The Odds of Staying on Budget

Christoph Haase and Stefan Kiefer

1 Laboratoire Spécification et Vérification (LSV), CNRS
École Normale Supérieure (ENS) de Cachan, France
2 Department of Computer Science
University of Oxford, UK

Abstract

Given Markov chains and Markov decision processes (MDPs) whose transitions are labelled with non-negative integer costs, we study the probability of paths whose accumulated cost satisfy a Boolean combination of inequalities. We investigate the computational complexity of deciding whether this probability exceeds a given threshold. For acyclic Markov chains, we show that this problem is PP-complete, whereas it is hard for the PosSLP problem and in PSPACE for general Markov chains. Moreover for acyclic and general MDPs, we prove PSPACE- and EXPTime-completeness, respectively. Our results significantly improve the state of the art of the complexity of computing reward quantiles in succinctly represented stochastic systems.

1998 ACM Subject Classification  F.1.1 Models of Computation

Keywords and phrases quantitative verification, Markov processes, quantiles, succinct representation, counter systems

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Computing the shortest path from \(s\) to \(t\) in a directed graph is a ubiquitous problem in computer science, so shortest-path algorithms such as Dijkstra’s algorithm are a staple for every computer scientist. These algorithms work in polynomial time even if the edges are weighted, thus questions of the following kind are easy to answer:

(I) Is it possible to travel from Lyon to Munich in less than 8 hours?

From a complexity-theoretic point of view, even computing the length of the shortest path lies in NC, the class of problems with “efficiently parallelisable” algorithms.

The shortest-path problem becomes more intricate as soon as uncertainties are taken into account. For example, additional information such as “there might be a traffic jam around Zürich, so the Zürich route will, with probability 10%, trigger a detour of 40 minutes” naturally leads to questions of the following kind:

(II) Is there a travel plan that avoids trips longer than 8 hours with probability \(\geq 0.9\)?

Markov decision processes (MDPs) are the established model to formalise problems such as (II). In each state of an MDP some actions are enabled, each of which is associated with a probability distribution over outgoing transitions. Each transition, in turn, determines the successor state and is equipped with a non-negative “weight”. The weight could be interpreted...
as time, distance, reward, or—as in this paper—as cost. For another example, imagine the plan of a research project whose workflow can be modelled by a directed weighted graph. In each project state the investigators can hire a programmer, travel to collaborators, acquire new equipment, etc., but each action costs money, and the result (i.e., the next project state) is probabilistic. The objective is to meet the goals of the project before exceeding its budget for the total accumulated cost. This leads to questions such as:

(III) Is there a strategy to stay on budget with probability $\geq 0.85$?

MDP problems like (II) and (III) become even more challenging when each transition is equipped with both a cost and a utility, e.g., in order to model problems that aim at maximising the probability that both a given budget is kept and a minimum total utility is achieved. Such cost-utility trade-offs have recently been studied in [4].

The problems (II) and (III) may become easier if there is no non-determinism, i.e., there are no actions. We then obtain Markov chains where the next state and the incurred transition cost are chosen in a purely probabilistic fashion. Referring to the project example above, the activities may be completely planned out, but their effects (i.e. cost and next state) may still be probabilistic, yielding problems of the kind:

(IV) Will the budget be kept with probability $\geq 0.85$?

Closely related to the aforementioned decision problems is the following optimisation problem, referred to as the quantile query in [2, 4, 22]. A quantile query asked by a funding body, for instance, could be the following:

(V) Given a probability threshold $\tau$, compute the smallest budget that suffices with probability at least $\tau$.

Non-stochastic problems like (I) are well understood. The purpose of this paper is to investigate the complexity of MDP problems such as (II) and (III), of Markov-chain problems such as (IV), and of quantile queries like (V). More formally, the models we consider are Markov chains and MDPs with non-negative integer costs, and the main focus of this paper is on the cost problem for those models: Given a budget constraint $\varphi$ represented as a Boolean combination of linear inequalities and a probability threshold $\tau$, we study the complexity of determining whether the probability of paths reaching a designated target state with cost consistent with $\varphi$ is at least $\tau$.

In order to highlight and separate our problems more clearly from those in the literature, let us briefly discuss two approaches that do not, at least not in an obvious way, resolve the core challenges. First, one approach to answer the MDP problems could be to compute a strategy that minimises the expected total cost, which is a classical problem in the MDP literature, solvable in polynomial time using linear programming methods [18]. However, minimising the expectation may not be optimal: if you don’t want to be late, it may be better to walk than to wait for the bus, even if the bus saves you time in average. The second approach with shortcomings is to phrase problems (II), (III) and (IV) as MDP or Markov-chain reachability problems, which are also known to be solvable in polynomial time. This, however, ignores the fact that numbers representing cost are commonly represented in their natural succinct binary encoding. Augmenting each state with possible accumulated costs leads to a blow-up of the state space which is exponential in the representation of the input, giving an \textsc{ExpTime} upper bound as in [4].

Our contribution. The main goal of this paper is to comprehensively investigate under which circumstances and to what extent the optimal complexity of the cost problem may be
lower than the \( \text{EXPTIME} \) upper bound. In particular, we distinguish between acyclic and general control graphs. We also provide new complementary lower bounds, much stronger than the best known \( \text{NP} \) lower bound derivable from [14]. In short, we show that the cost problem is

- \( \text{PP-complete} \) for acyclic Markov chains, and hard for the \( \text{PosSLP} \) problem and in \( \text{PSPACE} \) in the general case; and
- \( \text{PSpace-complete} \) for acyclic MDPs, and \( \text{EXPTIME-complete} \) for general MDPs.

In particular, the \( \text{PP} \) and \( \text{PSPACE} \) lower bounds in the acyclic cases already hold for quantile queries.

**Related Work.** The motivation for this paper comes from the work on quantile queries in [3, 4, 22] mentioned above and on model checking so-called durational probabilistic systems [14] with a probabilistic timed extensions of CTL. Answering quantile queries is, for instance, an essential task in systems engineering for the analysis and optimisation of real-time systems, see e.g. [9]. While the focus of [22] is mainly on “qualitative” problems where the probability threshold is either 0 or 1, an iterative linear-programming-based approach for solving quantile queries has been suggested in [3, 4]. The authors report satisfying experimental results, the worst-case complexity however remains exponential time. Settling the computational complexity of quantile queries has been identified as one of the current challenges in the conclusion of [4].

Recently, there has been considerable interest in models of stochastic systems that extend weighted graphs or counter systems. The work by Bruyère et al. [8] has also been motivated by the fact that minimising the expected total cost is not always an adequate solution to natural problems. For instance, they consider the problem of computing a scheduler in an MDP with positive integer weights that ensures that both the expected and the maximum incurred cost remain below some given values. Other recent work also investigated MDPs with a single counter ranging over the non-negative integers, see e.g. [6, 7]. However, in that work updates to the counter can be both positive and negative. For that reason, the analysis focuses on questions about the counter value zero, such as designing a strategy that maximises the probability of reaching counter value zero.

There is also a large body of work on the complexity of general MDPs, distinguishing between observable and unobservable MDPs, finite- and infinite-horizon problems, discounted and total reward, stationary and time-dependent schedulers, etc., see e.g. [15, 18] and the references therein. One of the main distinguishing features of our work is the focus on the distribution of accumulated reward/cost, rather than on (possibly discounted) expectations.

## 2 Preliminaries

We write \( \mathbb{N} = \{0, 1, 2, \ldots\} \). For a countable set \( X \) we write \( \text{dist}(X) \) for the set of probability distributions over \( X \); i.e., \( \text{dist}(X) \) consists of those functions \( f: X \to [0, 1] \) such that \( \sum_{x \in X} f(x) = 1 \).

**Markov Chains.** A Markov chain is a triple \( \mathcal{M} = (S, s_0, \delta) \), where \( S \) is a countable (finite or infinite) set of states, \( s_0 \in S \) is an initial state, and \( \delta: S \to \text{dist}(S) \) is a probabilistic transition function that maps a state to a probability distribution over the successor states. Given a Markov chain we also write \( s \xrightarrow{p} t \) or \( s \xrightarrow{} t \) to indicate that \( p = \delta(s)(t) > 0 \). A run is an infinite sequence \( s_0s_1 \cdots \in \{s_0\}^\omega \) with \( s_i \xrightarrow{} s_{i+1} \) for \( i \in \mathbb{N} \). We write \( \text{Run}(s_0 \cdots s_k) \) for the set of runs that start with \( s_0 \cdots s_k \). To \( \mathcal{M} \) we associate the standard probability space \( (\text{Run}(s_0), \mathcal{F}, \mathcal{P}) \) where \( \mathcal{F} \) is the \( \sigma \)-field generated by all basic cylinders \( \text{Run}(s_0 \cdots s_k) \).
with $s_0 \cdots s_k \in \{s_0\}S^*$, and $\mathcal{P} : \mathcal{F} \to [0,1]$ is the unique probability measure such that $\mathcal{P}(Ran(s_0 \cdots s_k)) = \prod_{i=1}^k \delta(s_{i-1})(s_i)$.

**Markov Decision Processes.** A Markov decision process (MDP) is a tuple $\mathcal{D} = (S, s_0, A, En, \delta)$, where $S$ is a countable set of states, $s_0 \in S$ is the initial state, $A$ is a finite set of actions, $En : S \to 2^A \setminus \emptyset$ is an action enabledness function that assigns to each state $s$ the set $En(s)$ of actions enabled in $s$, and $\delta : S \times A \to dist(S)$ is a probabilistic transition function that maps a state $s$ and an action $a \in En(s)$ enabled in $s$ to a probability distribution over the successor states. A (deterministic, memoryless) scheduler for $\mathcal{D}$ is a function $\sigma : S \to A$ with $\sigma(s) \in En(s)$ for all $s \in S$. A scheduler $\sigma$ induces a Markov chain $\mathcal{M}_\sigma = (S, s_0, \delta_\sigma)$ with $\delta_\sigma(s) = \delta(s, \sigma(s))$ for all $s \in S$. We write $\mathcal{P}_\sigma$ for the corresponding probability measure of $\mathcal{M}_\sigma$.

**Cost Processes.** A cost process is a tuple $\mathcal{C} = (Q, q_0, t, A, En, \Delta)$, where $Q$ is a finite set of control states, $q_0 \in Q$ is the initial control state, $t$ is the target control state, $A$ is a finite set of actions, $En : Q \to 2^A \setminus \emptyset$ is an action enabledness function that assigns to each control state $q$ the set $En(q)$ of actions enabled in $q$, and $\Delta : Q \times A \to dist(Q \times \mathbb{N})$ is a probabilistic transition function. Here, for $q, q' \in Q$, $a \in En(q)$ and $k \in \mathbb{N}$, the value $\Delta(q,a)(q',k) \in [0,1]$ is the probability that, if action $a$ is taken in control state $q$, the cost process transitions to control state $q'$ and cost $k$ is incurred. For the complexity results we define the size of $\mathcal{C}$ as the size of a succinct description, i.e., the costs are encoded in binary, the probabilities are encoded as fractions of integers in binary, and for each $q \in Q$ and $a \in En(q)$, the distribution $\Delta(q,a)$ is described by the list of triples $(q',k,p)$ with $\Delta(q,a)(q',k) = p > 0$. Consider the directed graph $G = (Q, E)$ with

$$E := \{(q, q') \in (Q \setminus \{t\}) \times Q : \exists a \in En(q) \exists k \in \mathbb{N}. \Delta(q,a)(q',k) > 0\}.$$  

We call $\mathcal{C}$ acyclic if $G$ is acyclic (which can be determined in linear time).

A cost process $\mathcal{C}$ induces an MDP $\mathcal{D}_C = (Q \times \mathbb{N}, (q_0,0), A, En', \delta')$ with $En'(q,c) = En(q)$ for all $q \in Q$ and $c \in \mathbb{N}$, and $\delta'((q,c),a)(q',c') = \Delta(q,a)(q',c' - c)$ for all $q, q' \in Q$ and $c, c' \in \mathbb{N}$ and $a \in A$. For a state $(q,c) \in Q \times \mathbb{N}$ in $\mathcal{D}_C$ we view $q$ as the current control state and $c$ as the current cost, i.e., the cost accumulated thus far.

We may refer to $\mathcal{C}$ as a cost chain if $|En(q)| = 1$ holds for all $q \in Q$. In this case one can view $\mathcal{D}_C$ as a Markov chain: namely, as the Markov chain induced by the unique scheduler of $\mathcal{D}_C$. For cost chains actions are not relevant, so we may describe cost chains just by the tuple $\mathcal{C} = (Q, q_0, t, \Delta)$.

Recall that we restrict schedulers to be deterministic and memoryless, as such schedulers will be sufficient for the objectives in this paper. Note, however, that our definition allows schedulers to depend on the current cost, i.e., we may have schedulers $\sigma$ with $\sigma(q, c) \neq \sigma(q, c')$.

**The accumulated cost $K$.** In this paper we will be interested in the cost accumulated during a run before reaching the target state $t$. In order to avoid technicalities that are orthogonal to the main issues, we assume that $En(t) = \{a\}$ holds for some $a \in A$ and $\Delta(t,a)(t,0) = 1$. Hence, runs that visit $t$ will not leave $t$ and accumulate only a finite cost. Furthermore we assume that for all schedulers the target state $t$ is almost surely reached, i.e., for all schedulers the probability of eventually visiting a state $(t,c)$ with $c \in \mathbb{N}$ is equal to one. This condition can be verified by graph algorithms in quadratic time, e.g., by computing the maximal end components of the MDP obtained from $\mathcal{C}$ by ignoring the cost, see e.g. [5] Algorithm 47.

Given a cost process $\mathcal{C}$ we define a random variable $K_C : Ran((q_0, 0)) \to \mathbb{N}$ such that $K_C((q_0,0) (q_1,c_1) \cdots) = c$ if there exists $i \in \mathbb{N}$ with $(q_i,c_i) = (t,c)$. We often drop the
C. Haase and S. Kiefer

subscript from $K_C$ if the cost process $C$ is clear from the context. From the above-mentioned assumptions on $t$ it follows that for any scheduler the random variable $K$ is almost surely defined. We view $K(w)$ as the accumulated cost of a run $w$.

**The cost problem.** Let $x$ be a fixed variable. An atomic cost formula is an inequality of the form $x \leq B$ where $B \in \mathbb{N}$ is encoded in binary. A cost formula is an arbitrary Boolean combination of atomic cost formulas. A number $n \in \mathbb{N}$ satisfies a cost formula $\varphi$, in symbols $n \models \varphi$, if $\varphi$ is true when $x$ is replaced by $n$.

This paper mainly deals with the following decision problem: given a cost process $C$, a cost formula $\varphi$, and a probability threshold $\tau \in [0, 1]$, the cost problem asks whether there exists a scheduler $\sigma$ with $P_{\sigma}(K_C \models \varphi) \geq \tau$. The case of an atomic cost formula $\varphi$ is an important special case. Clearly, for cost chains $C$ the cost problem simply asks whether $P(K_C \leq B) \geq \tau$. One can assume $\tau = 1/2$ without loss of generality, thanks to a simple construction, see Proposition 10 in Appendix A. Moreover, with an oracle for the cost problem at hand, one can use binary search over $\tau$ to approximate $P_{\sigma}(K \models \varphi)$: $i$ oracle queries suffice to approximate $P_{\sigma}(K \models \varphi)$ within an absolute error of $2^{-i}$.

By our definition the MDP $D_C$ is in general infinite as there is no upper bound on the accumulated cost. However, when solving the cost problem, there is no need to keep track of costs above $B$, where $B$ is the largest number appearing in $\varphi$. So one can solve the cost problem in so-called pseudo-polynomial time (i.e., polynomial in $B$, not in the size of the encoding of $B$) by computing an explicit representation of a restriction, say $\bar{D}_C$, of $D_C$ to costs up to $B$, and then applying classical linear-programming techniques [18] to compute the optimal scheduler for $\bar{D}_C$. In terms of our succinct representation we have:

▶ **Fact 1.** The cost problem is in EXPTIME.

Heuristic improvements to this approach were suggested in [22, 2]. The subject of this paper is to investigate to what extent the EXPTIME complexity is optimal.

### 3 Quantile Queries

In this section we consider the following function problem, referred to as quantile query in [22, 2, 4]. Given a cost chain $C$ and a probability threshold $\tau \in [0, 1]$, a quantile query asks for the smallest budget $B$ such that $P_{\sigma}(K_C \leq B) \geq \tau$. We show that polynomially many oracle queries to the cost problem for atomic cost formulas “$x \leq B$” suffice to answer a quantile query. This can be done using binary search over the budget $B$. The following proposition, proved in Appendix [3], provides a suitable general upper bound on this binary search, by exhibiting a concrete sufficient budget, computable in polynomial time:

▶ **Proposition 2.** Suppose $0 \leq \tau < 1$. Let $p_{\min}$ be the smallest non-zero probability and $k_{\max}$ be the largest cost in the description of the cost process. Let

$$B := k_{\max} \cdot \left[ |Q| \cdot \left( -\ln(1 - \tau) / p_{\min}^{|Q|} + 1 \right) \right].$$

Then $P_{\sigma}(K \leq B) \geq \tau$ holds for all schedulers $\sigma$.

The case $\tau = 1$ is covered by [22, Theorem 6], where it is shown that one can compute in polynomial time the smallest $B$ with $P_{\sigma}(K \leq B) = 1$ for all schedulers $\sigma$, if such $B$ exists. Thus we can conclude that quantile queries are polynomial-time inter-reducible with the cost problem for atomic cost formulas.
In this section we consider the cost problems for acyclic and general cost chains. Even in the general case we obtain PSPACE membership, avoiding the EXPTIME upper bound from Fact 1.

### 4.1 Acyclic Cost Chains

The complexity class PP \[11\] can be defined as the class of languages \(L\) that have an NP Turing machine \(M_L\) such that for all words \(x\) one has \(x \in L\) if and only if at least half of the computation paths of \(M_L\) on \(x\) are accepting. The class PP includes NP \[11\]. Closely related is the function class \#P, which consists of those functions \(f\) for which there exists an NP Turing machine \(M_f\) such that for all words \(x\) the function value \(f(x)\) is equal to the number of accepting computation paths of \(M_f\) on \(x\). PP is remarkably powerful: Toda’s Theorem \[20\] states that \(P^{#P}\) (which equals \(P^{\#P}\)) includes the polynomial-time hierarchy \(PH\). We show that the cost problem for acyclic cost chains is PP-complete.

**Theorem 3.** The cost problem for acyclic cost chains is in PP. It is PP-hard with respect to polynomial-time Turing reductions, even for atomic cost formulas.

**Proof sketch.** Membership in PP is straightforward, using the characterisation \[11\] of PP as the class of languages \(L\) that have a probabilistic polynomial-time bounded Turing machine \(M_L\) such that for all words \(x\) one has \(x \in L\) if and only if \(M_L\) accepts \(x\) with probability at least 1/2.

For the lower bound, we show that a deterministic polynomial-time Turing machine can compute the number of satisfying assignments of a given Boolean formula, provided that the Turing machine has access to an oracle that answers queries to the cost problem for acyclic cost chains. This suffices to decide the canonical PP-complete problem \(\text{MajSAT}\) \[11, 17\], which asks if a given Boolean formula is satisfied by at least one half of its variable assignments. A technique from \[14\], which encodes a counting version of the \(\text{SUBSETSUM}\) problem as a Markov chain, plays an essential role in this reduction. We provide more details in Appendix C.

The lower bound, PP-hardness with respect to polynomial-time Turing reductions, strengthens the NP-hardness result from \[14\] substantially: it follows by Toda’s theorem that any problem in the polynomial-time hierarchy can be solved by a deterministic polynomial-time bounded Turing machine that has oracle access to the cost problem for acyclic cost chains.

### 4.2 General Cost Chains

We now turn towards general cost chains whose underlying control graph may contain cycles. For the PP upper bound in Theorem 3 the absence of cycles is essential, and it does not appear to be possible to retain this PP upper bound in the presence of cycles. However, we can use cycles in order to show hardness of the cost problem for PosSLP, indicating that the complexity of the general case is inherently different from the acyclic case.

**Theorem 4.** The cost problem for cost chains is in PSPACE and hard for PosSLP.

For membership in PSPACE we use the fact that probabilistic PSPACE equals PSPACE, which was first proved in \[19\]. The argument from the beginning of the proof sketch for Theorems 3 is then easily adapted to general cost chains, replacing PP with probabilistic PSPACE.
In the rest of the section we sketch the proof of the PosSLP lower bound. Full details for the proof of Theorem 4 are given in Appendix C.

The PosSLP problem, introduced in [I], asks, given a straight-line program or, equivalently, an arithmetic circuit with operators +, −, *, and inputs 0 and 1, and a designated output gate, whether it outputs a positive integer. The PosSLP problem is a fundamental problem for numerical computation [1]. It can be decided in PSPACE; in fact, it is shown in [I] that PosSLP can be decided in the 4th level of the counting hierarchy (CH), an analogue to the polynomial-time hierarchy for classes like PP.

In the following we formalise the notion of an arithmetic circuit and at the same time take advantage of a normal form that avoids gates labelled with “−”.

This normal form was established in the proof of [9, Theorem 5.2]. An arithmetic circuit is a directed acyclic graph $G = (V, E)$ whose leaves are labelled with constants “0” and “1”, and whose vertices are labelled with operators “+” and “∗”. Subsequently, we refer to the elements of $V$ as gates.

With every gate we associate a level starting at which may introduce cycles in the structure of with “1” is simulated by two control states connected via a single transition, whereas this is the case with leaves. For levels greater than zero, gates on odd levels are labelled with “+” and on even levels with “∗”. Moreover, all gates on a level greater than zero have exactly two incoming edges from the preceding level. The upper part of Figure 1 illustrates an arithmetic circuit in this normal form. We can associate with every gate $v \in V$ a non-negative integer $\text{val}(v)$ the number of paths in a deterministic finite-state automaton (DFA) $\Pi(G, q, q')$, the set of all paths starting in $q$ and ending in $q'$.

In this form, the PosSLP problem asks, given an arithmetic circuit $G = (V, E)$ and two gates $v_1, v_2 \in V$, whether $\text{val}(v_1) \geq \text{val}(v_2)$ holds.

For obtaining the PosSLP lower bound we first describe an intermediate step, which we believe is of independent interest: we show that we can obtain the value of a gate of an arithmetic circuit as the number of paths in a deterministic finite-state automaton (DFA) with a certain Parikh image.

Formally, let $\mathcal{A} = (Q, \Sigma, \Delta)$ be a DFA such that $Q$ is a finite set of control states, $\Sigma = \{a_1, \ldots, a_k\}$ is a finite alphabet, and $\Delta \subseteq Q \times \Sigma \times Q$ is the set of transitions. A path $\pi$ in $\mathcal{A}$ is a sequence of transitions $\pi = \delta_1 \cdots \delta_n \in \Delta^*$ such that $\delta_i = (q_i, a_i, q'_i)$ and $\delta_{i+1} = (q_{i+1}, a_{i+1}, q'_{i+1})$ implies $q'_i = q_{i+1}$ for all $1 \leq i < n$. Let $q, q' \in Q$, we denote by $\Pi(\mathcal{A}, q, q')$ the set of all paths starting in $q$ and ending in $q'$. In this paper, a Parikh function is a function $f : \Sigma \rightarrow \mathbb{N}$. The Parikh image of a path $\pi$, denoted $\text{parikh}(\pi)$, is the unique Parikh function counting for every $a \in \Sigma$ the number of times $a$ occurs on a transition in $\pi$.

Proposition 5. Let $G = (V, E)$ be an arithmetic circuit and $v \in V$. There exists a log-space computable DFA $\mathcal{A} = (Q, \Sigma, \Delta)$ with distinguished control states $q, q' \in Q$ and a Parikh function $f : \Sigma \rightarrow \mathbb{N}$ such that

$$\text{val}(v) = |\{\pi \in \Pi(\mathcal{A}, q, q') : \text{parikh}(\pi) = f\}|.$$

Proof sketch. We only sketch the proof idea, full details are deferred to the appendix. The structure of the graph obtained from the transitions of $\mathcal{A}$ is illustrated in Figure 1. The idea is to construct $\mathcal{A}$ inductively according to the levels of $G$. For level 0, a gate labelled with “1” is simulated by two control states connected via a single transition, whereas this transition is missing for a gate labelled with “0”. For higher levels, a gate labelled with “+” is simulated by branching into the inductively constructed gadgets corresponding to the gates this gate connects to. Likewise, a gate labelled with “∗” is simulated by sequentially composing the gadgets corresponding to the gates this gate connects to. This is the case which may introduce cycles in the structure of $\mathcal{A}$. By choosing appropriate alphabet symbols and Parikh functions for each level of $G$, the statement follows.

By modifying this construction we show in the appendix:
Proposition 6. Let \( G = (V, E) \) be an arithmetic circuit. Let \( v \in V \) be a gate on level \( \ell \) with odd \( \ell \). There exist a log-space computable cost process \( C \) and \( T \in \mathbb{N} \) with \( P(K_C = T) = \frac{\text{val}(v)}{m} \), where \( m = \exp_d(2^{(\ell-1)/2+1-1}) \cdot \exp_d(2^{(\ell-1)/2+1-3}) \).

Applying this proposition, we reduce PosSLP to the cost problem for cost chains as follows. Let \( G = (V, E) \) be the given arithmetic circuit with \( v_1, v_2 \in V \). Without loss of generality we assume that \( v_1, v_2 \) are on level \( \ell \in \mathbb{N} \) with odd \( \ell \). In the following we construct in logarithmic space a cost chain \( C \) and a cost formula \( \varphi \) such that

\[
\text{val}(v_1) \geq \text{val}(v_2) \iff P(K_C = T) \geq \frac{1}{2}.
\]

Using Proposition 6, we first construct two cost chains \( C_1 = (Q, q_1, t, \Delta) \) and \( C_2 = (Q, q_2, t, \Delta) \) and \( T_1, T_2 \in \mathbb{N} \) such that \( P(K_{C_i} = T_i) = \frac{\text{val}(v_i)}{m} \) holds for \( i \in \{1, 2\} \) and for \( m \in \mathbb{N} \) as given by Proposition 6. We compute a number \( H \in \mathbb{N} \) with \( H \geq T_2 \) such that \( P(K_{C_2} > H) < 1/m \). By Proposition 2 it suffices to take

\[
H \geq \max \left\{ T_2, k_{\text{max}} \cdot \left| Q \right| \cdot \left( \ln(m + 1)/p_{\text{min}} + 1 \right) \right\},
\]

where \( k_{\text{max}} \) and \( p_{\text{min}} \) refer to \( C_2 \). Let

\[
\varepsilon := P(K_{C_2} > H \land K_{C_2} \neq H + 1 + T_1).
\]

We have

\[
0 \leq \varepsilon \leq P(K_{C_2} > H) < 1/m.
\]

We combine \( C_1 \) and \( C_2 \) to a cost chain \( C = (Q \uplus \{q_0\}, q_0, t, \tilde{\Delta}) \), where \( \tilde{\Delta} \) extends \( \Delta \) by

\[
\tilde{\Delta}(q_0)(q_1, H + 1) = 1/2 \quad \text{and} \quad \tilde{\Delta}(q_0)(q_2, 0) = 1/2.
\]

By this construction the new cost chain \( C \) initially either incurs cost \( H + 1 \) and then emulates \( C_1 \), or incurs cost 0 and then emulates \( C_2 \). Those possibilities have probability 1/2 each. We define the cost formula

\[
\varphi := (x \leq T_2 - 1) \lor (T_2 + 1 \leq x \leq H) \lor (x = H + 1 + T_1).
\]
By the construction of $C$ and the definition of $\varepsilon$ we have:

$$
\mathcal{P}(K_C \models \varphi) \\
= \frac{1}{2} \cdot \mathcal{P}(K_{C_1} = T_1) + \frac{1}{2} \cdot (\mathcal{P}(K_{C_2} \leq H \lor K_{C_2} = H + 1 + T_1) - \mathcal{P}(K_{C_2} = T_2)) \\
= \frac{1}{2} \cdot \text{val}(v_1)/m + \frac{1}{2} \cdot (1 - \varepsilon - \text{val}(v_2)/m)
$$

It follows that we have $\mathcal{P}(K_C \models \varphi) \geq \frac{1}{2}$ if and only if $\text{val}(v_1)/m \geq \text{val}(v_2)/m + \varepsilon$. Since $0 \leq \varepsilon < 1/m$ and $\text{val}(v_1), \text{val}(v_2)$ are integer numbers, we have shown the equivalence \[1\], completing the proof of the lower bound.

Two remarks are in order: First, the representation of $m$ from Proposition \[6\] is of exponential size. However, the computation of $H$ only requires an upper bound on the logarithm of $m + 1$. Therefore, the reduction can be performed in logarithmic space. Second, the structure of the cost formula $\varphi$, in particular the number of inequalities, is fixed. Only the constants $T_1, T_2, H$ depend on the instance.

## 5 Cost Processes

### 5.1 Acyclic Cost Processes

In this section we prove that the cost problem for acyclic cost processes is \textsc{PSPACE}-complete. The challenging part is to show that \textsc{PSPACE}-hardness even holds for atomic cost formulas. For our lower bound, we reduce from a generalisation of the classical \textsc{SubsetSum} problem: Given a tuple $(k_1, \ldots, k_n, T)$ of natural numbers with $n$ even, the \textsc{QSubsetSum} problem asks whether the following formula is true:

$$
\exists x_1 \in \{0, 1\} \ \forall x_2 \in \{0, 1\} \ \cdots \ \exists x_{n-1} \in \{0, 1\} \ \forall x_n \in \{0, 1\} : \sum_{i=1}^{n} x_ik_i = T
$$

Here the quantifiers $\exists$ and $\forall$ occur in strict alternation. It is shown in \[23\], Lemma 4] that \textsc{QSubsetSum} is \textsc{PSPACE}-complete. One can think of such a formula as a turn-based game, the \textsc{QSubsetSum} game, played between Player Odd and Player Even. If $i \in \{1, \ldots, n\}$ is odd (even), then turn $i$ is Player Odd’s (Player Even’s) turn, respectively. In turn $i$ the respective player decides to either take $k_i$ by setting $x_i = 1$, or not to take $k_i$ by setting $x_i = 0$. Player Odd’s objective is to make the sum of the taken numbers equal $T$, and Player Even tries to prevent that. If Player Even is replaced by a random player, then Player Odd has a strategy to win with probability 1 if and only if the given instance is a “yes” instance for \textsc{QSubsetSum}. This gives an easy \textsc{PSPACE}-hardness proof for the cost problem with non-atomic cost formulas $\varphi \equiv (x = T)$. In order to strengthen the lower bound to atomic cost formulas $\varphi \equiv (x \leq B)$ we have to give Player Odd an incentive to take numbers $k_i$, even though she is only interested in not exceeding the budget $B$. This challenge is addressed in the \textsc{PSPACE}-hardness proof.

The \textsc{PSPACE}-hardness result reflects the fact that the optimal strategy must take the current cost into account, even for atomic cost formulas. This may be somewhat counter-intuitive, as a good strategy should always “prefer small cost”. But if there always existed a strategy depending only on the control state, one could guess this strategy in NP and invoke the PP-result of Section \[4.1\] in order to obtain an \textsc{NP}P algorithm, implying \textsc{NP}P = \textsc{PSPACE} and hence a collapse of the counting hierarchy.

\^{Theorem 7.} The cost problem for acyclic cost processes is in \textsc{PSPACE}. It is \textsc{PSPACE}-hard, even for atomic cost formulas.
The Odds of Staying on Budget

Proof sketch. To prove membership in PSPACE we consider a procedure $\text{Opt}$ that, given $(q, c) \in Q \times \mathbb{N}$ as input, computes the optimal (i.e., maximised over all schedulers) probability $p_{q,c}$ that starting from $(q, c)$ one reaches $(t, d)$ with $d \models \varphi$. The procedure $\text{Opt}(q, c)$ relies on the following characterisation of $p_{q,c}$ for $q \neq t$:

$$p_{q,c} = \max_{a \in En(q)} \sum_{q' \in Q} \sum_{k \in \mathbb{N}} \Delta(q, a)(q', k) \cdot p_{q', c+k}$$

So $\text{Opt}(q, c)$ loops over all $a \in En(q)$ and all $(q', k) \in Q \times \mathbb{N}$ with $\Delta(q, a)(q', k) > 0$ and recursively computes $p_{q', c+k}$. Since the cost process is acyclic, the height of the recursion stack is at most $|Q|$. The representation size of the probabilities that occur in that computation is polynomial. To see that, consider the product $D$ of the denominators of the probabilities occurring in the description of $\Delta$. The encoding size of $D$ is polynomial. All probabilities occurring during the computation are integer multiples of $1/D$. Hence computing $\text{Opt}(q_0, 0)$ and comparing the result with $\tau$ gives a PSPACE procedure.

For the lower bound we reduce the above defined QSUBSETSUM problem to the cost problem for an atomic cost formula $x \leq B$. Given an instance $(k_1, \ldots, k_n, T)$, where $n$ is even, of the QSUBSETSUM problem, we construct an acyclic cost process $C = (Q, q_0, t, A, En, \Delta)$ as follows. We take $Q = \{q_0, q_1, \ldots, q_{n-2}, q_n, t\}$. Those control states reflect pairs of subsequent turns that the QSUBSETSUM game can be in. The transition rules $\Delta$ will be set up so that probably the control states $q_0, q_1, \ldots, q_{n-2}, t$ will be visited in that order, with the (improbable) possibility of shortcuts to $t$. For even $i$ with $0 \leq i \leq n-2$ we set $En(q_i) = \{a_0, a_1\}$. These actions correspond to Player Odd’s possible decisions of not taking, respectively taking $k_{i+1}$. Player Even’s response is modelled by the random choice of not taking, respectively taking $k_{i+2}$ (with probability $1/2$ each). In the cost process, taking a number $k_i$ corresponds to incurring cost $k_i$. We also add an additional cost $\ell$ in each transition, in order to prevent the possibility of reaching the full budget $B$ before an action in control state $q_{n-2}$ is played. Therefore we define our cost problem to have the atomic formula $x \leq B$ with

$$B := \frac{n}{2} \cdot \ell + T.$$

For a large number $M \in \mathbb{N}$, defined in detail in Appendix D, we set for all even $i \leq n-2$ and for $j \in \{0, 1\}$:

$$\Delta(q_i, a_j)(q_{i+2}, \ell+j \cdot k_{i+1}) = \frac{1}{2} \cdot \left(1 - \frac{\ell + j \cdot k_{i+1}}{M}\right)$$

$$\Delta(q_i, a_j)(t, \ell+j \cdot k_{i+1}) = \frac{1}{2} \cdot \frac{\ell + j \cdot k_{i+1}}{M}$$

$$\Delta(q_{i+2}, a_j)(q_i, \ell+j \cdot k_{i+1} + k_{i+2}) = \frac{1}{2} \cdot \left(1 - \frac{\ell + j \cdot k_{i+1} + k_{i+2}}{M}\right)$$

$$\Delta(q_i, a_j)(t, \ell+j \cdot k_{i+1} + k_{i+2}) = \frac{1}{2} \cdot \frac{\ell + j \cdot k_{i+1} + k_{i+2}}{M}$$

So with a high probability the MDP transitions from $q_i$ to $q_{i+2}$, and cost $\ell, \ell+k_{i+1}, \ell+k_{i+2}$, $\ell+k_{i+1}+k_{i+2}$ is incurred, depending on the scheduler’s (i.e., Player Odd’s) actions and on the random (Player Even) outcome. But with a small probability, which is proportional to the incurred cost, the MDP transitions to $t$, which is a “win” for the scheduler as long as the accumulated cost is within budget $B$. We make sure that the scheduler loses if $q_n$ is reached:

$$\Delta(q_n, a)(t, B+1) = 1 \quad \text{with } En(q_n) = \{a\}$$
The MDP is designed such that the scheduler probably “loses” (i.e., exceeds the budget $B$); but whenever cost $k$ is incurred, a winning opportunity with probability $k/M$ arises. Since $1/M$ is small, the overall probability of winning is approximately $C/M$ if total cost $C \leq B$ is incurred. In order to maximise this chance, the scheduler wants to maximise the total cost without exceeding $B$, so the optimal scheduler will target $B$ as total cost.

The values for $\ell$, $M$ and $\tau$ need to be chosen carefully, as the overall probability of winning is not exactly the sum of the probabilities of the individual winning opportunities. Rather, this sum is – by the “union bound” – only an upper bound. One needs to show that the sum approximates the real probability closely enough. In Appendix D we complete the proof.

\section{5.2 General Cost Processes}

We prove:

\begin{itemize}
\item \textbf{Theorem 8.} The cost problem is \textsc{Exptime}-complete.
\end{itemize}

The \textsc{Exptime} upper bound was stated in Fact 1. For \textsc{Exptime}-hardness we build on \textit{countdown games} [12]. Those games are described as “a simple class of turn-based 2-player games with discrete timing” in [12], and the authors of that paper prove that deciding the winner in a countdown game is \textsc{Exptime}-complete. Allbeit non-stochastic, countdown games are very close to our model: two players move along edges of a graph labelled with positive integer weights and thereby add corresponding values to a succinctly encoded counter. Player 1’s objective is to steer the value of the counter to a given number $T \in \mathbb{N}$, and Player 2 tries to prevent that. Our reduction from countdown games in Appendix D requires a small trick, as in our model the final control state $t$ needs to be reached with probability 1 regardless of the scheduler, and furthermore, the scheduler attempts to achieve the cost target $T$ when and only when the control state $t \in Q$ is visited.

The proof of Theorem 8 reveals that even the following problem is \textsc{Exptime}-hard. The \textit{qualitative cost problem} asks, given a cost process and a number $T \in \mathbb{N}$, whether there exists a scheduler $\sigma$ with $P_{\sigma}(K = T) = 1$. As mentioned in the introduction, MDPs with \textit{two} non-negative and non-decreasing integer counters, viewed as cost and utility, respectively, were considered in [2, 4]. Specifically, those works consider problems like computing the minimal cost $C$ such that the probability of gaining at least a given utility $U$ is at least $\tau$. Possibly the most fundamental of those problems is the following: the \textit{cost-utility problem} asks, given an MDP with both cost and utility, and numbers $C, U \in \mathbb{N}$, whether one can almost surely gain utility at least $U$ using cost at most $C$.

\begin{itemize}
\item \textbf{Corollary 9.} The cost-utility problem is \textsc{Exptime}-complete.
\end{itemize}

\textbf{Proof.} Membership in \textsc{Exptime} is easy, as in Fact 1. For hardness, reduce the qualitative cost problem to the cost-utility problem where both the cost and the utility in the new MDP are increased as the cost in the cost process. Then we have $P_{\sigma}(K = T) = 1$ in the cost process if and only if in the new MDP the cost is at most $T$ and the utility is at least $T$ with probability 1.

\section{6 Conclusions and Open Problems}

We have considered fundamental Markov-chain and MDP problems on a single non-negative and only increasing integer counter. Among other results, we have shown that the cost problem for Markov chains is in \textsc{Pspace} and both hard for \textsc{PP} and the \textsc{PosSLP} problem.
It would be fascinating and potentially challenging to prove either PSPACE-hardness or membership in the counting hierarchy: while the problem does not seem to lend itself to a PSPACE-hardness proof, the authors are not aware of natural problems, except BitSLP \cite{1}, that are in the counting hierarchy and known to be hard for both PP and PosSLP.

Regarding acyclic and general MDPs, we have proved PSPACE-completeness and EXP-Time-completeness, respectively. Our results leave open the possibility that the cost problem for atomic cost formulas is not EXPTime-hard and maybe even in PSPACE. The technique described in the proof sketch of Theorem \cite{7} cannot be applied to general cost processes, because there we have to deal with paths of exponential length, which, informally speaking, have double-exponentially small probabilities. Proving hardness in an analogous way would thus require a probability threshold $\tau$ of exponential representation size.

Acknowledgement. The authors thank Andreas Göbel for valuable hints.
18 M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 2008.
19 J. Simon. On the difference between one and many. In *Proceedings of ICALP*, volume 52 of *LNCS*, pages 480–491. Springer, 1977.
20 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.
21 S. Travers. The complexity of membership problems for circuits over sets of integers. *Theoretical Computer Science*, 369(1–3):211–229, 2006.
22 M. Ummels and C. Baier. Computing quantiles in Markov reward models. In *Proceedings of FoSSaCS*, volume 7794 of *LNCS*, pages 353–368. Springer, 2013.
A  Proofs of Section 2

Proposition 10. Let $\mathcal{C}$ be a cost process, $\varphi$ a cost formula with $n_0 \not\models \varphi$ and $n_1 \models \varphi$ for some $n_0, n_1 \in \mathbb{N}$, and $\tau \in [0, 1]$. One can construct in logarithmic space a cost process $\mathcal{C}'$ such that the following holds: There is a scheduler $\sigma$ for $\mathcal{C}$ with $\mathcal{P}_\sigma(K_{\mathcal{C}} \models \varphi) \geq \tau$ if and only if there is a scheduler $\sigma'$ for $\mathcal{C}'$ with $\mathcal{P}_{\sigma'}(K_{\mathcal{C}'} \models \varphi) \geq 1/2$. Moreover, $\mathcal{C}'$ is a cost chain if $\mathcal{C}$ is.

Proof. Let $\tau < 1/2$. Define $p := (1/2 - \tau)/(1 - \tau)$. To construct $\mathcal{C}'$ from $\mathcal{C}$, add a new initial state $s_{00}$ with exactly one enabled action, say $a$, and set $\Delta(s_{00}, a)(t, n_1) = p$ and $\Delta(s_{00}, a)(s_0, 0) = 1 - p$. In a straightforward sense any scheduler for $\mathcal{C}$ can be viewed as a scheduler for $\mathcal{C}'$ and vice versa. Thus for any scheduler $\sigma$ we have $\mathcal{P}_{\sigma'}(K \models \varphi) = p + (1 - p) \cdot \mathcal{P}_{\sigma}(K \models \varphi)$. The statement of the proposition now follows from a simple calculation.

Now let $\tau > 1/2$. Define $p := 1/(2\tau)$. In a similar way as before, add a new initial state $s_{00}$ with exactly one enabled action $a$, and set $\Delta(s_{00}, a)(t, n_0) = 1 - p$ and $\Delta(s_{00}, a)(s_0, 0) = p$. Thus we have $\mathcal{P}_{\sigma'}(K \models \varphi) = p \cdot \mathcal{P}_{\sigma}(K \models \varphi)$, and the statement of the proposition follows.

B  Proofs of Section 3

In this section we prove Proposition 2 from the main text:

Proposition 2. Suppose $0 \leq \tau < 1$. Let $p_{\min}$ be the smallest non-zero probability and $k_{\max}$ be the largest cost in the description of the cost process. Let

$$B := k_{\max} \cdot \left|Q\right| \cdot \left(-\ln(1 - \tau)/p_{\min}^{\left[Q\right]} + 1\right).$$

Then $\mathcal{P}_\sigma(K \leq B) \geq \tau$ holds for all schedulers $\sigma$.

Proof. Define $n := |Q|$. If $p_{\min} = 1$, then by our assumption on the almost-sure reachability of $t$, the state $t$ will be reached within $n$ steps, and the statement of the proposition follows easily. So we can assume $p_{\min} < 1$ for the rest of the proof.

Let $j \in \mathbb{N}$ be the smallest integer with

$$j \geq n \cdot \left(-\frac{\ln(1 - \tau)}{p_{\min}^n} + 1\right).$$

It follows:

$$\left\lfloor \frac{j}{n} \right\rfloor \geq \frac{-\ln(1 - \tau)}{p_{\min}^n} \geq \frac{\ln(1 - \tau)}{\ln(1 - p_{\min})} \quad \text{(as } x \leq -\ln(1 - x) \text{ for } x < 1\text{)}$$

(2)

For $i \in \mathbb{N}$ and $q \in Q$ and a scheduler $\sigma$, define $p_i(q, \sigma)$ as the probability that, if starting in $q$ and using the scheduler $\sigma$, more than $i$ steps are required to reach the target state $t$. Define $p_i := \max\{p_i(q, \sigma) : q \in Q, \sigma \text{ a scheduler}\}$. By our assumption on the almost-sure reachability of $t$, regardless of the scheduler, there is always a path to $t$ of length at most $n$. This path has probability at least $p_{\min}^n$, so $p_n \leq 1 - p_{\min}^n$. If a path of length $\ell \cdot n$ does not reach $t$, then none of its $\ell$ consecutive blocks of length $n$ reaches $t$, so we have $p_{\ell \cdot n} \leq p_n^\ell$. 


Hence we have:

\[
p_j \leq p_{\lfloor j/n \rfloor} \cdot n \quad \text{(as } p_i \geq p_{i+1} \text{ for all } i \in \mathbb{N})
\]

\[
\leq (1 - p_{\min}^n)^{\lfloor j/n \rfloor} \quad \text{(as argued above)}
\]

\[
= \exp \left( \ln(1 - p_{\min}^n) \cdot \lfloor j/n \rfloor \right) \quad \text{(by (2))}
\]

\[
\leq 1 - \tau \quad \text{(by (3))}
\]

\[\text{(3)}\]

Denote by \( T \) the random variable that assigns to a run the “time” (i.e., the number of steps) to reach \( t \) from \( s_0 \). Then we have for all schedulers \( \sigma \):

\[
\mathbb{P}_\sigma(K \leq B) = \mathbb{P}_\sigma(K \leq j \cdot k_{\max}) \quad \text{(by the definition of } B) \]

\[
\geq \mathbb{P}_\sigma(T \leq j) \quad \text{(each step costs at most } k_{\max}) \]

\[
= 1 - \mathbb{P}_\sigma(T > j) \quad \text{(by the definition of } T \text{ and } p_i) \]

\[
\geq 1 - p_j \quad \text{(by (3))}
\]

as claimed.

\[\Box\]

\section{C Proofs of Section 4}

\subsection{C.1 Proof of Theorem 3}

In this section we prove Theorem 3 from the main text:

\textbf{Theorem 3} \quad The cost problem for acyclic cost chains is in PP. It is PP-hard with respect to polynomial-time Turing reductions, even for atomic cost formulas.

\textbf{Proof.} First we prove membership in PP. The class PP can be equivalently defined as the class of languages \( L \) that have a probabilistic polynomial-time bounded Turing machine \( M_L \) such that for all words \( x \) one has \( x \in L \) if and only if \( M_L \) accepts \( x \) with probability at least \( 1/2 \), see [11] and note that PP is closed under complement [11]. By Proposition 10 it suffices to consider an instance of the cost problem with \( \tau = 1/2 \). The problem can be decided by a probabilistic Turing machine that simulates the cost chain as follows: The Turing machine keeps track of the control state and the cost, and branches according to the probabilities specified in the cost chain. It accepts if and only if the accumulated cost satisfies \( \varphi \). Note that the acyclicity of the cost chain guarantees the required polynomial time bound.

For the lower bound, we give a polynomial-time Turing reduction from the canonical PP-complete problem \textsc{MajSAT} [11, 17]. The problem \textsc{MajSAT} asks, given a Boolean formula \( \psi \) over \( n \) variables, does a majority (i.e., at least \( 2^{n-1} + 1 \)) of the variable assignments satisfy \( \psi \). Let \( \psi \) be the given formula. We will provide a polynomial-time Turing reduction that (even) computes the number of assignments that satisfy \( \psi \). It is argued in [10] that one can compute from \( \psi \) in polynomial time a tuple \((k_1, \ldots, k_\ell, T)\) of natural numbers so that the number of subsets of \( \{k_1, \ldots, k_\ell\} \) whose elements sum up to \( T \) is equal to the number of variable assignments that satisfy \( \psi \).\footnote{In more technical terms, it is argued in [10, p. 104] that there is a parsimonious reduction from \#SAT} Let \( k_1, \ldots, k_\ell, T \) denote those computed numbers,
and let \( m \) denote the sought number of subsets. Following [14] we construct the cost chain \( \mathcal{C} = (Q, q_0, q_\ell, \Delta) \) with \( Q = \{q_0, q_1, \ldots, q_\ell\} \) and \( \Delta(q_{i-1})(q_i, k_i) = 1/2 \) and \( \Delta(q_{i-1})(q_i, 0) = 1/2 \) for all \( i \in \{1, \ldots, \ell\} \). Now we have \( m = \mathcal{P}(K = T) \cdot 2^\ell \). To compute \( \mathcal{P}(K = T) \) we use polynomially many oracle queries to the cost problem for atomic cost formulas as follows. First we compute \( \mathcal{P}(K \leq T) \) using binary search over \( \tau \) (where \( \tau \) ranges over numbers \( i/2^\ell \) for \( i \in \{0, \ldots, 2^\ell\} \)). Then we compute \( \mathcal{P}(K \leq T - 1) \) using the same method. Then we compute \( \mathcal{P}(K = T) \) as \( \mathcal{P}(K \leq T) - \mathcal{P}(K \leq T - 1) \). Thus we can compute \( m = \mathcal{P}(K = T) \cdot 2^\ell \). 

\[ \square \]

C.2 Proof of Theorem 4

In this section we prove Theorem 4 from the main text:

\[ \textbf{Theorem 4} \]

The cost problem for cost chains is in \( \text{PSpace} \) and hard for \( \text{PosSLP} \).

We give some details on the upper bound in Section C.2.1. In Sections C.2.2 and C.2.3 we provide proofs for propositions from the main text that refer to the \( \text{PosSLP} \) lower bound.

C.2.1 Proof of the Upper Bound in Theorem 4

We show that the cost problem for cost chains is in \( \text{PSpace} \). As outlined in the main text we use the fact that probabilistic \( \text{PSpace} \) equals \( \text{PSpace} \). There is no standard definition of “probabilistic \( \text{PSpace} \)” in the literature. We define it in analogy to \( \text{PP} \) as follows: \( \text{Probabilistic \( \text{PSpace} \)} \) is the class of languages \( L \) that have a probabilistic polynomial-space bounded Turing machine \( M_L \) such that for all words \( x \) one has \( x \in L \) if and only if \( M_L \) accepts \( x \) with probability at least \( 1/2 \). The cost problem for cost chains is in this class. This can be shown in the same way as we showed in Theorem 3 that the cost problem for acyclic cost chains is in \( \text{PP} \). More concretely, given an instance of the cost problem for cost chains, we construct in logarithmic space a probabilistic \( \text{PSpace} \) Turing machine that simulates the cost chain and accepts if and only if the accumulated cost \( K \) satisfies the given cost formula.

The fact that (this definition of) probabilistic \( \text{PSpace} \) equals \( \text{PSpace} \) was first proved in [19]. A simpler proof can be obtained using a result by Ladner [13] that states that \( \#\text{PSpace} \) equals \( \text{FPSpace} \), see [13] for definitions. This was noted implicitly, e.g., in [15, Theorem 5.2]. We remark that the class \( \text{PPSpace} \) defined in [16] also equals \( \text{PSpace} \), but its definition (which is in terms of stochastic games) is different.

C.2.2 Proof of Proposition 5

In this section we prove Proposition 5 from the main text:

\[ \textbf{Proposition 5} \]

Let \( G = (V, E) \) be an arithmetic circuit and \( v \in V \). There exists a log-space computable DFA \( A = (Q, \Sigma, \Delta) \) with distinguished control states \( q, q' \in Q \) and a Parikh function \( f : \Sigma \rightarrow \mathbb{N} \) such that

\[ \text{val}(v) = |\{\pi \in \Pi(A, q, q') : \text{parikh}(\pi) = f\}|. \]

\[ \text{to} \ #\text{SubsetSum}, \text{where} \ #\text{SAT} \ \text{refers} \ \text{to} \ \text{the} \ \text{canonical} \ #\text{P}-\text{complete} \ \text{problem} \ \text{of} \ \text{counting} \ \text{the} \ \text{satisfying} \ \text{assignments} \ \text{of} \ \text{a} \ \text{Boolean} \ \text{formula}, \ \text{and} \ #\text{SubsetSum} \ \text{refers} \ \text{to} \ \text{the} \ \text{function} \ \text{problem} \ \text{that} \ \text{asks}, \ \text{given} \ \text{a} \ \text{tuple} \ (k_1, \ldots, k_t, T) \ \text{of} \ \text{natural} \ \text{numbers}, \ \text{for} \ \text{the} \ \text{number} \ \text{of} \ \text{subsets} \ \text{of} \ \{k_1, \ldots, k_t\} \ \text{whose} \ \text{elements} \ \text{sum} \ \text{up} \ \text{to} \ \text{T}. \ \text{This} \ \text{reduction} \ \text{establishes} \ \text{that} \ #\text{SubsetSum} \ \text{is} \ #\text{P}-\text{parsimonious-complete}, \ \text{which} \ \text{is} \ \text{an} \ \text{even} \ \text{stronger} \ \text{form} \ \text{of} \ \text{hardness} \ \text{than} \ \text{we} \ \text{would} \ \text{need}. \]
Proof. We construct $A$ by induction on the number of levels of $V$. For every level $i$, we define an alphabet $\Sigma_i$ and a Parikh function $f_i : \Sigma_i \to \mathbb{N}$. As an invariant, $\Sigma_i \subseteq \Sigma_{i+1}$ holds for all levels $i$. Subsequently, denote by $V(i)$ all gates on level $i$. For every $v \in V(i)$, we define a DFA $A_v$ such that each $A_v$ has two distinguished control locations $in(A_v)$ and $out(A_v)$. The construction is such that

$$val(v) = |\{\pi \in \Pi(A_v, in(A_v), out(A_v)) : \text{parikh}(\pi) = f_i\}|.$$  

For technical convenience, we allow transitions to be labelled with subsets of $\Sigma$ over $\Sigma$. Further we set $\Delta = \{\delta_1, \ldots, \delta_7\}$ such that

- $\delta_1 = (in(A_v), S_v, q)$, where $S_v = \{a_w, b_w, c_v : v \in V(i+1), v \neq w\}$;
- $\delta_2 = (q, a_v, q_1)$ and $\delta_3 = (q, b_v, q_2)$;
- $\delta_4 = (q_1, b_v, in(A_u))$ and $\delta_5 = (q_2, a_v, in(A_w))$; and
- $\delta_6 = (out(A_v), c_v, out(A))$ and $\delta_7 = (out(A_w), c_v, out(A))$.

Informally speaking, we simply branch at $q$ into $A_u$ and $A_w$, and this in turn enforces that the number of paths in $\Pi(A_v, in(A_v), out(A_v))$ on which $a_v$ occurs once equals the sum of $val(u)$ and $val(w)$. The reason behind using both $a_v$ and $b_v$ is that it ensures that the case $w = w$ is handled correctly. Setting $f_{i+1}(a) = 1$ if $a \in \Sigma_{i+1} \setminus \Sigma_i$, and $f_{i+1}(a) = f_i(a)$ otherwise, we consequently have that $\Pi(A_v, in(A_v), out(A_v))$ contains three additional control states $q, q_1, q_2$. Further we set $\Delta_v = \{\delta_1, \ldots, \delta_7\}$ such that

- $\delta_1 = (in(A_v), S_v, q)$, where $S_v = \{a_w, b_w, c_v : v \in V(i+1), v \neq w\}$;
- $\delta_2 = (q, a_v, q_1)$ and $\delta_3 = (q, b_v, q_2)$;
- $\delta_4 = (q_1, b_v, in(A_u))$ and $\delta_5 = (q_2, a_v, in(A_w))$; and
- $\delta_6 = (out(A_v), c_v, out(A))$ and $\delta_7 = (out(A_w), c_v, out(A))$.

The case of $i+1$ being even can be handled analogously, but instead of using branching we use sequential composition in order to simulate the computation of a gate labelled with “+”.

Apart from the control states $in(A_v)$ and $out(A_v)$, the set $Q_v$ contains an additional control state $q$. Further we set $\Delta_v = \{\delta_1, \ldots, \delta_4\}$ such that

- $\delta_1 = (in(A_v), S_v, q)$, where $S_v = \{a_w, b_w, c_w : v \in V(i+1), v \neq w\}$;
- $\delta_2 = (q, a_v, in(A_v))$;
- $\delta_3 = (out(A_v), b_v, in(A_v))$; and
- $\delta_4 = (out(A_v), c_v, out(A_v))$. 

As an invariant, the gates on this level are labelled with “+”, then apart from the control states $in(A_v)$ and $out(A_v)$, the set $Q_v$ contains two additional control states $q, q_1, q_2$. Further we set $\Delta_v = \{\delta_1, \ldots, \delta_4\}$ such that

- $\delta_1 = (in(A_v), S_v, q)$, where $S_v = \{a_w, b_w, c_v : v \in V(i+1), v \neq w\}$;
- $\delta_2 = (q, a_v, in(A_v))$;
- $\delta_3 = (out(A_v), b_v, in(A_v))$; and
- $\delta_4 = (out(A_v), c_v, out(A_v))$. 

The particularities of the construction depend on the type of $v$.
The Odds of Staying on Budget

A difference to the case where \( i + 1 \) is odd is that via the definition of \( f_{i+1} \) we have to allow for paths that can traverse both \( \mathcal{A}_u \) and \( \mathcal{A}_w \). Consequently, we define \( f_{i+1}(a) = 1 \) if \( a \in \Sigma_{i+1} \setminus \Sigma_i \), and \( f_{i+1}(a) = 2f_i(a) \) otherwise. Similarly as above, \( \text{(4)} \) holds since

\[
\left| \{ \pi \in \Pi(\mathcal{A}_u, \text{in}(\mathcal{A}_w), \text{out}(\mathcal{A}_u)) : \text{parikh}(\pi) = f_{i+1} \} \right| \\
= \left| \{ \pi \in \Pi(\mathcal{A}_u, \text{in}(\mathcal{A}_w), \text{out}(\mathcal{A}_w)) : \text{parikh}(\pi) = f_i \} \right| \\
\times \left| \{ \pi \in \Pi(\mathcal{A}_w, \text{in}(\mathcal{A}_u), \text{out}(\mathcal{A}_w)) : \text{parikh}(\pi) = f_i \} \right| \\
= \text{val}(u) \cdot \text{val}(w) \\
= \text{val}(v).
\]

Due to the inductive nature of the construction, the cautious reader may on the first sight cast doubt that the computation of \( \mathcal{A}_u \) and \( f \) can be performed in logarithmic space. However, a closer look reveals that the graph underlying \( \mathcal{A}_u \) has a simple structure and its list of edges can be constructed without prior knowledge of the DFA on lower levels. Likewise, even though \( f \) contains numbers which are exponential in the number of levels of \( G \), the structure of \( f \) is simple and only contains numbers which are powers of two, and hence \( f \) is computable in logarithmic space as well.

C.2.3 Proof of Proposition 6

In this section we prove Proposition 6 from the main text:

\[ \textbf{Proposition 6} \]

Let \( G = (V, E) \) be an arithmetic circuit. Let \( v \in V \) be a gate on level \( \ell \) with odd \( \ell \). There exist a log-space computable cost process \( C \) and \( T \in \mathbb{N} \) with \( P(K_C = T) = \text{val}(v)/m \), where \( m = \exp_2(2^{(\ell-1)/2+1} - 1) \cdot \exp_2(2^{(\ell-1)/2+1} - 3) \).

In order to prove Proposition 6 for a clearer proof structure we define an intermediate formalism between DFA and cost chains. A typed cost chain \( T = (Q, q_0, t, \Gamma, \Delta) \) is similar to a cost chain, but with costs (i.e., natural numbers) replaced with functions \( \Gamma \to \mathbb{N} \). The intuition is that instead of a single cost, a typed cost chain keeps track of several types of cost, and each type is identified with a symbol from \( \Gamma \). More precisely, \( Q \) is a finite set of control states, \( q_0 \in Q \) is the initial control state, \( t \) is the target control state, \( \Gamma \) is a finite alphabet, and \( \Delta : Q \to \text{dist}(Q \times \mathbb{N}^\Gamma) \) is a probabilistic transition function.

A typed cost chain \( T \) induces a Markov chain in the same way as a cost chain does, but the state space is \( Q \times \mathbb{N}^\Gamma \) rather than \( Q \times \mathbb{N} \). Formally, \( T \) induces the Markov chain \( \mathcal{D}_T = (Q \times \mathbb{N}^\Gamma, (q_0, 0), \delta) \), where by \( 0 \) we mean the function \( c : \Gamma \to \mathbb{N} \) with \( c(a) = 0 \) for all \( a \in \Gamma \), and \( \delta(q, c)(q', c') = \Delta(q)(q', c' - c) \) holds for all \( q, q' \in Q \) and \( c, c' \in \mathbb{N}^\Gamma \), where by \( c' - c \) we mean \( c' : \Gamma \to \mathbb{N} \) with \( c'(a) = c'(a) - c(a) \) for all \( a \in \Gamma \). As before, we assume that the target control state \( t \) is almost surely reached. We write \( K_T \) for the (multi-dimensional) random variable that assigns a run in \( \mathcal{D}_T \) the typed cost \( c : \Gamma \to \mathbb{N} \) that is accumulated upon reaching \( t \).

\[ \textbf{Lemma 11} \]

Let \( G = (V, E) \) be an arithmetic circuit. Let \( v \in V \) be a gate on level \( \ell \) with odd \( \ell \). Let \( d = |V| + 1 \). There exist a log-space computable typed cost chain \( T = (Q, q_0, t, \Gamma, \Delta) \) and \( c : \Gamma \to \mathbb{N} \) such that \( P(K_T = c) = \text{val}(v)/m \), where

\[
m = \exp_2(2^{(\ell-1)/2+1} - 1) \cdot \exp_2(2^{(\ell-1)/2+1} - 3).
\]

\textbf{Proof.} With no loss of generality we may assume that the maximum level of \( V \) is \( \ell \) and that \( v \) is the only gate on level \( \ell \). The idea is to translate the DFA obtained in Lemma 5 into a
suitable typed cost chain. Subsequently, we refer to the terminology used in the proof of Lemma 5.

Let $\mathcal{A} = (Q, \Sigma, \Delta)$ be the DFA, $q_0 = \text{in}(\mathcal{A}_w)$, $t = \text{out}(\mathcal{A}_w)$, and $f : \Sigma \rightarrow \mathbb{N}$ be the Parikh function obtained from Lemma 5. We define $\Gamma = \Sigma \cup \{e_j : 1 \leq j \leq d\}$ and alter $\mathcal{A}$ as follows:

- for the gate $w \in V$ on level 0 labelled with 0, we add an edge from $\text{in}(\mathcal{A}_w)$ to $t$ labelled with $e_1$; and
- for every $w \in V$ such that $w \neq v$, we add $k$ edges labelled with $e_1, \ldots, e_k$ from $\text{out}(\mathcal{A}_w)$ to $t$, where $k$ is the difference between $d$ and the number of outgoing edges from $\text{out}(\mathcal{A}_w)$.

The DFA $\mathcal{A}^\prime = (Q, \Gamma, \Delta^\prime)$ obtained from this construction has the property that $t$ can be reached from any control state, and that the number of outgoing edges from any $\text{out}(\mathcal{A}_w)$ for $w \neq v$ is uniform. Finally, we define $c : \Gamma \rightarrow \mathbb{N}$ such that $c$ coincides with $f$ for all $a \in \Sigma$ and $c(e_j) = 0$ for all $1 \leq j \leq k$. The intuition behind the $e_j$ is that they indicate errors, and once an edge with an $e_j$ is traversed it is impossible to reach $t$ with Parikh image $c$. Thus, in particular property (4) is preserved in $\mathcal{A}^\prime$.

We now transform $\mathcal{A}^\prime$ into a typed cost chain $\mathcal{T}$. Subsequently, for $a \in \Gamma$ let $c_a : \Gamma \rightarrow \{0, 1\}$ be the function such that $c_a(b) = 1$ if $b = a$ and $c_a(b) = 0$ otherwise. For our transformation, we perform the following steps:

- every alphabet letter $a \in \Gamma$ labelling a transition of $\mathcal{A}^\prime$ is replaced by $c_a$;
- the probability distribution over edges is chosen uniformly; and
- a self-loop labelled with 0 and probability 1 is added at $t$.

We observe that the transition probabilities of $\mathcal{T}$ are either $1/d$, $1/2$ or 1. Since $t$ can be reached from any control state, it is eventually reached with probability 1.

For every level $i$ and every $w \in V(i)$, let $p_w$ denote the probability that, starting from $\text{in}(\mathcal{A}_w)$, the control state $\text{out}(\mathcal{A}_w)$ is reached and typed cost $c_i$ is accumulated. Here, $c_i$ refers to the Parikh function $f_i$ constructed in the proof of Lemma 5, where we assert that $c_i(a) = 0$ for all $a \in \Gamma$ on which the “original” $f_i$ is undefined. Since $t = \text{out}(\mathcal{A}_w)$ is almost surely reached from $q_0 = \text{in}(\mathcal{A}_w)$, we have $p_w = \mathcal{P}(K_T = c_i)$. So in order to prove the lemma, it suffices to prove for all $i \in \mathbb{N}$:

$$p_w = \frac{\text{val}(w)}{m(i)} \quad \text{for all } w \in V(i),$$

where

$$m(i) = \begin{cases} \exp_d(2i/2^i - 1) \cdot \exp_d(2i/2^i - 4) & \text{if } i \text{ is even} \\ \exp_d(2(i-1)/2^i - 1) \cdot \exp_d(2(i-1)/2^i - 3) & \text{if } i \text{ is odd.} \end{cases}$$

We proceed by induction on the level $i$. Let $i = 0$. If $w$ is labelled with 1 then there is exactly one outgoing transition from $\text{in}(\mathcal{A}_w)$, and this transition goes to $\text{out}(\mathcal{A}_w)$ and incurs cost $c_0$. So we have $p_w = 1$ as required. If $w$ is labelled with 0, then the only outgoing transition from $\text{in}(\mathcal{A}_w)$ incurs cost $c$ with $c(e_1) = 1$, hence $p_w = 0$.

For the induction step, let $i \geq 0$. Let $w \in V(i+1)$ and let $u, u' \in V(i)$ be the gates connected to $w$. If $i+1$ is odd then $w$ is labelled with “+”, and by the construction of $\mathcal{A}_w$
The homomorphism $\gamma$ encodes any alphabet symbols into the digits of natural numbers represented in a suitable base.

\[ p_w = \frac{1}{d} \cdot \frac{1}{d} \cdot (p_u + p_w) \]
\[ = \frac{1}{d} \cdot \frac{1}{d} \cdot \frac{\text{val}(u) + \text{val}(u')}{\text{exp}_d(2^{i/2+1} - 2) \cdot \text{exp}_d(2^{i/2+1} - 4)} \]
\[ = \frac{\text{val}(u) + \text{val}(u')}{\text{exp}_d(2^{i/2+1} - 1) \cdot \text{exp}_d(2^{i/2+1} - 3)} \]
\[ = \frac{\text{val}(u)}{m(i+1)}. \]

The factor $1/2$ is the probability of branching into $\text{in}(A_u)$ or $\text{in}(A_w)$, and $1/d$ is the probability that when leaving $\text{out}(A_u)$ respectively $\text{out}(A_w)$, the transition to $\text{out}(A_w)$ is taken.

Otherwise, if $i + 1$ is even, we have

\[ p_w = \frac{1}{d^2} \cdot p_u \cdot p_w' \]
\[ = \frac{1}{d^2} \cdot \frac{\text{val}(u) \cdot \text{val}(u')}{\text{exp}_d(2^{(i-1)/2+1} - 1) \cdot \text{exp}_d(2^{(i-1)/2+1} - 3)} \]
\[ = \frac{1}{d^2} \cdot \frac{\text{val}(u) \cdot \text{val}(u')}{\text{exp}_d(2^{(i-1)/2+2} - 2) \cdot \text{exp}_d(2^{(i-1)/2+2} - 6)} \]
\[ = \frac{\text{val}(u) \cdot \text{val}(u')}{\text{exp}_d(2^{(i+1)/2+1} - 2) \cdot \text{exp}_d(2^{(i+1)/2+1} - 4)} \]
\[ = \frac{\text{val}(u)}{m(i+1)}. \]

Here, $1/d^2$ is the probability that when leaving $\text{out}(A_u)$ the transition to $\text{in}(A_w')$ is taken, and that when leaving $\text{out}(A_u')$ the transition to $\text{out}(A_w)$ is taken.

In order to complete the proof of Proposition 6, we now show how a typed cost chain can be transformed into a cost chain. The idea underlying the construction in the next lemma is that we can encode alphabet symbols into the digits of natural numbers represented in a suitable base.

**Lemma 12.** Let $\Gamma$ be a finite alphabet, and let $c, c_1, \ldots, c_n : \Gamma \to \mathbb{N}$ be functions represented as tuples with numbers encoded in binary. There exists a log-space computable homomorphism $h : \mathbb{N}^\Gamma \to \mathbb{N}$ such that for all $\lambda_1, \ldots, \lambda_n \in \mathbb{N}$ we have

\[ \sum_{i=1}^n \lambda_i c_i = c \iff \sum_{i=1}^n \lambda_i h(c_i) = h(c). \]

**Proof.** Let $\Gamma = \{ a_0, \ldots, a_{k-1} \}$, $m = \sum_{a \in \Gamma} c(a)$, and $b = m + 1$. We define $h : \mathbb{N}^\Gamma \to \mathbb{N}$ as

\[ h(d) = d(a_0) \cdot b^0 + d(a_1) \cdot b^1 + \cdots + d(a_{k-1}) \cdot b^{k-1} + \left( \sum_{a \in \Gamma} d(a) \right) \cdot b^k. \]

The homomorphism $h$ encodes any $d : \Gamma \to \mathbb{N}$ into the $k$ least significant digits of a natural number represented in base $b$, and the $k + 1$-th digit serves as a check digit.
Suppose $\sum_{i=1}^{n} \lambda_i c_i = c$. Then
\[
\sum_{i=1}^{n} \lambda_i h(c_i) = \sum_{j=0}^{k-1} \left( \sum_{i=1}^{n} \lambda_i c_i(a_j) \right) \cdot b^j + \left( \sum_{a \in \Gamma} \sum_{i=1}^{n} \lambda_i c_i(a) \right) \cdot b^k
\]
\[
= \sum_{j=0}^{k-1} c(a_j) \cdot b^j + \sum_{a \in \Gamma} c(a) \cdot b^k
\]
\[
= h(c).
\]
Conversely, assume that $\sum_{i=1}^{n} \lambda_i h(c_i) = h(c)$. By definition of $h$, the check digit $k+1$ ensures that
\[
\sum_{a \in \Gamma} \sum_{i=1}^{n} \lambda_i c_i(a) = \sum_{a \in \Gamma} c(a) = m < b.
\]
Thus, in particular for a fixed $a_j \in \Sigma$ we have
\[
\sum_{i=1}^{n} \lambda_i c_i(a_j) < b.
\]
But now, since $\sum_{i=1}^{n} \lambda_i h(c_i) = h(c)$ we have
\[
\sum_{i=1}^{n} \lambda_i c_i(a_j) = c(a_j),
\]
and consequently $\sum_{i=1}^{n} \lambda_i c_i = c$.

By replacing every typed cost function $c$ in $T$ with $h(c)$, an easy application of Lemma 12 now yields the following corollary.

**Corollary 13.** Let $T = (Q, q_0, t, \Gamma, \Delta)$ be a typed cost chain and $c : \Gamma \to \mathbb{N}$. There exist a log-space computable cost chain $C = (Q, q_0, t, \delta)$ and $n \in \mathbb{N}$ such that
\[
\mathcal{P}(K_T = c) = \mathcal{P}(K_C = n).
\]
Together with Lemma 11 this completes the proof of Proposition 6.

### D Proofs of Section 5

**Theorem 7.** The cost problem for acyclic cost processes is in PSpace. It is PSpace-hard, even for atomic cost formulas.

**Proof.** In the main body of the paper we proved the upper bound and gave a sketch of the PSpace-hardness construction. Following up on this, we now provide the details of that reduction.

Let $k_{\text{max}} := \max\{k_1, k_2, \ldots, k_n\}$. We choose $\ell := 1 + nk_{\text{max}}$. Before an action in control state $q_{n-2}$ is played, at most the following cost is incurred:
\[
\frac{n-2}{2} \cdot (\ell + 2k_{\text{max}}) = \frac{n}{2} \cdot \ell + nk_{\text{max}} - \ell - 2k_{\text{max}} < \frac{n}{2} \cdot \ell \leq \frac{n}{2} \cdot \ell + T = B,
\]
so one cannot reach the full budget $B$ before an action in control state $q_{n-2}$ is played. We choose
\[
M := 2^{n/2} n^2 \ell^2 \quad \text{and} \quad \tau := \left( B - \frac{1}{2} \cdot \frac{1}{2^{n/2}} \right) / M.
\]

(5)
For the sake of the argument we slightly change the standard old MDP to a new MDP, but without affecting the scheduler’s winning chances. The control state $t$ is removed. Any old transition $\delta$ from $q_i$ to $t$ is redirected: with the probability of $\delta$ the new MDP transitions to $q_{i+2}$ and incurs the cost of $\delta$; in addition, one marble is gained if the accumulated cost including the one of $\delta$ is at most the budget $B$. The idea is that a win in the old MDP (i.e., a transition to $t$ having kept within budget) should correspond exactly to gaining at least one marble in the new MDP. The new MDP will be easier to analyse.

We make the definition of the new MDP more precise: When in $q_i$, the new MDP transitions to $q_{i+2}$ with probability 1. The cost incurred and marbles gained during that transition depend on the action taken and on probabilistic decisions as follows. Suppose action $a_j$ (with $j \in \{0,1\}$) is taken in $q_i$, and cost $C_i$ has been accumulated up to $q_i$. Then:

1. $j' \in \{0,1\}$ is chosen with probability $1/2$ each.
2. Cost $\ell + j \cdot k_{i+1} + j' \cdot k_{i+2}$ is incurred.
3. If $C_i + \ell + j \cdot k_{i+1} + j' \cdot k_{i+2} \leq B$, then, in addition, one marble is gained with probability $\frac{\ell + j \cdot k_{i+1} + j' \cdot k_{i+2}}{M}$, and no marble is gained with probability $1 - \frac{\ell + j \cdot k_{i+1} + j' \cdot k_{i+2}}{M}$.

In the new MDP, the scheduler’s objective is, during the path from $q_i$ to $q_n$, to gain at least one marble. Since an optimal scheduler in the new MDP does not need to take into account whether or when marbles have been gained, we assume that schedulers in the new MDP do not take marbles into account. The new MDP is constructed so that the winning chances are the same in the old and the new MDP; in fact, any scheduler in the old MDP translates into a scheduler with the same winning chance in the new MDP, and vice versa.

Fix a scheduler $\sigma$ in the new MDP. A vector $x = (x_2, x_4, \ldots, x_n) \in \{0,1\}^{n/2}$ determines the cost incurred during a run, in the following way: when $\sigma$ takes action $a_j$ (for $j \in \{0,1\}$) in state $q_i$, then cost $c_i(\sigma, x) := \ell + j \cdot k_{i+1} + x_{i+2} \cdot k_{i+2}$ are added upon transitioning to $q_{i+2}$. Conversely, a run determines the vector $x$. Let $\hat{p}_i^\sigma(x)$ denote the conditional probability (conditioned under $x$) that a marble is gained upon transitioning from $q_i$ to $q_{i+2}$. We have:

$$\hat{p}_i^\sigma(x) = \begin{cases} 
c_i(\sigma, x)/M & \text{if } c_0(\sigma, x) + c_2(\sigma, x) + \cdots + c_i(\sigma, x) \leq B \\
0 & \text{otherwise} 
\end{cases}$$

It follows that we have:

$$\hat{p}_i^\sigma(x) \leq \frac{2\ell}{M} \quad \text{(6)}$$

$$\sum_{i=0}^{n-2} \hat{p}_i^\sigma(x) \leq \frac{B}{M} \quad \text{(7)}$$

Denote by $p_i^\sigma$ (for $i = 0, 2, \ldots, n-2$) the (total) probability that a marble is gained upon transitioning from $q_i$ to $q_{i+2}$. By the law of total probability we have

$$p_i^\sigma = \sum_{x \in \{0,1\}^{n/2}} \frac{1}{2^{n/2}} \cdot \hat{p}_i^\sigma(x) \quad \text{(9)}$$

and hence, by (7),

$$p_i^\sigma \leq \frac{2\ell}{M} \quad \text{(10)}$$

We show that Player Odd has a winning strategy in the QSUBSETSUM game if and only if the probability of winning in the new MDP is at least $\tau$.

Assume that Player Odd has a winning strategy in the QSUBSETSUM game. Let $\sigma$ be the scheduler in the new MDP that emulates Player Odd’s winning strategy from the
Q\textsc{SubsetSum} game. Using $\sigma$ the accumulated cost upon reaching $q_n$ is exactly $B$, with probability 1. So for all $x \in \{0,1\}^{n/2}$ we have:

$$c_0(\sigma, x) + c_2(\sigma, x) + \cdots + c_{n-2}(\sigma, x) = B \quad (11)$$

Thus:

$$\sum_{\text{even } i=0}^{n-2} p_i^\sigma = \sum_{x \in \{0,1\}^{n/2}} \frac{1}{2^{n/2}} \sum_{\text{even } i=0}^{n-2} \hat{p}_i^\sigma(x) \quad \text{by (9)}$$

$$= \sum_{x \in \{0,1\}^{n/2}} \frac{1}{2^{n/2}} \sum_{\text{even } i=0}^{n-2} c_i(\sigma, x) / M \quad \text{by (9) and (11)}$$

$$= B / M \quad \text{by (11)} \quad (12)$$

Further we have:

$$\sum_{\text{even } i,j \; i < j \leq n-2} p_i^\sigma p_j^\sigma \quad \text{by (10) and (11)}$$

$$\leq \left( \frac{n/2}{2} \right)^2 \left( \frac{2\ell}{M} \right)^2 = \frac{n^2}{2} \cdot \frac{(n/2 - 1) \cdot 4\ell^2}{2M^2} \quad (13)$$

Recall that the probability of winning equals the probability of gaining at least one marble. The latter probability is, by the inclusion-exclusion principle, bounded below as follows:

$$\sum_{\text{even } i=0}^{n-2} p_i^\sigma \quad \text{by (9)}$$

$$\leq \left( \frac{1}{2} \cdot \frac{n}{2^{n/2}} \right) / M \quad \text{by (9) and (11)}$$

$$\geq \left( B - \frac{1}{2} \cdot \frac{1}{2^{n/2}} \right) / M \quad \text{by (5) and (13)}$$

We conclude that the probability of winning is at least $\tau$.

Assume that Player Odd does not have a winning strategy in the \textsc{SubsetSum} game. Consider any scheduler $\sigma$ for the new MDP. Since the corresponding strategy in the \textsc{SubsetSum} game is not winning, there exists $y \in \{0,1\}^{n/2}$ with $c_0(\sigma, y) + c_2(\sigma, y) + \cdots + c_{n-2}(\sigma, y) \neq B$. By (6) it follows:

$$\sum_{\text{even } i=0}^{n-2} \hat{p}_i^\sigma(y) \leq (B - 1) / M \quad (14)$$

By the union bound the probability of gaining at least one marble is bounded above as follows:

$$\sum_{\text{even } i=0}^{n-2} p_i^\sigma = \sum_{x \in \{0,1\}^{n/2}} \frac{1}{2^{n/2}} \sum_{\text{even } i=0}^{n-2} \hat{p}_i^\sigma(x) \quad \text{by (9)}$$

$$\leq \left( 1 - \frac{1}{2^{n/2}} \right) \frac{B}{M} + \frac{1}{2^{n/2}} \cdot \frac{B - 1}{M} \quad \text{by (8) and (14)}$$

$$= \left( B - \frac{1}{2^{n/2}} \right) / M \quad \text{by (5)}$$

We conclude that the probability of winning is less than $\tau$. 


This completes the log-space reduction.

**Theorem** The cost problem is EXPTIME-complete.

**Proof.** We reduce from the problem of determining the winner in a countdown game [12]. A countdown game is a tuple $(S, \Rightarrow, s_0, T)$ where $S$ is a finite set of states, $\Rightarrow \subseteq S \times \mathbb{N} \setminus \{0\} \times S$ is a transition relation, $s_0 \in S$ is the initial state, and $T$ is the final value. We write $s \xrightarrow{k} r$ if $(s, k, r) \in \Rightarrow$. A configuration of the game is an element $(s, c) \in S \times \mathbb{N}$. The game starts in configuration $(s_0, 0)$ and proceeds in moves: if the current configuration is $(s, c) \in S \times \mathbb{N}$, first Player 1 chooses a number $k$ with $0 < k \leq T - c$ and $s \xrightarrow{k} r$ for at least one $r \in S$; then Player 2 chooses a state $r \in S$ with $s \xrightarrow{k} r$. The resulting new configuration is $(r, c + k)$. Player 1 wins if she hits a configuration from $S \times \{T\}$, and she loses if she cannot move (and has not yet won). (We have slightly paraphrased the game from [12] for technical convenience, rendering the term countdown game somewhat inept.)

The problem of determining the winner in a countdown game was shown EXPTIME-complete in [12]. Let $(S, \Rightarrow, s_0, T)$ be a countdown game. We construct a cost process $C = (Q, s_0, t, A, \mathcal{E}n, \Delta)$ so that Player 1 can win the countdown game if and only if there is a scheduler $\sigma$ with $P_\sigma(K = T) = 1$. The intuition is that Player 1 corresponds to the scheduler and Player 2 corresponds to randomness. We take

$$Q := S \cup \{q_i : i \in \mathbb{N}, 2^i \leq T\} \cup \{t\}.$$  

Intuitively, the states in $S$ are used in a first phase, which directly reflects the countdown game. The states $q_i$ are used in a second phase, which is acyclic and ends in the final control state $t$.

For all $s \in S$ we take

$$\mathcal{E}n(s) := \{a_{stop}\} \cup \{k \in \mathbb{N} \setminus \{0\} : \exists r \in S, s \xrightarrow{k} r\}.$$  

Whenever $s \xrightarrow{k} r$, we set $\Delta(s, k)(r, k) > 0$. (We do not specify the exact values of positive probabilities, as they do not matter. For concreteness one could take a uniform distribution.) Those transitions directly reflect the countdown game. Whenever $s \xrightarrow{k} r$, we also set $\Delta(s, k)(q_0, k) > 0$. Those transitions allow “randomness” to enter the second phase, which starts in $q_0$. Further, for all $s \in S$ we set $\Delta(s, a_{stop})(t, 0) = 1$. Those transitions allow the scheduler to jump directly to the final control state $t$, skipping the second phase.

Now we describe the transitions in the second phase. Let $i_{\text{max}} \in \mathbb{N}$ be the largest integer with $2^{i_{\text{max}}} \leq T$. For all $i \in \{0, 1, \ldots, i_{\text{max}}\}$ we take $\mathcal{E}n(q_i) = \{a_0, a_1\}$ and

$$\Delta(q_i, a_0)(q_{i+1}, 0) = 1 \quad \text{and} \quad \Delta(q_i, a_1)(q_{i+1}, 2^i) = 1,$$

where $q_{i+1}$ is identified with $t$. The second phase allows the scheduler to incur an arbitrary cost between 0 and $T$ (and possibly more). That phase is acyclic and leads to $t$.

Observe that $t$ is reached with probability 1. We show that Player 1 can win the countdown game if and only if the scheduler in the cost process can achieve $K = T$ with probability 1.

Assume Player 1 can win the countdown game. Then the scheduler can emulate Player 1’s winning strategy. If randomness enters the second phase while the cost $c$ accumulated so far is at most $T$, then the scheduler incurs additional cost $T - c$ in the second phase and wins. If and when accumulated cost exactly $T$ is reached in the first phase, the scheduler plays $a_{\text{stop}}$. 

\[\square\]
so it jumps to $t$ and wins. Since the scheduler emulates Player 1's winning strategy, it will not get in a state in which the accumulated cost is larger than $T$.

Conversely, assume Player 2 has a winning strategy in the countdown game. If the scheduler jumps to $t$ while accumulated cost $T$ has not yet been reached, the scheduler loses. If the scheduler does not do that, randomness emulates with non-zero probability Player 2’s winning strategy. This leads to a state $(s, c) \in S \times \mathbb{N}$ with $c > T$, from which the scheduler loses with probability 1. This completes the log-space reduction.