply reaching certain states, we need a means to specify rewards for specific trajectories through the state space. Generally, we want to associate rewards with *properties* of trajectories rather than rewarding individual trajectories. For example, we might reward an agent whenever condition Q is achieved within k steps of condition P, without regard for the *particular* trajectory the agent is traversing. Therefore, we associate rewards (or penalties) with desirable (or undesirable) *formulas* in a suitable temporal logic that describes such trajectory properties.

The logic we consider is "backward", or past looking. That is, the truth of a temporal formula depends on prior states only, not on what will happen in the future. This accords well with our view of reward processes because, in most contexts, rewards should be earned based on what has actually happened.

We present a past version of LTL [Eme90] called PLTL. We assume an underlying finite set of propositional constants **P**, the usual truth functional connectives, and the following temporal operators: S (since), $\boxminus$ (always in the past), $\diamondminus$ (once, or sometime in the past) and $\ominus$ (previously).[1] The

[1]These are the backward analogs of the LTL operators until,

formulas $\phi_1 \mathsf{S} \phi_2$, $\boxminus\phi_1$, $\diamondminus\phi_1$ and $\ominus\phi_1$ are well-formed when ϕ_1 and ϕ_2 are.[2] We use $\top$ and $\bot$ to denote truth and falsity, respectively. The semantics of PLTL is described with respect to models of the form $T = \langle s_0, \cdots, s_n\rangle$, $n \geq 0$, where each s_i is a *state* or valuation over $\mathbf{P}$ (i.e., $s_i \in 2^{\mathbf{P}}$). Such a T is called a *(finite) trajectory*, or partial history. For any trajectory $T = \langle s_0, \cdots, s_n\rangle$, and any $0 \leq i \leq n$, let $T(i)$ denote the *initial segment* $T(i) = \langle s_0, \cdots, s_i\rangle$.

Intuitively, a temporal formula is true of $T = \langle s_0, \cdots, s_n\rangle$ if it is true at the last (or *current* state) with respect to the history reflected in the trajectory. We define the truth of formulas inductively as follows:

- $T \models P$ iff $P \in s_n$, for $P \in \mathbf{P}$
- $T \models \phi_1 \wedge \phi_2$ iff $T \models \phi_1$ and $T \models \phi_2$
- $T \models \neg\phi$ iff $T \not\models \phi$
- $T \models \phi_1 \mathsf{S} \phi_2$ iff there is some $i \leq n$ s.t. $T(i) \models \phi_2$ and for all $i < j \leq n$, $T(j) \models \phi_1$ (intuitively, ϕ_1 has been true since the last time ϕ_2 held)
- $T \models \boxminus\phi$ iff for all $0 \leq i \leq n, T(i) \models \phi$ (ϕ has been true at each point in the past)
- $T \models \diamondminus\phi$ iff for some $0 \leq i \leq n, T(i) \models \phi$ (ϕ was true at some point in the past)
- $T \models \ominus\phi$ iff $n > 0$ and $T(n-1) \models \phi$ (ϕ was true at the previous state)

One notable consequence of this semantics is the fact that while $\{\ominus\phi, \ominus\neg\phi\}$ is unsatisfiable, $\{\neg\ominus\phi, \neg\ominus\neg\phi\}$ is satisfiable: any model of the form $\langle s\rangle$ satisfies the latter.

It is well-known that the modalities in LTL can be decomposed into present and future components [Eme90]. Similarly, modalities of PLTL can be decomposed into present and past components. For example, $\boxminus\phi$ is equivalent to $\ominus\boxminus\phi \wedge \phi$. That is, $\boxminus\phi$ is true iff ϕ is true of the current state and $\boxminus\phi$ is true of the previous state. Using these equivalences we can determine, for any formula ϕ, what must have been true in the previous state in order that ϕ be true now. We call this the *regression* of ϕ through the current state. Note that if the current component of ϕ is falsified by the current state, then nothing about the previous state can make ϕ true now. In this case the regression of ϕ is $\bot$.

Definition 2.1 *The* regression of ϕ through s, *denoted* $\mathrm{Regr}(\phi, s)$, *is a formula in PLTL such that, for all trajectories T of length $n > 1$ with final state s, we have*

$$T \models \phi \quad \text{iff} \quad T(n-1) \models \mathrm{Regr}(\phi, s) \qquad \blacksquare$$

$\mathrm{Regr}(\phi, s)$ can be computed recursively:

- If $\phi \in \mathbf{P}$, $\mathrm{Regr}(\phi, s) = \top$ if $s \models \phi$, and $\bot$ otherwise
- $\mathrm{Regr}(\phi_1 \wedge \phi_2, s) = \mathrm{Regr}(\phi_1, s) \wedge \mathrm{Regr}(\phi_2, s)$
- $\mathrm{Regr}(\neg\phi_i, s) = \neg\mathrm{Regr}(\phi_1, s)$
- $\mathrm{Regr}(\ominus\phi, s) = \phi$
- $\mathrm{Regr}(\phi_1 \mathsf{S} \phi_2, s) = \mathrm{Regr}(\phi_2, s) \vee (\mathrm{Regr}(\phi_1, s) \wedge (\phi_1 \mathsf{S} \phi_2))$
- $\mathrm{Regr}(\diamondminus\phi_1, s) = \mathrm{Regr}(\phi_1, s) \vee \diamondminus\phi_1$
- $\mathrm{Regr}(\boxminus\phi_1, s) = \mathrm{Regr}(\phi_1, s) \wedge \boxminus\phi_1$

Finally, we define some useful notation. For an MDP (or NMRDP) with actions A and transition probabilities Pr, a trajectory $\langle s_0, \cdots, s_n\rangle$ is *feasible* iff there are actions $a_1, \cdots, a_n \in A$ such that $\Pr(s_i, a_i, s_{i+1}) > 0$. If ϕ_1 and ϕ_2 are PLTL formulas, ϕ_1 *determines* ϕ_2 iff either $\phi_1 \models \phi_2$ or $\phi_1 \models \neg\phi_2$ hold. Given any PLTL formula ϕ, we define *Subformulas*(ϕ) to be the set of all subformulas of ϕ (including ϕ itself). Note that $|\mathit{Subformulas}(\phi)| \leq \mathit{length}(\phi)$.

2.3 Rewarding Temporally-Extended Behaviors

To reward behaviors, we must adopt a generalization of MDPs that allows the reward given at any stage of the process to depend on past history. A *decision process with non-Markovian reward*, or NMRDP, is similar to an MDP with the exception that the reward function R takes as its domain histories of the form $\langle s_0, \cdots, s_n\rangle$ for all n. Intuitively, the agent receives reward $R(\langle s_0, \cdots, s_n\rangle)$ at stage n if the process has passed through state s_i at stage i for all $i \leq n$. Clearly, the explicit specification of such a reward function is impossible since there are an infinite number of different histories. Instead, we assume that the reward function of an NMRDP can be specified more compactly. In particular, we assume that the reward function is defined by a finite set Φ of *reward formulas* expressed in PLTL, together with a real-valued reward r_i associated with each $\phi_i \in \Phi$ (we sometimes write this $\phi_i : r_i$). The temporally extended reward function (TERF) R is then defined as follows:

$$R(\langle s_0, \cdots, s_n\rangle) = \sum\{r_i : \langle s_0, \cdots, s_n\rangle \models \phi_i\}$$

This formulation gives a reward of r_i at each state that satisfies formula ϕ_i; if ϕ_i has a nontrivial temporal component then the reward is *history-dependent*. Because reward formulas are expressed in PLTL, rewards depend only on past states, and the TERF can be unambiguously evaluated at each stage of the process.[3]

Consideration should not be restricted to Markovian policies when dealing with NMRDPs. The value, and hence the choice, of action at any stage may depend on history. We thus take policies to be mappings from histories to actions. As usual, the value of a given policy π is taken to be the expectation of the discounted accumulated reward:

$$V_\pi(s_0) = E\{\sum_{n=0}^{\infty} \beta^n R(\langle s_0, s_1, \cdots, s_n\rangle)|\pi\}.$$

Since TERFs are finitely specified, we can find good ways of encoding and computing optimal policies (see Section 4). But first we examine the expressive power of TERFs.

always, eventually and next, respectively.

[2] We use the abbreviation $\ominus^k$ for k iterations of the $\ominus$ modality (e.g., $\ominus^3\phi \equiv \ominus\ominus\ominus\phi$), and $\ominus^{\leq k}$ to stand for the disjunction of $\ominus^i$ for $1 \leq i \leq k$, (e.g., $\ominus^{\leq 2}\phi \equiv \ominus\phi \vee \ominus\ominus\phi$).

[3] The r_i are assumed to be additive and independent (this is not restrictive). Any (history *independent*) MDP can be expressed this way by restricting Φ to contain no temporal operators.

3 Encoding Typical Forms of Behavior

To demonstrate that TERFs provide an appropriate and useful language in which to specify rewards for NMRDPs, we examine several common examples to see how they can be encoded in PLTL. We make no claim that all interesting rewards can be encoded in this way, but the evidence suggests that PLTL and TERFs can capture a very large and useful class of reward functions.

Among the common types of behaviors, simple *goal achievement* has retained a special place in classical planning. However, in a process-oriented model, like an MDP or NMRDP, a number of subtleties arise in giving "goal achievement" a precise interpretation. We describe several possibilities. Assume one such goal is the proposition G: we wish the agent to reach a state in which G holds and will reward it with r if it does so. The simplest reward formula for this goal is G. As a TERF, this rewards the agent at every state satisfying G, and hence the agent is more highly rewarded (roughly) the larger fraction of its time it spends in G-states. This provides incentive for the agent to constantly maintain G if r is greater than rewards it may receive for other behaviors.

In many cases, this is not the intended effect of specifying a goal G. If we only care that G is achieved once, there are several different interpretations that can be provided. The strictest offers reward r only to the *first* state at which G holds; that is, $(G \wedge \neg\ominus\diamondminus G) : r$. A more generous formula, $\diamondminus G : r$, rewards every state that follows the achievement of G. Finally, we may reward G periodically, but not encourage constant maintenance of G, by rewarding G at most once every k stages: formula $G \wedge \neg(\ominus^{\leq k} G) : r$ will reward G-states that have not occurred within k-stages of a previous G-state. Yet another option rewards any G-state that occurs within k-stages of some $\neg G$-state (allowing up to k consecutive G-rewards), using $G \wedge \ominus^{\leq k}\neg G : r$.

In addition, PLTL allows one to formulate temporally extended goal *sequences*. For instance, if the agent is to be rewarded for achieving G, followed immediately by H and then by I, the reward formula $\ominus^2 G \wedge \ominus H \wedge I$ can be used. Periodic reward of such behavior, or the similar behavior in which other steps are allowed to intervene between G, H, and I, can also be prescribed in a straightforward fashion.

The formulations above assume that there is some goal G that is constantly desirable, a vestige of the classical interpretation of goals. Such behaviors are more suited to background, maintenance goals. In a process-oriented setting, we are likely to want the agent to respond to requests or *commands* to bring about some goal. In these settings, goals are not constant: they arise periodically, can be fulfilled, forgotten, preempted, and might even expire. We model these in PLTL using *response formulas* which specify a relation between a *command* C and rewarded goal achievement G.

The most basic response formula is that of *eventual response*, $G \wedge \diamondminus C$—the agent is rewarded at any G-state that follows a C-state in which the command is given (or is outstanding). As usual, we may only wish to reward the first state at which G holds following the command, in which case $G \wedge \ominus(\neg G \mathsf{S} C)$ suffices.

Many requests must be achieved in a timely fashion. *Immediate response* formulas have the form $G \wedge \ominus C$, rewarding a goal achieved at the state following a command. More generally, we have *bounded response* formulas of the type $G \wedge \ominus^{\leq k} C$ which reward goal achievement within k steps of a request. This formula does not preclude multiple rewards for a single request, so we might instead prefer $G \wedge \ominus^{\leq k} C \wedge \ominus(\neg G \mathsf{S} C)$, which rewards only the first goal state. Finally, a *graded reward* can be given for faster achievement of G (within limits). For instance, the set

$$\left\{G \wedge \ominus C : r_1,\ \ G \wedge \ominus^{\leq 2} C : r_2,\ \ G \wedge \ominus^{\leq 3} C : r_3\right\}$$

rewards goal achievement in one step with reward $r_1+r_2+r_3$, in two steps with $r_2 + r_3$, and in three steps with r_3.

In a longer version of this paper, we describe additional types of behaviors, as well as the possibility of using other logics to express different kinds of reward.

4 Modeling NMRDPs with MDPs

As has been pointed out, constructing optimal policies in settings of non-Markovian reward can be computationally prohibitive. In this section, we describe a method of state-space expansion that determines the aspects of history that are *relevant* to an NMRDP (i.e., which must be recorded so that we can verify the truth of the temporal reward formulas), and encodes this history within the state. A straightforward transformation of the reward function, so that rewards are attached to such extended states rather than trajectories, restores the Markovian reward property. Together with an adjustment in action descriptions to deal with the new state space, we then have a (fully-observable) MDP that accurately reflects the NMRDP, that can be solved by standard (relatively efficient) methods. We begin by discussing the basic properties that such a transformation should satisfy, and then specialize to the case of rewards that are given by TERFs.

4.1 Markovian Transformations

To transform an NMRDP into an equivalent MDP requires that we *expand* the state space S of the NMRDP so that each new state in the expanded state space *ES* carries not just the original state information, but also any additional information required to render reward ascription independent of history.[4] As we shall see, we can think of expanded states as consisting of a base state annotated with a label that summarizes relevant history. If $G_S = (S, A, R)$ is the NMRDP in question, then we wish to produce an MDP $G_{ES} = (ES, A, R_{ES})$ with expanded space *ES*. The actions A available to the agent remain unchanged (since the aim is to produce a policy suitable for the original NMRDP), but the reward function R_{ES} is now Markovian: it assigns rewards to (expanded) states.

For the new MDP to be useful, we would expect it to bear a strong relationship to the NMRDP from which it was constructed. In particular, we define a strong correspondence between the two as follows:

[4] Here we are concerned only with reward ascription; the system dynamics are already Markovian.

Definition 4.1 *An MDP $G_{ES} = (ES, A, R_{ES})$ is an* expansion *of an NMRDP $G_S = (S, A, R)$ if there are functions $\tau : ES \mapsto S$ and $\sigma : S \mapsto ES$ such that:*

1. *For all $s \in S$, $\tau(\sigma(s)) = s$,*
2. *For all $s, s' \in S$ and $es \in ES$, if $\Pr(s, a, s') = p > 0$ and $\tau(es) = s$, then there is a unique es', $\tau(es') = s'$, such that $\Pr(es, a, es') = p$.*
3. *For any feasible trajectories $\langle s_0, \cdots, s_n\rangle$ in G_S and $\langle es_0, \cdots, es_n\rangle$ in G_{ES}, where $\tau(es_i) = s_i$ and $\sigma(s_0) = es_0$, we have $R(\langle s_0, \cdots, s_n\rangle) = R_{ES}(es_n)$.* ∎

Intuitively, $\tau(es)$ is the *base state* for *es*, the state in S extended by *es*. For this reason, we will often speak of extended states being *labeled* or annotated: each extended state can be written $s \circ l$, where $s \in S$ is the base state, and l is a label that distinguishes *es* from other extensions of s. However, among the extensions of s, we must pick out a unique $\sigma(s) \in ES$ as the "start state" corresponding to s. In other words, $\sigma(s)$ should be thought of as that annotation of s with an "empty" history; i.e., corresponding to an occurrence of s at the very start of a trajectory. We will see below why it is important to distinguish this extension of s from other extensions.

The important parts of this definition are clauses (2) and (3), which assert that G_{ES} and G_S are equivalent (with respect to base states) in both their dynamics and reward structure. In particular, clause (2) ensures, for any trajectory in G_S

$$s_0 \xrightarrow{a_1, p_1} s_1 \cdots s_{n-1} \xrightarrow{a_n, p_n} s_n$$

and extended state es_0 with base state s_0, that there is a trajectory in G_{ES} of similar structure

$$es_0 \xrightarrow{a_1, p_1} es_1 \cdots es_{n-1} \xrightarrow{a_n, p_n} es_n$$

where $\tau(es_i) = s_i$ for all i. We call $\langle es_0, \cdots, es_n\rangle$ and $\langle s_0, \cdots, s_n\rangle$ *weakly corresponding* trajectories in this case. Clause (3) imposes strong requirements on the reward assigned to the individual states in G_{ES}. In particular, if $\langle es_0, \cdots, es_n\rangle$ and $\langle s_0, \cdots, s_n\rangle$ are weakly corresponding, and $\sigma(s_0) = es_0$ (i.e., es_0 is a start state), we say these trajectories are *strongly corresponding*. It is not hard to see that this relationship is one-to-one: each $\langle s_0, \cdots, s_n\rangle$ has a unique strongly corresponding trajectory, and $\langle es_0, \cdots, es_n\rangle$ has a unique strongly corresponding trajectory iff es_0 is a start state. Clause (3) requires that R_{ES} assign rewards to extended states in such a manner that strongly corresponding trajectories receive the same reward. This need not be the case for weakly corresponding trajectories since, intuitively, different annotations (extensions) of s_0 correspond to different possible histories.

If we can produce an MDP G_{ES} that is an expansion of an NMRDP G_S as specified by Defn. 4.1, then we can find optimal policies for G_S by solving G_{ES} instead.

Definition 4.2 *Let π' be a policy for MDP G_{ES}. The* corresponding *policy π for the NMRDP G_S is defined as $\pi(\langle s_0, \cdots, s_n\rangle) = \pi'(es_n)$, where $\langle es_0, \cdots, es_n\rangle$ is the strongly corresponding trajectory for $\langle s_0, \cdots, s_n\rangle$.* ∎

Proposition 4.3 *For any policy π' for MDP G_{ES}, corresponding policy π for G_S, and $s \in S$, we have $V_\pi(s) = V_{\pi'}(\sigma(s))$.*

Corollary 4.4 *Let π' be an optimal policy for MDP G_{ES}. Then the corresponding policy π is optimal for NMRDP G_S.*

Thus, given a suitable expanded MDP and an optimal policy π', one can produce an optimal policy π for the NMRDP quite easily. In practice, the agent need not construct π explicitly. Instead, it can run π' over the expanded MDP. Once the agent knows what base state it starts in, it determines the corresponding extended state using the function σ. Furthermore, the dynamics of the expanded MDP ensures that it can keep track of the current extended state simply by observing the base state to which each transition is made.

Finally, we should consider the size of the expanded MDP. Often, we can fulfill the requirements of Defn. 4.1 with a trivial MDP, that has states encoding *complete* trajectory information over some finite horizon. But such an expanded space grows exponentially with the horizon. Furthermore, even simple rewards—like $\diamondminus G$, which only require one item of history (a bit indicating if a G state has been passed through)—can require in infinite amount of complete trajectory history using this naive approach. If possible, we want to encode only the *relevant* history, and find an MDP which has a few states as possible (subject to Defn. 4.1). Note that state-space size tends to be the dominant complexity-determining factor in standard MDP solution techniques, especially as applied to planning problems.[5]

4.2 Transformations using TERFs

The problem of finding a small MDP that expands a given NMRDP is made easier if the latter's rewards are given by a TERF. In this case, it is natural to label states with PLTL formulas that summarize history. More precisely, the new state space *ES* consists of *annotated states*, of the form $s \circ f$ where $s \in S$ and f is a formula in PLTL. These annotations will be meaningful and correct assertions about history, in a sense to be made precise below. We give an algorithm that constructs an expansion of the state space by producing labelings of states that are sufficient to determine future reward.

We begin with a simple example to illustrate the essential ideas. Consider a single reward formula $\phi_R = Q \wedge \ominus\ominus P$. Recall that our goal is to encode all relevant history in a state's annotation. Thus, for each state s in which ϕ_R might possibly be true, we need *at least* two distinct labels, one implying the truth of ϕ_R and one its falsity.

Next, imagine that we have an extended state $es = s \circ \psi$, where Q is true in s and $\psi = \ominus\ominus P$. (Thus *es* implies that ϕ_R is true.) Next, suppose that s is reachable from some other state s^- (i.e., there is some transition in the NMRDP from s^- to s). Since we must ensure that *es*'s label ψ is a correct assertion about its history, in the expanded MDP any transition from an extended version of s^- (es^-, say) to *es* must satisfy

[5] See [LDK95] on the complexity of solving MDPs. Generally, the state space is problematic in planning problems because it grows exponentially with the number of atomic propositions. Adding history "naively" to the domain exacerbates this problem considerably.

the "historical" constraints imposed by ψ. In this example, if there is a transition from es^- to es it must be the case that es^- satisfies $\ominus P$ (otherwise, es might not satisfy $\ominus\ominus P$). In general, we can use the regression operator to determine what must have been true at earlier states. A reward formula ϕ is true of a trajectory terminating in es iff $\mathrm{Regr}(\phi, s)$ holds at es's predecessor. Thus, the formula $\mathrm{Regr}(\phi, s)$—or a stronger formula implying $\mathrm{Regr}(\phi, s)$—must be part of any label attached to states that reach $es = s \circ \phi$ in one step. This process is, naturally, repeated (states reaching es^- must satisfy P, etc.).

To quickly summarize this example, suppose that every state is reachable from any other, and that P and Q are the only propositions (hence, there are exactly 4 base states). Then 12 extended states are necessary. For each base state where Q is false (i.e., $P \wedge \neg Q$ and $\neg P \wedge \neg Q$) we need one extension labeled with $\neg\ominus P$ and another with $\ominus P$. For each of the two base states in which Q is true, we need 4 extended states, with the labels $\ominus P \wedge \ominus\ominus P$, $\neg\ominus P \wedge \ominus\ominus P$, $\ominus P \wedge \neg\ominus\ominus P$, and $\neg\ominus P \wedge \neg\ominus\ominus P$. Note that every extended state has the property that we can easily tell whether the reward formula $Q \wedge \ominus\ominus P$ is true there. Furthermore, the regression constraints discussed above hold. For example, let $s^- \models P \wedge Q$ and $es^- = s^- \circ (\neg\ominus P \wedge \ominus\ominus P)$, and consider the transition to the base state s where $s \models \neg P \wedge Q$. It is necessary that there be some labeling of s, $es = s \circ \psi$, such that $\mathrm{Regr}(\psi, s)$ is implied by es^-. But this is so, because we can take ψ to be $\ominus P \wedge \neg\ominus\ominus P$. Note that if we had not been able to find such ψ, this would mean that es^-'s label did not encode enough history (because we would be unable to determine the correct subsequent label after a move to s).

Our algorithm constructs the set of extended phases somewhat indirectly, using a two phase approach. Phase I of our algorithm constructs *label sets* for each state, $ls(s)$, containing PLTL formulas that *might* be relevant to future reward. The elements of $ls(s)$ will not necessarily be the labels themselves, but are the ingredients out of which labels are constructed. In a certain sense (to be discussed in Section 4.3) it does not matter if we add "too many" (or too strong) formulas to $ls(s)$, so there are in fact several distinct implementations of Phase I. But as we have just seen, regression should be used to impose constraints on label sets. If $\phi \in ls(s)$, so that ϕ might be (part of) the label of a extension es, then $\mathrm{Regr}(\phi, s)$ must be implied by the annotation of any state es^- from which es is reachable.

Given that Phase I is correct (i.e., it finds all formulas that might be relevant), we can restrict attention to extended states whose labels are combinations of the formulas in $ls(s)$, asserting that some are true and others false. Formally:

Definition 4.5 *If Ψ is a set of PLTL formulas, the* atoms *of Ψ, denoted* ATOMS(Ψ)*, is the set of all conjunctions that can be formed from the members of Ψ and their negations. E.g., if $\Psi = \{q \wedge \ominus p, p\}$, then* ATOMS$(\Psi) = \{(q \wedge \ominus p) \wedge p, \neg(q \wedge \ominus p) \wedge p, (q \wedge \ominus p) \wedge \neg p, \neg(q \wedge \ominus p) \wedge \neg p\}$. ▮

Thus, the labels extending s will belong to ATOMS$(ls(s))$. In general, however, many of these atoms will be inconsistent, or simply not reachable given the set of feasible trajectories in the original NMRDP. Rather than performing theorem proving to check consistency, we will *generate* the extended states we require in a constructive fashion, by explicitly considering which states are reachable from a start state. This is Phase II of our algorithm.

To illustrate, suppose that we have determined $ls(s_1) = \{Q \wedge \ominus\ominus P, \top \wedge \ominus P, \bot \wedge \ominus P, P\}$, and that $s_1 \models \neg Q \wedge P$. There is only one atom over $ls(s_1)$ that can be true at s_1 in the (length 1) trajectory $\langle s_1 \rangle$, namely:

$$f = \neg(Q \wedge \ominus\ominus P) \wedge \neg(\top \wedge \ominus P) \wedge \neg(\bot \wedge \ominus P) \wedge P.$$

We thus include $s_1 \circ f$ in ES. (Note that $s_1 \circ f$ can be logically simplified, to $s_1 \circ \neg\ominus P$.) >From this extended state we consider, for each successor state s_2 of s_1, which atom over s_2's label set is true in the trajectory $\langle s_1, s_2 \rangle$. Again, this will be unique: for instance, if s_1 can succeed itself, we obtain a new extended state $s_1 \circ f'$ where

$$f' = \neg(Q \wedge \ominus\ominus P) \wedge (\top \wedge \ominus P) \wedge \neg(\bot \wedge \ominus P) \wedge P$$

(This also can be simplified, in this case to $s_2 \circ \ominus P$.) For any action a such that $\Pr(s_1, a, s_2) = p > 0$, we assert that $\Pr(s_1 \circ f, a, s_2 \circ f') = p$. By adding extended states to ES in this way, we will only add extended states that are reachable and whose history is meaningful. For instance, we see that while $\phi = Q \wedge \ominus\ominus P$ is in $ls(s_1)$, no label that makes ϕ true at s_1 will be reachable (recall s_1 makes Q false). This effectively eliminates ϕ from consideration at s_1.

The algorithm is described in Figure 1. We defer a discussion of Phases I and III until Section 4.3. Note, however, that an easy implementation of Phase I is to set $ls(s)$, for all s, equal to $\bigcup_{\phi_i \in \Phi}$ *Subformulas*(ϕ_i) where Φ is the set of reward formulas. All the results in this section apply to any suitable choice of $ls(\cdot)$, including this one.

The MDP G_{ES} generated by the algorithm is an expansion of G_S. To show this, it is useful to define a more general concept of which G_{ES} is an instance:

Definition 4.6 *$G_{ES} = (ES, A, R_{ES})$ is a* sound annotation *of $G_S = (S, A, R)$ if each state $es \in ES$ is of the form $s \circ f$ for $s \in S$ and some PLTL formula f, and:*

1. *Fixing $\tau(s \circ f) = s$, there exists $\sigma : S \mapsto ES$ such that clauses [1] and [2] of Definition 4.1 hold.*
2. *Let $\langle s_0 \circ f_0, s_1 \circ f_1, \ldots, s_n \circ f_n \rangle$, $n \geq 0$, be such that $\sigma(s_0) = s_0 \circ f_0$. Then:*

$$\langle s_0, s_1, \ldots, s_n \rangle \models f_n$$ ▮

This definition is similar to our definition of expansion (Defn. 4.1), except that we give the extended states a particular form: annotations using PLTL formulas. Furthermore, instead of requiring that annotations summarize enough history for the purposes of determining rewards, we no longer care *why* G_{ES} has the annotations it does; we only insist that whatever history is recorded in these annotations be accurate.

Because of its generality, the notion of sound annotation may have other applications. However, for our purposes we must make one more assumption: that G_{ES}'s labels are informative enough to determine rewards.

Definition 4.7 G_{ES} determines rewards *over a set of reward formulas Φ iff, for all $es = s \circ f \in ES$ and all $\phi_i \in \Phi$, f determines ϕ_i.*

Phase I Find label sets:

Choose any $ls : S \mapsto$ *subsets of* PLTL, such that:

For all $s \in S$, all $f \in$ ATOMS$(ls(s))$, and all formulas ϕ:
If $\phi \in \Phi$, the set of reward formulas for G_S, then:
f determines ϕ
If $\phi \in ls(s')$, where $\Pr(s, a, s') > 0$ for some $a \in A$, then:
f determines $\mathrm{Regr}(\phi, s')$

Note. See text for more discussion of this phase. However, $ls_{\mathrm{sub}}(s) = \cup_{\phi_i \in \Phi}$ *Subformulas*(ϕ_i) is always suitable.

Phase II Generate G_{ES}:

1. For all $s \in S$ do:
(a) Find $f \in$ ATOMS$(ls(s))$ such that $\langle s \rangle \models f$.
Note. Such an atom exists and is unique.
(b) Add $s \circ f$ to *ES*.
Note. This will be the start state corresponding to s.
(c) Mark $s \circ f$ unvisited.
2. While there exists an unvisited state $es \in ES$, $es = s \circ f$ do:
(a) For all s' such that $\Pr(s, a, s') > 0$ for some a do:
i. Find $f' \in$ ATOMS$(ls(s'))$ such that $f \models \mathrm{Regr}(f', s')$.
ii. If $s' \circ f' \notin ES$ then add $s' \circ f'$ to *ES* and mark it unvisited.
iii. Set $\Pr(s \circ f, a, s' \circ f')$ equal to $\Pr(s, a, s')$, for all a.
3. For $es = s \circ f$ in *ES*, set $R_{ES}(es)$ to $\sum_{\phi_i \in \Phi} \{r_i : f \models \phi_i\}$.
4. Set $\Pr(s \circ f, a, s' \circ f') = 0$ for all transition probabilities not previously assigned.

Phase III Minimization: See Section 4.3 for discussion.

Note. This phase is not always necessary.

Figure 1: Algorithm to find Annotated Expansion of G_S

Proposition 4.8 *If $G_{ES} = (ES, A, R_{ES})$ is sound annotation of G_S that determines rewards over Φ, and $R_{ES}(s \circ f) = \sum_{\phi_i \in \Phi} \{r_i : f \models \phi_i\}$, then G_{ES} is an expansion of G_S.*

The key to understanding our algorithm is realizing that it is designed to generate a MDP that satisfies the conditions of this proposition. Thus, by the results of Section 4.1, we have succeeded in our goal of finding an equivalent MDP G_{ES} for any NMRDP G_S whose rewards are given using a TERF. In particular, we have the following key result:

Theorem 4.9 *Let G_S be an NMRDP whose reward function is given by a TERF, over a set of formulas Φ. The Expansion Algorithm of Figure 1 constructs an MDP G_{ES} that is an expansion of G_S.*

Once this expansion G_{ES} is constructed, an optimal policy for the MDP G_{ES} can be computed using standard techniques. The correspondence presented in Section 4.1 shows that an agent executing this policy will behave optimally with respect to the original NMRDP. We note that the labels in G_{ES} determine the history that must be kept track of during policy execution. In particular, suppose we are given a policy π' defined on the extended space to apply to the NMRDP, and the process starts in state s_0. We take the extended state to be s_0's unique start state es_0 and perform $\pi'(es_0) = a$. An observation of the resulting state s_1 is made. The dynamics of the extended MDP ensure that there is a unique es_1 extending s_1 that is reachable from es_0 under action a. Thus, we next execute action $\pi'(es_1)$, and proceed as before. Note that we can keep track of the extended state that we are currently in even though we only get to directly observe base states.

4.3 Other Properties of the Algorithm

In this section, we very briefly discuss some of the other interesting issues raised by the expansion algorithm.

We begin by examining Phase I. As already noted, one possible implementation is $ls(s) = ls_{\mathrm{sub}}(s)$; i.e., the label sets consisting of all subformulas of Φ. An advantage of this choice is that Phase I becomes trivial, with complexity $O(L)$, where $L = \sum_{\phi_i \in \Phi}$ *length*(ϕ_i) is a bound on the number of subformulas we generate. Furthermore, we can bound the size of G_{ES}. Since there are at most 2^L atoms over $ls(s)$, each base state can receive at most this number of distinct labels. Thus G_{ES} can be *at most* this factor larger than G_S (although Phase II does not usually generate all conceivable labels.) The exponential here may seem discouraging, but there are simple, natural examples in which this number of historical distinctions is *required* for implementing an optimal policy. For instance, for the reward formula $\ominus^n P$, we need to keep track of when P was true among the previous n steps, leading to 2^n distinct annotations.

Nevertheless, the main disadvantage of $ls_{\mathrm{sub}}(\cdot)$ is that it can lead to unnecessarily fine distinctions among histories, so that G_{ES} as produced by Phase II is not guaranteed to be minimal (in the sense of having as few states as possible among valid expansions of G_S). If minimality is important, a separate step after Phase II is required. Fortunately, minimizing G_{ES} can be performed using a variant of standard algorithms for minimizing finite state automata [HU79]. We defer discussion to the full paper, but note that the complexity of doing this is only polynomial in the size of G_{ES}. Thus, so long as the intermediate G_{ES} produced by Phase II is of manageable size, minimization is fairly straightforward.[6]

A second implementation of Phase I constructs label sets ls_{w} with "weaker" formulas, subject to the stated requirements. More precisely, we initially set $ls_{\mathrm{w}}(s) = \Phi$, for all s. Then, so long as we can find s, s', such that s' is reachable from s and $\{\mathrm{Regr}(\phi, s') : \phi \in ls_{\mathrm{w}}(s')\} \not\subseteq ls_{\mathrm{w}}(s)$, we add $\{\mathrm{Regr}(\phi, s') : \phi \in ls_{\mathrm{w}}(s')\}$ to $ls_{\mathrm{w}}(s)$. We iterate until this terminates—which it will, so long as we are careful not to add different (but logically equivalent) formulas twice to $ls_{\mathrm{w}}(s)$. This procedure ensures the necessary properties of $ls(\cdot)$. For many natural examples of reward formula, this process terminates quickly, generating small label sets.

The major reason for considering $ls_{\mathrm{w}}(\cdot)$ is that G_{ES}, as constructed subsequently by Phase II, is then guaranteed to have minimal size. But $ls_{\mathrm{w}}(\cdot)$ has a serious drawback as well: Phase I can *potentially* become very complex. The number of iterations until termination can be exponential (in the size of the reward formulas) and the size of the label sets can grow double-exponentially. Perhaps the optimal strategy, then, is to begin to implement Phase I using $ls_{\mathrm{w}}(\cdot)$, but if any reward

[6] If G_{ES} is *much* larger than necessary, Phase II's complexity could cause difficulties.

formula proves troublesome, to then revert to the subformula technique at that point.

We conclude by noting that Phase II is, in comparison, unproblematic. Since each extended state is visited exactly once the complexity is linear in the size of the final answer (i.e., the size of G_{ES}.) Furthermore, none of the operations performed in Phase II are difficult. Steps 1.a and 2.a.i appear to involve theorem-proving, but this is misleading. Step 1.a is actually just *model checking* (over what is, furthermore, a very short trajectory) and in this particular case can be done in time proportional to $\sum_{\phi \in \mathrm{ls}(s)} \mathit{length}(\phi)$. Step 2.a.i can also be performed quickly; the details depend on exactly how Phase I is implemented, but in general (and in particular, for the two proposals discussed above) enough book-keeping information can be recorded during Phase I so that 2.a.i can be performed in time proportional to $|\mathit{ls}(s)|$. Again, space limitations prevent us from providing the details.

In conclusion, the annotation algorithm appears to be quite practical. The potential exists for exponential work (relative to the size of G_S) but this is generally the case exactly when we really do need to store a lot of history (i.e., when G_{ES} is necessarily large).

5 Concluding Remarks

While MDPs provide a useful framework for DTP, some of the necessary assumptions can be quite restrictive (at the very least, requiring that some planning problems be encoded in an unnatural way). We have presented a technique that weakens the impact of one of these assumptions, namely, the requirement of Markovian (or state-based) reward. The main contributions of this work are a methodology for the natural specification of temporally extended rewards, and an algorithm that automatically constructs an equivalent MDP, allowing standard MDP solution techniques to be used to construct optimal policies.

There are a number of interesting directions in which this work can be extended. First, similar techniques can be used to cope with non-Markovian dynamics, and can also be used with partially-observable processes. In addition, other temporal logics (such as more standard forward-looking logics) and process logics can potentially be used in a similar fashion to specify different classes of behaviors.

Another interesting idea is to use compact representations of MDPs to obviate the need for computation involving individual states. For instance, Bayes net representations have been used to specify actions for MDPs in [BDG95], and can be exploited in policy construction. Given an NMRDP specified in this way, we could produce new Bayes net action descriptions involving an expanded set of variables (or propositions) that render the underlying reward process Markovian, rather than expanding states explicitly.

Finally, our technique does not work well if the expanded MDP is large, which may be the case if a lot of history is necessary (note that this is inherent in formulating such a problem as an MDP, whether automatically constructed or not). The complexity of policy construction is typically dominated by the size of the state space. An important direction for future work is to combine policy construction with state space expansion. The hope is that one can avoid generating many expanded states using dominance arguments particular to the reward structure of the given NMRDP.

Acknowledgements

The work of Fahiem Bacchus and Craig Boutilier was supported by the Canadian government through their NSERC and IRIS programs.

We thank the anonymous referees for their thoughtful reviews. It is unfortunate that space limitations prevented us from responding to several of their valuable suggestions.

References

[AH90] R. Alur and T. Henzinger. Real-time logics: complexity and expressiveness. *LICS-90*, Philadelphia, 1990.

[BD94] C. Boutilier and R. Dearden. Using abstractions for decision-theoretic planning with time constraints. *AAAI-94*, pp.1016–1022, Seattle, 1994.

[BDG95] C. Boutilier, R. Dearden, and M. Goldszmidt. Exploiting structure in policy construction. *IJCAI-95*, pp.1104–1111, Montreal, 1995.

[Bel57] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[BP95] C. Boutilier and M. L. Puterman. Process-oriented planning and average-reward optimality. *IJCAI-95*, pp.1096–1103, Montreal, 1995.

[DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Comp. Intel.*, 5:142–150, 1989.

[DKKN93] T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson. Planning with deadlines in stochastic domains. *AAAI-93*, pp.574–579, Washington, D.C., 1993.

[Dru89] M. Drummond. Situated control rules. *KR-89*, pp.103–113, Toronto, 1989.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, ed., *Handbook Theor. Comp. Sci., Vol.B*, pp.997–1072, 1990.

[GK91] P. Godefroid and F. Kabanza. An efficient reactive planner for synthesizing reactive plans. *AAAI-91*, pp.640–645, 1991.

[HH92] P. Haddawy and S. Hanks. Representations for decision-theoretic planning: Utility functions for deadline goals. *KR-92*, pp.71–82, Cambridge, 1992.

[How60] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[Kab90] F. Kabanza. Synthesis of reactive plans for multi-path environments. *AAAI-90*, pp.164–169, 1990.

[KHW94] N. Kushmerick, S. Hanks and D. Weld. An algorithm for probabilistic least-commitment planning. *AAAI-94*, pp.1073–1078, Seattle, 1994.

[LDK95] M. Littman, T. L. Dean and L. P. Kaelbling. On the complexity of solving Markov decision problems. *UAI-95*, pp.394–402, Montreal, 1995.

[Put94] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.

[Sch87] M. J. Schoppers. Universal plans for reactive robots in unpredictable environments. *IJCAI-87*, 1039–1046, Milan, 1987.

[TR94] J. Tash and S. Russell. Control strategies for a stochastic planner. *AAAI-94*, 1079–1085, Seattle, 1994.