A Model-Checking Approach to Decision-Theoretic Planning with Non-Markovian Rewards

Sylvie Thiébaux[1], Froduald Kabanza[2], John Slaney[1]

Abstract. A popular approach to solving a decision process with non-Markovian rewards (NMRDP) is to exploit a compact representation of the reward function to automatically translate the NMRDP into an equivalent Markov decision process (MDP) amenable to our favorite MDP solution method. The contribution of this paper is a representation of non-Markovian reward functions and a translation into MDP aimed at making the best possible use of state-based anytime algorithms as the solution method. By explicitly constructing and exploring only parts of the state space, these algorithms are able to trade computation time for policy quality, and have proven quite effective in dealing with large MDPs. Our representation extends future linear temporal logic to express rewards. Our translation has the effect of embedding model-checking in the solution method and results in an MDP of the minimal size achievable without stepping outside the anytime framework.

1 Introduction

Markov decision processes (MDPs) are now widely accepted as the preferred model for decision-theoretic planning problems [6]. The fundamental assumption behind the MDP formulation is that not only the system dynamics but also the reward function are *Markovian*. Therefore, all information needed to determine the reward at a given state must be encoded in the state itself.

This requirement is not always easy to meet for planning problems, as many desirable behaviors are naturally expressed as properties of execution *sequences*, see e.g. [9, 11, 3, 17]. Typical cases include rewards for the maintenance of some property, for the periodic achievement of some goal, for the achievement of a goal within a given number of steps of the request being made, or even simply for the very first achievement of a goal which becomes irrelevant afterwards. A decision process in which rewards depend on the sequence of states passed through rather than merely on the current state is called a decision process with *non-Markovian rewards* (NMRDP) [1].

A difficulty with NMRDPs is that the most efficient MDP solution methods do not directly apply. The traditional way to circumvent this problem is to formulate the NMRDP as an equivalent MDP, whose states are those of the underlying system expanded to encode enough history-dependent information to determine the rewards. Hand crafting such an *expanded* MDP (XMDP) can however be very difficult in general. This is exacerbated by the fact that the size of the XMDP limits the applicability of many solution methods. Therefore, there has been interest in automating the translation into an XMDP, starting from a natural specification of non-Markovian rewards and of the system's dynamics [1, 2]. This is the problem we focus on.

When solving NMRDPs in this setting, the central issue is to define a non-Markovian reward specification language and a translation into an XMDP *adapted* to the class of MDP solution methods and representations we would like to use for the type of problems at hand. The two previous proposals within this line of research both rely on past linear temporal logic (PLTL) formulae to specify the behaviors to be rewarded, but target two very different types of solution methods and representations, and consequently adopt two very different translations. The first proposal, [1], targets classical *state-based* solution methods such as policy iteration [13]. These generate *complete* policies at the cost of enumerating all states in the entire MDP. Since they are extremely sensitive to the size of the MDP, attention is paid to translating the NMRDP into a XMDP of *minimal* size. To achieve minimality, the translation not only requires a pre-processing step enumerating the entire state space (this in itself is less damaging than it may appear in the context of the targeted solution methods, since they operate on the entire expanded structure in any case), but this pre-processing step requires double exponential space and a number of iterations (through the state space) exponential in the size of the reward formulae. The second proposal, [2], targets *structured* solution methods and representations, which do not require explicit state enumeration, see e.g. [7]. Accordingly, the translation keeps the expanded state space implicit, and amounts to adding temporal variables to the problem representation, together with the decision trees describing their dynamics. The translation is very efficient but rather crude. In particular, the encoded history features do not even vary from one state to the next, which strongly compromises the minimality of the XMDP. Again, this is in line with the targeted solution methods: because structured solution methods always keep the expanded state space implicit and are able to ignore some of the irrelevant variables, non-minimality is not as problematic as with the state-based approaches.

The aim of the present paper is provide a language and a translation adapted to another class of solution methods which have proven quite effective in dealing with large MDPs, namely *anytime* state-based methods such as [5, 8, 19, 12]. These typically start with a compact representation of the MDP based on probabilistic planning operators, and search forward from an initial state, constructing new states by expanding the envelope of the policy as time permits. They may produce an approximate and even incomplete policy, but only explicitly construct and explore a fraction of the MDP. Neither of the two previous proposals is well-suited to such solution methods, the first because the costly pre-processing phase annihilates the benefits

[1] Computer Sciences Laboratory The Australian National University Canberra, ACT 0200, Australia. email: {Sylvie.Thiebaux, John.Slaney}@anu.edu.au

[2] Département Mathématiques et Informatique, Université de Sherbrooke Sherbrooke, Québec, Canada. email: kabanza@dmi.usherb.ca

of anytime algorithms, and the second because the size of the XMDP obtained is an obstacle to the applicability of state-based methods.

Our approach has the following main features. The construction of the XMDP resulting from the translation is entirely embedded in the anytime solution method, to which full control is given as to which parts of the XMDP will be explicitly constructed. While the XMDP obtained is not minimal, it is of the minimal size achievable without stepping outside of the anytime framework, i.e., without requiring a pre-processing phase enumerating parts of the state or expanded state spaces. This relaxed notion of minimality, which we call *blind minimality* is the most appropriate in the context of anytime state-based solution methods. When rewarding behaviors are specified in PLTL, there does not appear to be a way of achieving a relaxed notion of minimality as powerful as blind minimality without a costly pre-processing phase. Therefore instead of PLTL, we adopt, unlike previous approaches, a variant of *future* linear temporal logic (FLTL) as our specification language, which we extend to handle rewards. While the language has a more complex semantics than PLTL, it enables a simple translation into a blind-minimal XMDP whose expanded states are labeled by formulae resulting from successive *progressions* of the reward formulae. Moreover, search control knowledge expressed in FLTL [4] fits particularly nicely in this model-checking framework, and can be used to dramatically reduce the fraction of the search space explored by anytime solution methods.

The paper is organised as follows. In Section 2, we begin with background material on MDPs, NMRDPs, XMDPs, and anytime state-based solution methods. In Section 3, we give the syntax and semantics of our language for specifying non-Markovian rewards, as well as a progression algorithm for it. In section 4, we define our translation into an XMDP along with the concept of blind minimality it achieves, and present our approach to the embedded construction and solution of the XMDP. Finally, in Section 5, we provide a detailed comparison with previous approaches, and conclude with some remarks about future work. Due to lack of space, we omit the proofs which appear in the technical report [20].

2 Background

2.1 MDPs

A Markov decision process of the type we consider consists of: (1) a finite set S of states which are fully observable, (2) an initial state[3] $s_0 \in S$, (3) a finite set A of actions – $A(s)$ denotes the subset of actions applicable in $s \in S$, (4) a family $\{T_{(a,s)} \mid s \in S, a \in A(s)\}$ of probability distributions over S, such that $T_{(a,s)}(s')$ is the probability of being in state s' after performing action a in state s, and (5) a reward function $R : S \mapsto \mathbb{R}$, such that $R(s)$ is the immediate reward for being in state s. It is well known that such an MDP can be compactly represented using probabilistic extensions of traditional planning languages, see e.g., [15, 19].

A stationary policy for an MDP is a function $\pi : S \mapsto A$, such that $\pi(s) \in A(s)$ is the action to be executed in state S. We note $E(\pi)$ the envelope of the policy, that is the set of states that are reachable (with a non-zero probability) from the initial state s_0 under the policy. If π is defined at all $s \in E(\pi)$, we say that the policy is complete, and that it is incomplete otherwise. We note $F(\pi)$ the set of states in $E(\pi)$ at which π is undefined. $F(\pi)$ is called the fringe of the policy. We stipulate that the fringe states are absorbing. The value of a policy π at any state $s \in E(\pi)$, noted $V_\pi(s)$ is the sum of the expected rewards to be received at each future time step, discounted by how far into the future they occur. That is, for a non-fringe state $s \in E(\pi) \setminus F(\pi)$:

$$V_\pi(s) = R(s) + \beta \sum_{s' \in E(\pi)} T_{(\pi(s),s)}(s') V_\pi(s')$$

where $0 \leq \beta \leq 1$ is the discounting factor controlling the contribution of distant rewards. For a fringe state $s \in F(\pi)$, $V_\pi(s)$ is heuristic or is the value at s of a complete default policy to be executed in absence of an explicit one.[4] For the type of MDP we consider, the value of a policy π is the value $V_\pi(s_0)$ of π at the initial state s_0, and the larger this value, the better the policy. This value can be computed in cubic time in the cardinality of $E(\pi)$.

2.2 State-based Anytime Algorithms

Traditional state-based solution methods such as policy iteration [13] can be used to produce an optimal complete policy. Policy iteration can also be viewed as an anytime algorithm, which returns a complete policy whose value increases with computation time and converges to optimal. The main drawback of policy iteration is that it explicitly enumerates all states that are reachable from s_0 in the entire MDP. Therefore, there has been interest in other anytime solution methods, which may produce incomplete policies, but only enumerate an increasing fraction of the states policy iteration requires.

For instance, [8] describes methods which deploy policy iteration on judiciously chosen larger and larger envelopes. Another example is [19], in which a backtracking forward search in the space of (possibly incomplete) policies rooted at s_0 is performed until interrupted, at which point the best policy found so far is returned. Real-time dynamic programming (RTDP) [5], is another popular anytime algorithm, which is to MDPs what learning real-time A* [14] is to deterministic domains. It can be run on-line, or off-line for a given number of steps or until interrupted. A more recent example is the LAO* algorithm [12] which combines dynamic programming with heuristic search.

All these algorithms eventually converge to the optimal policy but need not necessarily explore the entire state space to guarantee optimality[5]. When interrupted before convergence, they return a possibly incomplete but often useful policy. Another common point of these approaches is that they perform a forward search, starting from s_0 and repeatedly expanding the envelope of the current policy one step forward. Since planning operators are used to compactly represent the state space, these methods will only explicitly construct a subset of the MDP. In this paper, we will use these solution methods to solve decision processes with non-Markovian rewards which we define next.

2.3 NMRDPs and Equivalent XMDPs

Let S^* be the set of finite sequences of states over S, and S^ω be the set of possibly infinite state sequences. In the following, where 'Γ'

[3] The generalization to a probability distribution on initial states is straightforward.

[4] For instance, if the default policy is to wait and stay indefinitely in the current fringe state, we will take $V_\pi(s) = \frac{R(s)}{1-\beta}$.

[5] This is also true of the basic envelope expansion algorithm in [8], under the same conditions as for LAO*. Indeed, LAO* can be seen as a clever implementation of a particular case of the envelope expansion algorithm, where fringe states are given admissible heuristic values, where policy iteration is run up to convergence between envelope expansions, and where the clever implementation only runs policy iteration on the states whose optimal value can actually be affected when a new state is added to the envelope.

stands for a possibly infinite state sequence in S^ω and i is a natural number, by 'Γ_i' we mean the state of index i in Γ, by '$\Gamma(i)$' we mean the prefix $\langle \Gamma_0, \ldots, \Gamma_i \rangle \in S^*$ of Γ, and by '${}^i\Gamma$' we mean the suffix $\Gamma - \Gamma(i-1) \in S^\omega$. $\Gamma_1;\Gamma_2$ denotes the concatenation of $\Gamma_1 \in S^*$ and $\Gamma_2 \in S^\omega$. By $\mathsf{pre}(\Gamma)$ we mean the set of finite prefixes of Γ. For a decision process $D = \langle S, -, A, T, - \rangle$ and a state $s \in S$, $\tilde{D}(s)$ stands for the set of state sequences rooted at s that are possible under the actions in D, that is: $\tilde{D}(s) = \{\Gamma \in S^\omega \mid \Gamma_0 = s \text{ and } \forall i\ \exists a \in A\ \ T_{(a,\Gamma_i)}(\Gamma_{i+1}) > 0\}$

A decision process with non-Markovian rewards is identical to an MDP except that the domain of the reward function is S^*. The idea is that if the process has passed through state sequence $\Gamma(i)$ up to stage i, then the reward $R(\Gamma(i))$ is received at stage i. Like the reward function, a policy for an NMRDP depends on history, and is a mapping from S^* to A. As before, the value of policy π is the expectation of the discounted cumulative reward over an infinite horizon:

$$V_\pi(s_0) = \lim_{n\to\infty} \mathsf{E}\left[\sum_{i=0}^{n} \beta^i R(\Gamma(i)) \mid \pi, \Gamma_0 = s_0\right]$$

The clever algorithms developed to solve MDPs cannot be directly applied to NMRDPs. One way of dealing with this problem is to formulate the NMRDP as an equivalent MDP with an expanded state space [1, 2]. The expanded states in this XMDP augment the states of the NMRDP by encoding additional information sufficient to make the reward history-independent (i.e., a function from expanded states to the reals). An expanded state can be seen as labeled by a state of the NMRDP (via the function τ below) and by history information. The dynamics of NMRDPs being Markovian, the actions and their probabilistic effects in the XMDP are exactly those of the NMRDP. The following definition, taken from [1], makes this concept of equivalent XMDP precise.

Definition 1 *An MDP $D' = \langle S', s'_0, A', T', R' \rangle$ is an equivalent expansion (or XMDP) for an NMRDP $D = \langle S, s_0, A, T, R \rangle$ if there exists a function $\tau : S' \mapsto S$ such that:*

1. *$\tau(s'_0) = s_0$*
2. *$\forall s' \in S'$, $A'(s') = A(\tau(s'))$.*
3. *$\forall s_1, s_2 \in S$, $s'_1 \in S'$ and $a \in A$, if $T_{(a,s_1)}(s_2) = p > 0$ and $\tau(s'_1) = s_1$, then there is a unique s'_2, $\tau(s'_2) = s_2$, such that $T_{(a,s'_1)}(s'_2) = p$.*
4. *For any feasible state sequence $\Gamma \in \tilde{D}(s_0)$ and $\Gamma' \in \tilde{D}'(s'_0)$ such that $\tau(\Gamma'_i) = \Gamma_i$ for all i, we have: $R'(\Gamma'_i) = R(\Gamma(i))$ for all i.*

By solving an equivalent XMDP, provided we are able to produce one, we get a stationary policy that can be reinterpreted as a non-stationary policy for the NMRDP. Furthermore, it is shown in [1] that the two policies have identical values, and that consequently, solving an NMRDP optimally reduces to solving an equivalent XMDP optimally:

Proposition 1 *Let D be an NMRDP, D' an equivalent XMDP for it, and π' a policy for D'. Let π be the function defined on the sequence prefixes $\Gamma(i) \in \tilde{D}(s_0)$ by $\pi(\Gamma(i)) = \pi'(\Gamma'_i)$, where for all $j \leq i$ $\tau(\Gamma'_j) = \Gamma_j$. Then π is a policy for D such that $V_\pi(s_0) = V_{\pi'}(s'_0)$.*

When solving NMRDPs in this setting, the two key issues are how to specify the non-Markovian reward function compactly, and how to exploit this compact representation to automatically translate the NMRDP into an equivalent XMDP amenable to our favorite solution methods. The purpose of this paper is to provide a reward function specification language and a translation that are adapted to the anytime state-based solution methods previously mentioned. We take these problems in turn in the next two sections.

3 Rewarding Behaviors

3.1 Language and Semantics

Representing non-Markovian reward functions compactly reduces to compactly representing the behaviors of interest, where by *behavior* we mean a set of finite sequences of states (a subset of S^*). To do so, we adopt a version of future linear temporal logic (FLTL) augmented with a propositional constant '\$', intended to be read 'The reward is received now'. The language \$FLTL begins with a set of basic propositions $\mathcal{P}$ giving rise to literals:

$$\mathcal{L} ::= \mathcal{P} \mid \neg\mathcal{P} \mid \top \mid \bot \mid \$$$

The connectives are classical $\wedge$ and $\vee$, and the temporal modalities $\bigcirc$ (next) and U (*weak* until), giving formulae:

$$\mathcal{F} ::= \mathcal{L} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid \bigcirc\mathcal{F} \mid \mathcal{F}\,\mathsf{U}\,\mathcal{F}$$

Because our 'until' is weak ($f_1 \,\mathsf{U}\, f_2$ means that f_1 will be true from now on until f_2 is, if ever), we can define the useful operator (always): $\quad f \equiv f\,\mathsf{U}\,\bot$ (f will always be true from now on). We also adopt the notations $\bigcirc^k f$ for k iterations of the $\bigcirc$ modality (f will be true in exactly k steps), $\bigcirc^{\leq k} f$ for the disjunction of $\bigcirc^i$ for $1 \leq i \leq k$ (f will be true within the next k steps), and $\bigcirc_{\leq k} f$ for the conjunction of $\bigcirc^i f$ for $1 \leq i \leq k$ (f will be true at all the next k steps).

Although negation officially occurs only in literals, we allow ourselves to write formulae involving it in the usual way, provided that they have an equivalent in negation normal form. Not every formula has such an equivalent, because there is no such literal as $\neg$\$ and because eventualities ('f will eventually be true some time') are not expressible. These restrictions in expressivity are deliberate.

To specify the semantics of this language, we first define the valuation function $[\,.\,]_B$ which intuitively assigns to each literal L the set of sequence prefixes $\Gamma(i)$ such that L is true in Γ_i, assuming B is the behavior being rewarded:

- $[\,p\,]_B = \{\Gamma(i) \mid p \in \Gamma_i\}$ for $p \in \mathcal{P}$
- $[\,\neg p\,]_B = \{\Gamma(i) \mid p \not\in \Gamma_i\}$ for $p \in \mathcal{P}$
- $[\,\top\,]_B = S^*$
- $[\,\bot\,]_B = \emptyset$
- $[\,\$\,]_B = B$

Given this, we now define what it is for a formula to hold at stage i of sequence Γ for behavior B:

- $\Gamma, i \models_B L$ iff $\Gamma(i) \in [\,L\,]_B$ for $L \in \mathcal{L}$
- $\Gamma, i \models_B f_1 \wedge f_2$ iff $\Gamma, i \models_B f_1$ and $\Gamma, i \models_B f_2$
- $\Gamma, i \models_B f_1 \vee f_2$ iff $\Gamma, i \models_B f_1$ or $\Gamma, i \models_B f_2$
- $\Gamma, i \models_B \bigcirc f$ iff $\Gamma, i+1 \models_B f$
- $\Gamma, i \models_B f_1 \,\mathsf{U}\, f_2$ iff $\forall k \geq i\ (\Gamma, k \models_B f_1$ or $\exists i \leq j \leq k\ (\Gamma, j \models_B f_2))$

As a special case, we may say $\Gamma \models_B f$ iff $\Gamma, 0 \models_B f$. We also say $\Gamma \models f$ iff $\Gamma \models_B f$ for every $B \subseteq S^*$. Thus for \$-free formulae, the modelling relation $\models$ reduces to the usual one for FLTL semantics. Analogously, we say $\models_B f$ iff $\Gamma \models_B f$ for every $\Gamma \in S^\omega$. Note monotonicity: if $B \subseteq B'$ and $\Gamma, i \models_B f$ then $\Gamma, i \models_{B'} f$.

We define the behavior B_f which is rewarded according to f by:

Definition 2 $B_f \equiv \bigcap\{B \mid \models_B f\}$[6]

[6] If there is no B such that $\models_B f$, which is the case for any \$-free f which is not a logical theorem, then B_f is $\bigcap\emptyset$ – i.e. S^*. This limiting case is a little artificial, but since \$-free formulae do not describe the attribution of rewards, it does no harm.

That is, B_f will only reward a prefix if it is forced to because that prefix is in *every* behavior satisfying f. The reason for this ‘stingy’ semantics, making rewards minimal, is that f does not actually say that rewards are allocated to more prefixes than are required for its truth. For instance, let f be $p \rightarrow \$$. This *says* only that a reward is given if p is true, even though a more generous distribution of rewards would be *consistent* with it.

3.2 Examples

It is intuitively clear that many behaviors can be specified by means of $FLTL formulae. There is a list in [1] of behaviors expressible in PLTL which it might be useful to reward. All of those examples are expressible naturally in $FLTL, as follows.

A simple example is the classical goal formula G saying that a goal g is rewarded whenever it happens: $\quad(g \rightarrow \$)$. It is easy to see that B_G is the set of finite sequences of states such that g holds in the last state. If we only care that g is achieved once and get rewarded at each state from then on, we write $\quad(g \rightarrow \quad \$)$. By contrast, the formula $\neg g \,\mathsf{U}\, (g \wedge \$)$ stipulates that only the first occurrence of g is rewarded. To reward the occurrence of g at most once every k steps, we write $\quad((\bigcirc^{k+1} g \wedge \neg\bigcirc^{\leq k} g) \rightarrow \bigcirc^{k+1}\$)$.

Goal sequences are expressed very similarly as in PLTL. For instance, to be rewarded for achieving g_1, followed immediately by g_2, we write $\quad((g_1 \wedge \bigcirc g_2) \rightarrow \bigcirc\$)$. By taking $g_1 \wedge \bigcirc g_2$ as a compound goal g, we can express rewards for the periodic achievement of this goal or for its very first achievement, similarly as before. E.g, for the latter, we write $\neg(g_1 \wedge \bigcirc g_2) \,\mathsf{U}\, (g_1 \wedge \bigcirc g_2 \wedge \bigcirc\$)$.

For response formulae, where the achievement of g is triggered by the command c, we write $\quad(c \rightarrow \bigcirc \quad (g \rightarrow \$))$ to reward every state in which g is true following the first issue of the command. To reward only the first occurrence g after each command, we write $\quad(c \rightarrow \bigcirc(\neg g \,\mathsf{U}\, (g \wedge \$)))$. As for bounded variants for which we only reward goal achievement within k steps of the command, we write $\quad(c \rightarrow \bigcirc_{\leq k}(g \rightarrow \$))$ or $\quad(c \rightarrow \bigcirc^{\leq k} g \rightarrow (\neg g \,\mathsf{U}\, (g \wedge \$)))$ depending on whether all states or just the first state in which g is true is rewarded.

3.3 Determinacy, stability and normality

$FLTL is so expressive that it is possible to write formulae which describe “unnatural” allocations of rewards. For instance, they may make rewards depend on future behaviors rather than on the past, or they may leave open a choice as to which of several behaviors is to be rewarded. An example of the former is $\bigcirc p \rightarrow \$$, which stipulates a reward *now* if p is going to hold *next*. An example of the latter is $\quad(p \rightarrow \$) \vee \quad(\neg p \rightarrow \$)$ which says we should *either* reward all achievements of the goal p *or* reward achievements of the goal $\neg p$ but does not determine which. The semantic treatment of $FLTL specified above gives us the necessary concepts to make the intuitive distinction between natural and unnatural reward formulae precise.

The first desirable property of formulae is *reward-determinacy*.

Definition 3 *f is* reward-determinate *iff* $\models_{B_f} f$.

What reward-determinacy amounts to is that the set of behaviors modelling f has a unique minimum. As noted above, a typical reward-indeterminate formula is $\quad(p \rightarrow \$) \vee \quad(\neg p \rightarrow \$)$ which has two minimal satisfying behaviors, corresponding to the two disjuncts. B_f, their intersection, however, is the null set (nothing gets rewarded) which is not one of the behaviors sufficing to satisfy the formula.

Note that reward-determinate formulae must be consistent, since they are modeled by at least one behavior. Note also that in view of monotonicity, f is reward-determinate iff given any Γ, all supersets of $B_f \cap \mathsf{pre}(\Gamma)$ suffice for f:

$$\forall B \forall \Gamma (B_f \cap \mathsf{pre}(\Gamma) \subseteq B \;\Rightarrow\; \Gamma \models_B f)$$

The second desirable property, *reward-stability*, is the converse of reward-determinacy: that *only* supersets of $B_f \cap \mathsf{pre}(\Gamma)$ suffice to make Γ model f:

Definition 4 *f is* reward-stable *iff*

$$\forall B \forall \Gamma (\Gamma \models_B f \;\Rightarrow\; B_f \cap \mathsf{pre}(\Gamma) \subseteq B)$$

Intuitively, a reward-stable formula does not make present rewards depend on future states. The paradigm example of a reward-unstable formula is $f = \bigcirc p \rightarrow \$$ (reward now if p next). B_f turns out to be $\{\langle s \rangle \mid s \in S^*\}$, the set of singleton-sequences of states, meaning that the first state of any sequence should be rewarded. However, if for some Γ it so happens that $p \notin \Gamma_1$: then Γ satisfies f without the need to reward Γ_0, meaning that the distribution of rewards to prefixes according to f is not invariant across the sequences that extend those prefixes.

The ‘natural’ formulae satisfy both desirability criteria, which is the normal case:

Definition 5 *F is* reward-normal *iff it is both reward-determinate and reward-stable.*

The behavior described by a reward-normal formula is not too big and not too small, but just right to ensure that the formula holds. This is captured by the following theorem:

Theorem 1 *Let the* consequences *in Γ of formula f be those formulae g such that for every behavior B such that $\Gamma \models_B f$, $\Gamma \models_B g$. Let the* outcomes *in Γ of behavior B be those g such that $\Gamma \models_B g$. Then for any Γ:*

1. *If f is reward-determinate, then every consequence in Γ of f is an outcome in Γ of B_f.*
2. *If f is reward-stable, then every outcome in Γ of B_f is a consequence in Γ of f.*
3. *If f is reward-normal, then its consequences in Γ are exactly the outcomes in Γ of B_f.*

While reward-abnormal formulae may be interesting, and we do not wish to close off all investigation into them, for present purposes we restrict attention to reward-normal ones. Indeed, we claim (without proof) that for every formula f there exists a reward-normal f' such that $B_f = B_{f'}$. Naturally, all the formulae given as examples in Section 3.2 are reward-normal.

3.4 FLTL Formula Progression

Having defined a language to represent behaviors to be rewarded, we now turn to the problem of computing, given a reward formula, a minimum allocation of rewards to states actually encountered in an execution sequence in such a way as to satisfy the formula. Because we ultimately wish to use anytime solution methods which generate state sequences incrementally via forward search, this computation is best done on the fly, while the sequence is being generated. We therefore adapt an incremental model-checking technique normally used to check whether a state sequence is a model of an FLTL formula (see e.g [3]). This technique is called formula *progression*, because it progresses the formula through the sequence. It is particularly useful when states in the sequence become available one at a time, e.g. when the sequence is generated by a forward chaining planner, because it defers the evaluation of the part of the formula that refers to the future to the point where the next state becomes available. We first review the principles behind FLTL progression and then extend it to deal with rewards.

Let Γ_i be a state (say, the last state of the sequence prefix $\Gamma(i)$ that has been generated so far). The progression of a formula f through Γ_i, written $\mathrm{Progr}(\Gamma_i, f)$, is a new formula which exhibits the following property: f holds at Γ_i iff $\mathrm{Progr}(\Gamma_i, f)$ holds at the next (yet unavailable) state Γ_{i+1} in the sequence. The function Progr is computable in time linear in the length of f.

To check whether Γ is a model of a formula f_0, we first compute $f_1 = \mathrm{Progr}(\Gamma_0, f_0)$, then when Γ_1 becomes available, we compute $f_2 = \mathrm{Progr}(\Gamma_1, f_1)$, and so on. At any stage, $f_{i+1} = \mathrm{Progr}(\Gamma_i, f_i)$ can be either $\top$, $\perp$, or a non-trivial formula. If $f_{i+1} = \top$, this means that $\Gamma(i)$ already satisfies f_0, and so any extension of this prefix will. If $f_{i+1} = \perp$, $\Gamma(i)$ already violates f_0 and so any extension of this prefix will. If f_{i+1} is a non-trivial formula, some extensions of the prefix may satisfy f_0 and some may not.

The TLplan forward chaining planner [4] uses FLTL to specify domain-specific *search control knowledge* and formula progression to prune from the search space any sequence prefix violating this knowledge. This can be done as soon as the progression through the last state of the prefix returns $\perp$, and has been shown to provide enormous time gains. For instance, the effect of the control knowledge formula $(p \rightarrow \bigcirc q)$ is to prune from the search space any prefix in which p is not followed by q. TALplanner [16] won the hand-tailored track at the 2000 planning competition by making use of a similar model-checking approach to exploit control knowledge.

3.5 \$FLTL Formula Progression

We may use a variant of progression to assign rewards to sequence prefixes as follows. First, given a desired behavior G, choose a formula f_0 of \$FLTL to represent G; f_0 should be a formula such that $B_{f_0} = G$.[7] What we wish to reward is the behavior G restricted to Γ: that is $G \cap \mathsf{pre}(\Gamma)$. Provided f_0 is reward-normal, this is the minimal behavior B such that $\Gamma \models_B f_0$.The technique is therefore to let progression define a sequence $\langle f_0, f_1, \ldots\rangle$ of formulae, where each f_i is required to hold at the corresponding state Γ_i and to be such that the minimum behavior ensuring that it does is the result of truncating sequences in B on the left (removing their first i states).

The definition of Progr is much as for FLTL, except that at each state we have a choice as to whether \$ progresses to $\top$ or to $\perp$, and to ensure minimality it progresses to $\perp$ whenever possible. That is, there are three progression functions Progr, Progr^+ and Progr^- as follows:

- $\mathrm{Progr}^+(s, \$) = \top$
- $\mathrm{Progr}^-(s, \$) = \perp$
- for $p \in \mathcal{P}$, $\mathrm{Progr}^+(s,p) = \mathrm{Progr}^-(s,p) = \top$ iff $p \in s$ and $\perp$ otherwise
- $\mathrm{Progr}^+(s, \neg f) = \neg\mathrm{Progr}^+(s, f)$
- $\mathrm{Progr}^-(s, \neg f) = \neg\mathrm{Progr}^-(s, f)$
- $\mathrm{Progr}^+(s, f_1 \wedge f_2) = \mathrm{Progr}^+(s, f_1) \wedge \mathrm{Progr}^+(s, f_2)$
- $\mathrm{Progr}^-(s, f_1 \wedge f_2) = \mathrm{Progr}^-(s, f_1) \wedge \mathrm{Progr}^-(s, f_2)$
- $\mathrm{Progr}^+(s, f_1 \vee f_2) = \mathrm{Progr}^+(s, f_1) \vee \mathrm{Progr}^+(s, f_2)$
- $\mathrm{Progr}^-(s, f_1 \vee f_2) = \mathrm{Progr}^-(s, f_1) \vee \mathrm{Progr}^-(s, f_2)$
- $\mathrm{Progr}^+(s, \bigcirc f) = \mathrm{Progr}^-(s, \bigcirc f) = f$
- $\mathrm{Progr}^+(s, f_1 \mathrm{U} f_2) = \mathrm{Progr}^+(s, f_2) \vee (\mathrm{Progr}^+(s, f_1) \wedge f_1 \,\mathsf{U}\, f_2)$
- $\mathrm{Progr}^-(s, f_1 \mathrm{U} f_2) = \mathrm{Progr}^-(s, f_2) \vee (\mathrm{Progr}^-(s, f_1) \wedge f_1 \,\mathsf{U}\, f_2)$
- $\mathrm{Progr}(s, f) = \mathrm{Progr}^+(s, f)$ if $\mathrm{Progr}^-(s, f) = \perp$ and $\mathrm{Progr}^-(s, f)$ otherwise

[7] Since there are more behaviors than formulae, it may be that no such formula exists. In that case, temporal logic (with past or future operators) is not the right vehicle for representing G: but *every* formalism suffers from *that* sort of limitation.

The function Progr so defined generates a sequence $\langle f_0, f_1 \ldots\rangle$ of formulae satisfying the requirement stated above. This fundamental property of \$FLTL progression is captured in the following theorem. Where B is a behavior, let $\mathsf{tail}(B) = \{{}^1\beta \mid \beta \in B\}$ be B with the first state of each sequence removed, then:

Theorem 2 ${}^i\Gamma \models_B f$ *iff* ${}^{i+1}\Gamma \models_{\mathsf{tail}(B)} \mathrm{Progr}(\Gamma_i, f)$

If some reward formula f_i eventually progresses to $\perp$, something must have gone wrong: at some point, \$ was made false where it should have been made true. The usual explanation is that the original f_0 was not reward-normal, though other (admittedly bizarre) possibilities exist: for example, although $\$ \vee \bigcirc p$ is reward-abnormal, and will progress to $\perp$ in the next state if p is false there, its substitution instance $\$ \vee \bigcirc\bigcirc\perp$, which also goes to $\perp$ in a few steps, is logically equivalent to \$ and is reward-normal. If the progression method is to deliver the correct minimal behavior in all cases (even in all reward-normal cases) it has to backtrack. In the interests of efficiency, we choose not to allow backtracking. Instead, our algorithm raises an exception whenever a reward formula progresses to $\perp$, and informs the user of the sequence which caused the problem. The onus is thus placed on the domain modeller to select sensible reward formulae so as avoid possible progression to $\perp$.

The following theorem states that under weak assumptions, rewards are correctly allocated by progressing the formulae representing them.

Theorem 3 *Let f_0 be reward-normal, and let $\langle f_0, f_1, \ldots\rangle$ be the result of progressing it through the successive states of a sequence Γ. Then, provided no f_i is $\perp$, $\forall i$ $\mathrm{Progr}^-(\Gamma_i, f_i) = \perp$ iff $\Gamma(i) \in B_{f_0}$.*

If backtracking were allowed in the progression algorithm, the condition that no f_i is $\perp$ would become vacuous and could be dropped from the statement of the theorem. As it is, we retain it to rule out the cases in which exceptions are raised.

3.6 Reward Functions

With the language defined so far, we are able to compactly represent behaviors. The extension to a non-Markovian reward function is straightforward. We represent such a function by a set $\Phi \subseteq$ \$FLTL $\times\, \mathbb{R}$ of formulae associated with real valued rewards. We call Φ a *reward function specification*. Where formula f is associated with reward r in Φ, we write '$(f : r) \in \Phi$'. The rewards are assumed to be independent and additive, so that the reward function R_Φ represented by Φ is given by:

Definition 6 $R_\Phi(\Gamma(i)) = \sum_{(f:r)\in\Phi} \{r \mid \Gamma(i) \in B_f\}$

Again, we can progress a reward function specification Φ_0 to compute the reward at all stages i of Γ. As before, progression defines a sequence $\langle \Phi_0, \Phi_1, \ldots \rangle$ of reward function specifications, where $\Phi_{i+1} = \text{SProgr}(\Gamma_i, \Phi_i)$, and where SProgr is the function that applies Progr to all formulae in a reward function specification:

$$\text{SProgr}(s, \Phi) = \{(\text{Progr}(s, f) : r) \mid (f : r) \in \Phi\}$$

Then, the total reward received at stage i is simply the sum of the real-valued rewards granted by the progression function to the behaviors picked up by the formulae in Φ_i:[8]

$$\sum_{(f:r)\in\Phi_i} \{r \mid \text{Progr}^-(\Gamma_i, f) = \bot\}$$

By proceeding that way, we get the expected analog of Theorem 3, which states that progression correctly computes non-Markovian reward functions.

Theorem 4 *Let Φ_0 be a reward-normal*[9] *reward function specification, and let $\langle \Phi_0, \Phi_1 \ldots \rangle$ be the sequence of reward function specifications obtained by progressing Φ_0 through the successive states of a sequence Γ. Then, provided $(\bot : r) \notin \Phi_i$ for any i, then* $\sum_{(f:r)\in\Phi_i} \{r \mid \text{Progr}^-(\Gamma_i, f) = \bot\} = R_{\Phi_0}(\Gamma(i))$.

4 Solving NMRDPs

4.1 Translation into XMDP

We now exploit the compact representation of a non-Markovian reward function as a reward function specification to translate an NMRDP into an equivalent XMDP amenable by state-based anytime solution methods. Recall from Section 2.3 that each expanded state (*e-states* for short) is labeled by a state of the NMRDP and by history information sufficient to determine the immediate reward. In the case of a compact representation as a reward function specification Φ_0, this additional information can be summarized by the progression of Φ_0 through the sequence of states passed through to the current one. So an e-state will be of the form $\langle s, \Phi \rangle$, where $s \in S$ is a state, and $\Phi \subseteq \$\text{FLTL} \times \mathbb{R}$ is a reward function specification (obtained by progression). Two e-states are equal iff they are labeled by the same s and by semantically equivalent Φs. More formally:

Definition 7 *Let $D = \langle S, s_0, A, T, R \rangle$ be an NMRDP, and Φ_0 be a reward function specification representing R (i.e., $R_{\Phi_0} = R$, see Definition 6). We translate D into the XMDP $D' = \langle S', s'_0, A', T', R' \rangle$ defined as follows:*

1. $S' \subseteq S \times 2^{\$FLTL \times \mathbb{R}}$
2. $s'_0 = \langle s_0, \Phi_0 \rangle$
3. $a \in A'(\langle s, \Phi \rangle) \equiv a \in A(s)$
4. *If* $a \in A'(\langle s, \Phi \rangle)$, *then* $T'_{(a,\langle s,\Phi\rangle)}(\langle s', \Phi' \rangle) =$

$$\begin{array}{ll} T_{(a,s)}(s') & \text{if } \Phi' = \mathit{SProgr}(s, \Phi) \\ 0 & \textit{otherwise} \end{array}$$

If $a \notin A'(\langle s, \Phi \rangle)$, *then* $T'_{(a,\langle s,\Phi\rangle)}$ *is undefined.*

5. $R'(\langle s, \Phi \rangle) = \sum_{(f:r)\in\Phi} \{r \mid \text{Progr}^-(s, f) = \bot\}$

Item 1 says that the e-states are labeled by a state and a reward function specification. Item 2 says that the initial e-state is labeled with the initial state and with the original reward function specification. Imte 3 says that an action is applicable in an e-state if it is applicable in the state labeling it. Item 4 explains how successor e-states are and their probabilities are computed. Given an action a applicable in an e-state $\langle s, \Phi \rangle$, each successor e-state will be labeled by a successor state s' of s via a in the NMRDP and by the progression of Φ through s. The probability of that e-state is $T_{(a,s)}(s')$ as in the NMRDP. Note that the cost of computing T' is linear in that of computing T and in the sum of the lengths of the formulae in Φ. Item 5 has been motivated before (see Section 3.6).

It is not difficult to show that this translation lead to an equivalent XMDP, as defined in Definition 1:

Theorem 5 *Let D' be the translation of D according to Definition 7. D' is an equivalent XMDP for D.*

4.2 Blind Minimality

The size of the XMDP obtained, i.e. the number of e-states it contains is a key issue for us, as it has to be amenable to state-based solution methods. Ideally, we would like the XMDP to be of minimal size. Unfortunately, we do not know of a method building the minimal equivalent XMDP incrementally, and since in the worst case it can be larger than the NMRDP by a factor exponential in the length of the reward formulae, building it entirely would nullify the interest of anytime solution methods.

However, Definition 7 leads to an equivalent XMDP exhibiting a relaxed notion of minimality, as we now explain. By inspection, we may observe that wherever an e-state $\langle s, \Phi \rangle$ has a successor $\langle s', \Phi' \rangle$ via action a, this means that in order to succeed in rewarding the behaviors described in Φ by means of execution sequences that start by going from s to s' via a, it is necessary that the future starting with s' succeeds in rewarding the behaviors described in Φ'. If $\langle s, \Phi \rangle$ is in the minimal equivalent XMDP, and if there really are such execution sequences succeeding in rewarding the behaviors described in Φ, then $\langle s', \Phi' \rangle$ must also be in the minimal XMDP. That is, construction by progression exhibits a relaxed notion of minimality, in that it can only introduce e-states which are *a priori* needed. Note that an e-state that is *a priori* needed may not *really* be needed: there may in fact be no execution sequence using the available actions that exhibits a given behavior. For instance, consider the response formula $(p \rightarrow \bigcirc^k q \rightarrow \bigcirc^k \$)$, i.e., every time command p is issued, we will be rewarded k steps later provided q is true then. Obviously, whether p is true at some stage affects the way future states should be rewarded. However, if k steps from there a state satisfying q can never be reached, then a posteriori p is irrelevant, and there was no need to label e-states with differently according to whether p was true or not. To detect such cases, we would have to look perhaps quite deep into possible futures. Hence the relaxed notion which we call *blind minimality* does not always coincide with absolute minimality.

We now formalise the difference between true and blind minimality. To simplify notation (avoiding functions like the τ of Definition 1), we represent each e-state as a pair $\langle s, r \rangle$ where $s \in S$ and r is a function from S^* to $\mathbb{R}$ intuitively assigning rewards to sequences in the NMRDP starting from s. A given s may be paired with several functions r corresponding to relevantly different histories of s. The

[8] Recall that a reward if only granted if not granting it falsifies the reward formula.

[9] We extend the definition of reward-normality to reward specification functions the obvious way, by requiring that all reward formulae involved be reward normal.

XMDP is minimal if every such r is *needed* to distinguish between reward patterns in the *feasible* futures of s:

Theorem 6 *Let S' be the set of e-states in a minimal equivalent XMDP D' for $D = \langle S, s_0, A, T, R\rangle$. Then for each e-state $\langle s, r\rangle \in S'$ there exists a sequence prefix $\Gamma(i) \in \tilde{D}(s_0)$ such that $\Gamma_i = s$ and $\forall\Delta \in S^*$:*

$$r(\Delta) = \begin{array}{ll} R(\Gamma(i-1);\Delta) & \textit{if } \Delta \in \tilde{D}(s) \\ 0 & \textit{otherwise} \end{array}$$

Blind minimality is similar, except that, since there is no looking ahead, no distinction can be drawn between feasible trajectories and others in the future of s:

Definition 8 *Let S' be the set of e-states in an equivalent XMDP D' for $D = \langle S, s_0, A, T, R\rangle$. D' is* blind minimal *iff for each e-state $\langle s, r\rangle \in S'$ there exists a sequence prefix $\Gamma(i) \in \tilde{D}(s_0)$ such that $\Gamma_i = s$ and $\forall\Delta \in S^*$:*

$$r(\Delta) = \begin{array}{ll} R(\Gamma(i-1);\Delta) & \textit{if } \Delta_0 = s \\ 0 & \textit{otherwise} \end{array}$$

Theorem 7 *Let D' be the translation of D according to Definition 7. D' is a blind minimal equivalent XMDP for D.*

4.3 XMDP Embedded Solution/Construction

Blind minimality is essentially the best achievable with anytime state-based solution methods which typically extend their envelope one step forward without looking deeper into the future. Our translation into a blind-minimal XMDP can be trivially embedded in any of these solution methods. This will result in an 'on-line construction' of the XMDP: the method will entirely drive the construction of those parts of the XMDP which it feels the need to explore, and leave the others implicit. If time is short, a suboptimal or even incomplete policy may be returned, but only a fraction of the state and expanded state spaces will be constructed. Note that the solution method should raise an exception as soon as one of the reward formulae progresses to $\perp$, i.e., as soon as an expanded state $\langle s, \Phi\rangle$ is built such that $(\perp : r) \in \Phi$, since this acts as a detector of unsuitable reward function specifications.

To the extent enabled by blind minimality, our approach allows for a dynamic analysis of the reward formulae, much as in [2]. Indeed, only the execution sequences realisable under a particular policy actually explored by the solution method contributes to the analysis of rewards for that policy. Specifically, the reward formulae generated by progression for a given policy are determined by the prefixes of the execution sequences realisable under this policy. This dynamic analysis is particularly useful, since relevance of reward formulae to particular policies (e.g. the optimal policy) cannot be detected a priori.

Because our approach is based on progression, it provides an elegant way to exploit search control knowledge. This may result in a dramatic reduction of the fraction of the XMDP to be constructed and explored, and therefore in substantially better policies by the deadline. To use control knowledge, we simply progress the control formula alongside the reward function specification, making e-states triples $\langle s, \Phi, c\rangle$ where $c \in$ FLTL is the control-knowledge. To prevent the solution method to apply an action that leads to the control knowledge being violated, the action applicability condition (item 3 of the translation) becomes: $a \in A'(\langle s, \Phi, c\rangle)$ iff $a \in A(s)$ and $c \neq \perp$. The other changes are straightforward. Although this paper focuses on non-Markovian rewards rather than dynamics, it should be noted that FLTL formulae can also be used to express non-Markovian constraints on the system's dynamics, which can be incorporated in our approach exactly as we do for the control knowledge.

5 Related and Future Work

It is evident that our thinking about solving NMRDPs and the use of temporal logic to represent them draws on [1]. Both this paper and [2] advocate the use of PLTL over a finite past to specify non-Markovian rewards. In the PLTL style of specification, we describe the past conditions under which we get rewarded now, while with \$FLTL we describe the conditions on the present and future under which future states will be rewarded. While the behaviors and rewards may be the same in each scheme, the naturalness of thinking in one style or the other depends on the case. Letting the kids have a strawberry dessert because they have been good all day fits naturally into a past-oriented account of rewards, whereas promising that they may watch a movie if they tidy their room (indeed, making sense of the whole notion of promising) goes more naturally with FLTL. One advantage of the PLTL formulation is that it trivially enforces the principle that present rewards do not depend on future states. In \$FLTL, this responsibility is placed on the domain modeller. Since deciding whether a formula is reward-normal is as hard as theorem proving for FLTL, all we can offer is an exception mechanism to recognise mistakes when their effects appear. On the other hand, the greater expressive power of \$FLTL[10] opens the possibility of considering a richer class of decision processes, e.g. with uncertainty as to which rewards are received (the dessert or the movie) and when (some time this week-end, before Mum comes back). This is a topic for future work. At any rate, as we now explain, \$FLTL is better suited than PLTL to solving NMRDPs using anytime state-based solution methods.

[1] proposes a method whereby an e-state is labeled by a set of subformulae of the PLTL reward formulae. For the labeling, two extreme cases are considered: one very simple and the other elaborate. In the simple case, an e-state is labeled by the set of all subformulae which are true at it. The computation of such simple labels can be done forward starting from the initial state, and so could be embedded in an anytime solution method. However, because the structure of the original reward formulae is lost when considering subformulae, fine distinctions between histories are drawn which are totally irrelevant to the reward function. Consequently, the expanded state space easily becomes exponentially bigger than the blind-minimal one. This is problematic with the solution methods we consider (including with real-time search), because their performance in solution quality is severely affected by the size of the expanded state space.

In the elaborate case, a pre-processing phase uses regression to find sets of subformulae as potential labels for possible predecessor states, so that the subsequent generation phase builds an XMDP representing all and only the histories which make a difference to the way *actually possible* execution sequences should be rewarded. The XMDP produced is minimal, and so in the best case exponentially smaller than the blind-minimal one. However, the prohibitive cost of the pre-processing phase makes it unusable for anytime solution methods. We do not consider that any method based on PLTL and regression will achieve a meaningful relaxed notion of minimality

[10] Many extensions of the language are straightforward: adding eventualities, unrestricted negation, first-class reward propositions, quantitative time, etc. Of course, dealing with them via progression without backtracking is another matter.

without a costly pre-processing phase. Our main contribution is an approach based on FLTL and progression which does precisely that, letting the solution method resolve the tradeoff between quality and cost in a principled way intermediate between the two extreme suggestions above.

In another sense, too, our work also represents a middle way, combining the advantages conferred by state-based and structured approaches, e.g. by [1] on one side, and [2] on the other. From the former we inherit a meaningful notion of minimality. As with the latter, approximate solution methods can be used and can perform a restricted dynamic analysis of the reward formulae. The type of solution methods targeted by [2] is different from ours. In virtue of the size of the expanded state space produced, the translation proposed in [2] is clearly usuitable to anytime state-based methods. The question of the appropriateness of our translation to structured solution methods, however, cannot be settled as clearly. On the one hand, our approach does not preclude the exploitation of a structured representation of *system's states*[11], and formula progression enables even state-based methods to exploit *some* of the structure in '$FLTL space'. On the other hand, the gap between blind and true minimality indicates that progression alone is insufficient to always *fully* exploit that latter structure (reachability is not exploited). With our translation, even structured solution methods will not remedy this. There is a hope that [2] is able to take advantage of the full structure of the reward function, but also a possibility that it will fail to exploit even as much structure as our approach would, as efficiently.

The most important item for future work is an empirical comparison of the three approaches to answer this question and to identify the domain features favoring one over the other. Ours has been fully implemented as an extension (rewards, probabilities) of TLplan's planning language, which includes functions and bounded quantification [4]. To allow for comparisons to be reported in a longuer version of this paper, we are in the process of implementing the other two approaches, no implementation of either of them being reported in the literature. Another exciting area for future work is the investigation of temporal logic formalisms for specifying heuristics for NMRDPs or more generally for search problems with temporally extended goals, as good heuristics are important to some of the solution methods we are targeting. Related to this is the problem of extending, to temporally extended goals, the GOALP predicate [4] which is the key to the specification of reusable control knowledge in the case of reachability goals. Finally, we should investigate the precise relationship between our line of work and recent work on planning for weak temporally extended goals in non-deterministic domains, such as attempted reachability and maintenance goals [18].

References

[1] F. Bacchus, C. Boutilier, and A. Grove, 'Rewarding behaviors', in *Proc. AAAI-96*, pp. 1160–1167, (1996).

[2] F. Bacchus, C. Boutilier, and A. Grove, 'Structured solution methods for non-markovian decision processes', in *Proc. AAAI-97*, pp. 112–117, (1997).

[3] F. Bacchus and F. Kabanza, 'Planning for temporally extended goals', *Annals of Mathematics and Artificial Intelligence*, **22**, 5–27, (1998).

[4] F. Bacchus and F. Kabanza, 'Using temporal logic to express search control knowledge for planning', *Artificial Intelligence*, **116**(1-2), (2000).

[5] A.G. Barto, S.L. Bardtke, and S.P. Singh, 'Learning to act using real-time dynamic programming', *Artificial Intelligence*, **72**, 81–138, (1995).

[6] C. Boutilier, T. Dean, and S. Hanks, 'Decision-theoretic planning: Structural assumptions and computational leverage', in *Journal of Artificial Intelligence Research*, volume 11, pp. 1–94, (1999).

[7] C. Boutilier, R. Dearden, and M. Goldszmidt, 'Stochastic dynamic programming with factored representations', *Artificial Intelligence*, **121**(1-2), 49–107, (2000).

[8] T. Dean, L. Kaelbling, J. Kirman, and A. Nicholson, 'Planning under time constraints in stochastic domains', *Artificial Intelligence*, **76**, 35–74, (1995).

[9] M. Drummond, 'Situated control rules', in *Proc. KR-89*, pp. 103–113, (1989).

[10] Z. Feng and E. Hansen, 'Symbolic LAO* search for factored markov decision processes', in *Proc. AIPS-02 Workshop on Planning via Model-Checking*, (2002). To appear.

[11] P. Haddawy and S. Hanks, 'Representations for decision-theoretic planning: Utility functions and deadline goals', in *Proc. KR-92*, pp. 71–82, (1992).

[12] E. Hansen and S. Zilberstein, 'LAO*: A heuristic search algorithm that finds solutions with loops', *Artificial Intelligence*, **129**, 35–62, (2001).

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

[14] R. Korf, 'Real-time heuristic search', *Artificial Intelligence*, **42**, 189–211, (1990).

[15] N. Kushmerick, S. Hanks, and D. Weld, 'An algorithm for probabilistic planning', *Artificial Intelligence*, **76**, 239–286, (1995).

[16] J. Kvarnström, P. Doherty, and P. Haslum, 'Extending TALplanner with concurrency and ressources', in *Proc. ECAI-00*, pp. 501–505, (2000).

[17] M. Pistore and P. Traverso, 'Planning as model-checking for extended goals in non-deterministic domains', in *Proc. IJCAI-01*, pp. 479–484, (2001).

[18] M. Pistore and P. Traverso, 'Planning with a language for extended goals', in *Proc. AAAI-02*, (2002). To appear.

[19] S. Thiébaux, J. Hertzberg, W. Shoaff, and M. Schneider, 'A stochastic model of actions and plans for anytime planning under uncertainty', *International Journal of Intelligent Systems*, **10**(2), 155–183, (1995).

[20] S. Thiébaux, F. Kabanza, and J. Slaney, 'A model-checking approach to decision-theoretic planning with non-markovian rewards', Technical report, The Australian National University, Computer Sciences Laboratory, (2002). http://csl.anu.edu.au/∼thiebaux/papers/nmr.ps.gz.

[11] Symbolic implementations of the solution methods we consider, e.g. [10], as well as formula progression in the context of symbolic state representations [17] could be investigated for that purpose.