# Multiple Mean-Payoff Optimization Under Local Stability Constraints

**David Klaška, Antonín Kučera, Vojtěch Kůr, Vít Musil, Vojtěch Řehák**

Masaryk University, Brno, Czechia
{david.klaska, vojtech.kur}@mail.muni.cz, {tony, musil, rehak}@fi.muni.cz

## Abstract

The long-run average payoff per transition (mean payoff) is the main tool for specifying the performance and dependability properties of discrete systems. The problem of constructing a controller (strategy) simultaneously optimizing several mean payoffs has been deeply studied for stochastic and game-theoretic models. One common issue of the constructed controllers is the *instability* of the mean payoffs, measured by the deviations of the average rewards per transition computed in a finite "window" sliding along a run. Unfortunately, the problem of simultaneously optimizing the mean payoffs under local stability constraints is computationally hard, and the existing works do not provide a practically usable algorithm even for non-stochastic models such as two-player games. In this paper, we design and evaluate the first efficient and scalable solution to this problem applicable to Markov decision processes.

**Code** — https://gitlab.fi.muni.cz/formela/2025-aaai-mmp
**Extended version** — https://arxiv.org/abs/2412.13369

## Introduction

*Mean payoff*, i.e., the long-run average payoff per time unit, is the standard tool for specifying and evaluating the long-run performance of dynamic systems. The overall performance is typically characterized by a tuple of mean payoffs computed for multiple payoff functions representing expenditures, income, resource consumption, and other relevant aspects. The basic task of *multiple mean-payoff optimization* is to design a controller (strategy) jointly optimizing these mean payoffs. Efficient strategy synthesis algorithms have been designed for various models, such as Markov decision processes or two-player games (see Related work).

A fundamental problem of the standard mean payoff optimization is the *lack of local stability guarantees*. For example, consider a payoff function modeling the expenditures of a company. Even if the long-run average expenditures per day (mean payoff) are \$1000, there are no guarantees on the maximal average expenditures per day in a *bounded time horizon* of, say, one month. It is possible that the company pays between \$800 and \$1200 per day every month, which is fine and sustainable. However, it may also happen that there are "good" and "bad" months with average daily expenditures of \$0 and \$5000, respectively, where the second case is not so frequent but represents a critical cashflow problem that may ruin the company. The mean payoff does not reflect this difference; hence, optimizing the mean payoff (minimizing the overall long-run expenditures) may lead to disastrous outcomes in certain situations.

The lack of local stability guarantees has motivated the study of *window mean payoff objectives*, where the task is to optimize the average payoff in a "window" of finite length sliding along a run. The window size represents a bounded time horizon, and the task is to construct a strategy such that the average payoff computed for the states in the window stays within given bounds. Thus, one can express both long-run average performance and stability constraints. For a single payoff function, some technical variants of this problem are solvable efficiently, while others are PSPACE-hard. For *multiple* window mean payoffs, intractability results have been established for two-player games (see Related work).

The main concern of the previous works on window mean payoffs is classifying the computational complexity of constructing an optimal strategy. The obtained algorithms are based on reductions to other problems, and their complexity matches the established lower bounds. To the best of our knowledge, there have been no attempts to tackle the high complexity of strategy synthesis by designing scalable algorithms at the cost of some (unavoidable but acceptable) compromises. This open challenge and the problem's high practical relevance are the primary motivations for our work.

**Our Contribution.** We design the first efficient and scalable strategy synthesis algorithm for optimizing multiple window mean payoffs in a given Markov decision process.

We start by establishing the principal limits of our efforts by showing that the problem is NP-hard even for simple instances where the underlying MDP is a graph. Consequently, *every* efficient algorithm attempting to solve the problem must inevitably suffer from some limitations. Our algorithm trades efficiency for optimality guarantees, i.e., the computed solutions are not necessarily optimal. Nevertheless, our experiments show that the algorithm can construct high-quality (and sometimes quite sophisticated) strategies for complicated instances of non-trivial size. Thus, we obtain the first practically usable solution to the problem of multi-

ple window mean payoff optimization.

More concretely, the optimization objective is specified by a function *Eval* measuring the "badness" of the achieved tuple of window mean payoffs, and the task is to *minimize* the expected value of *Eval* (we refer to the next section for precise definitions). The *Eval* function can specify complex requirements on the tuple of achieved mean payoffs. This overcomes another limitation of the previous works, where it was only possible to specify the desired upper/lower bounds for each window mean payoff separately.

The core of our strategy synthesis algorithm is a novel procedure based on dynamic programming, computing the distribution of local mean payoffs for a given starting state. Remarkably, this procedure is differentiable, and the corresponding gradient can be calculated (in the backward pass) at essentially the same computational costs.

A strategy maximizing the expected value of *Eval* may require memory. Our strategy synthesis algorithm produces *finite-memory randomized* strategies, where the memory size is a parameter. Using larger memory may produce better strategies but also substantially increases computational costs. From the scalability perspective, using randomization is *essential* because randomized strategies may achieve much better performance than deterministic strategies with the same amount of memory. This is demonstrated by the simple example below.

**Example 1.** Consider a graph $D$ with two states $A, B$ and two payoff functions where $Pay_1(A)=1$, $Pay_1(B)=0$, $Pay_2(A)=0$, and $Pay_2(B)=8$. We aim to construct a finite-memory strategy such that the pair of expected window mean payoffs in a window of length 8 is positive in both components and as close to $(1, 1)$ as possible with respect to $L_1$ (Manhattan) distance. Formally, for a given pair of window mean-payoffs $(wmp_1, wmp_2)$, the function *Eval* returns the $L_1$ distance to the vector $(1, 1)$ if both $wmp_1$ and $wmp_2$ are positive; otherwise, *Eval* returns a "penalty" equal to 5.

Suppose that the available memory has $K$ different states. Then, for every $K < 7$, there is a *randomized* strategy with $K$ memory states achieving a strictly better expected value of *Eval* than the best *deterministic* strategy with $K$ memory states. For $K = 1, 2$, such strategies are shown in Fig. 1 (for $K = 1$, the best deterministic strategy achieves the window mean payoffs $(\frac{1}{2}, 4)$ in *every* window of length 8, and hence the expected $L_1$ distance to $(1, 1)$ is 3.5; for $K = 2$, $2/3$ of the windows have mean payoffs $(\frac{5}{8}, 3)$, and $1/3$ (those beginning with $AA$) have $(\frac{3}{4}, 2)$, hence $\mathbb{E}[Eval] = 2$). The *optimal* finite-memory strategy achieving the window mean payoffs $(\frac{7}{8}, 1)$ in every window of length 8 (i.e., the expected value of *Eval* equal to $\frac{1}{8}$) requires 7 memory elements. □

In our experiments, we concentrate on the following:

(1) Demonstrating the improvement achieved by the dynamic procedure described above, where the baseline is a "naive" DFS-based procedure.

(2) Evaluating the scalability of our algorithm and the quality of the constructed strategies.

(3) Analyzing the power of randomization to compensate for insufficient memory.

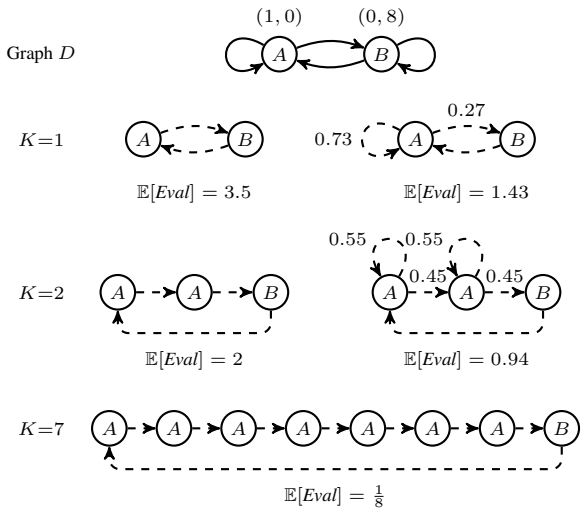


Figure 1: For a memory with $K$ states where $K \in \{1, 2\}$, the best deterministic strategy (left) is worse than a suitable randomized strategy (right). The optimal finite memory strategy requires 7 memory elements.

In (2), a natural baseline for evaluating the quality of a constructed strategy is the best value achievable by a finite-memory strategy. However, there is no algorithm for computing the best value. This issue is overcome by constructing a parameterized family of graphs where the best achievable value and the amount of required memory can be determined by a careful *manual analysis*. The structure of our parameterized graphs is similar to the ones constructed in our NP-hardness proofs mentioned above. Thus, we avoid bias towards simple instances of the problem. In (3), we analyze the quality of randomized strategies constructed for memory of increasing size. Roughly speaking, our experiments show that even in scenarios where the number of memory elements is *insufficient* for implementing an optimal strategy, the quality of the obtained strategies is still relatively close to the optimum. Hence, randomization can "compensate" for insufficient memory. This is encouraging because the number of memory elements is one of the crucial parameters influencing the computational costs.

**Related Work.** Single mean-payoff optimization for Markov decision processes (MDPs) is a classical problem studied for decades (see, e.g., (Puterman 1994)). For multiple mean-payoff optimization in MDPs, a polynomial-time algorithm computing optimal strategies and approximating the Pareto curve has been given in (Brázdil et al. 2011), and the solution has been successfully integrated into the software tool PRISM (Brázdil et al. 2015).

The window mean payoff objectives have been first studied for a single payoff function and non-stochastic two-player games in (Chatterjee et al. 2015), and then for Markov decision processes in (Bordais, Guha, and Raskin 2019; Brihaye et al. 2020). The complexity of the strategy synthesis algorithms proposed in these works ranges from polynomial time to polynomial space, depending on the con-

crete technical variant of the problem. The multiple window mean payoff objectives have been considered for two-player games in (Chatterjee et al. 2015), where the problem of computing an optimal strategy is classified as provably intractable (EXPTIME-hard). The optimization objective studied in these works is a simple conjunction of upper/lower bounds imposed on the achieved window mean payoffs. The main focus is on classifying the computational complexity of the studied problems, where the upper bounds are typically obtained by reductions to other game-theoretic problems. To the best of our knowledge, our work gives the first scalable algorithm for multiple window mean payoff optimization applicable to Markov decision processes of considerable size.

In a broader context, a related problem of window parity objective has been studied in (Chatterjee and Henzinger 2006; Horn 2007; Chatterjee, Henzinger, and Horn 2009). An alternative approach to capturing local stability of mean payoff based on bounding the variance of relevant random variables has been proposed in (Brázdil et al. 2017).

## The Model

We assume familiarity with basic notions of probability theory (probability distribution, expected value, conditional random variables, etc.) and Markov chain theory. The set of all probability distributions over a finite set $A$ is denoted by $\mathit{Dist}(A)$. We use $\mathbb{N}$ and $\mathbb{Q}_+$ do denote the set of non-negative integers and non-negative rationals, respectively.

**Markov chains.** A *Markov chain* is a triple $M = (S, \mathit{Prob}, \mu)$ where $S$ is a finite set of *states*, $\mathit{Prob} : S \times S \to [0,1]$ is a *stochastic matrix* where $\sum_{t \in S} \mathit{Prob}(s,t) = 1$ for all $s \in S$, and $\mu \in \mathit{Dist}(S)$ is an *initial distribution*.

A *run* in $M$ is an infinite sequence of states. We use $\mathbb{P}$ to denote the standard probability measure over the runs of $M$ (see, e.g., (Norris 1998))

A state $t$ is *reachable* from a state $s$ if $\mathit{Prob}^n(s,t) > 0$ for some $n \geq 1$. We say that $M$ is *strongly connected* (or *irreducible*) if all states are mutually reachable from each other. For an irreducible $M$, we use $\mathbb{I} \in \mathit{Dist}(S)$ to denote the unique *invariant distribution* satisfying $\mathbb{I} = \mathbb{I} \cdot \mathit{Prob}$ (note that $\mathbb{I}$ is independent of $\mu$). By ergodic theorem (Norris 1998), $\mathbb{I}$ is the limit frequency of visits to the states of $S$ along a run. More precisely, let $w = s_0, s_1, \ldots$ be a run of $M$. For every $s \in S$, let

$$\mathit{Freq}_s(w) = \lim_{n \to \infty} \frac{\#_s(s_0, \ldots, s_{n-1})}{n}$$

where $\#_s(s_0, \ldots, s_{n-1})$ is the number of occurrences of $s$ in $s_0, \ldots, s_{n-1}$. If the above limit does not exist, we put $\mathit{Freq}_s(w) = 0$. Furthermore, let $\mathit{Freq}(w) : S \to [0,1]$ be the vector of all $\mathit{Freq}_s(w)$ where $s \in S$. The ergodic theorem says that $\mathbb{P}[\mathit{Freq} = \mathbb{I}] = 1$.

A *bottom strongly connected component (BSCC)* of $M$ is a maximal $B \subseteq S$ such that $B$ is strongly connected and closed under reachable states. Note that if $M$ is irreducible, then the set $S$ is the only BSCC of $M$. Otherwise, $M$ can have multiple BSCCs, and each such $B$ can be seen as an irreducible Markov chain where the set of states is $B$ and the probability matrix is the restriction of $\mathit{Prob}$ to $B \times B$.

For the rest of this section, we fix an irreducible Markov chain $M = (S, \mathit{Prob}, \mu)$.

**Global and Local Mean Payoff.** Let $\mathit{Pay} : S \to \mathbb{N}$ be a *payoff function*. For every run $w = s_0, s_1, \ldots$, let

$$\mathrm{GMP}(w) = \lim_{n \to \infty} \frac{1}{n} \sum_{i=0}^{n-1} \mathit{Pay}(s_i)$$

be the limit-average payoff per transition along the run $w$. If the limit does not exist, we put $\mathrm{GMP}(w) = 0$. An immediate consequence of the ergodic theorem (see above) is that the defining limit of $\mathrm{GMP}(w)$ exists and takes the same value $\mathit{gmp} = \sum_{s \in S} \mathbb{I}(s) \cdot \mathit{Pay}(s)$ for almost all runs, i.e., $\mathbb{P}[\mathrm{GMP} = \mathit{gmp}] = 1$. We refer to the (unique) value $\mathit{gmp}$ of GMP as the *global mean payoff*.

Let $d \in \mathbb{N}$ be a *time horizon*. For every $j \in \mathbb{N}$, let

$$\mathrm{WMP}[d,j](w) = \frac{1}{d} \sum_{i=j}^{j+d-1} \mathit{Pay}(s_i)$$

be the average payoff computed for the sequence of $d$ consecutive states in $w$ starting with $s_j$. We refer to the value $\mathrm{WMP}[d,j]$ as the *window mean payoff after $j$ steps* (assuming some fixed $d$).

Note that as $d \to \infty$, the value of $\mathrm{WMP}[d,j]$ is arbitrarily close to $\mathit{gmp}$ with arbitrarily large probability, independently of $j$. However, for a given $d$, the value of $\mathrm{WMP}[d,j]$ depends on $j$ and can be rather different from $\mathit{gmp}$. Also observe that $\mathrm{WMP}[d,j]$ is a discrete random variable, and the underlying distribution depends only on the state $s_j$. More precisely, for all $j, m \in \mathbb{N}$ and $s \in S$ such that $\mathbb{P}[s_j = s] > 0$ and $\mathbb{P}[s_m = s] > 0$, we have that the conditional random variables $\mathrm{WMP}[d,j] \mid s_j = s$ and $\mathrm{WMP}[d,m] \mid s_m = s$ are identically distributed. In the following, we write just $\mathrm{WMP}[d,s]$ instead of $\mathrm{WMP}[d,j] \mid s_j = s$ where $\mathbb{P}[s_j = s] > 0$.

**Mean Payoff Objectives.** Let $\mathit{Pay}_1, \ldots, \mathit{Pay}_k : S \to \mathbb{N}$ be payoff functions. The standard multi-objective optimization problem for Markov Decision Processes (see below) is to jointly maximize the global mean payoffs for $\mathit{Pay}_1, \ldots, \mathit{Pay}_k$. In this paper, we use a more general approach where the "badness" of the achieved mean payoffs is measured by a dedicated function $\mathit{Eval} : \mathbb{R}^k \to \mathbb{R}$. A smaller value of $\mathit{Eval}$ indicates more appropriate payoffs.

For example, the joint maximization of the mean payoffs can be encoded by $\mathit{Eval}_{\max}(\vec{x}) = \sum_{i=1}^{k} (\mathit{Pay}_i^{\max} - \vec{x}_i)$, where $\mathit{Pay}_i^{\max} = \max_{s \in S} \mathit{Pay}_i(s)$. The defining sum can also be weighted to reflect certain priorities among the Pareto optimal solutions. In general, $\mathit{Eval}$ can encode the preference of keeping the mean payoffs close to some values, within some interval, or even enforce some mutual dependencies among the mean payoffs.

As we shall see, our strategy synthesis algorithm works for an arbitrary $\mathit{Eval}$ that is *decomposable*. Intuitively, the decomposability condition enables more efficient evaluation/differentiation of $\mathit{Eval}$ by dynamic programming without substantially restricting the expressive power.

Now we can define the *global* and *local* value of a run as follows:

$$\mathit{GVAL}(w) = \mathit{Eval}(\mathrm{GMP}_1(w), \ldots, \mathrm{GMP}_k(w))$$

$$\mathit{WVAL}(w) = \lim_{n\to\infty} \frac{1}{n} \sum_{j=0}^{n-1} \mathit{Eval}\big(\mathrm{WMP}_1[d,j], \ldots, \mathrm{WMP}_k[d,j]\big)$$

where $\mathrm{GMP}_i(w)$ and $\mathrm{WMP}_i[d,j]$ are the global and window mean payoff determined by $\mathit{Pay}_i$. Note that $\mathit{WVAL}(w)$ corresponds to the "limit-average badness" of the window mean-payoffs along the run $w$.

Clearly, $\mathbb{P}[\mathit{GVAL}{=}\mathit{gval}] = 1$, where $\mathit{gval} = \mathit{Eval}(\mathit{gmp}_1, \ldots, \mathit{gmp}_k)$. Furthermore, by applying the ergodic theorem and the above observations leading to the definition of $\mathrm{WMP}_i[d,s]$, we obtain that $\mathbb{P}[\mathit{WVAL}{=}\mathit{wval}] = 1$ where

$$\mathit{wval} = \sum_{s\in S} \mathbb{I}(s) \cdot \mathbb{E}\big[\mathit{Eval}(\mathrm{WMP}_1[d,s], \ldots, \mathrm{WMP}_k[d,s])\big] \quad (1)$$

We refer to *gval* and *wval* as the *global* and *window mean payoff value*. In this paper, we study the problem of minimizing *wval* in a given Markov decision process.

**Markov decision processes.** A *Markov decision process (MDP)*[1] is a triple $D = (V, E, p)$ where $V$ is a finite set of *vertices* partitioned into subsets $(V_N, V_S)$ of *non-deterministic* and *stochastic* vertices, $E \subseteq V{\times}V$ is a set of *edges* s.t. each vertex has at least one out-going edge, and $p\colon V_S{\to}\mathit{Dist}(V)$ is a *probability assignment* such that $p(v)(v'){>}0$ only if $(v,v') \in E$. We say that $D$ is a *graph* if $V_S{=}\emptyset$.

Outgoing edges in non-deterministic vertices are selected by a *strategy*. In this paper, we consider *finite-memory randomized (FR)* strategies where the selection depends not only on the vertex currently visited but also on some finite information about the history of vertices visited previously.

**FR strategies.** Let $D = (V, E, p)$ be an MDP and $M{\neq}\emptyset$ a finite set of *memory states* that are used to "remember" some information about the history. For a given pair $(v, m)$ where $v$ is a currently visited vertex and $m$ a current memory state, a strategy randomly selects a new pair $(v', m')$ such that $(v, v') \in E$.

Formally, let $\alpha\colon V \to 2^M$ be a *memory allocation*, and let $\overline{V} = \{(v,m) \mid v \in V, m \in \alpha(v)\}$ be the set of *augmented vertices*. A *finite-memory (FR) strategy* is a function $\sigma\colon \overline{V} \to \mathit{Dist}(\overline{V})$ such that for all $(v,m) \in \overline{V}$ where $v \in V_S$ and every $(v,v') \in E$ we have that

$$\sum_{m'\in\alpha(v')} \sigma(v,m)(v',m') \;=\; p(v)(v')\,.$$

An FR strategy is *memoryless* (or *Markovian*) if $M$ is a singleton. In the following, we use $\overline{v}$ to denote an augmented vertex of the form $(v,m)$ for some $m \in \alpha(v)$.

Every FR strategy $\sigma$ together with a probability distribution $\mu \in \mathit{Dist}(\overline{V})$ determine the Markov chain $D^\sigma = (\overline{V}, \mathit{Prob}, \mu)$ where $\mathit{Prob}(\overline{v},\overline{u}) = \sigma(\overline{v})(\overline{u})$.

[1] Our definition of MDPs is standard in the area of graph games. Although it is equivalent to the "classical" MDP definition of (Puterman 1994), it is more convenient for our purposes and leads to simpler notation.

## The Optimization Problem

In this section, we define the multiple window mean-payoff optimization problem and examine the principal limits of its computational tractability.

Let $D = (V, E, p)$ be an MDP, $\mathit{Pay}_1, \ldots, \mathit{Pay}_k : V{\to}\mathbb{N}$ payoff functions, and $\mathit{Eval} : \mathbb{R}^k \to \mathbb{R}$ an evaluation function. Furthermore, let $d \in \mathbb{N}$ be a time horizon. The task is to construct an FR strategy $\sigma$ and an initial augmented vertex so that the value of *wval* is *minimized*.

More precisely, for a given FR strategy $\sigma$, the function *wval* is evaluated as follows. Every $\mathit{Pay}_i$ is extended to the augmented vertices of $D^\sigma$ by defining $\mathit{Pay}_i(\overline{v}) = \mathit{Pay}_i(v)$. Let $C_1, \ldots, C_n$ be the BSCCs of the Markov chain $D^\sigma$. Recall that every $C_i$ can be seen as an irreducible Markov chain. We use $\mathit{wval}(C_i)$ to denote the window mean payoff value computed for $C_i$. The value of *wval* achieved by $\sigma$, denoted by $\mathit{wval}^\sigma$, is defined as

$$\mathit{wval}^\sigma \;=\; \min\{\mathit{wval}(C_i) \mid 1 \le i \le n\}$$

Recall that the initial augmented vertex can be chosen freely, which is reflected in the above definition (the strategy $\sigma$ can be initiated directly in the "best" BSCC and thus achieve the value $\mathit{wval}^\sigma$).

A FR-strategy $\sigma$ is *$\varepsilon$-optimal* for a given $\varepsilon \ge 0$ if

$$\mathit{wval}^\sigma - \varepsilon \;\le\; \inf_\pi \mathit{wval}^\pi$$

where $\pi$ ranges over all FR strategies. A 0-optimal strategy is called *optimal*.

The next theorem shows that computing an optimal strategy is computationally hard, even for restricted subsets of instances.

**Theorem 1** *Let $D = (V, E, p)$ be a graph, $\mathit{Pay}_1, \ldots, \mathit{Pay}_k$ payoff functions, $d \le |V|$ a time horizon, and Eval an evaluation function. The problem of whether there exists a FR strategy $\sigma$ such that $\mathit{wval}^\sigma \le 0$ is* NP*-hard.*

*Furthermore, the problem is* NP*-hard even if the set of eligible instances is restricted so that an optimal* memoryless *strategy is guaranteed to exist, and one of the following three conditions is satisfied:*

- *A.* $k{=}1$ *(i.e., there is only* one *payoff function).*
- *B.* $k{=}2$*, and there are thresholds $c_1, c_2 \ge 0$ such that $\mathit{Eval}(\kappa_1,\kappa_2) = 0$ iff $\kappa_1 \ge c_1$ and $\kappa_2 \ge c_2$.*
- *C. There is a constant $r \in \mathbb{N}$ such that $k \le |V|^{\frac{1}{r}}$ (i.e., the number of payoff functions is "substantially smaller" than the number of vertices), and the co-domain of every $\mathit{Pay}_i$ is $\{0,1\}$.*

Intuitively, (A) says that one payoff function is sufficient for NP-hardness, (B) says that for two payoff functions we have NP-hardness even if we just aim to push both window mean payoffs above certain thresholds, and (C) says that the range of all payoff functions can be restricted to $\{0,1\}$ even if the number of payoff functions is at most $|V|^{\frac{1}{r}}$. Furthermore, the *Eval* function can be constructed so that it ranges over $\{-t, 0\}$ where $t \in \mathbb{N}$ is an arbitrary constant, and $\mathit{wval}^\sigma = 0$ iff $\sigma$ is optimal. Consequently, an optimal (and even $t{-}1$-optimal) strategy cannot be constructed in polynomial time unless $\mathsf{P} = \mathsf{NP}$.

## The Algorithm

For the rest of this section, we fix an MDP $D = (V, E, p)$, a time horizon $d$, payoff functions $Pay_1, \ldots, Pay_k$, and an evaluation function *Eval*.

Our algorithm is based on optimizing *wval* by the methods of differentiable programming. The core ingredient is a dynamic procedure for computing the expected value of $Eval(\mathrm{WMP}_1[d,s], \ldots, \mathrm{WMP}_k[d,s])$ (see (1)). The procedure is designed so that the gradient of *wval* can be computed using automatic differentiation. Then, we show how to incorporate this procedure into a strategy-improvement algorithm for *wval*. For the rest of this section, we fix a memory allocation $\alpha : V \to 2^M$.

**Representing FR Strategies.** For every pair of augmented vertices $(\overline{v}, \overline{u})$ such that $(v, u) \in E$, we fix a real-valued parameter representing $\sigma(\overline{v})(\overline{u})$. Note that if $\overline{v}$ is stochastic, then the parameter actually represents the probability of selecting the memory state of $\overline{u}$. These parameters are initialized to random values, and we use the standard SOFTMAX function to transform these parameters into probability distributions. Thus, every function $F$ depending on $\sigma$ becomes a function of the parameters, and we use $\nabla F$ to denote the corresponding gradient.

**Computing $wval^\sigma$.** Let $\sigma$ be an FR strategy where $\alpha$ is the memory allocation. We show how to compute $wval^\sigma$ interpreted as a function of the parameters representing $\sigma$.

Recall that $wval^\sigma = \min\{wval(C_i) \mid 1 \leq i \leq n\}$. Hence, the first step is to compute all BSCCs of $D^\sigma$ by the Tarjan's algorithm (Tarjan 1972). Then, for each BSCC $C$, we compute $wval(C)$ in the following way.

The invariant distribution $\mathbb{I}_C$ is computed as the unique solution of the following system of linear equations: For every $\overline{v} \in C$, we fix a fresh variable $x_{\overline{v}}$ and the corresponding equation $x_{\overline{v}} = \sum_{\overline{u} \in C} x_{\overline{u}} \cdot \sigma(\overline{u})(\overline{v})$. Furthermore, we add the equation $\sum_{\overline{u} \in C} x_{\overline{u}} = 1$. Recall that $\mathbb{I}_C$ is the unique distribution satisfying $\mathbb{I}_C = \mathbb{I}_C \cdot Prob_C$, where $Prob_C$ is the probability matrix of $C$. Hence, $\mathbb{I}_C$ is the unique solution of the constructed system.

The main challenge is to compute the expected value

$$\mathbb{E}\big[Eval(\mathrm{WMP}_1[d, \overline{v}^*], \ldots, \mathrm{WMP}_k[d, \overline{v}^*])\big] \qquad (2)$$

for each $\overline{v}^* \in C$. This is achieved by Algorithm 1 described below. Then, $wval(C)$ is calculated using (1).

**Computing the Expected Value of *Eval* by Dynamic Programming.** In this section, we show how to compute (2) by dynamic programming for a given $\overline{v}^* \in C$. We use $\mathbb{P}$ to denote the probability measure over the runs in $C$, where the initial distribution assigns 1 to $\overline{v}^*$.

Let $\mathbb{N}^C$ be the set of vectors of non-negative integers indexed by the augmented vertices $\overline{v} \in C$. For each $t \in \mathbb{N}$, let $\mathbb{N}^C_t = \{x \in \mathbb{N}^C \mid \ell_x = t\}$, where $\ell_x = \sum_{\overline{u} \in C} x(\overline{u})$. For every $x \in \mathbb{N}^C$, let $Occ_x$ be an indicator assigning to every run $w = \overline{v}_0, \overline{v}_1, \ldots$ of $C$ either 1 or 0 so that $Occ_x(w) = 1$ iff $\#_{\overline{v}}(\overline{v}_0, \ldots \overline{v}_{\ell_x - 1}) = x(\overline{v})$ for every $\overline{v} \in C$.

For every $x \in \mathbb{N}^C$, let

$$\mathrm{WMP}_i(x) = \frac{\sum_{\overline{v} \in C} x(\overline{v}) \cdot Pay_i(\overline{v})}{\ell_x}.$$

Algorithm 1: Evaluation via dynamic programming

```
ϱ* = m(r({0,...,0}), v̄*)
map_0[(v̄*, ϱ*)] = 1.
for t ∈ {1,...,d−1} do
    for ((v̄, ϱ), p) ∈ map_0 do
        for ū ∈ C do
            ϱ' = m(ϱ, ū)
            map_1[(ū, ϱ')] += p · σ[v̄][ū]
    swap(map_0, map_1)
    map_1.clear()
rsl = 0.
for ((v̄, ϱ), p) ∈ map_0 do
    rsl += p · e(ϱ)
```

Then, (2) is equal to

$$\sum_{x \in \mathbb{N}^C_d} \mathbb{P}[Occ_x{=}1] \cdot Eval(\mathrm{WMP}_1(x), \ldots, \mathrm{WMP}_k(x)). \quad (3)$$

Calculating (3) directly is time-consuming. However, $Eval(\mathrm{WMP}_1(x), \ldots, \mathrm{WMP}_k(x))$ is the same for many different $x \in \mathbb{N}^C_d$, and our dynamic algorithm avoids these redundant computations. The algorithm is applicable to a subclass of *decomposable Eval* functions defined below. The decomposability condition is not too restrictive, and it does not influence the NP-hardness of the considered optimization problem (the *Eval* functions constructed in the proof of Theorem 1 are decomposable).

For all $x, y \in \mathbb{N}^C$ and $\overline{v} \in C$, we write $x \to^{\overline{v}} y$ if $y(\overline{v}) = x(\overline{v}) + 1$ and $y(\overline{u}) = x(\overline{u})$ for all $\overline{u} \neq \overline{v}$. We say that *Eval* is *decomposable* if there is a set $R$ of *representatives* and efficiently computable functions $r\colon \mathbb{N}^C \to R$, $e\colon R \to \mathbb{R}$ and $m\colon R \times C \to R$ satisfying the following conditions:

- $Eval(\mathrm{WMP}_1(x), \ldots, \mathrm{WMP}_k(x)) = e(r(x))$ for every $x \in \mathbb{N}^C_d$, i.e., the value of *Eval* for a given $x \in \mathbb{N}^C_d$ is efficiently computable by the function $e$ just from the representative of $x$.
- For each $x \to^{\overline{v}} y$, we have that $r(y) = m(r(x), \overline{v})$. That is, when a path is prolonged by a vertex $\overline{v}$, the representative can be efficiently updated by the function $m$.

A concrete example of a decomposable *Eval* and the associated $r, e, m$ functions is given in a special subsection below.

For each $t \in \{1, \ldots, d\}$, let $R_t = \{r(x) \mid x \in \mathbb{N}^C_t\}$. Furthermore, for every representative $\varrho \in R_t$, let $\mathbb{P}[\varrho] = \sum_{x \in \mathbb{N}^C_t \cap r^{-1}(\varrho)} \mathbb{P}[Occ_x{=}1]$. Then (3) can be rewritten into

$$\sum_{\varrho \in R_d} \mathbb{P}[\varrho] \cdot e(\varrho). \qquad (4)$$

Algorithm 1 computes $\mathbb{P}[\varrho]$ for all $t \in \{1, \ldots, d\}$ and $\varrho \in R_t$ by dynamic programming. Moreover, only reachable representatives (i.e., those with $\mathbb{P}[\varrho] > 0$) are considered during the computation. Thus, Algorithm 1 avoids the redundancies of the direct computation of (3).

More specifically, Algorithm 1 uses two associative arrays (such as C++ *unordered_map*), called $map_0$ and $map_1$, to gather information about the representatives and the corresponding probabilities. In the $t$-th iteration of the cycle,

Algorithm 2: Evaluation via DFS

```
1: if n < d then
2:     for ū ∈ C do
3:         DFS(ū, p · σ[v̄][ū], n + 1, payoff_vector + Pay(v̄))
4: else
5:     rsl += p · Eval(payoff_vector)
```

$map_0$ contains items corresponding to all reachable $\varrho \in R_t$, and the items corresponding to all reachable $\varrho \in R_{t+1}$ are gradually gathered in $map_1$. In particular, each $map_i$ is indexed by the elements $(\overline{v}, \varrho) \in C \times R$. Intuitively, each such element represents the set of all runs $w = \overline{v}_0, \overline{v}_1, \ldots$ initiated in $\overline{v}^*$ such that there exists $x \in \mathbb{N}^C_{t+i}$ satisfying $r(x) = \varrho$, $Occ_x(w) = 1$, and $\overline{v}_{\ell_x - 1} = \overline{v}$. The value associated to $(\overline{v}, \varrho)$ is the total probability of all such runs $w$.

**A Simple DFS Procedure.** As a baseline for measuring the improvement achieved by the dynamic algorithm described in the previous paragraph, we use a simple DFS-based procedure of Algorithm 2. For simplicity, the vector $(Pay_1(\overline{v}), \ldots, Pay_k(\overline{v}))$ is denoted by $Pay(\overline{v})$.

The DFS procedure inputs the following parameters:

- the current augmented vertex $\overline{v}$;
- the probability $p$ of the current path;
- the length $n$ of the current path;
- the vector $payoff\_vector$ of the individual payoffs accumulated along the path.

The procedure is called as DFS($\overline{v}^*, 1, 1, Pay(\overline{v})$) for each $\overline{v}^*$ in the currently examined BSCC $C$. At the end of the computation, the global variable $rsl$ holds the value of (2).

**A Strategy Improvement Algorithm.** In this section, we describe a strategy improvement algorithm WINMPSYNT that inputs an MDP $D = (V, E, p)$, payoff functions $Pay_1, \ldots, Pay_k : V \to \mathbb{N}$, a decomposable $Eval : \mathbb{R}^k \to \mathbb{R}$, and a time horizon $d \in \mathbb{N}$, and computes an FR strategy $\sigma$ with the aim of minimizing $wval^\sigma$.

The memory allocation function is a hyperparameter (by default, all memory states are assigned to every vertex). The algorithm proceeds by randomly choosing the parameters representing a strategy. The values are sampled from LOGUNIFORM distribution so that no prior knowledge about the solution is imposed. Then, the algorithm computes the BSCCs of $D^\sigma$ and identifies a BSCC $C$ with the best $wval(C)$. Subsequently, $wval(C)$ is improved by gradient descent. The crucial ingredient of WINMPSYNT is Algorithm 1, allowing to compute $wval(C)$ and its gradient by automatic differentiation. After that, the point representing the current strategy is updated in the direction of the steepest descent. The intermediate solutions and the corresponding $wval(C)$ values are stored, and the best solution found within STEPS optimization steps is returned (the value of STEPS is a hyper-parameter). Our implementation uses PYTORCH framework (Paszke et al. 2019) and its automatic differentiation with ADAM optimizer (Kingma and Ba 2015). Observe that WINMPSYNT is equally efficient for general MDPs and graphs. The only difference is that stochastic vertices generate fewer parameters.

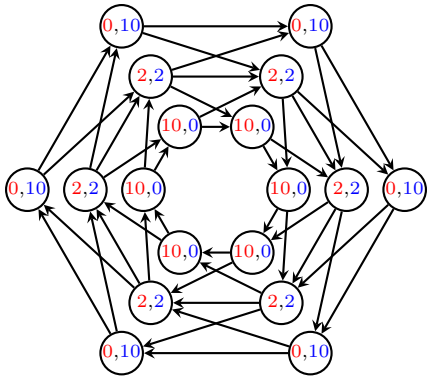


Figure 2: The graph $D_6$ and the payoffs assigned to vertices.

**An Example of a Decomposable *Eval* Function.** Let *Pay* be a payoff function, and let $Eval : \mathbb{R} \to \mathbb{R}$ where $Eval(P)$ is either 0 or 1 depending on whether $P \in [3, 5]$ or not, respectively. Assume $d = 5$. Then, we can put $R = \{0, \ldots, 26\}$ and define

$$r(x) = \min\{a, 26\}, \text{ where } a = \sum_{\overline{v} \in C} x(\overline{v}) \cdot Pay(\overline{v})$$

$$e(\varrho) = \begin{cases} 0 & \text{if } \varrho/d \in [3, 5], \\ 1 & \text{otherwise.} \end{cases}$$

$$m(\varrho, \overline{v}) = \min\{\varrho + Pay(\overline{v}), 26\}$$

Note that as soon as the accumulated payoff exceeds 25, there is no reason to remember the exact value because *Eval* inevitably becomes one. Note that $R$ contains 27 elements *independently* of the size of $C$, while the total number of all $x \in \mathbb{N}^C_5$ such that $\mathbb{P}[Occ_x{=}1] > 0$ may exceed $\binom{|C|}{4}$, and the total number of paths, all of which are considered separately by the naive DFS-based algorithm, may reach $|C|^4$.

## Experiments

We perform our experiments on graphs to separate the probabilistic choice introduced by the constructed strategies from the internal probabilistic choice performed in stochastic vertices. The graphs are structurally similar to the ones constructed in the NP-hardness proof of Theorem 1. This avoids bias towards simple instances. Recall that the problem of constructing a (sub)optimal strategy for such graphs is NP-hard even if just one memory state is allocated to every vertex, there are only two payoff functions, and we aim at pushing the window mean payoffs above certain thresholds (see item B. in Theorem 1).

**The graphs $D_\ell$.** For every $\ell \geq 2$, we construct a directed ring $D_\ell$ with three "layers" where every vertex in the inner, middle, and outer layer is assigned a pair of payoffs $(10, 0)$, $(2, 2)$, and $(0, 10)$, respectively. The vertices are connected in the way shown in Fig. 2.

**The *Eval* function.** A *scenario* is a pair $(\ell, d)$ where $\ell, d$ are even integers in the interval $[2, 20]$ representing $D_\ell$ and the window length $d$. For every scenario, we aim to push both window mean payoffs simultaneously above a bound $b$,

where $b$ is as large as possible. For each $(\ell, d)$, the *maximal* bound achievable by an FR strategy is denoted by $b_{\ell,d}$, and can be determined by a careful *manual analysis*, together with the least number of memory states $K_{\ell,d}$ required by an optimal FR strategy (we have that $K_{\ell,d} \leq 5$ for all $(\ell, d)$). Let us note that the manual analysis of $D_\ell$ is enabled by the regular structure of $D_\ell$, but this regularity does not bring *any advantage* to WINMPSYNT.

For a given scenario $(\ell, d)$, we use the evaluation function $Eval_{\ell,d}$ defined as follows:

$$Eval_{\ell,d}(P_1, P_2) = \frac{\max\{0, b_{\ell,d} - P_1\} + \max\{b_{\ell,d} - P_2\}}{2 \cdot b_{\ell,d}}.$$

By the definition of $b_{\ell,d}$, there always exists an optimal FR strategy $\sigma$ for the scenario $(\ell, d)$ achieving the bound $b_{\ell,d}$, i.e., $wval^\sigma = 0$. Due to the normalizing denominator, the maximal value of $Eval_{\ell,d}$ is bounded by 1, simplifying the comparison of strategies constructed for different scenarios.

**The experiments.** For all scenarios $(\ell, d)$ and $K \in \{1, \ldots, 5\}$, we invoked WINMPSYNT 100 times, where $K$ memory states are assigned to every vertex. The number of optimization steps is set to 1000. Thus, for all $(\ell, d)$ and $K$, we obtained a set $\Sigma_{\ell,d,K}$ of 100 strategies and the corresponding values. We use $\Sigma_{\ell,d}$ to denote $\bigcup_{K=1}^{5} \Sigma_{\ell,d,K}$, and $\Sigma$ to denote the union of all $\Sigma_{\ell,d}$.

**The quality of the obtained strategies.** Due to the definition of $Eval_{\ell,d}$, for every $\sigma \in \Sigma_{\ell,d}$, the value $wval^\sigma$ can be interpreted as a normalized distance to the *optimal* strategy with value 0. The percentage of scenarios where the value of the *best* strategy found by WINMPSYNT is bounded by 0, 0.05, and 0.1 is 40%, 52%, and 100%, respectively. Hence, the best strategies found by WINMPSYNT are of *very high quality*. Since the best strategy is selected out of 500 strategies constructed for a given scenario, a natural question is how good are these strategies "on average". The percentage of all $\sigma \in \Sigma$ whose value is bounded by 0, 0.1, and 0.3 is 6%, 27%, and 99%, respectively. Hence, the quality of an "average" strategy is substantially worse, which is consistent with intuitive expectations (since the problem is computationally hard, obtaining a high-quality solution for non-trivial instances cannot be easy).

The roles of memory states and randomization are demonstrated in Fig. 3. The scenarios are split into five disjoint subsets with the same $K_{\ell,d}$ (horizontal axis). For each subset, we report the values achieved by strategies with $1, \ldots, 5$ memory states assigned to every vertex (indicated by different colors). Note that for the subset where $K_{\ell,d} = 2$, an optimal strategy is computed when 2 or more memory states are available. For the subset where $K_{\ell,d} = 3$, an optimal strategy is found only for 5 memory states. For all subsets, increasing the number of memory states decreases the average strategy value. Furthermore, even if the number of memory states is smaller than $K_{\ell,d}$, the value of the constructed strategies is still relatively small on average. Hence, randomization effectively compensates for the lack of memory.

**The improvement achieved by dynamic programming.** The baseline for evaluating the efficiency improvement

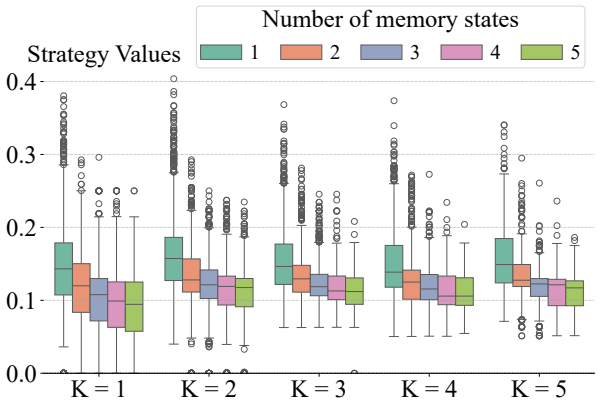


Figure 3: More memory states lead to better strategies.

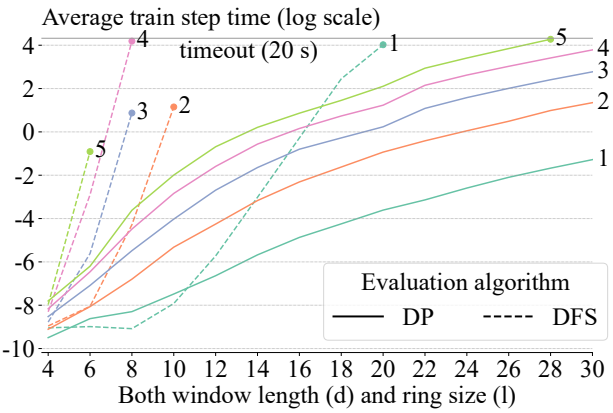


Figure 4: Dynamic evaluation procedure outperforms the DFS-based one.

achieved by the dynamic procedure of Algorithm 1 is the simple DFS-based procedure of Algorithm 2. For every instance $(\ell, d)$ where $4 \leq \ell{=}d \leq 30$, we report the average running time of one training step for an FR strategy with $1, \ldots, 5$ memory states (different colors) using *logarithmic* scale. The timeout is set to 20 secs. For one memory state, the DFS-based procedure reaches the timeout for all scenarios $(\ell, d)$ where $\ell{=}d \leq 20$, whereas the dynamic procedure needs less than one second even for the $(30, 30)$ scenario. Hence, the dynamic procedure substantially outperforms the DFS-based one, and the same holds when the number of memory states increases.

## Conclusions

We have designed an efficient strategy synthesis algorithm for optimizing multiple window mean payoffs capable of producing high-quality solutions for instances of considerable size. An interesting question is whether the proposed approach is applicable to a larger class of window-based optimization objectives such as the window parity objectives.

## References


Bordais, B.; Guha, S.; and Raskin, J.-F. 2019. Expected Window Mean-Payoff. In *Proceedings of FST&TCS 2019*, volume 150 of *Leibniz International Proceedings in Informatics*, 32:1–32:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

Brázdil, T.; Brožek, V.; Chatterjee, K.; Forejt, V.; and Kučera, A. 2011. Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. In *Proceedings of LICS 2011*. IEEE Computer Society Press.

Brázdil, T.; Chatterjee, K.; Forejt, V.; and Kučera, A. 2015. MultiGain: A Controller Synthesis Tool for MDPs with Multiple Mean-Payoff Objectives. In *Proceedings of TACAS 2015*, volume 9035 of *Lecture Notes in Computer Science*, 181–187. Springer.

Brázdil, T.; Chatterjee, K.; Forejt, V.; and Kučera, A. 2017. Trading performance for stability in Markov decision processes. *Journal of Computer and System Sciences*, 84: 144–170.

Brihaye, T.; Delgrange, F.; Oualhadj, Y.; and Randour, M. 2020. Life is Random, Time is Not: Markov Decision Processes with Window Objectives. *Logical Methods in Computer Science*, 16(4).

Chatterjee, K.; Doyen, L.; Randour, M.; and Raskin, J.-F. 2015. Looking at Mean-Payoff and Total-Payoff through Windows. *Information and Computation*, 242: 25–52.

Chatterjee, K.; and Henzinger, T. 2006. Finitary winning in $\omega$-regular games. In *Proceedings of TACAS 2006*, volume 3920 of *Lecture Notes in Computer Science*, 257–271. Springer.

Chatterjee, K.; Henzinger, T.; and Horn, F. 2009. Stochastic games with finitary objectives. In *Proceedings of MFCS 2009*, volume 5734 of *Lecture Notes in Computer Science*, 34–54. Springer.

Horn, F. 2007. Faster Algorithms for Finitary Games. In *Proceedings of TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, 472–484. Springer.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of ICLR 2015*.

Norris, J. 1998. *Markov Chains*. Cambridge University Press.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc.

Puterman, M. 1994. *Markov Decision Processes*. Wiley.

Tarjan, R. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM Journal of Computing*, 1(2).