LTL-Constrained Steady-State Policy Synthesis

Jan Křetínský
Technical University of Munich
jan.kretinsky@tum.de

Abstract

Decision-making policies for agents are often synthesized with the constraint that a formal specification of behaviour is satisfied. Here we focus on infinite-horizon properties. On the one hand, Linear Temporal Logic (LTL) is a popular example of a formalism for qualitative specifications. On the other hand, Steady-State Policy Synthesis (SSPS) has recently received considerable attention as it provides a more quantitative and more behavioural perspective on specifications, in terms of the frequency with which states are visited. Finally, rewards provide a classic framework for quantitative properties. In this paper, we study Markov decision processes (MDP) with the specification combining all these three types. The derived policy maximizes the reward among all policies ensuring the LTL specification with the given probability and adhering to the steady-state constraints. To this end, we provide a unified solution reducing the multi-type specification to a multi-dimensional long-run average reward. This is enabled by Limit-Deterministic Büchi Automata (LDBA), recently studied in the context of LTL model checking on MDP, and allows for an elegant solution through a simple linear programme. The algorithm also extends to the general ω-regular properties and runs in time polynomial in the sizes of the MDP as well as the LDBA.

1 Introduction

Markov decision processes (MDP), e.g. [Puterman, 1994], are a basic model for agents operating in uncertain environments since it features both non-determinism and stochasticity. An important class of problems is to automatically synthesize a policy that resolves the non-determinism in such a way that a given formal specification of some type is satisfied on the induced Markov chain. Here we focus on the classic formalisms capable of specifying properties over *infinite* horizon.

Infinite-horizon properties. On the one hand, *Linear Temporal Logic* (LTL) [Pnueli, 1977] is a popular formalism for qualitative specifications, capable of expressing complex temporal relationships, abstracting from the concrete quantitative timing, e.g., *after every request there is a grant* (not saying when exactly). It has found applications in verification of programs [Baier and Katoen, 2008] as well as high-level robot control, e.g. [Kress-Gazit *et al.*, 2009], or preference-based planning in PDDL [Benton *et al.*, 2012] to name a few.

On the other hand, *Steady-State Control* (SSC, a.k.a. Steady-State Policy Synthesis) [Akshay *et al.*, 2013] constrains the frequency with which states are visited, providing a more quantitative and more behavioural perspective (in terms of states of the system, as opposed to logic-based or reward-based specifications). Recently, it has started receiving more attention also in AI planning [Velasquez, 2019; Atia *et al.*, 2020], improving the theoretical complexity and its applicability to a wider class of MDP (although still being quite restrictive on the class of policies, see below).

Finally, rewards provide a classic framework for quantitative properties. In the setting of infinite horizon, a key role is played by the *long-run average reward* (LRA), e.g. [Puterman, 1994], which constrains the reward gained on average per step and, over the decades, it has found numerous applications [Feinberg and Shwartz, 2012].

Our contribution. In this paper, we study Markov decision processes (MDP) with the specifications *combining all these three types*, yielding a more balanced and holistic perspective. We synthesize a policy maximizing the LRA reward among all policies ensuring the LTL specification (with the given probability) and adhering to the steady-state constraints. To this end, we provide a *unified solution* conceptually reducing the problem with the heterogeneous specification combining three types of properties to a problem with a single-type specification, namely the multi-dimensional LRA reward. This in turn can be solved using a single simple linear programme, which is easy to understand and shorter than previous solutions to the special cases considered in the literature.

Not only does the unified approach allow us to use the powerful results from literature on rewards, but it also allows for easy extensions and modifications such as considering various classes of policies (depending on the memory available or further requirements on the induced chain) or variants of the objectives (e.g. constraints on frequency of satisfying recurrent goals, multiple rewards, trade-offs between objectives), which we also describe.

Our reduction is particularly elegant due to the use of Limit-Deterministic Büchi Automata (LDBA), recently studied in the context of LTL model checking on MDP. Moreover, the solution extends trivially from LTL to the general ω-regular properties (given as automata) and runs in time polynomial in the sizes of the MDP as well as the LDBA.

In summary, our contribution is as follows:

- We introduce the heterogeneous LTL-SSC-LRA specification for maximizing the reward under the LTL and steady-state constraints.
- We provide a unified solution framework via recent results on LDBA automata and on multi-dimensional LRA optimization.
- The resulting solution is generic, as documented on a number of extensions and variants of the problem we discuss, as well as simple, generating a linear programme with a structure close to classic reward optimization.

1.1 Related Work

The *steady-state control* was introduced in [Akshay *et al.*, 2013], treating the case of recurrent MDP and showing the problem is in PSPACE by quadratic programming. It is combined with LRA reward maximization, giving rise to *steady-state policy synthesis*, in [Velasquez, 2019]. A polynomial-time solution is provided via linear programming, even for general (multi-chain) MDP, but restricting to stationary policies inducing recurrent Markov chains. In [Atia *et al.*, 2020], general (multi-chain) MDP and a wider class of policies are considered. However, the "EP" policies prohibit playing in some non-bottom end components, which makes stationary policies optimal within this class, but is very restrictive in contrast to general strategies. In this work, we add also the LTL specification and we have neither restriction on the MDP nor on the class of strategies considered and chains induced. Our linear programme is simpler and extends the correspondence between solutions and the policies with their steady-state distributions to the general ones (with no recurrence assumed). On the other hand, we also treat the restricted settings of finite-memory, recurrence, and further modifications to show the versatility of our approach.

The main idea of our approach is an underlying conceptual reduction to *multi-dimensional LRA rewards* allowing for a simple linear programme. The classic linear-programming solution for a single LRA reward [Puterman, 1994] has been extended to various settings such as multi-dimensional reachability and ω-regular properties [Etessami *et al.*, 2008] or multi-dimensional LRA rewards [Brázdil *et al.*, 2014; Chatterjee *et al.*, 2017]. Some of these results [Brázdil *et al.*, 2014] are also implemented [Brázdil *et al.*, 2015] on top of the PRISM model checker [Kwiatkowska *et al.*, 2011]. Rewards have also been combined with LTL on finite paths in [Brafman *et al.*, 2018; Giacomo *et al.*, 2019]. Our results strongly rely particularly on [Chatterjee *et al.*, 2017]; however, that work treats neither SSC nor LTL.

The other element simplifying our work is the *limit-deterministic Büchi automaton* [Courcoubetis and Yannakakis, 1995]. It has been shown to be usable for LTL model checking on MDP under some conditions [Hahn *et al.*, 2015; Sickert *et al.*, 2016], these conditions are satisfied by an efficient translation from LTL [Sickert *et al.*, 2016]. Tools for the translation [Křetínský *et al.*, 2018] as well as the model-checking [Sickert and Křetínský, 2016] are also available.

2 Preliminaries

2.1 Basic Definitions

We use $\mathbb{N}, \mathbb{Q}, \mathbb{R}$ to denote the sets of positive integers, rational and real numbers, respectively. The Kronecker function $\mathbf{1}_x(y)$ yields 1 if $x = y$ and 0 otherwise. The set of all distributions over a countable set X is denoted by $Dist(X)$. A distribution $d \in Dist(X)$ is Dirac on $x \in X$ if $d = \mathbf{1}_x$.

Markov chains. A *Markov chain* (MC) is a tuple $\mathcal{M} = (L, P, \mu, \nu)$ where L is a countable set of locations, $P : L \to Dist(L)$ is a probabilistic transition function, $\mu \in Dist(L)$ is the initial probability distribution, and $\nu : L \to 2^{Ap}$ is a labelling function.

A *run* in $\mathcal{M}$ is an infinite sequence $\rho = \ell_1 \ell_2 \cdots$ of locations, a *path* in $\mathcal{M}$ is a finite prefix of a run. Each path w in $\mathcal{M}$ determines the set $\mathsf{Cone}(w)$ consisting of all runs that start with w. To $\mathcal{M}$ we associate the probability space $(\mathsf{Runs}, \mathcal{F}, \mathbb{P})$, where Runs is the set of all runs in $\mathcal{M}$, $\mathcal{F}$ is the σ-field generated by all $\mathsf{Cone}(w)$, and $\mathbb{P}$ is the unique probability measure such that $\mathbb{P}(\mathsf{Cone}(\ell_1 \cdots \ell_k)) = \mu(\ell_1) \cdot \prod_{i=1}^{k-1} P(\ell_i)(\ell_{i+1})$.

Markov decision processes. A *Markov decision process* (MDP) is a tuple $\mathcal{G} = (S, A, Act, \delta, \hat{s}, \nu)$ where S is a finite set of states, A is a finite set of actions, $Act : S \to 2^A \setminus \{\emptyset\}$ assigns to each state s the set $Act(s)$ of actions enabled in s so that $\{Act(s) \mid s \in S\}$ is a partitioning of A, $\delta : A \to Dist(S)$ is a probabilistic transition function that given an action a gives a probability distribution over the successor states, $\hat{s}$ is the initial state, and $\nu : S \to 2^{Ap}$ is a labelling function. Note that every action is enabled in exactly one state (w.l.o.g. by renaming).

A *run* in $\mathcal{G}$ is an infinite alternating sequence of states and actions $\rho = s_1 a_1 s_2 a_2 \cdots$ such that for all $i \geq 1$, we have $a_i \in Act(s_i)$ and $\delta(a_i)(s_{i+1}) > 0$. A *path* of length k in $\mathcal{G}$ is a finite prefix $w = s_1 a_1 \cdots a_{k-1} s_k$ of a run in $\mathcal{G}$.

Policies and plays. Intuitively, a policy (a.k.a. strategy) in an MDP is a "recipe" to choose actions. A policy is formally defined as a function $\sigma : (SA)^* S \to Dist(A)$ that given a finite path w, representing the history of a play, gives a probability distribution over the actions enabled in the last state of w.

A *play* of $\mathcal{G}$ determined by a policy σ is a Markov chain $\mathcal{G}^\sigma$ where the set of locations is the set of paths $S(AS)^*$, the valuation depends on the current state only $\nu(ws) = \nu(s)$, the initial distribution is Dirac on $\hat{s}$, and transition function P is defined by $P(w)(was) = \sigma(w)(a) \cdot \delta(a)(s)$. Hence, $\mathcal{G}^\sigma$ starts in the initial state of the MDP and actions are chosen according to σ evaluated on the current history of the play, and the next state is chosen according to the transition function evaluated on the chosen action. The induced probability measure is denoted by $\mathbb{P}^\sigma$ and "almost surely" ("a.s.") or "almost all runs" refers to happening with probability 1 according to this measure. The respective expected value of a random variable $F : \mathsf{Runs} \to \mathbb{R}$ is $\mathbb{E}^\sigma[F] = \int_{\mathsf{Runs}} F \, d\,\mathbb{P}^\sigma$. For $t \in \mathbb{N}$, random variables S_t, A_t return the t-th state and action on the run.

A policy is *memoryless* if it only depends on the last state of the current history. A policy is *bounded-memory* (or *finite-memory*) if it stores only finite information about the history; technically, it can written down using a finite-state machine, as detailed in Appendix A.

End components. A set $T \cup B$ with $\emptyset \neq T \subseteq S$ and $B \subseteq \bigcup_{t \in T} Act(t)$ is an *end component* of $\mathcal{G}$ if (1) for all $a \in B$, whenever $\delta(a)(s') > 0$ then $s' \in T$; and (2) for all $s, t \in T$ there is a path $w = s_1 a_1 \cdots a_{k-1} s_k$ such that $s_1 = s$, $s_k = t$, and all states and actions that appear in w belong to T and B, respectively. An end component $T \cup B$ is a *maximal end component (MEC)* if it is maximal with respect to the subset ordering. Given an MDP, the set of MECs is denoted by MEC and can be computed in polynomial time [Courcoubetis and Yannakakis, 1995].

2.2 Specifications

In order to define our problem, we first recall definitions of the considered classes of properties.

LTL and ω-regular properties. *Linear Temporal Logic* (LTL) [Pnueli, 1977] over the finite set Ap of atomic propositions is given by the syntax

$$\varphi ::= p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi\mathbf{U}\varphi \qquad p \in Ap$$

and is interpreted over a run $\rho = s_1 s_2 \cdots$ of $\mathcal{M}$ by

$\rho \models p$ if $p \in L(s_1)$

$\rho \models \mathbf{X}\varphi$ if $s_2 s_3 \cdots \models \varphi$

$\rho \models \varphi\mathbf{U}\psi$ if $\exists k : s_k s_{k+1} \cdots \models \psi, \forall j < k : s_j s_{j+1} \cdots \models \varphi$

and the usual semantics of Boolean connectives. The event that φ is satisfied is denoted by $\models \varphi$. Given an MDP $\mathcal{G}$ and a policy σ, $\mathbb{P}^\sigma[\models \varphi]$ then denotes the probability that φ is satisfied on $\mathcal{G}^\sigma$.

A *Büchi Automaton* (BA) is a tuple $\mathcal{B} = (\Sigma, Q, \Delta, q_0, F)$ where Σ is a finite alphabet, Q is a finite set of states, $\Delta \colon Q \times \Sigma \to 2^Q$ is a transition function, writing $q \overset{\alpha}{\to} r$ for $r \in \Delta(q, \alpha)$, q_0 is the initial state, and $F \subseteq Q$ is the set of accepting states. $\mathcal{B}$ *accepts* the language $\mathsf{L}(\mathcal{B}) = \{\alpha_1\alpha_2 \cdots \in \Sigma^\omega \mid \exists q_0 \overset{\alpha_1}{\to} q_1 \overset{\alpha_2}{\to} q_2 \cdots : q_i \in F \text{ for infinitely many } i\}$ of infinite words that can pass through F infinitely often.

A BA $\mathcal{B}$ is *limit-deterministic* (LDBA) [Sickert *et al.*, 2016] if Q can be partitioned $Q = Q_\mathcal{N} \uplus Q_\mathcal{D}$ so that

1. $\Delta(q, \alpha) \subseteq Q_\mathcal{D}$ and $|\Delta(q, \alpha)| = 1$ for $q \in Q_\mathcal{D}$, $\alpha \in \Sigma$,
2. $F \subseteq Q_\mathcal{D}$, and
3. $|\Delta(q, \alpha) \cap Q_\mathcal{N}| = 1$ for $q \in Q_\mathcal{N}$, $\alpha \in \Sigma$.

Intuitively, a BA is LDBA if it can be split into a non-deterministic part without accepting transitions and a deterministic component, where it has to remain forever but may now accept. The third constraint is less standard and not essential, but eases the argumentation. Intuitively, it implies that the only choice that is made on a run is when and where to do the single non-deterministic "jump" from $Q_\mathcal{N}$ to $Q_\mathcal{D}$ [Sickert *et al.*, 2016].

Every LTL formula can be translated into an equivalent BA $BA(\varphi)$ in exponential time and into LDBA $LDBA(\varphi)$ in double exponential time [Sickert *et al.*, 2016], so that both automata are over the alphabet 2^{Ap} and accept exactly the language $\mathsf{L}(\varphi)$ of words satisfying φ.[1]

A set is an ω-regular language if it can be written as $\mathsf{L}(\mathcal{B})$ for some BA (or equivalently LDBA) $\mathcal{B}$. The standard algorithm for LTL model checking on MDP, i.e. to decide whether $\exists\sigma : \mathbb{P}^\sigma[\models \varphi] \geq p$, is to translate it to the more general problem of model checking ω-regular specification $\exists\sigma : \mathbb{P}^\sigma[\nu(\rho) \in \mathsf{L}(\mathcal{A})] \geq p$ for some automaton $\mathcal{A}$ accepting $\mathsf{L}(\varphi)$. Typically, $\mathcal{A}$ is deterministic with a more complex acceptance condition [Baier and Katoen, 2008], but also LDBA (with the simple Büchi condition of visiting F infinitely often) can be used if they satisfy a certain condition, ensured by the recent translation from LTL [Sickert *et al.*, 2016]. Then the problem can be solved by combining the MDP $\mathcal{G}$ and the automaton $\mathcal{B} := LDBA(\varphi)$ into the *product* $\mathcal{G} \times \mathcal{B}$, where the automaton "monitors" the run of the MDP, and its subsequent analysis. The product of $\mathcal{G} = (S, A, Act, \delta, \hat{s}, \nu)$ and $\mathcal{B} = (2^{Ap}, Q, \Delta, q_0, F)$ is the MDP $\mathcal{G} \times \mathcal{B} = (S \times Q, A', Act', \delta', \langle \hat{s}, q_0 \rangle, \nu')$ where

- $A' = A \times Q \times Q$ ensuring uniqueness of actions, the elements are written as $a_{q \to r}$, intuitively combining a with $q \to r$;
- $Act'(\langle s, q \rangle) = \{a_{q \to r} \mid a \in Act(s), q \overset{\nu(s)}{\to} r\}$
- $\delta'(\langle s, q \rangle, a_{q \to r})(\langle t, r' \rangle) = \delta(s, a)(t) \cdot \mathbf{1}_r(r')$
- $\nu'(\langle s, q \rangle) = \nu(s)$

Note that indexing the actions preserves their uniqueness. A state $\langle s, q \rangle$ of the product is *accepting* if $q \in F$. Observe that a run of the original MDP satisfies φ iff there is a choice of the jump such that the corresponding run of the product visits accepting states infinitely often. A MEC which contains an accepting state is called an *accepting MEC* and the set of accepting MECs is denoted by AMEC. Observe that once in AMEC, any policy that uses all actions of the current MEC (and none that leave it) ensures visiting all its states hence also acceptance. Consequently, $\max_\sigma \mathbb{P}^\sigma[\models \varphi] = \max_\sigma \mathbb{P}^\sigma[\bigcup \mathsf{AMEC} \text{ is reached}]$ by [Sickert *et al.*, 2016].

Steady-state constraints. The classic way of constraining the steady state is to require that $\pi(s) \in [\ell, u]$ where $\pi(s)$ is the steady-state distribution applied to the state s and $[\ell, u] \subseteq [0, 1]$ is the constraining interval. We slightly generalize the definition in two directions: Firstly, we can constrain (not necessarily disjoint) *sets* of states, to match the ability of the logical specification. Secondly, we consider general cases where the steady-state distribution does not exist, to cater for general policies (in contrast to restricted classes of policies investigated in previous works), which may cause such behaviour. Formally, a *steady-state specification* (SSS) is a set $\mathcal{S}$ of steady-state constraints of the form $(p, \ell, u) \in Ap \times [0, 1] \times [0, 1]$. A Markov chain $\mathcal{M}$ satisfies $\mathcal{S}$, written $\mathcal{M} \models \mathcal{S}$ if for each $(p, \ell, u) \in \mathcal{S}$ we have

$$\ell \leq \pi_{\inf}(p) := \liminf_{T \to \infty} \tfrac{1}{T} \textstyle\sum_{t=1}^{T} \mathbb{P}[p \in \nu(S_t)]$$
$$\pi_{\sup}(p) := \limsup_{T \to \infty} \tfrac{1}{T} \textstyle\sum_{t=1}^{T} \mathbb{P}[p \in \nu(S_t)] \leq u$$

[1] Technically, $\mathsf{L}(\varphi)$ is the set such that for any run $\rho = s_1 s_2 \cdots$ with $\nu(s_i) = \alpha_i$, we have $\rho \models \varphi$ iff $\alpha_1\alpha_2 \cdots \in \mathsf{L}(\varphi)$.

Intuitively, we measure the frequency with which p is satisfied and the average frequency must remain in the constraining interval at all times (even when the limit does not exist), except for an initial "heat-up" phase. It is worth noting that the policies we will synthesize always have even the limit defined. If an atomic proposition p_s is only satisfied in state s, then the constraint (s, x, x) is equivalent to the usual constraint $\pi(s) = x$ for the steady-state distribution π.

Further, $\mathcal{M}$ *δ-satisfies* $\mathcal{S}$ for some $\delta \geq 0$, written $M \models_\delta \mathcal{S}$, if $\mathcal{M} \models \{(p, \ell - \delta, u + \delta) \mid (p, \ell, u) \in \mathcal{S}\}$. Intuitively, δ-satisfaction allows for δ imprecision in satisfying the constraints. Such a relaxation is often used, for instance, to obtain "δ-decidability" of otherwise undecidable problems, e.g. on systems with complex continuous dynamics [Gao *et al.*, 2012; Rungger and Tabuada, 2017]. In our case, we show it allows for a polynomial-time solution even when restricting to finite-memory policies.

Long-run average reward. Let $r : A \to \mathbb{Q}$ be a *reward function*. Recall that A_t is a random variable returning the action played at time t. Similarly to the steady-state distribution, the random variable given by the limit-average function $\mathrm{lr}(r) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} r(A_t)$ may be undefined for some runs, so we consider maximizing the respective point-wise limit inferior:

$$\mathrm{lr}_{\inf}(r) = \liminf_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} r(A_t)$$

Although we could also maximize limit superior, it is less interesting technically, see [Brázdil *et al.*, 2014], and less relevant from the perspective of ensuring the required performance. Further, the respective minimizing problems can be solved by maximization with opposite rewards.

3 Problem Statement and Examples

This paper is concerned with the following tasks:

Satisfiability: Given an MDP $\mathcal{G}$ with the *long-run specification* $\mathfrak{LR} = ((\varphi, \theta), \mathcal{S}, (r, R))$, where φ is an LTL formula, $\theta \in [0, 1]$ is a probability threshold, $\mathcal{S}$ is a steady-state specification, r is a reward function, $R \in \mathbb{Q}$ is a reward threshold, decide whether there is a policy σ such that

- $\mathbb{P}^\sigma[\models \varphi] \geq \theta$ (LTL)
- $\mathcal{G}^\sigma \models \mathcal{S}$ (SSS)
- $\mathbb{E}^\sigma[\mathrm{lr}_{\inf}(r)] \geq R$ (LRA)

Policy synthesis: If satisfiable, construct a policy satisfying the requirements, i.e. inducing a Markov chain satisfying them.

δ-satisfying finite-memory policy synthesis (for $\delta > 0$)**:** If satisfiable, construct a finite-memory policy σ ensuring

- $\mathbb{P}^\sigma[\models \varphi] \geq \theta$ (LTL)
- $\mathcal{G}^\sigma \models_\delta \mathcal{S}$ (δ-SSS)
- $\mathbb{E}^\sigma[\mathrm{lr}_{\inf}(r)] \geq R - \delta$ (δ-LRA)

We illustrate the problem, some of its intricacies, the effect of the bound on the memory and the consequent importance of δ-satisfaction on the following examples.

Example 1. Consider the MDP in Fig. 1 with the specification $\pi(s) = 0.5 = \pi(t)$. On the one hand, every memoryless policy yields either $\pi(s) = 1$ or $\pi(t) = 1$, the latter occurring iff b is played in s with non-zero probability. On the other hand, a history-dependent policy σ defined by $\sigma(sa \cdots s)(a) = 1$ and $\sigma(\lambda)(a) = 0.5 = \sigma(\lambda)(b)$ satisfies the specification. In terms of the automata representation, the policy is 2-memory, remembering whether a step has been already taken, see Fig. 5 in Appendix A. △

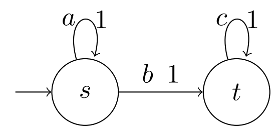

Figure 1: A (multichain) MDP where memory is required for satisfying a steady-state specification (SSS)

Consequently, memory may be necessary, in contrast to the claim of [Velasquez, 2019] that memoryless policies are sufficient by [Akshay *et al.*, 2013], which holds only for the setting with recurrent chains. Moreover, the combination with LTL may require even unbounded memory:

Example 2. Consider the MDP in Fig. 2 with the LTL specification $(\mathbf{GF}p_t, 1)$ where p_t holds in t only, and steady-state constraint $(s, 1, 1)$. Every policy satisfying the specification must play action b infinitely often. On the one hand, a history-dependent policy σ defined by $\sigma(ws)(b) = 1/|w|$, where $|w|$ denotes the length of w, satisfies the specification. Indeed, t is still visited infinitely often almost surely, but with frequency decreasing below every positive bound. While σ is Markovian (does not depend on the history, only on the time), it means it still uses unbounded memory.

On the other hand, every policy with finite memory satisfying the LTL specification visits t with positive frequency, in fact with at least $1/p^{|S|\cdot m}$ where m is the size of the memory and p the minimum probability occurring in the policy. Hence no finite-memory policy can satisfy the whole specification. However, in order to δ-satisfy it, a memoryless policy is sufficient, defined by $\sigma(s)(b) = 1/\delta$. △

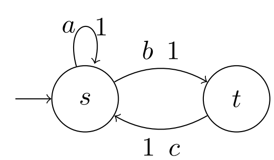

Figure 2: A (unichain) MDP where unbounded memory is required for satisfying SSS and LTL specifications, but finite memory is sufficient for δ-satisfaction

4 LTL-Constrained Steady-State Policy Synthesis

Fig. 3 depicts a set of linear constraints capturing (both transient and recurrent) behaviour of a policy as a "flow" [Chatterjee *et al.*, 2017], with variables capturing how much each action is used. Together with maximization of a certain linear

Input: MDP $(S, A, Act, \delta, \hat{s}, \nu)$

Beside requiring all variables y_a, y_s, x_a for $a \in A, s \in S$ be non-negative, the *policy-flow* constraints are the following:

1. transient flow: for $s \in S$
$$\mathbf{1}_{\hat{s}}(s) + \sum_{a \in A} y_a \cdot \delta(a)(s) = \sum_{a \in Act(s)} y_a + y_s$$
2. almost-sure switching to recurrent behaviour:
$$\sum_{s \in C \in \mathsf{MEC}} y_s = 1$$
3. probability of switching in a MEC is the frequency of using its actions: for $C \in \mathsf{MEC}$
$$\sum_{s \in C} y_s = \sum_{a \in C} x_a$$
4. recurrent flow: for $s \in S$
$$\sum_{a \in A} x_a \cdot \delta(a)(s) = \sum_{a \in Act(s)} x_a$$

Figure 3: Linear program for policy synthesis in general (multichain) MDP

Input: MDP $(S, A, Act, \delta, \hat{s}, \nu)$, long-run specification $\mathfrak{LR} = ((\varphi, \theta), \mathcal{S}, (r, R))$, AMEC due to the assumption that the MDP is a product of another MDP with $\mathcal{LDBA}(\varphi)$

The *specification constraints* are the following:

5. LTL specification:
$$\sum_{a \in A \setminus \bigcup \mathsf{AMEC}} x_a \leq 1 - \theta$$
6. steady-state specification: for every $(p, \ell, u) \in \mathcal{S}$
$$\ell \leq \sum_{\substack{p \in \nu(s) \\ a \in Act(s)}} x_a \leq u$$
7. LRA rewards:
$$\sum_{a \in A} x_a \cdot r(a) \geq R$$

Figure 4: Linear program for the long-run specification on the product MDP (together with the policy-flow constraints of the LP in Fig. 3)

function it yields the standard linear programme for maximizing LRA reward in general (multichain) MDP [Puterman, 1994]. Since then it has been used in various contexts, including multiple reachability objectives [Etessami *et al.*, 2008] or multi-dimensional rewards [Brázdil *et al.*, 2014; Chatterjee *et al.*, 2017].

Intuitively, x_a is the expected frequency of using a on the long run; Equation 4 thus expresses the recurrent flow in MECs and $\pi(s) := \sum_{a \in Act(s)} x_a$ is the steady-state distribution for state s. However, before we can play according to x-variables, we have to reach MECs and switch from the transient behaviour to this recurrent behaviour. Equation 1 expresses the transient flow before switching. Variables y_a are the expected number of using a until we switch to the recurrent behaviour in MECs and y_s is the probability of this switch upon reaching s. To relate y- and x-variables, Equation 3 states that the probability to switch and remain within a given MEC is the same whether viewed from the transient or recurrent flow perspective.[2] Finally, Equation 2 states that switching happens almost surely.[3]

Lemma 1 ([Chatterjee *et al.*, 2017])**.** *Existence of a solution to the equations in Fig. 3 is equivalent to existence of a policy inducing the steady-state distribution $\pi(s) = \sum_{a \in Act(s)} x_a$ for all $s \in S$.* □

[2]Actually, one could eliminate variables y_s and use directly x_a in Equation 1 and leave out Equation 3 completely, in the spirit of [Puterman, 1994]. However, the form with explicit y_s is closer to the more recent related work, more intuitive, and more convenient for correctness proofs.

[3]Note that summing Equation 1 over all $s \in S$ yields $\sum_{s \in S} y_s = 1$. Since y_s can be shown to equal 0 for state s not in MEC, Equation 2 is again redundant but more convenient.

Theorem 1 (Linear program for LTL-constrained steady-state policy synthesis on general MDP)**.** *For an MDP $\mathcal{G}$ and a long-run specification $\mathfrak{LR}$, let L be the system consisting of linear policy-flow constraints of Fig. 3 and specification constraints of Fig. 4 both on input $\mathcal{G} \times \mathcal{LDBA}(\varphi)$ where φ is the LTL formula of $\mathfrak{LR}$. Then:*

1. *Every policy satisfying $\mathfrak{LR}$ induces a solution to L.*
2. *Every solution to L effectively induces a (possibly unbounded-memory) policy satisfying $\mathfrak{LR}$ and, moreover for every $\delta > 0$, also a finite-memory policy δ-satisfying $\mathfrak{LR}$.*

Proof sketch. The full proof can be found in [Křetínský, 2021, Appendix B]. It heavily relies on results about the linear programming solution for the multi-dimensional long-run average reward [Chatterjee *et al.*, 2017] and also on the LTL model checking algorithm on MDP via LDBA [Sickert *et al.*, 2016]. Essentially, each steady-state constraint can be seen as a LRA constraint on a new reward that is 1 whenever p holds and 0 otherwise. Besides, δ-satisfaction of the LTL constraint with finite memory is equivalent to positive reward collected on accepting states.

The constructed policy first reaches the desired MECs where it "switches" to staying in the MEC and playing a memoryless policy whose frequencies of using each action a are close to x_a, and hence we get a 2-memory policy; alternatively, the switch might be followed by a policy achieving exactly x_a but possibly requiring unbounded memory. The policy's distribution over actions then depends on the length of the current history. □

Corollary 1. *Given an MDP $\mathcal{G}$ with a long-run specification $\mathfrak{LR}$ (possibly with a trivial reward threshold, e.g. the minimum reward), a policy maximizing reward among policies satisfying $\mathfrak{LR}$ can be effectively synthesized.*

Proof. First, we construct the product $\mathcal{G} \times \mathcal{LDBA}(\varphi)$, where φ is the LTL formula of $\mathfrak{LR}$. Second, we construct the linear programme with the objective function

$$\max \sum_{a \in Act(s)} x_a \cdot r(a)$$

and the constraints 1.–7., or in the case of trivial reward threshold only 1.–6. Thirdly, a solution to the programme induces the satisfying policy (as described in detail in the proof of Theorem 1. □

5 Complexity

Observe that the *number of variables* is linear in the size of the product and the *size* of the LP is quadratic, hence the size is polynomial in the size of $\mathcal{G}$ and $\mathcal{LDBA}(\varphi)$. Since linear programming is solvable in polynomial time [Khachiyan, 1979], our problems can also be solved in time polynomial in $\mathcal{G}$ and $\mathcal{LDBA}(\varphi)$.

Consequently, the satisfiability problem can be solved in double exponential time due to the translation of LTL to LDBA with this complexity. In fact, it is 2-EXPTIME-complete, inheriting the hardness from its special case of LTL model checking of MDP. While this sounds pessimistic at first glance, the worst-case complexity is rarely any issue in practice whenever it stems only from the translation as here. Indeed, the sizes of the LDBA produced for practically occurring formulae are typically very small, see [Sickert *et al.*, 2016; Sickert and Křetínský, 2016; Křetínský *et al.*, 2018; Esparza *et al.*, 2020] for examples and more details. Moreover, in the frequent case of qualitative (almost sure) LTL satisfaction, Constraint 5 takes the form: for every $a \in A \setminus \bigcup \mathsf{AMEC} : x_a = 0$, effectively eliminating many variables from the LP.

Further, for long-run specifications with ω-regular properties (instead of LTL), the problem can be solved in polynomial time if the input specification is provided directly as an LDBA. If instead a fully non-deterministic BA is used, single exponential time is required due to the semi-determinization to LDBA [Courcoubetis and Yannakakis, 1995].

In any case, the algorithm is polynomial with respect to the MDP and there are no extra variables for the steady-state or reward specifications. Hence, if the LTL property is trivial (omitted), there is almost no overhead compared the LP of Fig. 3 (or the LP for rewards which contains one constraint (7.) more). In particular, for point specifications with constraints (p, x, x) as in [Akshay *et al.*, 2013], Constraint 6 takes the form $\sum_a x_a = x$, which can be substituted into the system, even decreasing the number of its variables.

The final remark is concerned with the type of the automaton used. While we used LDBA, one could use other, deterministic automata such as the traditionally used Rabin or parity [Baier and Katoen, 2008]. However, there are two drawbacks of doing so. Firstly, while the translation from LTL is for all the cases double exponential in the worst case, practically it is often worse for Rabin than for LDBA and yet significantly worse for parity, see e.g. [Křetínský *et al.*, 2018]. Secondly, the acceptance condition is more complex. Instead of the Büchi condition with infinitely many visits of a set of states, for Rabin there is disjunction of several options, each a conjunction of the Büchi condition and prohibiting some other states. A reduction to LRA would require several copies of these parts in the product, each with the prohibited parts removed, resulting in a larger system. For parity, one copy would be sufficient together with a pre-processing by the attractor construction, yet, as already mentioned, the automaton is typically significantly larger than the LDBA, see [Esparza *et al.*, 2017].

6 Discussion

We have seen we can synthesize satisfying policies if they exist. The generality of the framework can be documented by the ease of extending our solution to various modifications of the problem. For instance, the general policies suffer from several deficiencies, which we can easily address here:

- *Unbounded memory* may not be realistic for implementation. As discussed, we can solve the δ-satisfaction problem in polynomial time for steady-state specifications (or also for general long-run specification provided the $\mathcal{LDBA}(\varphi)$ is given on input), yielding finite-memory policies, where the imprecision δ can be arbitrarily small.
- The unbounded-memory solution may lead to policies visiting the accepting states *less and less frequently* over time. Avoiding this is easy through an additional constraint
$$\sum_{a \in Acc A \cap C \in \mathsf{AMEC}} x_a \geq \frac{1}{f}$$
ensuring the frequency does not decrease below once per f steps on average. Alternatively, if we want to enforce the frequency bound for almost all satisfying runs, then we can instead use the frequency bound in each MEC:
$$\sum_{a \in Acc A \cap C} x_a \geq \frac{1}{f} \cdot \sum_{a \in A \cap C} x_a$$

Further modifications and their solutions include:

- Similarly to the frequency bounds for visiting accepting states, we may require bounds on *frequency of satisfying recurrent subgoals*, i.e. subformulae of φ. Indeed, the translation from LTL to LDBA [Sickert *et al.*, 2016] labels the states with the progress of satisfying subformulae. For each subformula, there are its "breakpoint" states where the subformula is satisfied, and whose x variables we can constrain as above.
- LRA rewards can be *multi-dimensional*, i.e., take the vector form $\boldsymbol{r} : A \to \mathbb{Q}^n$. Then the single Constraint 7 appears once for each dimension, with negligible additional computational requirements. Moreover, we can optimize any weighted combination of the LRA rewards or consider the trade-offs among them. To this end, we

can ε-approximate the Pareto frontier capturing all pointwise optimal combinations, in time polynomial in the size of the product and in $1/\varepsilon$ [Papadimitriou and Yannakakis, 2000; Chatterjee *et al.*, 2017].

- We can *optimize the satisfaction probability of the LTL* formula instead of the reward or consider the trade-offs. To this end, it is important that our solution uses a single linear programme. A possible approach of analyzing each MEC separately and then merging the fixed solutions would suffer from the mutual inflexibility and could not efficiently address these trade-offs.
- In contrast, if only *unichain*[4] solutions should be considered, see [Velasquez, 2019], we can check each accepting MEC separately (through our LP) whether it also satisfies the steady-state specification and whether it can be reached almost surely.

As future work, one may consider combinations with non-linear properties, such as those expressible in branching-time logics, or with finite-horizon or discounted rewards.

Acknowledgments

This work has been partially supported by the German Research Foundation (DFG) projects 427755713 (KR 4890/3-1) *Group-By Objectives in Probabilistic Verification (GOPro)* and 317422601 (KR 4890/1-1) *Verified Model Checkers (VMC)*.

Appendix

A Finite-Memory Policies

In order to work with finite-memory policies, we also use a slightly different (though equivalent—see [Brázdil *et al.*, 2014, Section 6]) definition of policies, which is more convenient in some settings. Let M be a countable set of *memory elements*. A *policy* is a triple $\sigma = (\sigma_u, \sigma_n, \alpha)$, where $\sigma_u : A \times S \times \mathsf{M} \to \mathit{Dist}(\mathsf{M})$ and $\sigma_n : S \times \mathsf{M} \to \mathit{Dist}(A)$ are *memory update* and *next move* functions, respectively, and α is the initial distribution on memory elements. We require that, for all $(s, m) \in S \times \mathsf{M}$, the distribution $\sigma_n(s, m)$ assigns a positive value only to actions enabled at s, i.e. $\sigma_n(s, m) \in \mathit{Dist}(\mathit{Act}(s))$.

A *play* of $\mathcal{G}$ determined by a policy σ is a Markov chain $\mathcal{G}^\sigma$ where the set of locations is $S \times \mathsf{M} \times A$, the initial distribution μ is zero except for $\mu(\hat{s}, m, a) = \alpha(m) \cdot \sigma_n(\hat{s}, m)(a)$, and

$$P(s, m, a)(s', m', a') = \delta(a)(s') \cdot \sigma_u(a, s', m)(m') \cdot \sigma_n(s', m')(a') \,.$$

Hence, $\mathcal{G}^\sigma$ starts in a location chosen randomly according to α and σ_n. In a current location (s, m, a), the next action to be performed is a, hence the probability of entering s' is $\delta(a)(s')$. The probability of updating the memory to m' is $\sigma_u(a, s', m)(m')$, and the probability of selecting a' as the next action is $\sigma_n(s', m')(a')$. Note that these choices are independent, and thus we obtain the product above.

[4] A Markov chain is a unichain if it only has one bottom strongly connected component (one recurrent class) and all other states are transient.

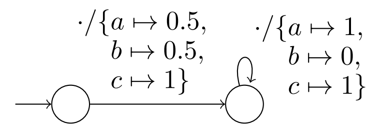

Figure 5: An automaton representation of the 2-memory strategy considered in Example 1 depicted in Fig. 1

In general, a policy may use infinite memory M, and both σ_u and σ_n may randomize, the former called *stochastic-update*, the latter *randomization* in the narrower sense.

We can now classify the policies according to the size of memory they use. Important subclasses are *memoryless* policies, in which M is a singleton, *n-memory* policies, in which M has exactly n elements, and *finite-memory* policies, in which M is finite. Other policies are *unbounded-memory* or *infinite-memory* policies.

References

[Akshay *et al.*, 2013] S. Akshay, Nathalie Bertrand, Serge Haddad, and Loïc Hélouët. The steady-state control problem for Markov decision processes. In *QEST*, volume 8054 of *Lecture Notes in Computer Science*, pages 290–304. Springer, 2013.

[Atia *et al.*, 2020] George K. Atia, Andre Beckus, Ismail Alkhouri, and Alvaro Velasquez. Steady-state policy synthesis in multichain Markov decision processes. In *IJCAI*, pages 4069–4075. ijcai.org, 2020.

[Baier and Katoen, 2008] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[Benton *et al.*, 2012] J. Benton, Amanda Jane Coles, and Andrew Coles. Temporal planning with preferences and time-dependent continuous costs. In *ICAPS*. AAAI, 2012.

[Brafman *et al.*, 2018] Ronen I. Brafman, Giuseppe De Giacomo, and Fabio Patrizi. LTLf/LDLf non-Markovian rewards. In *AAAI*, pages 1771–1778. AAAI Press, 2018.

[Brázdil *et al.*, 2014] Tomás Brázdil, Václav Brozek, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Two views on multiple mean-payoff objectives in Markov decision processes. *Log. Methods Comput. Sci.*, 10(1), 2014.

[Brázdil *et al.*, 2015] Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Multigain: A controller synthesis tool for mdps with multiple mean-payoff objectives. In *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 181–187. Springer, 2015.

[Chatterjee *et al.*, 2017] Krishnendu Chatterjee, Zuzana Křetínská, and Jan Křetínský. Unifying two views on multiple mean-payoff objectives in Markov decision processes. *Log. Methods Comput. Sci.*, 13(2), 2017.

[Courcoubetis and Yannakakis, 1995] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[Esparza *et al.*, 2017] Javier Esparza, Jan Křetínský, Jean-François Raskin, and Salomon Sickert. From LTL and limit-deterministic Büchi automata to deterministic parity automata. In *TACAS (1)*, volume 10205 of *Lecture Notes in Computer Science*, pages 426–442, 2017.

[Esparza *et al.*, 2020] Javier Esparza, Jan Křetínský, and Salomon Sickert. A unified translation of linear temporal logic to ω-automata. *J. ACM*, 67(6):33:1–33:61, 2020.

[Etessami *et al.*, 2008] K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. *LMCS*, 4(4):1–21, 2008.

[Feinberg and Shwartz, 2012] Eugene A Feinberg and Adam Shwartz. *Handbook of Markov decision processes: methods and applications*, volume 40. Springer Science & Business Media, 2012.

[Gao *et al.*, 2012] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. Delta-decidability over the reals. In *LICS*, pages 305–314. IEEE Computer Society, 2012.

[Giacomo *et al.*, 2019] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In *ICAPS*, pages 128–136. AAAI Press, 2019.

[Hahn *et al.*, 2015] Ernst Moritz Hahn, Guangyuan Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. Lazy probabilistic model checking without determinisation. In *CONCUR*, volume 42 of *LIPIcs*, pages 354–367. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[Khachiyan, 1979] L.G. Khachiyan. A polynomial algorithm in linear programming. *Dokl. Akad. Nauk SSSR*, 244:1093–1096, 1979.

[Kress-Gazit *et al.*, 2009] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Trans. Robotics*, 25(6):1370–1381, 2009.

[Křetínský *et al.*, 2018] Jan Křetínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to your favourite deterministic automaton. In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 567–577. Springer, 2018.

[Křetínský, 2021] Jan Křetínský. LTL-constrained steady-state policy synthesis. *CoRR (arXiv)*, abs/2105.14894, 2021.

[Kwiatkowska *et al.*, 2011] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591, 2011.

[Papadimitriou and Yannakakis, 2000] C. H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *FOCS*, pages 86–92, 2000.

[Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[Puterman, 1994] M.L. Puterman. *Markov Decision Processes*. John Wiley and Sons, 1994.

[Rungger and Tabuada, 2017] Matthias Rungger and Paulo Tabuada. Computing robust controlled invariant sets of linear systems. *IEEE Trans. Autom. Control.*, 62(7):3665–3670, 2017.

[Sickert and Křetínský, 2016] Salomon Sickert and Jan Křetínský. Mochiba: Probabilistic LTL model checking using limit-deterministic Büchi automata. In *ATVA*, volume 9938 of *Lecture Notes in Computer Science*, pages 130–137, 2016.

[Sickert *et al.*, 2016] Salomon Sickert, Javier Esparza, Stefan Jaax, and Jan Křetínský. Limit-deterministic Büchi automata for linear temporal logic. In *CAV*, pages 312–332, 2016.

[Velasquez, 2019] Alvaro Velasquez. Steady-state policy synthesis for verifiable control. In *IJCAI*, pages 5653–5661. ijcai.org, 2019.