Looking at Mean-Payoff and Total-Payoff through Windows

Krishnendu Chatterjee[1,⋆], Laurent Doyen[2], Mickael Randour[3,†], and Jean-François Raskin[4,‡]

[1] IST Austria (Institute of Science and Technology Austria)
[2] LSV - ENS Cachan, France
[3] Computer Science Department, Université de Mons (UMONS), Belgium
[4] Département d'Informatique, Université Libre de Bruxelles (U.L.B.), Belgium

Abstract. We consider two-player games played on weighted directed graphs with mean-payoff and total-payoff objectives, two classical quantitative objectives. While for single-dimensional games the complexity and memory bounds for both objectives coincide, we show that in contrast to multi-dimensional mean-payoff games that are known to be coNP-complete, multi-dimensional total-payoff games are undecidable. We introduce conservative approximations of these objectives, where the payoff is considered over a local finite window sliding along a play, instead of the whole play. For single dimension, we show that (i) if the window size is polynomial, deciding the winner takes polynomial time, and (ii) the existence of a bounded window can be decided in NP $\cap$ coNP, and is at least as hard as solving mean-payoff games. For multiple dimensions, we show that (i) the problem with fixed window size is EXPTIME-complete, and (ii) there is no primitive-recursive algorithm to decide the existence of a bounded window.

1 Introduction

Mean-payoff and total-payoff games. Two-player mean-payoff and total-payoff games are played on finite weighted directed graphs (in which every edge has an integer weight) with two types of vertices: in player-1 vertices, player 1 chooses the successor vertex from the set of outgoing edges; in player-2 vertices, player 2 does likewise. The game results in an infinite path through the graph, called a *play*. The mean-payoff (resp. total-payoff) value of a play is the long-run average (resp. sum) of the edge-weights along the path. While traditionally games on graphs with ω-regular objectives have been studied for system analysis, research efforts have recently focused on quantitative extensions to model resource constraints of embedded systems, such as power consumption, or buffer size [2]. Quantitative games, such as mean-payoff games, are crucial for the formal analysis of resource-constrained reactive systems. For the analysis of systems with multiple resources, multi-dimension games, where edge weights are integer vectors, provide the appropriate framework.

⋆ Author supported by Austrian Science Fund (FWF) Grant No P 23499-N23, FWF NFN Grant No S11407 (RiSE), ERC Start Grant (279307: Graph Games), Microsoft faculty fellowship.
† Author supported by F.R.S.-FNRS. fellowship.
‡ Author supported by ERC Starting Grant (279499: inVEST).

Decision problems. The decision problem for mean-payoff and total-payoff games asks, given a starting vertex, whether player 1 has a strategy that against all strategies of the opponent ensures a play with value at least 0. For both objectives, *memoryless* winning strategies exist for both players (where a memoryless strategy is independent of the past and depends only on the current state) [9,12]. This ensures that the decision problems belong to NP $\cap$ coNP; and they belong to the intriguing class of problems that are in NP $\cap$ coNP but whether they are in P (deterministic polynomial time) are long-standing open questions. The study of mean-payoff games has also been extended to multiple dimensions where the problem is shown to be coNP-complete [21,4]. While for one dimension all the results for mean-payoff and total-payoff coincide, our first contribution shows that quite unexpectedly (in contrast to multi-dimensional mean-payoff games) the multi-dimensional total-payoff games are undecidable.

Window objectives. On the one hand, the complexity of single-dimensional mean-payoff and total-payoff games is a long-standing open problem, and on the other hand, the multi-dimensional problem is undecidable for total-payoff games. In this work, we propose to study variants of these objectives, namely, *bounded window mean-payoff* and *fixed window mean-payoff* objectives. In a bounded window mean-payoff objective instead of the long-run average along the whole play we consider payoffs over a local bounded window sliding along a play, and the objective is that the average weight must be at least zero over every bounded window from some point on. This objective can be seen as a strengthening of the mean-payoff objective (resp. of the total-payoff objective if we require that the window objective is satisfied from the beginning of the play rather than from some point on), i.e., winning for the bounded window mean-payoff objective implies winning for the mean-payoff objective. In the fixed window mean-payoff objective the window length is fixed and given as a parameter. Observe that winning for the fixed window objective implies winning for the bounded window objective.

Attractive features for window objectives. First, they are a strengthening of the mean-payoff objectives and hence provide conservative approximations for mean-payoff objectives. Second, the window variant is very natural to study in system analysis. Mean-payoff objectives require average to satisfy certain threshold in the long-run (or in the limit of the infinite path), whereas the window objectives require to provide guarantee on the average, not in the limit, but within a bounded time, and thus provide better time guarantee than the mean-payoff objectives. Third, the window parameter provides flexibility, as it can be adjusted specific to applications requirement of strong or weak time guarantee for system behaviors. Finally, we will establish that our variant in the single dimension is more computationally tractable, which makes it an attractive alternative to mean-payoff objectives.

Our contributions. The main contributions of this work (along with the undecidability of multi-dimensional total-payoff games) are as follows:

1. *Single dimension.* For the single-dimensional case we present an algorithm for the fixed window problem that is polynomial in the size of the game graph times the length of the binary encoding of weights times the size of the fixed window. Thus if the window size is polynomial, we have a polynomial-time algorithm. For the bounded window problem we show that the decision problem is in NP $\cap$ coNP, and

	one-dimension			k-dimension		
	complexity	$\mathcal{P}_1$ mem.	$\mathcal{P}_2$ mem.	complexity	$\mathcal{P}_1$ mem.	$\mathcal{P}_2$ mem.
$\underline{\mathsf{MP}}$ / $\overline{\mathsf{MP}}$	NP∩coNP	mem-less		coNP-c. / NP∩coNP	infinite	mem-less
$\underline{\mathsf{TP}}$ / $\overline{\mathsf{TP}}$	NP∩coNP	mem-less		**undec.** (Thm. 1)	-	-
WMP: fixed polynomial window	**P-c.** (Thm. 2)	**mem. req.** $\leq$ **linear**($\lvert S\rvert \cdot l_{\max}$) (Thm. 2)		**PSPACE-h.** (Thm. 4) **EXP-easy** (Thm. 4)	**exponential** (Thm. 4)	
WMP: fixed arbitrary window	**P**($\lvert S\rvert, V, l_{\max}$) (Thm. 2)			**EXP-c.** (Thm. 4)		
WMP: bounded window problem	**NP∩coNP** (Thm. 3)	**mem-less** (Thm. 3)	**infinite** (Thm. 3)	**NPR-h.** (Thm. 5)	-	-

Table 1: Complexity of deciding the winner and memory required, with $|S|$ the number of states of the game (vertices in the graph), V the length of the binary encoding of weights, and $l_{\max}$ the window size. New results in bold (h. for hard and c. for complete).

at least as hard as solving mean-payoff games. However, winning for mean-payoff games does not imply winning for the bounded window mean-payoff objective, i.e., the winning sets for mean-payoff games and bounded window mean-payoff games do not coincide. Moreover, the structure of winning strategies is also very different, e.g., in mean-payoff games both players have memoryless winning strategies, but in bounded window mean-payoff games we show that player 2 requires infinite memory. We also show that if player 1 wins the bounded window mean-payoff objective, then a window of size $(|S|-1)\cdot(|S|\cdot W+1)$ is sufficient where S is the state space (the set of vertices of the graph), and W is the largest absolute weight value. Finally, we show that (i) a winning strategy for the bounded window mean-payoff objective ensures that the mean-payoff is at least 0 regardless of the strategy of the opponent, and (ii) a strategy that ensures that the mean-payoff is strictly greater than 0 is winning for the bounded window mean-payoff objective.

2. *Multiple dimensions*. For multiple dimensions, we show that the fixed window problem is EXPTIME-complete (both for arbitrary dimensions with weights in $\{-1,0,1\}$ and for two dimensions with arbitrary weights); and if the window size is polynomial, then the problem is PSPACE-hard. For the bounded window problem we show that the problem is non-primitive recursive hard (i.e., there is no primitive recursive algorithm to decide the problem).
3. *Memory requirements.* For all the problems for which we prove decidability we also characterize the memory required by winning strategies.

The relevant results are summarized in Table 1: our results are in bold fonts. In summary, the fixed window problem provides an attractive approximation of the mean-payoff and total-payoff games that we show have better algorithmic complexity. In contrast to the long-standing open problem of mean-payoff games, the one-dimension fixed window problem with polynomial window size can be solved in polynomial time; and in contrast to the undecidability of multi-dimensional total-payoff games, the multi-dimension fixed window problem is EXPTIME-complete.

Related works. An extended version of this work, including proofs, can be found in [5]. Mean-payoff games have been first studied by Ehrenfeucht and Mycielski in [9] where it is shown that memoryless winning strategies exist for both players. This result entails that the decision problem lies in NP ∩ coNP [17,22], and it was later shown to belong to

UP ∩ coUP [15]. Despite many efforts [13,22,19,18,14], no polynomial-time algorithm for the mean-payoff games problem is known so far. Gurvich, Karzanov, Khachivan and Lebedev [13,17] provided the first (exponential) algorithm for mean-payoff games, later extended by Pisaruk [19]. The first pseudo-polynomial-time algorithm for mean-payoff games was given in [22] and was improved in [1]. Lifshits and Pavlov [18] propose an algorithm which is polynomial in the encoding of weights but exponential in the number of vertices of the graph: it is based on a graph decomposition procedure. Bjorklund and Vorobyov [14] present a *randomized* algorithm which is both subexponential and pseudo-polynomial. While all the above works are for single dimension, multi-dimensional mean-payoff games have been studied in [21,4,7]. One-dimension total-payoff games have been studied in [11] where it is shown that memoryless winning strategies exist for both players and the decision problem is in UP ∩ coUP.

2 Multi-Dimensional Mean-Payoff and Total-Payoff Objectives

We consider two-player turn-based games and denote the two *players* by $\mathcal{P}_1$ and $\mathcal{P}_2$.

Multi-weighted two-player game structures. *Multi-weighted two-player game structures* are weighted graphs $G = (S_1, S_2, E, k, w)$ where (*i*) S_1 and S_2 resp. denote the finite sets of vertices, called *states*, belonging to $\mathcal{P}_1$ and $\mathcal{P}_2$, with $S_1 \cap S_2 = \emptyset$ and $S = S_1 \cup S_2$; (*ii*) $E \subseteq S \times S$ is the set of *edges* such that for all $s \in S$, there exists $s' \in S$ with $(s, s') \in E$; (*iii*) $k \in \mathbb{N}$ is the *dimension* of the weight vectors; and (*iv*) $w \colon E \to \mathbb{Z}^k$ is the multi-weight labeling function. When it is clear from the context that a game G is one-dimensional ($k = 1$), we omit k and write it as $G = (S_1, S_2, E, w)$. The game structure G is *one-player* if $S_2 = \emptyset$. We denote by W the largest absolute weight that appears in the game. For complexity issues, we assume that weights are encoded in binary. Hence we differentiate between pseudo-polynomial algorithms (polynomial in W) and truly polynomial algorithms (polynomial in $V = \lceil \log_2 W \rceil$, the number of bits needed to encode the weights).

A *play* in G from an initial state $s_{\mathsf{init}} \in S$ is an infinite sequence of states $\pi = s_0 s_1 s_2 \ldots$ such that $s_0 = s_{\mathsf{init}}$ and $(s_i, s_{i+1}) \in E$ for all $i \geq 0$. The *prefix* up to the n-th state of π is the finite sequence $\pi(n) = s_0 s_1 \ldots s_n$. Let $\mathsf{Last}(\pi(n)) = s_n$ denote the last state of $\pi(n)$. A prefix $\pi(n)$ belongs to $\mathcal{P}_i$, $i \in \{1, 2\}$, if $\mathsf{Last}(\pi(n)) \in S_i$. The set of plays of G is denoted by $\mathsf{Plays}(G)$ and the corresponding set of prefixes is denoted by $\mathsf{Prefs}(G)$. The set of prefixes that belong to $\mathcal{P}_i$ is denoted by $\mathsf{Prefs}_i(G)$. The infinite suffix of a play starting in s_n is denoted $\pi(n, \infty)$.

The *total-payoff* of a prefix $\rho = s_0 s_1 \ldots s_n$ is $\mathsf{TP}(\rho) = \sum_{i=0}^{i=n-1} w(s_i, s_{i+1})$, and its *mean-payoff* is $\mathsf{MP}(\rho) = \frac{1}{n}\mathsf{TP}(\rho)$. This is naturally extended to plays by considering the componentwise limit behavior (i.e., limit taken on each dimension). The *infimum (resp. supremum) total-payoff* of a play π is $\underline{\mathsf{TP}}(\pi) = \liminf_{n\to\infty} \mathsf{TP}(\pi(n))$ (resp. $\overline{\mathsf{TP}}(\pi) = \limsup_{n\to\infty} \mathsf{TP}(\pi(n))$). The *infimum (resp. supremum) mean-payoff* of π is $\underline{\mathsf{MP}}(\pi) = \liminf_{n\to\infty} \mathsf{MP}(\pi(n))$ (resp. $\overline{\mathsf{MP}}(\pi) = \limsup_{n\to\infty} \mathsf{MP}(\pi(n))$).

Strategies. A *strategy* for $\mathcal{P}_i$, $i \in \{1, 2\}$, in G is a function $\lambda_i \colon \mathsf{Prefs}_i(G) \to S$ such that $(\mathsf{Last}(\rho), \lambda_i(\rho)) \in E$ for all $\rho \in \mathsf{Prefs}_i(G)$. A strategy λ_i for $\mathcal{P}_i$ has *finite-memory* if it can be encoded by a deterministic finite state machine with outputs (Moore machine). It is *memoryless* if it does not depend on history but only on the current state of the

game. A play π is said to be *consistent* with a strategy λ_i of $\mathcal{P}_i$ if for all $n \geq 0$ such that $\mathsf{Last}(\pi(n)) \in S_i$, we have $\mathsf{Last}(\pi(n+1)) = \lambda_i(\pi(n))$. Given an initial state $s_{\mathsf{init}} \in S$, and two strategies, λ_1 for $\mathcal{P}_1$ and λ_2 for $\mathcal{P}_2$, the unique play from s_{init} consistent with both strategies is the *outcome* of the game, denoted by $\mathsf{Outcome}_G(s_{\mathsf{init}}, \lambda_1, \lambda_2)$.

Attractors. The *attractor* for $\mathcal{P}_1$ of a set $A \subseteq S$ in G is denoted by $\mathsf{Attr}_G^{\mathcal{P}_1}(A)$ and computed as the fixed point of the sequence $\mathsf{Attr}_G^{\mathcal{P}_1,n+1}(A) = \mathsf{Attr}_G^{\mathcal{P}_1,n}(A) \cup \{s \in S_1 \mid \exists (s,t) \in E, t \in \mathsf{Attr}_G^{\mathcal{P}_1,n}(A)\} \cup \{s \in S_2 \mid \forall (s,t) \in E, t \in \mathsf{Attr}_G^{\mathcal{P}_1,n}(A)\}$, with $\mathsf{Attr}_G^{\mathcal{P}_1,0}(A) = A$. The attractor $\mathsf{Attr}_G^{\mathcal{P}_1}(A)$ is exactly the set of states from which $\mathcal{P}_1$ can ensure to reach A no matter what $\mathcal{P}_2$ does. The attractor $\mathsf{Attr}_G^{\mathcal{P}_2}(A)$ for $\mathcal{P}_2$ is defined symmetrically.

Objectives. An *objective* for $\mathcal{P}_1$ in G is a set of plays $\phi \subseteq \mathsf{Plays}(G)$. A play $\pi \in \mathsf{Plays}(G)$ is *winning* for an objective ϕ if $\pi \in \phi$. Given a game G and an initial state $s_{\mathsf{init}} \in S$, a strategy λ_1 of $\mathcal{P}_1$ is winning if $\mathsf{Outcome}_G(s_{\mathsf{init}}, \lambda_1, \lambda_2) \in \phi$ for all strategies λ_2 of $\mathcal{P}_2$. Given a rational threshold vector $v \in \mathbb{Q}^k$, we define the *infimum (resp. supremum) total-payoff (resp. mean-payoff) objectives* as follows:

- $\mathsf{TotalInf}_G(v) = \{\pi \in \mathsf{Plays}(G) \mid \underline{\mathsf{TP}}(\pi) \geq v\}$
- $\mathsf{TotalSup}_G(v) = \{\pi \in \mathsf{Plays}(G) \mid \overline{\mathsf{TP}}(\pi) \geq v\}$
- $\mathsf{MeanInf}_G(v) = \{\pi \in \mathsf{Plays}(G) \mid \underline{\mathsf{MP}}(\pi) \geq v\}$
- $\mathsf{MeanSup}_G(v) = \{\pi \in \mathsf{Plays}(G) \mid \overline{\mathsf{MP}}(\pi) \geq v\}$

Decision problem. Given a game structure G, an initial state $s_{\mathsf{init}} \in S$, and an inf./sup. total-payoff/mean-payoff objective $\phi \subseteq \mathsf{Plays}(G)$, the *threshold problem* asks to decide if $\mathcal{P}_1$ has a winning strategy for this objective. In one-dimension games, both mean-payoff and total-payoff threshold problems lie in NP $\cap$ coNP [11]. In multi-dimension, the mean-payoff threshold problem lies in coNP [21]. In contrast, we show that multi-dimension total-payoff games are undecidable.

Theorem 1. *The threshold problem for infimum and supremum total-payoff objectives is undecidable in multi-dimension games, for five dimensions.*

We reduce the halting problem for two-counter machines to the threshold problem for two-player total-payoff games with five dimensions. Counters take values $(v_1, v_2) \in \mathbb{N}^2$ along an execution, and can be incremented or decremented (if positive). A counter can be tested for equality to zero, and the machine can branch accordingly. We build a game with a sup. (resp. inf.) total-payoff objective of threshold $(0,0,0,0,0)$ for $\mathcal{P}_1$, in which $\mathcal{P}_1$ has to faithfully simulate an execution of the machine, and $\mathcal{P}_2$ can retaliate if he does not. We present gadgets by which $\mathcal{P}_2$ checks that *(a)* the counters are always non-negative, and that *(b)* a zero test is only passed if the value of the counter is really zero. The current value of counters (v_1, v_2) along an execution is encoded as the total sum of weights since the start of the game, $(v_1, -v_1, v_2, -v_2, -v_3)$, with v_3 being the number of steps of the computation. Hence, along a faithful execution, the 1st and 3rd dimensions are always non-negative, while the 2nd, 4th and 5th are always non-positive. To check that counters never go below zero, $\mathcal{P}_2$ is always able to go to an absorbing state with a self-loop of weight $(0,1,1,1,1)$ (resp. $(1,1,0,1,1)$). To check that all zero tests on counter 1 (resp. 2) are faithful, $\mathcal{P}_2$ can branch after a test to an absorbing state with a self-loop of weight $(1,0,1,1,1)$ (resp. $(1,1,1,0,1)$). Using these gadgets, $\mathcal{P}_2$ can punish

an unfaithful simulation as he ensures that the sum in the dimension on which $\mathcal{P}_1$ has cheated always stays strictly negative and the outcome is thus losing (it is only the case if $\mathcal{P}_1$ cheats, otherwise all dimensions become non-negative). When an execution halts (with counters equal to zero w.l.o.g.) after a faithful execution, it goes to an absorbing state with weight $(0,0,0,0,1)$, ensuring a winning outcome for $\mathcal{P}_1$ for the total-payoff objective. If an execution does not halt, the 5th dimension stays strictly negative and the outcome is losing.

In multi-weighted total-payoff games, $\mathcal{P}_1$ may need infinite memory. Consider a game with only one state and two self-loops of weights $(1,-2)$ and $(-2,1)$. For any threshold $v \in \mathbb{Q}^2$, $\mathcal{P}_1$ has an infinite-memory strategy to win the sup. total-payoff objective: alternating between the two loops for longer and longer periods, each time waiting to get back above the threshold in the considered dimension before switching. There exists no finite-memory one as the negative amount to compensate grows boundlessly with each alternation.

3 Window Mean-Payoff: Definition

In one dimension, no polynomial algorithm is known for mean-payoff and total-payoff, and in multi dimensions, total-payoff is undecidable. We introduce the *window mean-payoff objective*, a conservative approximation for which local deviations from the threshold must be compensated in a parametrized number of steps. We consider a *window*, sliding along a play, within which the compensation must happen. Our approach can be applied to mean-payoff and total-payoff objectives. Since we consider *finite* windows, both versions coincide for threshold zero. Hence we present our results for mean-payoff.

Objectives and decision problems. Given a multi-weighted two-player game $G = (S_1, S_2, E, k, w)$ and a rational threshold $v \in \mathbb{Q}^k$, we define the following objectives.[5]

- Given $l_{\max} \in \mathbb{N}_0$, the *good window* objective

$$\mathsf{GW}_G(v, l_{\max}) = \Big\{ \pi \mid \forall t,\, 1 \le t \le k,\, \exists l \le l_{\max},\, \frac{1}{l} \sum_{p=0}^{l-1} w\Big(e_\pi(p, p+1)\Big)(t) \ge v(t) \Big\}, \quad (1)$$

  where $e_\pi(p, p+1)$ is the edge $(\mathsf{Last}(\pi(p)), \mathsf{Last}(\pi(p+1)))$, requires that for all dimensions, there exists a window starting in the first position and bounded by $l_{\max}$ over which the mean-payoff is at least equal to the threshold.
- Given $l_{\max} \in \mathbb{N}_0$, the *fixed window mean-payoff* objective

$$\mathsf{FixWMP}_G(v, l_{\max}) = \Big\{ \pi \mid \exists i \ge 0,\, \forall j \ge i,\, \pi(j, \infty) \in \mathsf{GW}_G(v, l_{\max}) \Big\} \quad (2)$$

  requires that there exists a position i such that in all subsequent positions, good windows bounded by $l_{\max}$ exist.
- The *bounded window mean-payoff* objective

$$\mathsf{BndWMP}_G(v) = \Big\{ \pi \mid \exists l_{\max} > 0,\, \pi \in \mathsf{FixWMP}_G(v, l_{\max}) \Big\} \quad (3)$$

  asks that there exists a bound $l_{\max}$ such that the play satisfies the fixed objective.

[5] For brevity, we omit that $\pi \in \mathsf{Plays}(G)$.

We define *direct* versions of the objectives by fixing $i = 0$ rather than quantifying it existentially. For any $v \in \mathbb{Q}^k$ and $l_{\max} \in \mathbb{N}_0$, the following inclusions are true:

$$\mathsf{DirFixWMP}_G(v, l_{\max}) \subseteq \mathsf{FixWMP}_G(v, l_{\max}) \subseteq \mathsf{BndWMP}_G(v), \tag{4}$$

$$\mathsf{DirFixWMP}_G(v, l_{\max}) \subseteq \mathsf{DirBndWMP}_G(v) \subseteq \mathsf{BndWMP}_G(v). \tag{5}$$

The threshold v can be taken equal to $\{0\}^k$ (where $\{0\}^k$ denotes the k-dimension zero vector) w.l.o.g. as we can transform the weight function w to $b \cdot w - a$ for any threshold $\frac{a}{b}$, $a \in \mathbb{Z}^k$, $b \in \mathbb{N}_0 = \mathbb{N} \setminus \{0\}$. Hence, given any variant of the objective, the associated *decision problem* is to decide the existence of a winning strategy for $\mathcal{P}_1$ for threshold $\{0\}^k$. Lastly, for complexity purposes, we make a difference between *polynomial* (in the size of the game) and *arbitrary* (i.e., non-polynomial) window sizes.

Let $\pi = s_0 s_1 s_2 \dots$ be a play. Fix any dimension $t, 1 \leq t \leq k$. The window from position j to j', $0 \leq j < j'$, is *closed* iff there exists j'', $j < j'' \leq j'$ such that the sum of weights in dimension t over the sequence $s_j \dots s_{j''}$ is non-negative. Otherwise the window is *open*. Given a position j' in π, a window is still open in j' iff there exists a position $0 \leq j < j'$ such that the window from j to j' is open. Consider any edge (s_i, s_{i+1}) appearing along π. If the edge is non-negative in dimension t, the window starting in i immediately closes. If not, a window opens that must be closed within $l_{\max}$ steps. Consider the *first* position i' such that this window closes, then we have that all intermediary opened windows also get closed by i', that is, for any i'', $i < i'' \leq i'$, the window starting in i'' is closed before or when reaching position i'. Indeed, the sum of weights over the window from i'' to i' is strictly greater than the sum over the window from i to i', which is non-negative. We call this fact the *inductive property of windows*.

Illustration. Consider the game depicted in Fig. 1. It has a unique outcome, and it is winning for the classical mean-payoff objective of threshold 0, as well as for the infimum (resp. supremum) total-payoff objective of threshold -1 (resp. 0). Consider the fixed window mean-payoff objective for threshold 0. If the size of the window is bounded by 1, the play is losing.[6] However, if the window size is at least 2, the play is winning, as in s_3 we close the window in two steps and in s_4 in one step. Notice that by definition of the objective, it is clear that it is also satisfied for all larger sizes.[7] As the fixed window objective is satisfied for size 2, the bounded window objective is also satisfied. On the other hand, if we restrict the objectives to their direct variants, then none is satisfied, as from s_2, no window, no matter how large it is, gets closed.

Consider the game of Fig. 2. Again, the unique strategy of $\mathcal{P}_1$ satisfies the mean-payoff objective for threshold 0. It also ensures value -1 for the infimum and supremum total-payoffs. Consider the strategy of $\mathcal{P}_2$ that takes the self-loop once on the first visit of s_2, twice on the second, and so on. Clearly, it ensures that windows starting in s_1 stay open for longer and longer numbers of steps (we say that $\mathcal{P}_2$ *delays* the closing of the window), hence making the outcome losing for the bounded window objective (and thus the fixed window objective for any $l_{\max} \in \mathbb{N}_0$). This illustrates the added guarantee

[6] A window size of one actually requires that all infinitely often visited edges are of non-negative weights.

[7] The existential quantification on the window size l, bounded by $l_{\max}$, is indeed crucial in eq. (1) to ensure monotonicity with increasing maximal window sizes, a desired behavior of the definition for theoretical properties and intuitive use in specifications.

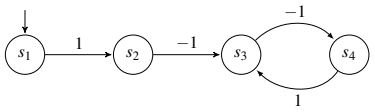

Fig. 1: Fixed window is satisfied for $l_{\max} \geq 2$, whereas even direct bounded window is not.

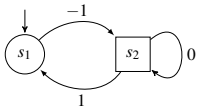

Fig. 2: Mean-payoff is satisfied but none of the window objectives is.

(compared to mean-payoff) asked by the window objective: in this case, no upper bound can be given on the time needed for a window to close, i.e., on the time needed to get the local sum back to non-negative. Note that $\mathcal{P}_2$ has to go back to s_1 at some point: otherwise, the prefix-independence of the objectives[8] allows $\mathcal{P}_1$ to wait for $\mathcal{P}_2$ to settle on cycling and win. For the direct variants, $\mathcal{P}_2$ has a simpler winning strategy consisting in looping forever, as enforcing one permanently open window is sufficient.

Relation with classical objectives. We introduce the bounded window objectives as conservative approximations of mean-payoff and total-payoff in one-dimension games. Indeed, in Lemma 1, we show that winning the bounded window (resp. direct bounded window) objective implies winning the mean-payoff (resp. total-payoff) objective while the reverse implication is only true if a strictly positive mean-payoff (resp. arbitrary high total-payoff) can be ensured.

Lemma 1. *Given a one-dimension game $G = (S_1, S_2, E, w)$, the following assertions hold.*

(a) If the answer to the bounded window mean-payoff problem is YES*, then the answer to the mean-payoff threshold problem for threshold zero is also* YES.
(b) If there exists $\varepsilon > 0$ such that the answer to the mean-payoff threshold problem for threshold ε is YES*, then the answer to the bounded window mean-payoff problem is also* YES.
(c) If the answer to the direct bounded window mean-payoff problem is YES*, then the answer to the supremum total-payoff threshold problem for threshold zero is also* YES.
(d) If the answer to the supremum total-payoff threshold problem is YES *for all integer thresholds (i.e., the total-payoff value is ∞), then the answer to the direct bounded window mean-payoff problem is also* YES.

Assertions *(a)* and *(c)* follow from the decomposition of winning plays into bounded windows of non-negative weights. The key idea for assertions *(b)* and *(d)* is that mean-payoff and total-payoff objectives always admit *memoryless* winning strategies, for which the consistent outcomes can be decomposed into *simple cycles* (i.e., with no repeated edge) over which the mean-payoff is at least equal to the threshold and which length is bounded. Hence they correspond to closing windows. Note that strict equivalence with the classical objectives is not verified, as witnessed before (Fig. 2).

[8] Fixed and bounded window mean-payoff objectives are prefix-independent: for all $\rho \in \mathsf{Prefs}(G)$, $\pi \in \mathsf{Plays}(G)$, we have that $\rho \cdot \pi$ is winning if and only if π is winning.

4 Window Mean-Payoff: One-Dimension Games

Fixed window. Given a game $G = (S_1, S_2, E, w)$ and a window size $l_{\max} \in \mathbb{N}_0$, we present an iterative algorithm FWMP (Alg. 1) to compute the winning states of $\mathcal{P}_1$ for the objective $\mathsf{FixWMP}_G(0, l_{\max})$. Initially, all states are potentially losing for $\mathcal{P}_1$. The algorithm iteratively declares states to be winning, removes them, and continues the computation on the remaining subgame as follows. In every iteration, *i)* DirectFWMP computes the set W_d of states from which $\mathcal{P}_1$ can win the direct fixed window objective; *ii)* it computes the attractor to W_d; and then proceeds to the next iteration on the remaining subgame (the restriction of G to a subset of states $A \subseteq S$ is denoted $G \restriction A$). In every iteration, the states of the computed set W_d are obviously winning for the fixed window objective. Thanks to the prefix-independence of the fixed window objective, the attractor to W_d is also winning. Since $\mathcal{P}_2$ must avoid entering this attractor, $\mathcal{P}_2$ must restrict his choices to stay in the subgame, and hence we iterate on the remaining subgame. Thus states removed over all iterations are winning for $\mathcal{P}_1$. The key argument to establish correctness is as follows: when the algorithm stops, the remaining set of states $\overline{W}$ is such that $\mathcal{P}_2$ can ensure to stay in $\overline{W}$ and falsify the direct fixed window objective by forcing the appearance of one open window larger than $l_{\max}$. Since he stays in $\overline{W}$, he can repeatedly use this strategy to falsify the fixed window objective. Thus the remaining set $\overline{W}$ is winning for $\mathcal{P}_2$, and the correctness of the algorithm follows.

Algorithm 1 $\mathsf{FWMP}(G, l_{\max})$

Require: $G = (S_1, S_2, E, w)$ and $l_{\max} \in \mathbb{N}_0$
Ensure: W is the set of winning states for $\mathcal{P}_1$ for $\mathsf{FixWMP}_G(0, l_{\max})$
  $n := 0$; $W := \emptyset$
  repeat
    $W_d^n := \mathsf{DirectFWMP}(G, l_{\max})$
    $W_{attr}^n := \mathsf{Attr}_G^{\mathcal{P}_1}(W_d^n)$ {attractor for $\mathcal{P}_1$}
    $W := W \cup W_{attr}^n$; $G := G \restriction (S \setminus W)$; $n := n + 1$
  until $W = S$ or $W_{attr}^{n-1} = \emptyset$
  return W

Algorithm 2 $\mathsf{DirectFWMP}(G, l_{\max})$

Require: $G = (S_1, S_2, E, w)$ and $l_{\max} \in \mathbb{N}_0$
Ensure: W_d is the set of winning states for $\mathcal{P}_1$ for $\mathsf{DirFixWMP}_G(0, l_{\max})$
  $W_{gw} := \mathsf{GoodWin}(G, l_{\max})$
  if $W_{gw} = S$ or $W_{gw} = \emptyset$ **then**
    $W_d := W_{gw}$
  else
    $W_d := \mathsf{DirectFWMP}(G \restriction W_{gw}, l_{\max})$
  return W_d

Algorithm 3 $\mathsf{GoodWin}(G, l_{\max})$

Require: $G = (S_1, S_2, E, w)$ and $l_{\max} \in \mathbb{N}_0$
Ensure: W_{gw} is the set of winning states for $\mathsf{GW}_G(0, l_{\max})$
  for all $s \in S$ **do**
    $C_0(s) := 0$
  for all $i \in \{1, \ldots, l_{\max}\}$ **do**
    for all $s \in S_1$ **do**
      $C_i(s) := \max_{(s,s') \in E} \{w((s,s')) + C_{i-1}(s')\}$
    for all $s \in S_2$ **do**
      $C_i(s) := \min_{(s,s') \in E} \{w((s,s')) + C_{i-1}(s')\}$
  return $W_{gw} := \{s \in S \mid \exists i,\ 1 \le i \le l_{\max}, C_i(s) \ge 0\}$

The main idea of algorithm DirectFWMP (Alg. 2) is that to win the direct fixed window objective, $\mathcal{P}_1$ must be able to repeatedly win the good window objective, which consists in ensuring a non-negative sum in at most $l_{\max}$ steps. A winning strategy of $\mathcal{P}_1$ in a state s is thus a strategy that enforces a non-negative sum and, *as soon as the sum turns non-negative* (in some state s'), starts doing the same from s'. It is important to start again immediately as it ensures that all suffixes along the path from s to s' also have a non-negative sum thanks to the inductive property. The states from which $\mathcal{P}_1$

can win the good window objective are computed by subroutine GoodWin (Alg. 3): given a state $s \in S$ and a number of steps $i \geq 1$, the value $C_i(s)$ is computed iteratively (from $C_{i-1}(s)$) and represents the best sum that $\mathcal{P}_1$ can ensure from s in exactly i steps. Hence, the set of winning states for $\mathcal{P}_1$ is the set of states for which there exists some i, $1 \leq i \leq l_{\max}$ such that $C_i(s) \geq 0$. The construction implies linear bounds (in $|S| \cdot l_{\max}$) on the memory needed for both players. We show that the fixed window problem is P-hard even for $l_{\max} = 1$ and weights $\{-1, 1\}$ via a simple reduction from reachability games.

Theorem 2. *In two-player one-dimension games, (a) the fixed arbitrary window mean-payoff problem is decidable in time $\mathcal{O}\left(|S|^3 \cdot |E| \cdot l_{\max} \cdot V\right)$, with $V = \lceil \log_2 W \rceil$, the length of the binary encoding of weights, and (b) the fixed polynomial window mean-payoff problem is P-complete. In general, both players require memory, and memory of size linear in $|S| \cdot l_{\max}$ is sufficient.*

Bounded window. We establish a NP $\cap$ coNP algorithm for bounded window mean-payoff objective using two intermediate results. First, if $\mathcal{P}_1$ has a strategy to win the sup. total-payoff objective, then he wins the good window objective for $l_{\max} = (|S|-1) \cdot (|S| \cdot W + 1)$. Second, if $\mathcal{P}_2$ has a memoryless strategy to ensure that the sup. total-payoff is strictly negative, then all consistent outcomes violate the direct bounded window mean-payoff objective. As a corollary, we obtain that the sets of winning states coincide for objectives $\mathsf{FixWMP}_G(0, l_{\max} = (|S|-1) \cdot (|S| \cdot W + 1))$ and $\mathsf{BndWMP}_G(0)$.

Algorithm 4 $\mathsf{BoundedProblem}(G)$

Require: Game $G = (S_1, S_2, E, w)$
Ensure: W_{bp} is the set of winning states for $\mathcal{P}_1$ for the bounded window mean-payoff problem
$W_{bp} := \emptyset$
$L := \mathsf{UnbOpenWindow}(G)$
while $L \neq S \setminus W_{bp}$ **do**
$W_{bp} := \mathsf{Attr}_G^{\mathcal{P}_1}(S \setminus L)$
$L := \mathsf{UnbOpenWindow}\Big(G \restriction (S \setminus W_{bp})\Big)$
return W_{bp}

Algorithm 5 $\mathsf{UnbOpenWindow}(G)$

Require: Game $G = (S_1, S_2, E, w)$
Ensure: L is the set of states from which $\mathcal{P}_2$ can force a position for which the window never closes
$p := 0$; $L_0 := \emptyset$
repeat
$L_{p+1} := L_p \cup \mathsf{Attr}^{\mathcal{P}_2}_{G \restriction (S \setminus L_p)}\Big(\mathsf{NegSupTP}\big(G \restriction (S \setminus L_p)\big)\Big)$
$p := p + 1$
until $L_p = L_{p-1}$
return $L := L_p$

Algorithm BoundedProblem (Alg. 4) computes via subroutine UnbOpenWindow the states from which $\mathcal{P}_2$ can force the visit of a position such that the window opening in this position never closes. To prevent $\mathcal{P}_1$ from winning the bounded window problem, $\mathcal{P}_2$ must be able to do so repeatedly as the prefix-independence of the objective otherwise gives the possibility to wait that all such bad positions are encountered before taking the windows into account. Thus, the states that are not in $\mathsf{UnbOpenWindow}(G)$, as well as their attractor, are winning for $\mathcal{P}_1$. Since the choices of $\mathcal{P}_2$ are reduced because of the attractor of $\mathcal{P}_1$ being declared winning, we compute in several steps, adding new states to the set of winning states for $\mathcal{P}_1$ up to stabilization. Subroutine UnbOpenWindow (Alg. 5) computes the attractor for $\mathcal{P}_2$ of the set of states from which $\mathcal{P}_2$ can enforce a strictly negative supremum total-payoff. Routine NegSupTP returns this set in NP $\cap$ coNP complexity [11]. Again, we compute the fixed point of the sequence as at each iteration, the choices of $\mathcal{P}_1$ are reduced. The main idea of the correctness proof is that from all states in $\overline{W_{bp}}$, $\mathcal{P}_2$ has an infinite-memory winning strategy which is played in rounds, and in round n ensures an open window of size at least n

by playing the total-payoff strategy of $\mathcal{P}_2$ for at most $n \cdot |S|$ steps, and then proceeds to round $(n+1)$ to ensure an open window of size $(n+1)$, and so on. Hence, windows stay open for arbitrary large periods and the bounded window objective is falsified.

The algorithm gives memoryless winning strategies for $\mathcal{P}_1$. The game of Fig. 2 shows that infinite memory is necessary for $\mathcal{P}_2$: he needs to cycle in the zero loop for longer and longer. Mean-payoff games reduce polynomially to bounded window games by simply modifying the weight structure.

Theorem 3. *In two-player one-dimension games, the bounded window mean-payoff problem is in NP$\cap$coNP and at least as hard as mean-payoff games. Memoryless strategies suffice for $\mathcal{P}_1$ and infinite-memory strategies are required for $\mathcal{P}_2$ in general.*

5 Window Mean-Payoff: Multi-Dimension Games

Fixed window. Given $G = (S_1, S_2, E, k, w)$ and $l_{\max} \in \mathbb{N}_0$, the fixed window problem is solved in time $\mathcal{O}(|S|^2 \cdot (l_{\max})^{4 \cdot k} \cdot W^{2 \cdot k})$ via reduction to an exponentially larger co-Büchi game (where the objective of $\mathcal{P}_1$ is to avoid visiting a set of bad states infinitely often). Co-Büchi games are solvable in quadratic time [6]. A winning play is such that, starting in some position $i \geq 0$, in all dimensions, all opening windows are closed in at most $l_{\max}$ steps. We keep a counter of the sum over the sequence of edges and as soon as it turns non-negative, we reset the sum counter and start a new sequence. Hence, the reduction is based on accounting for each dimension the current negative sum of weights since the last reset, and the number of steps that remain to achieve a non-negative sum. This accounting is encoded in the states of $G^c = (S_1^c, S_2^c, E^c)$, as from the original state space S, we go to $S \times (\{-l_{\max} \cdot W, \ldots, 0\} \times \{1, \ldots, l_{\max}\})^k$: states of G^c are tuples representing a state of G and the current status of open windows in all dimensions (sum and remaining steps). We add states reached whenever a window reaches its maximum size $l_{\max}$ without closing. We label those as *bad* states. We have one bad state for every state of G. Transitions in G^c are built in order to accurately model the effect of transitions of G on open windows: each time a transition (s, s') in the original game G is taken, the game G^c is updated to a state $(s', (\sigma^1, \tau^1), \ldots, (\sigma^k, \tau^k))$ such that (a) if the current sum becomes positive in some dimension, the corresponding sum counter is reset to zero and the step counter is reset to its maximum value, $l_{\max}$, (b) if the sum is still strictly negative in some dimension and the window for this dimension is not at its maximal size, the sum is updated and the step counter is decreased, and (c) if the sum stays strictly negative and the maximal size is reached in any dimension, the game visits the corresponding bad state and then, all counters are reset for all dimensions and the game continues from the corresponding state $(s', (0, l_{\max}), \ldots, (0, l_{\max}))$. Clearly, a play is winning for the fixed window problem if and only if the corresponding play in G^c is winning for the co-Büchi objective that asks that the set of bad states is not visited infinitely often, as that means that from some point on, all windows close in the required number of steps.

We prove that the fixed *arbitrary* window problem is EXPTIME-hard for $\{-1, 0, 1\}$ weights and arbitrary dimensions via a reduction from the *membership problem for alternating polynomial space Turing machines (APTMs)* [3]. Given an APTM $\mathcal{M}$ and a word $\zeta \in \{0,1\}^*$, such that the tape contains at most $p(|\zeta|)$ cells, where p is a polynomial function, the membership problem asks to decide if $\mathcal{M}$ accepts ζ. We build a fixed

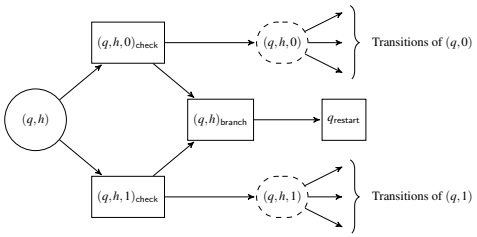

Fig. 3: Gadget ensuring a correct simulation of the APTM on tape cell h.

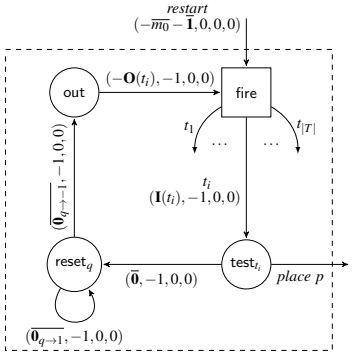

Fig. 4: Gadget simulating an execution of the reset net.

arbitrary window game G so that $\mathcal{P}_1$ has to simulate the run of $\mathcal{M}$ on ζ, and $\mathcal{P}_1$ has a winning strategy in G iff the word is accepted. For each tape cell $h \in \{1,2,\ldots,p(|\zeta|)\}$, we have two dimensions, $(h,0)$ and $(h,1)$ such that a sum of weights of value -1 (i.e., an open window) in dimension (h,i), $i \in \{0,1\}$ encodes that in the current configuration of $\mathcal{M}$, tape cell h contains a bit of value i. In each step of the simulation (Fig. 3), $\mathcal{P}_1$ has to disclose the symbol under the tape head: if in position h, $\mathcal{P}_1$ discloses a 0 (resp. a 1), he obtains a reward 1 in dimension $(h,0)$ (resp. $(h,1)$). To ensure that $\mathcal{P}_1$ was faithful, $\mathcal{P}_2$ is then given the choice to either let the simulation continue, or assign a reward 1 in all dimensions except $(h,0)$ and $(h,1)$ and then restart the game after looping in a zero self-loop for an arbitrary long time. If $\mathcal{P}_1$ cheats by not disclosing the correct symbol under tape cell h, $\mathcal{P}_2$ can punish him by branching to the restart state and ensuring a sufficiently long open window in the corresponding dimension before restarting (as in Fig. 2). But if $\mathcal{P}_1$ discloses the correct symbol and $\mathcal{P}_2$ still branches, all windows close. In the accepting state, all windows are closed and the game is restarted. The window size $l_{\max}$ of the game is function of the existing bound on the length of an accepting run. To force $\mathcal{P}_1$ to go to the accepting state, we add an additional dimension, with weight -1 on the initial edge of the game and weight 1 on reaching the accepting state.

We also prove EXPTIME-hardness for two dimensions and arbitrary weights by establishing a reduction from *countdown games* [16]. A countdown game $\mathcal{C}$ consists of a weighted graph $(\mathcal{S},\mathcal{T})$, with $\mathcal{S}$ the set of states and $\mathcal{T} \subseteq \mathcal{S} \times \mathbb{N}_0 \times \mathcal{S}$ the transition relation. Configurations are of the form (s,c), $s \in \mathcal{S}$, $c \in \mathbb{N}$. The game starts in an initial configuration (s_{init},c_0) and transitions from a configuration (s,c) are performed as follows: first $\mathcal{P}_1$ chooses a duration d, $0 < d \leq c$ such that there exists $t = (s,d,s') \in \mathcal{T}$ for some $s' \in \mathcal{S}$, second $\mathcal{P}_2$ chooses a state $s' \in \mathcal{S}$ such that $t = (s,d,s') \in \mathcal{T}$. Then, the game advances to $(s',c-d)$. Terminal configurations are reached whenever no legitimate move is available. If such a configuration is of the form $(s,0)$, $\mathcal{P}_1$ wins the play. Otherwise, $\mathcal{P}_2$ wins the play. Deciding the winner in countdown games given an initial configuration (s_{init},c_0) is EXPTIME-complete [16]. Given a countdown game $\mathcal{C}$ and an initial configuration (s_{init},c_0), we create a game $G = (S_1,S_2,E,k,w)$ with $k = 2$ and a fixed window objective for $l_{\max} = 2 \cdot c_0 + 2$. The two dimensions are used to store the value of the countdown counter and its opposite. Each time a duration d is chosen, an

edge of value of value $(-d,d)$ is taken. The game simulates the moves available in $\mathcal{C}$: a strict alternation between states of $\mathcal{P}_1$ (representing states of $\mathcal{S}$) and states of $\mathcal{P}_2$ (representing transitions available from a state of $\mathcal{S}$ once a duration has been chosen). On states of $\mathcal{P}_1$, we add the possibility to branch to a state s_{restart} of $\mathcal{P}_2$, in which $\mathcal{P}_2$ can either take a zero cycle, or go back to the initial state and force a restart of the game. By placing weights $(0,-c_0)$ on the initial edge, and $(c_0,0)$ on the edge branching to s_{restart}, we ensure that the only way to win for $\mathcal{P}_1$ is to accumulate a value exactly equal to c_0 in the game before switching to s_{restart}. This is possible if and only if $\mathcal{P}_1$ can reach a configuration of value zero in $\mathcal{C}$.

For the case of polynomial windows, we prove PSPACE-hardness via a reduction from generalized reachability games [10]. Filling the gap with the EXPTIME membership is an open problem. The generalized reachability objective is a conjunction of reachability objectives: a winning play has to visit a state of each of a series of k reachability sets. If $\mathcal{P}_1$ has a winning strategy in a generalized reachability game $G^r = (S_1^r, S_2^r, E^r)$, then he has one that guarantees visit of all sets within $k \cdot |S^r|$ steps. We create a modified weighted version of the game, $G = (S_1, S_2, E, k, w)$, such that the weights are k-dimension vectors. The game starts by opening a window in all dimensions and the only way for $\mathcal{P}_1$ to close the window in dimension t, $1 \leq t \leq k$ is to reach a state of the t-th reachability set. We modify the game by giving $\mathcal{P}_2$ the ability to close all open windows and restart the game such that the prefix-independence of the fixed window objective cannot help $\mathcal{P}_1$ to win without reaching the target sets. Then, a play is winning in G for the fixed window objective of size $l_{\max} = 2 \cdot k \cdot |S^r|$ if and only if it is winning for the generalized reachability objective in G^r. This reduction also provides exponential lower bounds on memory for both players, while exponential upper bounds follow from the reduction to co-Büchi games.

Theorem 4. *In two-player multi-dimension games, the fixed arbitrary window mean-payoff problem is EXPTIME-complete, and the fixed polynomial window mean-payoff problem is PSPACE-hard. For both players, exponential memory is sufficient and is required in general.*

Bounded window. We show non-primitive recursive hardness through a reduction from the problem of deciding the existence of an infinite execution in a *marked reset net*, also known as the *termination problem*. Hence, there is no hope for efficient algorithms on the complete class of two-player multi-weighted games. A marked reset net [8] is a Petri net with *reset arcs* together with an initial marking of its places. Reset arcs are special arcs that reset a place (i.e., empty it of all its tokens). The termination problem for reset nets is decidable but non-primitive recursive hard (as follows from [20]).

Given a reset net $\mathcal{N}$ with an initial marking $\overline{m_0} \in \mathbb{N}^{|P|}$ (where P is the set of places of the net), we build a two-player multi-weighted game G with $k = |P| + 3$ dimensions such that $\mathcal{P}_1$ wins the bounded window objective for threshold $\{0\}^k$ if and only if $\mathcal{N}$ does not have an infinite execution from $\overline{m_0}$. A high level description of our reduction is as follows. The structure of the game (Fig. 5) is based on the alternance between two gadgets simulating the net (Fig. 4).[9] Edges are labeled by k-dimension weight vectors

[9] $\overline{\mathbf{1}} = (1,\dots,1)$, $\overline{\mathbf{0}} = (0,\dots,0)$, and, for $a,b \in \mathbb{Z}$, $p \in P$, the vector $\overline{\mathbf{a}_{p\to b}}$ represents the vector $(a,\dots,a,b,a,\dots,a)$ which has value b in dimension p and a in the other dimensions.

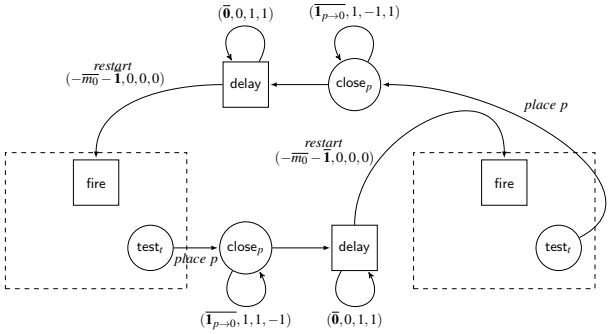

Fig. 5: Careful alternation between gadgets is needed in order for $\mathcal{P}_1$ to win.

such that the first $|P|$ dimensions are used to encode the number of tokens in each place. In each gadget, $\mathcal{P}_2$ chooses transitions to simulate an execution of the net. During a faithful simulation, there is always a running open window in all the first $|P|$ dimensions: if place p contains n tokens then the negative sum from the start of the simulation is $-(n+1)$. This is achieved as follows: if a transition t consumes $\mathbf{I}(t)(p)$ tokens from p, then this value is added on the corresponding dimension, and if t produces $\mathbf{O}(t)(p)$ tokens in p, then $\mathbf{O}(t)(p)$ is removed from the corresponding dimension. When a place p is reset, a gadget ensures that dimension p reaches value -1 (the coding of zero tokens). This is thanks to the monotonicity property of reset nets: if $\mathcal{P}_1$ does not simulate a full reset, then the situation gets easier for $\mathcal{P}_2$ as it leaves him more tokens available. If all executions terminate, $\mathcal{P}_2$ has to choose an unfireable transition at some point, consuming unavailable tokens from some place $p \in P$. If so, the window in dimension p closes. After each transition choice of $\mathcal{P}_2$, $\mathcal{P}_1$ can either continue the simulation or branch out of the gadget to close all windows, except in some dimension p of his choice. Then $\mathcal{P}_2$ can arbitrarily extend any still open window in the first $(|P|+1)$ dimensions and restart the game afterwards. Dimension $(|P|+1)$ prevents $\mathcal{P}_1$ from staying forever in a gadget. If an infinite execution exists, $\mathcal{P}_2$ simulates it and never has to choose an unfireable transition. Hence, when $\mathcal{P}_1$ branches out, the window in some dimension p stays open. The last two dimensions force him to alternate between gadgets so that he cannot take profit of the prefix-independence to win after a faithful simulation. So, $\mathcal{P}_2$ can delay the closing of the open window for longer and longer, thus winning the game.

Theorem 5. *In two-player multi-dimension games, the bounded window mean-payoff problem is non-primitive recursive hard.*

The decidability of the bounded window mean-payoff problem remains open.

6 Conclusion

The strong relation between mean-payoff and total-payoff breaks in multi-weighted games as the total-payoff threshold problem becomes undecidable. Window objectives provide conservative approximations with timing guarantees. Some variants prove to be more computationally tractable than the corresponding classical objectives.

References

1. L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J.-F. Raskin. Faster algorithms for mean-payoff games. *Formal Methods in System Design*, 38(2):97–118, 2011.
2. A. Chakrabarti, L. de Alfaro, T.A. Henzinger, and M. Stoelinga. Resource interfaces. In *Proc. of EMSOFT*, LNCS 2855, pages 117–133. Springer, 2003.
3. A.K. Chandra, D. Kozen, and L.J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
4. K. Chatterjee, L. Doyen, T.A. Henzinger, and J.-F. Raskin. Generalized mean-payoff and energy games. In *Proc. of FSTTCS*, LIPIcs 8, pages 505–516. Schloss Dagstuhl - LZI, 2010.
5. K. Chatterjee, L. Doyen, M. Randour, and J.F. Raskin. Looking at mean-payoff and total-payoff through windows. *CoRR*, abs/1302.4248, 2013. `http://arxiv.org/abs/1302.4248`.
6. K. Chatterjee and M. Henzinger. An $\mathcal{O}(n^2)$ time algorithm for alternating büchi games. In *Proc. of SODA*, pages 1386–1399. SIAM, 2012.
7. K. Chatterjee, M. Randour, and J.-F. Raskin. Strategy synthesis for multi-dimensional quantitative objectives. In *Proc. of CONCUR*, LNCS 7454, pages 115–131. Springer, 2012.
8. C. Dufourd, A. Finkel, and P. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. of ICALP*, LNCS 1443, pages 103–115. Springer, 1998.
9. A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *Int. Journal of Game Theory*, 8(2):109–113, 1979.
10. N. Fijalkow and F. Horn. The surprizing complexity of generalized reachability games. *CoRR*, abs/1010.2420, 2010.
11. T. Gawlitza and H. Seidl. Games through nested fixpoints. In *Proc. of CAV*, LNCS 5643, pages 291–305. Springer, 2009.
12. H. Gimbert and W. Zielonka. When can you play positionally? In *Proc. of MFCS*, LNCS 3153, pages 686–697. Springer, 2004.
13. V.A. Gurvich, A.V. Karzanov, and L.G. Khachivan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Computational Mathematics and Mathematical Physics*, 28(5):85–91, 1988.
14. H. Bjorklund and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155:210–229, 2007.
15. M. Jurdziński. Deciding the winner in parity games is in UP $\cap$ co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998.
16. M. Jurdziński, J. Sproston, and F. Laroussinie. Model checking probabilistic timed automata with one or two clocks. *Logical Methods in Computer Science*, 4(3), 2008.
17. A.V. Karzanov and V.N. Lebedev. Cyclical games with prohibitions. *Math. Program.*, 60:277–293, 1993.
18. Y.M. Lifshits and D.S. Pavlov. Potential theory for mean payoff games. *Journal of Mathematical Sciences*, 145(3):4967–4974, 2007.
19. N.N. Pisaruk. Mean cost cyclical games. *Mathematics of Operations Research*, 24(4):817–828, 1999.
20. P. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Inf. Process. Lett.*, 83(5):251–261, 2002.
21. Y. Velner and A. Rabinovich. Church synthesis problem for noisy input. In *Proc. of FOSSACS*, LNCS 6604, pages 275–289. Springer, 2011.
22. U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.