Computational Approaches for Stochastic Shortest Path on Succinct MDPs

Krishnendu Chatterjee\(^1\), Hongfei Fu\(^2\), Amir Kafshdar Goharshady\(^1\), and Nastaran Okati\(^3\)
\(^1\) IST Austria (Institute of Science and Technology Austria)
\(^2\) Shanghai Jiao Tong University
\(^3\) Ferdowsi University of Mashhad

kchatterjee@ist.ac.at, fuhf@cs.sjtu.edu.cn, goharshady@ist.ac.at, nastaran.okati@mail.um.ac.ir

Abstract

We consider the stochastic shortest path (SSP) problem for succinct Markov decision processes (MDPs), where the MDP consists of a set of variables, and a set of nondeterministic rules that update the variables. First, we show that several examples from the AI literature can be modeled as succinct MDPs. Then we present computational approaches for upper and lower bounds for the SSP problem: (a) for computing upper bounds, our method is polynomial-time in the implicit description of the MDP; (b) for lower bounds, we present a polynomial-time (in the size of the implicit description) reduction to quadratic programming. Our approach is applicable even to infinite-state MDPs. Finally, we present experimental results to demonstrate the effectiveness of our approach on several classical examples from the AI literature.

1 Introduction

Markov decision processes (MDPs) [Howard, 1960] are a standard mathematical model for sequential decision making, with a wide range of applications in artificial intelligence and beyond [Puterman, 1994; Filar and Vrieze, 1997; Bertsekas, 2005]. An MDP consists of a set of states, a finite set of actions (that represent the nondeterministic choices), and a probabilistic transition function that describes the transition probability over the next states, given the current state and action. One of the most classical optimization objectives in MDPs is the stochastic shortest path (SSP) problem, where the transitions of the MDP are labeled with rewards/costs, and the goal is to optimize the expected total rewards until a target set is reached.

Curse of dimensionality. In many typical applications, the computational analysis of MDPs suffers from the curse of dimensionality. The state space of the MDP is huge as it represents valuations to many variables that constitute the MDP. If the state and action are denoted by \( s \) and \( a \), respectively, the state transition function is denoted as \( P(s'|s,a) \), where \( s' \) denotes the state at the next time step. The transition and reward functions can act on the current state, action, and/or the state transition. In many practical applications, the state space of the MDP is very large, and the action space is also huge. One way to reduce the curse of dimensionality is to consider factored MDPs [Guestrin et al., 2003; Delgado et al., 2011] that allow a mathematical approach to constraint solving to obtain a mathematical approach to compute bounds for the SSP problem and our algorithmic bounds are as follows: (a) for the sup-value (resp. inf-value) we show that an upper (resp., lower) bound can be computed in polynomial time in the implicit description of the MDP; (b) for the sup-value (resp. inf-value) we show that a lower (resp., upper) bound can be computed by a polynomial-time (in the implicit description) reduction to quadratic programming. Our approach is as follows: we use results from probability theory to establish a mathematical approach to compute bounds for the SSP problem for succinct MDPs (Section 3), and reduce the mathematical approach to constraint solving to obtain a computational method (Section 4).

Our contributions. Our main contributions are as follows:

1. First, we show that many examples from the AI literature (e.g., Gambler’s Ruin, Robot Planning, and variants of Roulette) can be naturally modeled as succinct MDPs.
2. Second, we present mathematical and computational results for the SSP problem for succinct MDPs. For the SSP problem the sup-value (resp. inf-value) represents the expected shortest path value with supremum (resp., infimum) over all policies. We consider linear bounds for the SSP problem and our algorithmic bounds are as follows: (a) for the sup-value (resp. inf-value) we show that an upper (resp., lower) bound can be computed in polynomial time in the implicit description of the MDP; (b) for the sup-value (resp. inf-value) we show that a lower (resp., upper) bound can be computed by a polynomial-time (in the implicit description) reduction to quadratic programming. Our approach is as follows: we use results from probability theory to establish a mathematical approach to compute bounds for the SSP problem for succinct MDPs (Section 3), and reduce the mathematical approach to constraint solving to obtain a computational method (Section 4).
3. Finally, we present experimental results on several classical examples from the literature where our method computes tight bounds (i.e., lower and upper bounds are quite close) on the SSP problem extremely efficiently.

Comparison with approaches for factored MDPs. Some key advantages of our approach are the following. First, our approach gives a provably polynomial-time algorithm
values. The succinct MDP is described by a simple while loop program and sampling variables can take integer, or even real numbers, independently wrt some probability distribution. In general, both program variables are normal variables, while sampling variables as its right-hand-side. The operator $\square$ is the nondeterministic choice which means that the decision as to which $Q_i$ will be executed in the current loop iteration depends on a scheduler (or policy) that resolves nondeterminism. We first provide the formal syntax and then a simple example.

Formal syntax of simple-while-loop programs. A succinct MDP is specified by a simple-while-loop program equipped with probability distributions for sampling variables. We now formalize the intuitive description provided in Equation (1).

Given a simple-while-loop program can be produced using the following grammar, where each $\langle pvar \rangle$ is chosen from a finite fixed set $X$ of program variables, each $\langle svar \rangle$ from a finite fixed set $R$ of sampling variables and each $\langle constant \rangle$ denotes a floating point number:

\[
\langle \text{simple-while-loop-program} \rangle ::= \text{\textquoteleft while} \langle \text{guard} \rangle \text{\textquoteleft do} \langle \text{nondet-block-list} \rangle \text{\textquoteleft od} \\
\langle \text{guard} \rangle ::= \langle \text{linear-pvar-expr} \rangle \langle \text{cmp} \rangle \langle \text{linear-pvar-expr} \rangle \\
\langle \text{linear-pvar-expr} \rangle ::= \langle \text{constant} \rangle \langle \text{constant} \rangle \langle \text{pvar} \rangle \\
\langle \text{cmp} \rangle ::= \langle \text{\textquoteleft \textgreater\textgreater} \rangle \langle \text{\textless\textless} \rangle \langle \text{\textless\textless} \rangle \\
\langle \text{nondet-block-list} \rangle ::= \langle \text{block} \rangle \langle \text{nondet-block-list} \rangle \\
\langle \text{block} \rangle ::= \langle \text{assignment} \rangle \langle \text{assignment} \rangle \langle \text{block} \rangle \\
\langle \text{assignment} \rangle ::= \langle \text{pvar} \rangle \langle \text{\textquoteleft =\textquoteleft} \rangle \langle \text{linear-expr} \rangle \langle \text{\textquoteleft ;\textquoteleft} \rangle \\
\langle \text{linear-expr} \rangle ::= \langle \text{constant} \rangle \langle \text{constant} \rangle \langle \text{pvar} \rangle \\
\langle \text{example 1}. \text{Consider the following program} \\
\text{while } x \geq 1 \text{ do } x ::= x + r \text{ od } x ::= x - 1 \text{ od}
\]

where $x$ is a program variable and $r$ is a sampling variable that observes the two-point distribution $P(r = -1) = P(r = 1) = \frac{1}{2}$. Informally, the program performs either decrement or random increment/decrement on $x$ until its value is zero.

Informal description of the semantics. Given a simple-while-loop program in the form (1), an MDP is obtained as follows: the state space $S$ consists of values assigned to program variables (i.e., valuations for program variables); the action space $A$ corresponds to the non-deterministic choice between $Q_1, \ldots, Q_k$; and the transition function $P$ depends on the sampling of the sampling variables, which given the current valuation for program variables probabilistically updates the valuation. The assignments are linear functions, and the loop guard $\phi$ describes the target states as those which do not satisfy $\phi$. Given the above components, the notion of a policy

\[
\text{while } \phi \text{ do } Q_1 \square \ldots \square Q_k \text{ od} \tag{1}
\]

- $\phi$ is the loop guard defined as a single comparison between linear arithmetic expressions over program variables (e.g. $x \leq y + 1$);
(or scheduler) σ that resolves the non-deterministic choices, and that of the probability space \( \mathbb{P}^\sigma \) given a policy is standard [Puterman, 1994]. For a more formal treatment of the semantics, see Section 2.3.

**Remark 1** (Simple-while-loop MDPs and Factored MDPs). The principle behind factored MDPs and simple-while-loop succinct MDPs is similar. Both aim to consider high-dimensional and large MDPs described implicitly in a succinct way. In factored MDPs the transitions and reward functions can be factored based on small sets of variables, but the dependency on the variables in these sets can be arbitrary. In contrast, in simple-while-loop succinct MDPs, we only allow linear arithmetic expressions as guards and assignments. Moreover, our MDPs do not allow nesting of while loops. The goal of our work is to consider linear upper and lower bounds, and nesting of linear loops can result in super-linear bounds. Hence we do not consider nesting of loops. Therefore, simple-while-loop succinct MDPs are a special class of factored MDPs.

For algorithmic approaches with theoretical guarantees on computational complexity, the analysis of factored MDPs has typically been restricted to finite-state MDPs. However, we will present solutions for simple-while-loop programs, where the variables can take integer or real values, and thus the underlying state space is infinite or even uncountable. Thus the class of simple-while-loop MDPs consists of large finite-state MDPs (when the variables are bounded); countable state MDPs (variables are integer); and even uncountable state MDPs (variables are real-valued). Moreover, our algorithmic approaches provide computational complexity guarantees on the input size of the implicit representation of the MDP. Note that for finite-state MDPs, the implicit representation can be exponentially more compact than the explicit MDP. For example, \( n \) boolean variables lead to a state-space of size \( 2^n \).

In the sequel we consider MDPs obtained from simple-while-loop programs and, for brevity, call them succinct MDPs.

### 2.2 Stochastic Shortest Path on Succinct MDPs

We consider the stochastic shortest path (SSP) problem on succinct MDPs. Below we fix a succinct MDP described by a while-loop in the form \( (1) \).

**Reward function.** We consider a reward function \( R \) that assigns a reward \( R(u, \ell) \) when the sampling valuation for sampling variables is \( u \) and the non-deterministic choice is \( Q_\ell \) (in a loop iteration). We assume that there is a maximal constant \( R_{\max} \geq 0 \) such that \( |R(u, \ell)| \leq R_{\max} \) for all \( u, \ell \). The rewards need not be nonnegative as our approach is able to handle negative rewards as well.

**Stochastic shortest path.** Given an initial valuation \( v \) for program variables and a policy \( \sigma \), the definition of expected total reward/cost until termination is standard. The inf-value (resp., sup-value) of a succinct MDP, given an initial valuation \( v \) for program variables, is the infimum (resp., supremum) expected reward value over all policies that ensure finite expected termination time, which we denote as \( \text{infval}(v) \) (resp., \( \text{supval}(v) \)).

**Computational problem.** We consider the computational problem of obtaining upper and lower bounds for the inf-value and sup-value for succinct MDPs. Due to the similarity of the problems, we will focus only on computing lower and upper bounds for the sup-value, and the results for inf-value are similar and omitted.

---

**2.3 Illustrative Examples**

**Example 2** (Gambler’s Ruin). We start with a simple and classical example. A gambler has \( x_0 \) tokens for gambling. In each round he can choose one of the two types of gambling. In type 1, he wins with probability \( p_1 < 1/2 \) and in type 2 with probability \( p_2 < 1/2 \). A win leads to a reward of \( w \) and an extra token. A loss costs one token. The player gambles as long as he has tokens left. His goal is to maximize overall expected reward. Letting \( p_1 = 0.4, p_2 = 0.3 \) and \( w = 1 \), a succinct MDP for this example is shown in Figure 1

```plaintext
while x ≥ 1 do
  if (0.4) { x := x + 1 reward=1 } else { x := x - 1 }
  if (0.3) { x := x + 1 reward=1 } else { x := x - 1 }
```

Figure 1: Gambler’s Ruin as a succinct MDP

**Remark 2.** Note that above we use probabilistic if as syntactic sugar, where the assignments in if(\( p \)) run with probability \( p \) and those in the else part with probability \( 1 - p \). Given that the assignments are linear, this can be translated back to a succinct MDP, e.g. the first if-else block in Figure 1 is equivalent to:

\[
x := x + r \quad \text{reward=0.4}
\]

where \( r \) is a sampling variable with \( \mathbb{P}(r = 1) = 0.4 \) and \( \mathbb{P}(r = -1) = 0.6 \).

**Example 3** (Continuous variant). We can also consider a continuous variant of the example where the sampling variable \( r \) is chosen from some continuous distribution with expected value \( -0.2 \) (e.g., uniform distribution \([-0.8, 0.4]\)).

---

**2.4 Technical Details of Semantics and SSP**

**Formal Semantics**

We formalize our semantics by introducing the notion of valuations which specify current values for program and sampling variables. Below we fix a program \( P \) in the form \( (1) \) and let \( X \) (resp. \( R \)) be the set of program (resp. sampling) variables appearing in \( P \). The sizes of \( X \) and \( R \) are denoted by \( |X| \) and \( |R| \), respectively. We also impose arbitrary linear orders on both of \( X \) and \( R \), and assume that for each sampling variable
$r_i \in R$, a distribution $\delta_i$ is given and each time $r_i$ appears in the program, its value is stochastically sampled according to $\delta_i$.

Valuations. A program valuation is a vector $v \in \mathbb{R}^{|X|}$. Intuitively, a valuation $v$ specifies that for each $x \in X$ with rank $i$, the variable $x$ holds a value in the linear order, the value assigned is the $i$th coordinate $v[i]$ of $v$. Likewise, a sampling valuation is a vector $u \in \mathbb{R}^{|R|}$.

For each program valuation $v$, we say that $v$ satisfies the loop guard $\phi$, denoted by $v \models \phi$, if the formula $\phi$ holds when every appearance of a program variable is replaced by its corresponding value in $v$. Moreover, each $Q_m$ in $P$ $(1 \leq m \leq k)$ now encodes a function $F_m : \mathbb{R}^{|X|} \times \mathbb{R}^{|R|} \rightarrow \mathbb{R}^{|X|}$ which transforms the program valuation $v$ before the execution of $Q_m$ and the sampled values in $u$ into the program valuation $F_m(v, u)$. By our linear setting, each $F_m$ is also linear.

Our semantics are based on paths. Intuitively, a path is an infinite sequence of valuations where the valuations at the $n$th step reflects the current values for program and sampling variables at the $n$th step.

Paths. A path is a finite or infinite sequence of triples $((v_n, u_n, \ell_n))_{n \geq 0}$ such that each $v_n$ (resp. $u_n, \ell_n$) is a program valuation (resp. sampling valuation, nondeterministic choice). The intuition is that each $v_n$ (resp. $u_n$) is the current program valuation (resp. sampling valuation, nondeterministic choice) right before the $n$th loop-iteration of $P$.

The program $P$ may involve nondeterministic choices (i.e., the operator $\square$) which are still unspecified. To resolve non-determinism, we need the standard notion of schedulers.

Schedulers. A scheduler is a function which maps each finite path $(v_0, u_0, \ell_0), \ldots, (v_{n-1}, u_{n-1}, \ell_{n-1})$ and current program valuation $v_n$, to a number $m$ in $\{1, 2, \ldots, k\}$ representing the choice of $Q_m$ $(1 \leq m \leq k)$ at the $n$th loop iteration.

Intuitively, a scheduler resolves the nondeterministic choice at each iteration of $P$ by choosing which $Q_m$ to run in the loop body. The resolution at the $n$th iteration may depend on all previous valuations $P$ has traversed before.

Intuitive Semantics. Consider an initial program valuation $v$ and a scheduler $\sigma$. An infinite path $((v_n, u_n, \ell_n))_{n \in \mathbb{N}_0}$ is constructed as follows. Initially, $v_0 = v$. Then at each step $n$ ($n \geq 0$): first, a sampling valuation $u_n$ is obtained through samplings for all sampling variables, where the sampling of each sampling variable observes a predefined probability distribution for the variable; second, if $v_n \models \phi$ then the program enters the loop and $\ell_n := \sigma(((v_i, u_i))_{1 \leq i \leq n-1}, v_n)$, $v_{n+1} := F_{\ell_n}(v_n, u_n)$, otherwise the program terminates and $v_{n+1} := v_n$.

Probability Space. A probability space is a triple $(\Omega, \mathcal{F}, \mathbb{P})$, where $\Omega$ is a nonempty set (so-called sample space), $\mathcal{F}$ is a $\sigma$-algebra over $\Omega$ (i.e., a collection of subsets of $\Omega$ that contains the empty set $\emptyset$ and is closed under complementation and countable union), and $\mathbb{P}$ is a probability measure on $\mathcal{F}$, i.e., a function $\mathbb{P} : \mathcal{F} \rightarrow [0, 1]$ such that (i) $\mathbb{P}(\Omega) = 1$ and (ii) for all set-sequences $A_1, A_2, \ldots \in \mathcal{F}$ that are pairwise-disjoint (i.e., $A_i \cap A_j = \emptyset$ whenever $i \neq j$) it holds that $\sum_{i=1}^{\infty} \mathbb{P}(A_i) = \mathbb{P}\left(\bigcup_{i=1}^{\infty} A_i\right)$. Elements $A \in \mathcal{F}$ are usually called events. An event $A \in \mathcal{F}$ is said to hold almost surely (a.s.) if $\mathbb{P}(A) = 1$.

Formal Semantics. Given an initial program valuation $v$ and a scheduler $\sigma$, we build the probability space $(\Omega, \mathcal{F}, \mathbb{P})^v$ for the program $P$ as follows. First, $\Omega$ is the set of all infinite paths. Then, we construct $\mathcal{F}, \mathbb{P}^v$ through general state-space Markov chains. In detail, we build the kernel function on the set of all finite paths. The construction of the kernel function follows exactly from our aforementioned intuitive semantics. Then the probability space on infinite paths is generated uniquely from the kernel function.

Formal Details of the SSP Problem

We consider accumulated cost until program termination. We first define several classic notions of probability theory.

Random Variables. A random variable $X$ for a probability space $(\Omega, \mathcal{F}, \mathbb{P})$ is an $\mathcal{F}$-measurable function $X : \Omega \rightarrow \mathbb{R} \cup \{-\infty, +\infty\}$, i.e., a function satisfying the condition that for all $d \in \mathbb{R} \cup \{-\infty, +\infty\}$, the set $\{\omega \in \Omega \mid X(\omega) < d\}$ belongs to $\mathcal{F}$; By convention, we abbreviate $+\infty$ as $\infty$.

Below we fix a program $P$ in the form $(\mathcal{L})$ with program variables $X$ and sampling variables $R$. We first establish the notion of cost functions for $P$ which measures the cost consumed in one loop iteration. In this paper, we consider that the cost is only related to the sampling valuation and the nondeterministic choice before the execution of the loop body.

Definition 1. A cost function is a function $C : \mathbb{R}^{|R|} \times \{1, \ldots, k\} \rightarrow [0, \infty)$.

The next definition introduces the notion of accumulated cost.

Definition 2. For each $m \in \mathbb{N}$, we define the random variable $C_m$ for cost at the $m$th step by:

$$C_m((v_n, u_n, \ell_n))_{n \in \mathbb{N}} := \begin{cases} R(u_m, \ell_m) & \text{if } v_m \models \phi \\ 0 & \text{otherwise} \end{cases}$$

for any infinite path $((v_n, u_n, \ell_n))_{n \in \mathbb{N}}$. Then the random variable $C_{\infty}$, for accumulated cost until termination is defined as $C_{\infty}(\rho) := \sum_{m=0}^{\infty} C_m(\rho)$ for all infinite paths $\rho$.

Expectation. The expected value of a random variable $X$ from a probability space $(\Omega, \mathcal{F}, \mathbb{P})$, denoted by $\mathbb{E}(X)$, is defined as the Lebesgue integral of $X$ w.r.t $\mathbb{P}$, i.e., $\mathbb{E}(X) := \int X \, d\mathbb{P}$; the precise definition of Lebesgue integral is somewhat technical and is omitted here (cf. [Williams, 1991 Chapter 5] for a formal definition).

We study the expected accumulated cost $\mathbb{E}(C_{\infty})$, which is an important criterion for measuring how much cost the program consumes upon termination. In the presence of nondeterminism, we consider maximum and minimum accumulated cost over all schedulers.

Definition 3. The maximum (resp. minimum) expected accumulated cost $\sup_{\sigma} \mathbb{E}^\sigma(C_{\infty})$ (resp. $\inf_{\sigma} \mathbb{E}^\sigma(C_{\infty})$ w.r.t an initial program valuation $v$) is defined as $\sup_{\sigma} \mathbb{E}^\sigma(C_{\infty})$.
(resp. \( \infval(v) := \inf_{\sigma} \mathbb{E}_\sigma^\cdot(C_{\infty}) \)), where \( \sigma \) ranges over all schedulers and \( \mathbb{E}_\sigma^\cdot(\cdot) \) is the expectation under the probability space \((\Omega, \mathcal{F}, \mathbb{P})\) for infinite paths.

### 3 Theoretical Results

In this section we present the main theoretical results which forms the basis of our algorithms of the following section.

**Notation.** Given a succinct MDP in the form (1), we let \( X \) be the set of program variables in the program, \(|X|\) be the size of \( X, \mathbb{R}^{|X|} \) be the set of \(|X|\)-dimensional real-valued vectors, \( v \in \mathbb{R}^{|X|} \) be a valuation for program variables such that the \( i \)th program variable is the \( i \)th coordinate of \( v \), \( u \) be a valuation for sampling variables, \( N := \{1, \ldots, k\} \) be the set of non-deterministic choices, \( \ell \in N \) be a non-deterministic choice, and \( F_\ell (\ell \in N) \) be the function such that \( F_\ell (v, u) \) is the valuation for program variables resulting from executing \( Q_\ell \) with the valuation \( v \) for program variables and the sampled valuation \( u \) for sampling variables. Table 1 summarizes the notation.

| Notation | Meaning |
|----------|---------|
| \( X \) | the set of program variables |
| \(|X|\) | the size of \( X \) |
| \( \mathbb{R}^{|X|} \) | the set of \(|X|\)-dimensional real column vectors |
| \( v \) | a valuation for program variables |
| \( u \) | a valuation for sampling variables |
| \( \infval(v) \) | the inf-value for initial valuation \( v \) |
| \( \supval(v) \) | the sup-value for initial valuation \( v \) |
| \( N \) | the set of non-deterministic choices |
| \( \ell \) | a non-deterministic choice in \( N \) |
| \( F_\ell \) | the transformation function of \( Q_\ell \) mapping valuations before its execution to the resulting valuation after it |
| \( R \) | the reward function |
| \( H_{\max} \) | an upperbound for the absolute value of the rewards |

Table 1: A Summary of Notation

#### Upper bounds

We first introduce the main concept for computing an upper bound for \( \supval(v) \) formally, and then present the informal description.

**Definition 4** (Linear Upper Potential Functions (LUPFs)). A linear upper potential function (LUPF) is a function \( h: \mathbb{R}^{|X|} \rightarrow \mathbb{R} \) that satisfies the following conditions:

1. (C1) \( h \) is linear, i.e., there exist \( a \in \mathbb{R}^{|X|} \) and \( b \in \mathbb{R} \) such that for all \( v \in \mathbb{R}^{|X|} \), we have \( h(v) = a^T \cdot v + b \);
2. (C2) for all \( v, u, \ell \) such that \( v \models \phi \), \( F_\ell (v, u) \models \neg \phi \) and \( \ell \in N \), we have \( K \leq h(F_\ell (v, u)) \leq K' \) for some fixed constants \( K \) and \( K' \);
3. (C3) for all \( \ell \in N \) and all valuations \( v \) such that \( v \models \phi \),
   \[
   h(v) \geq \mathbb{E}_u(h(F_\ell (v, u))) + \mathbb{E}_u(R(u, \ell))
   \]
   where \( \mathbb{E}_u(h(F_\ell (v, u))) \), \( \mathbb{E}_u(R(u, \ell)) \) are the expected values over the sampling \( u \) when fixing \( v \) and \( \ell \);
4. (C4) for all \( v \) such that \( v \models \phi \), all sampling valuations \( u \) and all \( \ell \in N \), we have \( |h(v) - h(F_\ell (v, u))| \leq M \) for some fixed constant \( M \geq 0 \).

Informally, (C1) specifies the linearity of LUPFs, (C2) specifies that the value of the function at terminating valuations should be bounded, (C3) specifies that the current value of the function at \( v \) is no less than that of the next step at \( F_\ell (v, u) \) plus the cost/reward at the current step, and finally (C4) specifies that the change of value between the current step \( v \) and the next step \( F_\ell (v, u) \) is bounded. Note that \( \mathbb{E}_u(h(F_\ell (v, u))) \) is linear in \( v \) since \( h \) and \( F_\ell \) are linear. Note that the function \( h \) (only) depends on the valuations of the variables before the loop execution, and hence it is only loop-dependent (but not dependent on each assignment).

The following theorem shows that LUPFs indeed serve as upper bounds for \( \supval(*) \).

**Theorem 1.** If \( h \) is an LUPF, then \( \supval(v) \leq h(v) - K \) for all valuations \( v \in \mathbb{R}^{|X|} \) such that \( v \models \phi \).

**Proof sketch.** The key ideas of the proof are as follows: Fix any scheduler \( \sigma \) that ensures finite expected termination time.

- We first construct a stochastic process based on \( h \). Using the condition (C3) which is non-increasing property we establish that the stochastic process obtained is a supermartingale (for definitions of supermartingale see [Williams, 1991, Chapter 10]). The supermartingale in essence preserve the non-increasing property.
- Given the supermartingale, we apply Optional Stopping Theorem (OST) ([Williams, 1991, Chapter 10]), and use condition (C4) to establish the required boundedness condition of OST, to arrive at the desired result.

While conditions (C1) and (C2) are not central to the proof, the condition (C1) ensures linearity, which will be required by our algorithms, and the condition (C2) is the boundedness after termination, that derives the desired upper bounds (i.e., contribution of the term \( K \) comes from condition (C2)).

**Formal proof of Theorem 7** In order to formally prove the theorem, we need the fundamental mathematical notions of filtrations, stochastic processes and conditional expectation.

**filtrations.** A filtration of a probability space \((\Omega, \mathcal{F}, \mathbb{P})\) is an infinite sequence \( \{\mathcal{F}_n\}_{n \in \mathbb{N}_0} \) of \( \sigma \)-algebras over \( \Omega \) such that \( \mathcal{F}_n \subseteq \mathcal{F}_{n+1} \subseteq \mathcal{F} \) for all \( n \in \mathbb{N}_0 \).

**Discrete-Time Stochastic Processes.** A discrete-time stochastic process is a sequence \( \Gamma = \{X_n\}_{n \in \mathbb{N}_0} \) of random variables where \( X_n \)'s are all for some probability space (say, \((\Omega, \mathcal{F}, \mathbb{P})\)). \( \Gamma \) is adapted to a filtration \( \{\mathcal{F}_n\}_{n \in \mathbb{N}_0} \) of sub-\( \sigma \)-algebras of \( \mathcal{F} \) if for all \( n \in \mathbb{N}_0, X_n \in \mathcal{F}_n \)-measurable.

**Conditional Expectation.** Let \( X \) be any random variable for a probability space \((\Omega, \mathcal{F}, \mathbb{P})\) such that \( \mathbb{E}(\cdot) < \infty \). Then given any \( \sigma \)-algebra \( \mathcal{G} \subseteq \mathcal{F} \), there exists a random variable (for \((\Omega, \mathcal{F}, \mathbb{P})\)), conventionally denoted by \( \mathbb{E}(\cdot|\mathcal{G}) \), such that:

- (E1) \( \mathbb{E}(X|\mathcal{G}) \) is \( \mathcal{G} \)-measurable, and
- (E2) \( \mathbb{E}(\mathbb{E}(X|\mathcal{G})) < \infty \), and
- (E3) for all \( A \in \mathcal{G} \), we have \( \int_A \mathbb{E}(X|\mathcal{G}) \ d\mathbb{P} = \int_A X \ d\mathbb{P} \).
The random variable $\mathbb{E}(X|G)$ is called the conditional expectation of $X$ given $G$, and is unique in the sense that if $Y$ is another random variable satisfying (E1)–(E3), then $\mathbb{P}(Y = \mathbb{E}(X|G)) = 1$. See [Williams, 1991] Chapter 9 for more details.

**Proof of Theorem 7.** Fix any scheduler $\sigma$ and initial valuation $v$ for our simple while loop. Let $T$ be the random variable that measures the number of loop iterations. By our assumption, $\mathbb{E}(T) < \infty$ under $\sigma$. Define the following sequences of (vectors of) random variables:

- $\mathbf{v}_0, \mathbf{v}_1, \ldots$ where each $\mathbf{v}_n$ represents the valuation before the $n$th loop iteration of the while loop (so that $\mathbf{v}_0 = v$);
- $\mathbf{u}_0, \mathbf{u}_1, \ldots$ where each $\mathbf{u}_n$ represents the sampled valuation for the $n$th loop iteration of the whole loop;
- $\mathbf{t}_0, \mathbf{t}_1, \ldots$ where each $\mathbf{t}_n$ represents the nondeterministic choice from the scheduler for the $n$th loop iteration.

We also recall the random variables $C_0, C_1, \ldots$ where each $C_n$ represents the cost/reward during the $n$th iteration.

By the execution of the loop, we have:

$$
\mathbf{v}_{n+1} = \begin{cases} 
F_n(\mathbf{v}_n, \mathbf{u}_n) & \text{if } \mathbf{v}_n \models \phi \\
\mathbf{v}_n & \text{otherwise}
\end{cases},
$$

$$
C_n = \begin{cases} 
R(\mathbf{u}_n, \mathbf{t}_n) & \text{if } \mathbf{v}_n \models \phi \\
0 & \text{otherwise}
\end{cases}.
$$

We also have that $T = \min\{n \mid \mathbf{v}_n \models \neg \phi\}$. Then we define the stochastic process $Y_0, Y_1, \ldots$ by:

$$
Y_n := h(\mathbf{v}_n) + \sum_{m=1}^{n-1} C_m.
$$

We accompany $Y_0, Y_1, \ldots$ with the filtration $\mathcal{F}_0, \mathcal{F}_1, \ldots$ such that each $\mathcal{F}_n$ is the smallest sigma-algebra that makes all random variables from $\mathbf{v}_0, \mathbf{v}_1, \mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{n-1}$ and $\mathbf{t}_0, \mathbf{t}_1, \ldots, \mathbf{t}_{n-1}$ measurable. Hence,

$$
\mathbb{E}(Y_{n+1}|\mathcal{F}_n) = \mathbb{E}(Y_n + h(\mathbf{v}_n + 1) - h(\mathbf{v}_n) + C_n|\mathcal{F}_n)
$$

$$
= Y_n + \mathbb{E}(h(\mathbf{v}_n + 1) + C_n|\mathcal{F}_n) - h(\mathbf{v}_n)
$$

$$
= Y_n - h(\mathbf{v}_n) + 1_{\mathbf{v}_n \models \phi} \cdot h(\mathbf{v}_n) +
$$

$$
1_{\mathbf{v}_n \models \neg \phi} \cdot \left[ \mathbb{E}(h(F_{\mathbf{t}_n}(\mathbf{v}_n, \mathbf{u}_n))|\mathcal{F}_n) + \mathbb{E}(R(\mathbf{u}_n, \mathbf{t}_n)|\mathcal{F}_n) \right]
$$

$$
= Y_n - h(\mathbf{v}_n) + 1_{\mathbf{v}_n \models \phi} \cdot h(\mathbf{v}_n)
$$

$$
+ 1_{\mathbf{v}_n \models \neg \phi} \cdot \left[ \mathbb{E}_u(h(F_{\mathbf{t}_n}(\mathbf{v}_n, \mathbf{u}))) + \mathbb{E}_u(R(\mathbf{u}, \mathbf{t}_n)) \right]
$$

$$
\leq Y_n \quad \text{(by (C3))}
$$

where $1_{\mathbf{v}_n \models \phi}$ is the random variable such that

$$
1_{\mathbf{v}_n \models \phi} = \begin{cases} 
1 & \text{if } \mathbf{v}_n \models \phi \\
0 & \text{otherwise}
\end{cases}
$$

and $1_{\mathbf{v}_n \models \neg \phi} = 1 - 1_{\mathbf{v}_n \models \phi}$. Hence, $\{Y_n\}_{n \in \mathbb{N}}$ is a supermartingale. Moreover, we have from (C4) that $|Y_{n+1} - Y_n| \leq M + R_{max}$. Thus, by applying Optional Stopping Theorem, we obtain immediately that $\mathbb{E}(Y_T) \leq \mathbb{E}(Y_0)$. By definition,

$$
Y_T = h(\mathbf{v}_T) + \sum_{m=1}^{T-1} C_m.
$$

It follows from (C2) that $\mathbb{E}(C_\infty) = \mathbb{E}(\sum_{m=1}^{T-1} C_m) \leq \mathbb{E}(Y_0) - K = h(v) - K$. Since the scheduler $\sigma$ is chosen arbitrarily, we obtain that $supval(v) \leq h(v) - K$.

**Lower bounds**

For the lower bound, we have the following definition:

**Definition 5 (Linear Lower Potential Functions (LLPFs)).** A linear lower potential function (LLPF) is a function $h: \mathbb{R}^{|X|} \to \mathbb{R}$ that satisfies (C1),(C2),(C4) and in addition (C3') (instead of (C3)) as follows:

(C3') there exists $\ell \in N$ such that

$$
h(v) \leq \mathbb{E}_u(h(F_\ell(v, u))) + \mathbb{E}_u(R(u, \ell))
$$

for all $v$ satisfying $v \models \phi$.

Similar to Theorem 1 we obtain the following result on lower bounds for $supval(*)$.

**Theorem 2.** If $h$ is an LLPF, then $supval(v) \geq h(v) - K'$ for all valuations $v \in \mathbb{R}^{|X|}$ such that $v \models \phi$.

**Proof.** This theorem can be proved in the same manner as Theorem 1.

4 Computational Results

By Theorem 1 and Theorem 2 to obtain tight upper and lower bounds for the SSP problem, we need an algorithm to obtain good LUPFs and LLPFs, respectively. We present the results for upper and lower bounds separately.

4.1 Computational Approach for Upper Bound

The key steps to obtain an algorithmic approach is as follows: (i) we first establish a linear template with unknown coefficients for a LUPF from (C1); (ii) then we transform logical conditions (C2)–(C4) equivalently into inclusion assertions between polyhedrons; (iii) next we transform inclusion assertions into linear inequalities through Farkas’ Lemma; (iv) finally we solve the linear inequalities through linear programming, where the solution for unknown coefficients in the template synthesizes a concrete LUPF that serves as an upper bound for the sup-value. Below we recall the well-known Farkas’ Lemma.

**Theorem 3 (Farkas’ Lemma [Farkas, 1894]).** Let $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$ and $d \in \mathbb{R}$. Suppose that $\{x \in \mathbb{R}^n \mid Ax \leq b\} \neq \emptyset$. Then

$$
\{x \in \mathbb{R}^n \mid Ax \leq b\} \subseteq \{x \in \mathbb{R}^n \mid c^T x \leq d\}
$$

iff there exists $y \in \mathbb{R}^m$ such that $y \geq 0$, $A^T y = c$ and $b^T y \leq d$.

Intuitively, Farkas’ Lemma transforms the inclusion problem of a nonempty polyhedron within a halfspace into a feasibility problem of a system of linear inequalities. As a result, one can decide the inclusion problem in polynomial time through linear programming.
The Algorithm

1. **Template.** The algorithm sets up a column vector $a$ of $|X|$ fresh variables and a fresh scalar variable $b$ such that the template for an LUPF $h$ is $h(v) = a^T \cdot v + b$.

2. **Constraints on $a$ and $b.$** The algorithm first encodes the condition (C2) for the template $h$ as the inclusion assertion

$$\{(v, u) \mid v \models \phi, F_i(v, u) \models \neg \phi\}$$

$$\subseteq \{(v, u) \mid K < a^T \cdot F_i(v, u) + b \leq K'\}$$

parameterized with $a, b, K, K'$ for every $\ell \in N$, where $K, K'$ are fresh unknown constants. Then for every $\ell \in N$, the algorithm encodes (C3) as

$$\{v \mid v \models \phi\} \subseteq \{v \mid c^T \cdot v \leq d_\ell\}$$

where $c_\ell, d_\ell$ are unique linear combinations of unknown coefficients $a, b$ satisfying that $c^T_\ell \cdot v \leq d_\ell$ is equivalent to $h(v) \geq \mathbb{E}_u (h(F_i(v, u))) + \mathbb{E}_u (R_i(v, u, \ell))$. Finally, the algorithm encodes (C4) as inclusion assertions with a fresh unknown constant $M$ using similar transformations. All the inclusion assertions (with parameters $a, b, K, K', M$) are grouped conjunctively so that these inclusions should all hold.

3. **Applying Farkas’ Lemma.** The algorithm applies Farkas’ Lemma to all the inclusion assertions generated in the previous step and obtains a system of linear inequalities involving the parameters $a, b, K, K', M$.

4. **Constraint Solving.** The algorithm calls a linear programming solver on the linear program consisting of the system of linear inequalities generated in the previous step and the minimizing objective function $a^T \cdot v_0 + b - K$ where $v_0$ is an initial valuation for program variables.

**Correctness and running time.** The above algorithm obtains concrete values for $a, b, K, K', M$ and leads to a concrete LUPF $h$. The correctness that $a^T \cdot v + b - K$ is an upper bound for the sup-value follows from Theorem[1]. The main optimization solution required by the algorithm is linear programming, and thus our algorithm runs in polynomial time in the size of the input succinct MDP.

**Theorem 4.** Given a succinct MDP and the SSP problem, the best linear upper bound (wrt an initial valuation) on the sup-value can be computed in polynomial-time in the implicit description of the MDP.

**Example 4.** Consider the Gambler’s Ruin example (from Section[2.3] Figure[7]).

Let $h : \mathbb{R} \to \mathbb{R}$ be an LUPF for this example, we have:

(C1) $\exists \lambda_1, \lambda_2 \in \mathbb{R} \forall x \in \mathbb{R} \ h(x) = \lambda_1 x + \lambda_2$

(C2) $\exists K, K' \in \mathbb{R} \forall x \in [1, 2] \ K \leq h(x) \leq K'$

(C3) $\forall x \in [1, \infty) \ h(x) \geq 0.4 \cdot (1 + h(x+1)) + 0.6 \cdot h(x-1)$

(C4) $\exists M \in [0, \infty) \forall x \in [1, \infty) \ h(x) - h(x - 1) \leq M$

Note that for condition (C2) we need to quantify over $x \in [1, 2]$, as if $x$ is not in this range, then the loop does not terminate in the next iteration. Given condition (C1), the two (C4) conditions are equivalent to $M \geq \lambda_1$. Also, the (C2) condition is equivalent to $K \leq \min \{h(1), h(3)\}$ and $K' \geq \max \{h(1), h(2)\}$ or more precisely $K \leq \lambda_1 + \lambda_2$, $K \leq 2\lambda_1 + \lambda_2$, $K' \geq \lambda_1 + \lambda_2$ and $K' \geq 2\lambda_1 + \lambda_2$. By expanding the occurrences of $h$ in the first (C3) condition and simplifying, we get $\forall x \in [1, \infty) \ 0 \geq 0.4 - 0.2\lambda_1$ and we can drop the quantification given that $x$ does not appear. Similarly, the second (C3) condition is equivalent to $0 \geq 0.3 - 0.4\lambda_1$.

In our method, such equivalences are automatically obtained by applying Farkas’ Lemma rather than manual inspection of the inequalities. Now that all necessary conditions are replaced by equivalent linear inequalities, we can solve the linear program to find an LUPF. An optimal answer (with minimal $\lambda_1$) is the following: $\lambda_1 = M = 2, \lambda_2 = K = 0, K' = 4$. Therefore by Theorem[1] we have $\supval(x_0) \leq 2x_0$ for all initial valuations $x_0$ that satisfy the loop guard.

**Remark 3.** Note that our approach is applicable to succinct MDPs with integer as well as real-valued variables, (i.e., the underlying state-space of the MDP is infinite). Even when we consider integer variables, since $h$ gives upper bounds, the reduction is to linear programming, rather than integer linear programming. Note that our approach only depends on expectation of sampling variables, and thus applicable even to continuous sampling variables with same expectation, e.g., our results apply uniformly to Example[2] and Example[3] given that the sampling variable $\tau$ has same expectation.

### 4.2 Computational Approach for Lower Bound

The algorithm for lower bound is similar to the upper bound, however, there are some subtle and key differences. An important difference is that while in Step 2 of the algorithm for upper bound, there is a conjunction of constraints, for the lower bound problem it requires a disjunction. This has two important implications: first, we need to consider a generalization of Farkas’ lemma and in this case we use Motzkin’s Transposition Theorem (which extends Farkas’ Lemma with strict inequalities) and second, instead of linear programming we require quadratic programming.

**Theorem 5 (Motzkin’s Transposition Theorem [Motzkin, 1936]).** Let $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m, c \in \mathbb{R}^n$. Assume that $\{x \in \mathbb{R}^n \mid Ax \leq b\} \neq \emptyset$. Then

$$\{x \in \mathbb{R}^n \mid Ax \leq b\} \cap \{x \in \mathbb{R}^n \mid Bx < c\} = \emptyset$$

iff there exist $y \in \mathbb{R}^m$ and $z \in \mathbb{R}^k$ such that $y, z \geq 0, A^T \cdot y + B^T \cdot z = 0$ and $b^T \cdot y + c^T \cdot z \leq 0$.

**The Algorithm LowerBound.** The algorithm here is similar to UpperBound described previously. For the sake of brevity, we explain the differences only.

1. **Template $h$.** Same as in the Algorithm UpperBound.
2. **Constraints on $a$ and $b$.** The algorithm first encodes (C2) and (C4) as inclusion assertions in the same way as in UpperBound and transforms them into linear inequalities over $a, b, K, K', M$ through Farkas’ Lemma. Then
the algorithm transforms (C3') equivalently into the inclusion assertion

\[ \{ v \mid v \models \phi \} \subseteq \bigcup_{\ell \in N} \{ v \mid c^T \ell \cdot v \leq d_{\ell} \} \]

where \( c_{\ell}, d_{\ell} \) are determined in the same way as in UpperBound. Furthermore, this inclusion assertion is equivalently written as

\[ \{ v \mid v \models \phi \} \cap \bigcap_{\ell \in N} \{ v \mid c^T_{\ell} \cdot v > d_{\ell} \} = \emptyset \]

and then transformed into a system of quadratic inequalities over \( a \) and \( b \) through Motzkin’s Transposition Theorem. The system may involve quadratic inequalities since \( c_{\ell} \) contains the unknown parameters \( a \) and \( b \).

### 3. Constraint Solving

The algorithm calls a nonlinear-programming solver on the system of linear and quadratic inequalities generated in the previous step with the maximizing objective function \( a^T \cdot v_0 + b - K' \) where \( v_0 \) is an appropriate initial valuation.

**Correctness and optimization problem.** As in the upper bound algorithm, once \( a, b, K, K', M \) are found, we obtain an concrete lower bound \( a^T \cdot v + b - K' \) for the sup-value from Theorem \( \ref{th:correctness_upper} \) establishing the correctness of our algorithm. The reduction leads to a quadratic optimization problem of polynomial size in the size of the succinct MDP, implying the following result.

**Theorem 6.** Given a succinct MDP and the SSP problem, the best linear lower bound (wrt the initial valuation) on the sup-value can be computed via a polynomial (in the implicit description of the MDP) reduction to quadratic programming.

**Example 5.** Let \( h : \mathbb{R} \to \mathbb{R} \) be an LLPF for the Gambler’s Ruin example (Figure \( \ref{fig:ruin} \) in Section \( \ref{sec:example} \)). The conditions of the form \( (C1), (C2) \) and \( (C4) \) are exactly the same as in Example \( \ref{ex:ruin} \). In addition, \( h \) must also satisfy the following condition:

\[ (C3') \quad \forall x \in [1, \infty) \]

\[ h(x) \leq 0.4 \cdot (1 + h(x + 1)) + 0.6 \cdot h(x - 1) \]

or

\[ h(x) \leq 0.3 \cdot (1 + h(x + 1)) + 0.7 \cdot h(x - 1) \]

Expanding the occurrences of \( h \) in the condition above using \( h(x) = \lambda_1 x + \lambda_2 \), and discarding the quantification, we obtain the following equivalent disjunctive system of inequalities: \( 0 \leq 0.4 - 0.2\lambda_1 \) or \( 0 \leq 0.3 - 0.4\lambda_1 \). This system is obviously equivalent to \( \lambda_1 \leq 2 \). Note that in general we have disjunction of linear inequalities, which require quadratic programming. As explained previously, such equivalences are automatically obtained by our algorithm using Motzkin’s transposition theorem.

Adding the equivalent linear forms of conditions \( (C1), (C2) \) and \( (C4) \) as in Example \( \ref{ex:ruin} \) and considering the resulting linear program with the objective of maximizing \( \lambda_1 \) leads to the following solution: \( \lambda_1 = M = 2, \lambda_2 = -2, K = 0, K' = 2 \).

Therefore, by Theorem \( \ref{th:correctness_upper} \), \( \text{supval}(x_0) \geq 2x_0 \) for every initial valuation \( x_0 \) that satisfies the loop guard. Given that this is the exact same upper bound we found in Example \( \ref{ex:ruin} \), the bound is tight.

### 4.3 Remarks

**Remark 4 (Sampling Distributions).** For simplicity, we only considered discrete and uniform sampling distributions in our examples in this paper. However, as mentioned in Remark \( \ref{rem:sampling} \), since we only use the mean of random variables, our approach extends to other sampling distributions with known means.

**Remark 5 (Dependence of Rewards on Program Variables).** Our proof (of Theorem \( \ref{th:correctness_upper} \) that is the basis of all results) depends on the Optional Stopping Theorem that requires bounded difference in all steps. We considered that the rewards depend only on sampling variables and nondeterminism. This assumption is sufficient, but not always necessary, to ensure bounded difference. Our approach is also applicable if the rewards depend on program variables, provided that they remain bounded.

**Remark 6.** Our approach computes the best linear upper and lower bounds wrt the initial valuations. However, a succinct MDP might collect logarithmic reward. For example, the succinct MDP shown in Figure \( \ref{fig:log} \) leaves the variable \( x \) until it becomes less than or equal to 1, and has a unit reward at each step. Hence, it collects \( \log x \) reward in total. In such cases, the obtained bounds, which are linear, can be arbitrarily bad.

\[ \text{while } x \geq 1 \text{ do} \]
\[ \text{ } x := x/2 \quad \text{reward}=1 \]

*Figure 2: A succinct MDP with logarithmic total reward*

**Future work.** While in this work we consider succinct MDPs, which are special case of factored MDPs, extending our approach to factored MDPs with linear dependency on the variables, but without restriction on nesting structure of while-loops, is an interesting direction of future work.

### 5 Case Studies and Experiments

We present more case studies and our experimental results.

#### 5.1 Additional Examples

We consider several other classical problems in probabilistic planning that can be described as succinct MDPs. We provide examples of Robot Planning and two variants of Roulette as typically played in casinos.

**Two-dimensional Robot Planning.** Consider a robot that is placed on an initial point \((x_0, y_0)\) of a two-dimensional grid. A player controls the robot and at each step, can order it to move one unit in either direction (left, right, up, down). However, the robot is not perfect. It follows the order with probability \( p < 1 \) and ignores it and goes to the left with probability \( 1 - p \). The process ends when the robot leaves the \( x \geq y \) half-plane and the player’s objective is to keep the robot in this half-plane. The player gets a reward of 1 each.
Multi-robot Planning. Our approach can handle as many variables as necessary and is only polynomially dependent on the succinct representation of the MDP. To demonstrate this, we consider a scenario similar to the previous case, in which there are now two robots $r_1$ and $r_2$ starting at positions $(x_0, y_0)$ and $(x′_0, y′_0)$. The robot $r_1$ follows the orders with probability $p_1$ and malfunctions and goes right with probability $1 - p_1$. Similarly, $r_2$ follows the commands with probability $p_2$ and goes left with probability $1 - p_2$. The player’s goal is to keep $r_1$ to the left of $r_2$, i.e. to keep the robots in the four-dimensional half-space $x \leq x′$. The process ends when the robots leave this half-space and the player gets a reward of 1 per step as long as the process has not terminated.

Mini-roulette. We model Mini-roulette which is a popular casino game based on a 13-slot wheel. A player starts the Mini-roulette.

We implemented our approach in Java and obtained experimental results the upper and lower bounds are tight. Thus our approach provides precise bounds on the SSP problem for several classical examples. Second, our results apply to infinite-state MDPs: in all the above examples, we consider infinite-state MDPs, where algorithmic approaches for factored MDPs do not work. Finally, in the above examples, instead of infinite-state MDPs if we consider large finite-state MDP variants (e.g., bounding the variable $x$ in Gambler’s Ruin with a large domain), then as the MDP becomes larger, the SSP value of the finite-state MDP approaches the infinite-state value. Since we provide tight bounds on the SSP value for this infinite-state limit, our approach provides efficient approximation even for large finite-state MDP variants.

5.2 Experimental Results

We implemented our approach in Java and obtained experimental results on all examples mentioned previously. The results are summarized in Table 3 where “Parameters” shows concrete parameters for our examples (where for the last two examples there is no parameter), “Upper bound” (resp. “Lower bound”) presents the LUPFs (resp. LLPFs) obtained through our approach, and finally “Time” shows the running time in milliseconds. The reported upper bounds on sup-values are the results of $h(v) - K$ as in Theorem 1. Similarly, the reported lower bounds on sup-values are obtained from $h(v) - K′$ as in Theorem 2. All initial valuations lead to the same results in our experiments. Finally, our approach is not sensitive to parameters as varying parameters will only change coefficients of our LUPFs/LLPFs.

Runtime and Platform. Note that our approach is extremely efficient and can handle all these MDPs, even those with large succinct representation such as the American Roulette MDP, in less than a second. The results were obtained on a machine with Intel Core i5-2520M dual-core processor (2.5GHZ), running Ubuntu Linux 16.04.3 LTS. We used Ip-solve [Berkelaar et al., 2004], JavaLP [Lukasiewycz, 2008] and JOptimizer [Tivellato, 2017] for solving linear and quadratic optimization tasks.

Significance of our results. First, observe that in most experimental results the upper and lower bounds are tight. Thus our approach provides precise bounds on the SSP problem for several classical examples. Second, our results apply to infinite-state MDPs: in all the above examples, we consider infinite-state MDPs, where algorithmic approaches for factored MDPs do not work. Finally, in the above examples, instead of infinite-state MDPs if we consider large finite-state MDP variants (e.g., bounding the variable $x$ in Gambler’s Ruin with a large domain), then as the MDP becomes larger, the SSP value of the finite-state MDP approaches the infinite-state value. Since we provide tight bounds on the SSP value for this infinite-state limit, our approach provides efficient approximation even for large finite-state MDP variants.

6 Related Works

MDPs. MDPs have been studied quite widely and deeply in the AI literature [Sigaud and Buffet, 2010] [Dean et al., 1997] [Singh et al., 1994; Williams and Young, 2007 Poupart et al., 2015] [Gilbert et al., 2017] [Topin et al., 2015] [Perrault and Boutilier, 2017] Boutilier and Lu, 2016 Ferns et al., 2004]; and factored MDPs have also been considered as an efficient algorithmic approach [Guestrin et al., 2003]. In this line of work, we
introduce succinct MDPs and efficient algorithms for them which are applicable to several problems in AI.

**PPDDL and RDDL.** There are a variety of languages for specifying MDPs and especially factored MDPs. Two of the most commonly used are PPDDL [Younes and Littman, 2004] and RDDL [Sanner, 2010]. These languages are general languages that capture all factored MDPs. Instead, we consider succinct MDPs where guards and assignments are linear expressions and we do not consider nested while-loops, and thus our language is simpler than PPDDL and RDDL.

**Programming language results.** Besides the AI community, research in programming languages also considers probabilistic programs and algorithmic approaches [Chakarov and Sankaranarayanan, 2013; Chatterjee et al., 2016]: but the main focus is termination with probability 1 or in finite time, whereas we consider the SSP problem and compute precise bounds for it. While both approaches use the theory of martingales as the underlying mathematical tool, the key differences are as follows:

- **Problem difference:** The work of [Chatterjee et al., 2016] considers the number of steps to termination. There is no notion of rewards or stochastic shortest path (SSP). In contrast, we consider rewards and SSP. In particular, we have negative rewards that cannot be modeled by the notion of steps.

- **Result difference:** [Chatterjee et al., 2016] considers the qualitative question of whether expected termination time is finite or not, and then applies Azuma’s inequality to martingales for concentration bounds. In contrast, we present upper and lower bounds on expected SSP. Thus our results provide quantitative (rather than qualitative) bounds for expected SSP that significantly generalize expected termination time. However, our results are applicable to a more restricted class of programs.

- **Proof-technique difference:** [Chatterjee et al., 2016] considers qualitative expected termination time characterization, and the main mathematical tool is martingale convergence that does not handle negative rewards. In contrast, we present quantitative bounds for SSP and our mathematical tool is Optional Stopping Theorem.

**Comparison with [Hansen and Abdoulahi, 2015].** This work provides convergence tests for heuristic search value-iteration algorithms. While both approaches provide bounds for SSPs, the main differences are as follows: (i) our approach can handle negative costs whereas [Hansen and Abdoulahi, 2015] can handle only positive costs; (ii) our results are on the implicit representation of MDPs, while [Hansen and Abdoulahi, 2015] evaluates parts of the explicit MDP; and (iii) our approach presents polynomial reductions to optimization problems and is not dependent on value-iteration.

7 Conclusion

We consider succinct MDPs, which can model several classical examples from the AI literature, and present algorithmic approaches for the SSP problem on them. There are several interesting directions for future work. The first direction would be to consider other algorithmic approaches for succinct MDPs. In our work, we consider linear templates for efficient algorithmic analysis. Generalization of our approach to more general templates is another interesting direction for future work. Finally, whether our approach can be extended to other class of MDPs is also an interesting problem to investigate.

**References**

[Berkelaar et al., 2004] M. Berkelaar, K. Eikland, P. Notebaert, et al. LpSolve: Open source (mixed-integer) linear programming system. Eindhoven U. of Technology, 2004.

[Bertsekas, 2005] D. P. Bertsekas. Dynamic programming and optimal control, 3rd Ed. Athena Scientific, 2005.

[Boutilier and Lu, 2016] C. Boutilier and T. Lu. Budget allocation using weakly coupled, constrained Markov decision processes. In UAI, 2016.

[Chakarov and Sankaranarayanan, 2013] A. Chakarov and S. Sankaranarayanan. Probabilistic program analysis with martingales. In CAV, 2013.

[Chatterjee et al., 2016] K. Chatterjee, H. Fu, P. Novotný, and R. Hasheminezhad. Algorithmic analysis of qualitative and quantitative termination problems for affine probabilistic programs. In POPL, 2016.

[Dean et al., 1997] T. Dean, R. Givan, and S. Leach. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In UAI, 1997.

[Delgado et al., 2011] K. Delgado, S. Sanner, and L. Nunes de Barros. Efficient solutions to factored MDPs with imprecise transition probabilities. Artif. Intell., 175(9-10):1498–1527, 2011.

[Farkas, 1894] J. Farkas. A Fourier-Féle mechanikai elv alkalmazásai (Hungarian). Matematikai és Természettudományi Értesítő, 12:457–472, 1894.

[Ferns et al., 2004] N. Ferns, P. Panangaden, and D. Precup. Metrics for finite Markov decision processes. In UAI, 2004.

[Filar and Vrieze, 1997] J. Filar and K. Vrieze. Competitive Markov Decision Processes. Springer, 1997.

[Gilbert et al., 2017] H. Gilbert, P. Weng, and Y. Xu. Optimizing quantiles in preference-based Markov decision processes. In AAAI, 2017.
