Faster Algorithms for Mean-Payoff Parity Games

Krishnendu Chatterjee[1], **Monika Henzinger**[2], **and Alexander Svozil**[3]

1 **IST Austria**
`krish.chat@ist.ac.at`
2 **University of Vienna, Faculty of Computer Science, Vienna, Austria**
`monika.henzinger@univie.ac.at`
3 **University of Vienna, Faculty of Computer Science, Vienna, Austria**
`alexander.svozil@univie.ac.at`

Abstract

Graph games provide the foundation for modeling and synthesis of reactive processes. Such games are played over graphs where the vertices are controlled by two adversarial players. We consider graph games where the objective of the first player is the conjunction of a qualitative objective (specified as a parity condition) and a quantitative objective (specified as a mean-payoff condition). There are two variants of the problem, namely, the *threshold* problem where the quantitative goal is to ensure that the mean-payoff value is above a threshold, and the *value* problem where the quantitative goal is to ensure the optimal mean-payoff value; in both cases ensuring the qualitative parity objective. The previous best-known algorithms for game graphs with n vertices, m edges, parity objectives with d priorities, and maximal absolute reward value W for mean-payoff objectives, are as follows: $\mathcal{O}(n^{d+1} \cdot m \cdot W)$ for the threshold problem, and $\mathcal{O}(n^{d+2} \cdot m \cdot W)$ for the value problem. Our main contributions are faster algorithms, and the running times of our algorithms are as follows: $\mathcal{O}(n^{d-1} \cdot m \cdot W)$ for the threshold problem, and $\mathcal{O}(n^{d} \cdot m \cdot W \cdot \log(n \cdot W))$ for the value problem. For mean-payoff parity objectives with two priorities, our algorithms match the best-known bounds of the algorithms for mean-payoff games (without conjunction with parity objectives). Our results are relevant in synthesis of reactive systems with both functional requirement (given as a qualitative objective) and performance requirement (given as a quantitative objective).

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.39

1 Introduction

Graph games. A graph game is played on a finite directed graph with two players, namely, player 1 and player 2 (the adversary of player 1). The vertex set is partitioned into player-1 and player-2 vertices. At player-1 vertices, player 1 chooses a successor vertex; and at player-2 vertices, player 2 does likewise. The result of playing the game forever is an infinite path through the graph. There has been a long history of using graph games for modeling and synthesizing reactive processes [6, 17, 18]: a reactive system and its environment represent the two players, whose states and transitions are specified by the vertices and edges of a game graph. Consequently, graph games provide the theoretical foundation for modeling and synthesizing reactive processes.

Qualitative and quantitative objectives. For reactive systems, the objective is given as a set of desired paths (such as ω-regular specifications), or as a quantitative optimization objective with a payoff function on the paths. The class of ω-regular specifications provide a robust framework to express all commonly used specifications for reactive systems in verification and synthesis. Parity objectives are a canonical way to express ω-regular objectives [19], where an integer priority is assigned to every vertex, and a path satisfies the parity objective for player 1 if the minimum priority visited infinitely often is even. One of the classical and most well-studied quantitative objectives is the mean-payoff objective, where a reward is associated with every edge, and the payoff of a path is the long-run average of the rewards of the path.

Mean-payoff parity objectives. Traditionally the verification and the synthesis problems were considered with qualitative objectives. However, recently combinations of qualitative and quantitative objectives have received a lot of attention. Qualitative objectives such as ω-regular objectives specify the functional requirements of reactive systems, whereas the quantitative objectives specify resource consumption requirements (such as for embedded systems or power-limited systems). Combining quantitative and qualitative objectives is crucial in the design of reactive systems with both resource constraints and functional requirements [8, 13, 3, 2]. For example, mean-payoff parity objectives are relevant in synthesis of optimal performance lock-synchronization for programs [7], where one player is the synchronizer, the opponent is the environment; the performance criteria is specified as mean-payoff objective; and the functional requirement (e.g., data-race freedom or liveness) as an ω-regular objective. Mean-payoff parity objectives have been used in several other applications, e.g., define permissivity for parity games [4] and robustness in synthesis [1].

Threshold and value problems. For graph games with mean-payoff and parity objectives there are two variants of the problem. First, the *threshold* problem, where a threshold ν is given for the mean-payoff objective, and player 1 must ensure the parity objective and that the mean-payoff is at least ν. Second, the *value* problem, where player 1 maximizes the mean-payoff value while ensuring the parity objective. In the sequel of this section, we will refer to graph games with mean-payoff and parity objectives as mean-payoff parity games.

Previous results. Mean-payoff parity games were first studied in [13], and algorithms for the value problem were presented. It was shown in [9] that the decision problem for mean-payoff parity games lies in NP $\cap$ coNP (similar to the status of mean-payoff games and parity games). For game graphs with n vertices, m edges, parity objectives with d priorities, and maximal absolute reward value W for the mean-payoff objective, the previous known algorithmic bounds for mean-payoff parity games are as follows: For the threshold problem the results of [9] give an $\mathcal{O}(n^{d+4} \cdot m \cdot d \cdot W)$-time algorithm. This algorithmic bound was improved in [4] where an $\mathcal{O}(n^{d+2} \cdot m \cdot W)$-time algorithm was presented for the value problem. The result of [4] does not explicitly present any other better bound for the threshold problem. However, the recursive algorithm of [4] uses value mean-payoff games as a sub-routine, and replacing value mean-payoff games with threshold mean-payoff games gives an $\mathcal{O}(n)$-factor saving, and yields an $\mathcal{O}(n^{d+1} \cdot m \cdot W)$-time algorithm for the threshold problem for mean-payoff parity games.

Contributions. In this work our main contributions are faster algorithms to solve mean-payoff parity games. Previous and our results are summarized in Table 1.

1. *Threshold problem.* We present an $\mathcal{O}(n^{d-1} \cdot m \cdot W)$-time algorithm for the threshold problem for mean-payoff parity games, improving the previous $\mathcal{O}(n^{d+1} \cdot m \cdot W)$ bound. The important special case of parity objectives with two priorities correspond to Büchi and coBüchi objectives. Our bound for mean-payoff Büchi games and mean-payoff coBüchi games is $\mathcal{O}(n \cdot m \cdot W)$, which matches the best-known bound to solve the threshold problem for mean-payoff objectives [5], and improves the previous known $\mathcal{O}(n^3 \cdot m \cdot W)$ bound [4].
2. *Value problem.* We present an $\mathcal{O}(n^d \cdot m \cdot W \cdot \log(n \cdot W))$-time algorithm for the value problem for mean-payoff parity games, improving the previous $\mathcal{O}(n^{d+2} \cdot m \cdot W)$ bound. Our bound for mean-payoff Büchi games and mean-payoff coBüchi games is $\mathcal{O}(n^2 \cdot m \cdot W \cdot \log(n \cdot W))$, which matches the bound of [5] to solve the value problem for mean-payoff objectives, and improves the previous known $\mathcal{O}(n^4 \cdot m \cdot W)$ bound.

	threshold problem		value problem	
	Previous	Our	Previous	Our
MP-Büchi	$\mathcal{O}(n^3 \cdot m \cdot W)$	$\mathcal{O}(n \cdot m \cdot W)$	$\mathcal{O}(n^4 \cdot m \cdot W)$	$\mathcal{O}(n^2 \cdot m \cdot W \cdot \log(nW))$
MP-coBüchi	$\mathcal{O}(n^3 \cdot m \cdot W)$	$\mathcal{O}(n \cdot m \cdot W)$	$\mathcal{O}(n^4 \cdot m \cdot W)$	$\mathcal{O}(n^2 \cdot m \cdot W \cdot \log(nW))$
MP-parity	$\mathcal{O}(n^{d+1} \cdot m \cdot W)$	$\mathcal{O}(n^{d-1} \cdot m \cdot W)$	$\mathcal{O}(n^{d+2} \cdot m \cdot W)$	$\mathcal{O}(n^d \cdot m \cdot W \cdot \log(nW))$

Table 1 Algorithmic bounds for mean-payoff (MP) and parity objectives, and special cases: threshold problem (left) and value problem (right).

Technical contributions. Our main technical contributions are as follows:

1. First, for the threshold problem, we present a decremental algorithm for mean-payoff games that supports a sequence of vertex-set deletions along with their player-2 reachability set. We show that the total running time is $\mathcal{O}(n \cdot m \cdot W)$, which matches the best-known bound for the static algorithm to solve mean-payoff games. We show that using our decremental algorithm we can solve the threshold problem for mean-payoff Büchi games in time $\mathcal{O}(n \cdot m \cdot W)$.
2. Second, for mean-payoff coBüchi games, the decremental approach does not work. We present a new static algorithm for threshold mean-payoff games that identifies subsets X of the winning set for player 1, where the time complexity is $\mathcal{O}(|X| \cdot m \cdot W)$, i.e., it replaces n with the size of the set identified. We show that with our new static algorithm we can solve the threshold problem for mean-payoff coBüchi games in time $\mathcal{O}(n \cdot m \cdot W)$.
3. Finally, we show for all mean-payoff parity objectives, given an algorithm for the threshold problem, the value problem can be solved in time $n \cdot \log(n \cdot W)$ times the complexity of the threshold problem.

Related works. The problem of graph games with mean-payoff parity objectives was first studied in [13]. The NP $\cap$ coNP complexity bound was established in [9], and an improved algorithm for the problem was given in [4]. The mean-payoff parity objectives has also been considered in other stochastic setting such as Markov decision processes [10, 11] and stochastic games [12]. The algorithmic approaches for stochastic games build on the results for non-stochastic games. In this work, we present faster algorithms for mean-payoff parity games.

2 Preliminaries

Graphs. A graph $G = (V, E)$ consists of a finite set V of vertices and a finite set of edges $E \subseteq V \times V$. Given a graph $G = (V, E)$ and a subset $U \subseteq V$ we denote by $G \restriction U = (V', E')$ the subgraph of G induced by U, i.e., $V' = U$, $E' = (U \times U) \cap E$. For $v \in V$ we denote by $In(v)$ (resp., $Out(v)$) the set of incoming (resp., outgoing) vertices, i.e., $In(v) = \{v' \mid (v', v) \in E\}$, and $Out(v) = \{v' \mid (v, v') \in E\}$.

Game graphs. A game graph $\Gamma = (V, E, \langle V_1, V_2 \rangle)$ is a graph whose vertex set is partitioned into V_1 and V_2, (i.e., $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$). In a game graph every vertex $v \in V$ has a successor $v' \in V$, i.e., $Out(v) \neq \emptyset$ for all $v \in V$. Given a game graph Γ and a set U such that for all vertices u in U we have $Out(u) \cap U \neq \emptyset$, we denote by $\Gamma \restriction U$ the subgame induced by U.

Plays. Given a game graph Γ and a starting vertex v_0, the game proceeds in rounds. In each round, if the current vertex belongs to player 1, then player 1 chooses a successor vertex, and player 2 does likewise if the current vertex belongs to player 2. The result is a *play* ρ which is an infinite path from v_0, i.e., $\rho = v_0 v_1 \ldots$, where every $(v_i, v_{i+1}) \in E$ for all $i \geq 0$. We denote by $Plays(\Gamma)$ the set of all plays of the game graph.

Strategies. Strategies are recipes to extend prefixes of plays by choosing the next vertex. Formally, a strategy for player-1 is a function $\sigma_1 : V^* \cdot V_1 \mapsto V$ such that $(v, \sigma_1(\rho \cdot v)) \in E$ for all $v \in V_1$ and all $\rho \in V^*$. We define strategies σ_2 for player 2 analogously. We denote by Σ_1 and Σ_2 the set of all strategies for player 1 and player 2, respectively. Given strategies σ_1 and σ_2 for player 1 and player 2, and a starting vertex v_0, there is a unique play $\rho = v_0 v_1 \dots$ such that for all $i \geq 0$, (a) if $v_i \in V_1$ then $v_{i+1} = \sigma_1(v_0 \dots v_i)$; and (b) if $v_i \in V_2$ then $v_{i+1} = \sigma_2(v_0 \dots v_i)$. We denote the unique play as $outcome(v_0, \sigma_1, \sigma_2)$. A strategy is *memoryless* if it is independent of the past and depends only on the current vertex, and hence can be defined as a function $\sigma_1 : V_1 \mapsto V$ and $\sigma_2 : V_2 \mapsto V$, respectively.

Objectives and parity objectives. An objective for a game graph Γ is a subset of the possible plays, i.e., $\phi \subseteq Plays(\Gamma)$. Given a play ρ we denote by $Inf(\rho)$ the set of vertices that appear infinitely often in ρ. A *parity* objective is defined with a priority function p that maps every vertex to a non-negative integer priority, and a play satisfies the parity objective for player 1 if the minimum priority vertex that appear infinitely often is even. Formally, the parity objective is $Parity_\Gamma(p) = \{\rho \in Plays(\Gamma) \mid \min\{p(v) \mid v \in Inf(\rho)\} \text{ is even}\}$. The Büchi and coBüchi objectives are special cases of parity objectives with two priorities only. We have $p : V \mapsto \{0, 1\}$ for Büchi objectives and $p : V \mapsto \{1, 2\}$ for the coBüchi objectives.

Payoff functions. Consider a game graph Γ, and a weight function $w : E \mapsto \mathbb{Z}$ that maps every edge to an integer. The mean-payoff function maps every play to a real-number and is defined as follows: For a play $\rho = v_0 v_1 \dots$ in $Plays(\Gamma)$ we have $MP(w, \rho) = \liminf_{n \mapsto \infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} w(v_i, v_{i+1})$. The mean-payoff parity function also maps every play to a real-number or $-\infty$ as follows: if the parity objective is satisfied, then the value is the mean-payoff value, else it is $-\infty$. Formally, for a play ρ, we have

$$MPP_\Gamma(w, p, \rho) = \begin{cases} MP_\Gamma(w, \rho) & \text{if } \rho \in Parity_\Gamma(p); \\ -\infty & \text{if } \rho \notin Parity_\Gamma(p). \end{cases}$$

Threshold mean-payoff parity objectives. Given a threshold $\nu \in \mathbb{Q}$, the *threshold mean-payoff objective* $MeanPayoff_\Gamma(\nu) = \{\rho \in Plays(\Gamma) \mid MP(\rho) \geq \nu\}$ requires that the mean-payoff value is at least ν. The *threshold mean-payoff parity objective* is a conjunction of a parity objective and a threshold mean-payoff objective, i.e., $Parity_\Gamma(p) \cap MeanPayoff_\Gamma(\nu)$.

Winning strategies. Given an objective (such as parity, threshold mean-payoff, or threshold mean-payoff parity) ϕ, a vertex v is winning for player 1, if there is a strategy σ_1 such that for all strategies σ_2 of player 2, the play $outcome(v, \sigma_1, \sigma_2) \in \phi$ (i.e., the play satisfies the objective). We denote by $W_1(\phi)$ the set of winning vertices (or the winning region) for player 1 for the objective ϕ. The notation $W_2(\overline{\phi})$ for complementary objectives $\overline{\phi}$ for player 2 is similar.

Value functions. Given a payoff function f (such as the mean-payoff function, or the mean-payoff parity function), the value for player 1 is the maximal payoff that she can guarantee against all strategies of player 2. Formally,

$$val_\Gamma(f)(v) = \sup_{\sigma_1 \in \Sigma_1} \inf_{\sigma_2 \in \Sigma_2} f(outcome(v, \sigma_1, \sigma_2)).$$

Attractors. The player-1 attractor $Attr_1(S)$ of a given set $S \subseteq V$ is the set of vertices from which player-1 can force to reach a vertex in S. It is defined as the limit of the sequence $A_0 = S; A_{i+1} = A_i \cup \{v \in V_1 \mid Out(v) \cap A_i \neq \emptyset\} \cup \{v \in V_2 \mid Out(v) \subseteq A_i\}$ for all $i \geq 0$. Th Player-2 attractor $Attr_2(S)$ is defined analogously exchanging the roles of player 1 and player 2. The complement of an attractor induces a game graph, as in the complement every vertex has an outgoing edge in the complement set.

Relevant parameters. In this work we will consider computing the winning region for threshold mean-payoff parity objectives, and the value function for mean-payoff parity objectives. We will consider the following relevant parameters: n denotes the number of vertices, m denotes the number of edges, d denotes the number of priorities of the parity function p, and W is the maximum absolute value of the weight function w.

3 Decremental Algorithm for Threshold Mean-Payoff Games

In this section we present a decremental algorithm for threshold mean-payoff games that supports deleting a sequence of sets of vertices along with their player-2 attractors. The overall running time of the algorithm is $\mathcal{O}(n \cdot m \cdot W)$.

Key idea. A static algorithm based on the notion of progress measure for mean-payoff games was presented in [5]. We show that the progress measure is monotonic wrt to the deletion of vertices and their player-2 attractors. We use an amortized analysis to obtain the running time of our algorithm.

Mean-payoff progress measure. Let Γ be a mean-payoff game with threshold ν. Progress measure is a function f which maps every vertex in Γ to an element of the set $C_\Gamma = \{i \in \mathbb{N} \mid i \leq nW\} \cup \{\top\}$, i.e., $f : V \mapsto C_\Gamma$. Let $(\preceq, C_\Gamma)$ be a total order, where $x \preceq y$ for $x, y \in C_\Gamma$ holds iff $x \leq y \leq nW$ or $y = \top$. We define the operation $\ominus : C_\Gamma \times \mathbb{Z} \mapsto C_\Gamma$ for all $a \in C_\Gamma$ and $b \in \mathbb{Z}$ as follows:

$$a \ominus b = \begin{cases} \max(0, a-b) & \text{if } a \neq \top \text{ and } a - b \leq nW, \\ \top & \text{otherwise.} \end{cases}$$

A player-1 vertex v is *consistent* if $f(v) \succeq f(v') \ominus w(v,v')$ for *any* $v' \in \mathit{Out}(v)$. A player-2 vertex v is *consistent* if $f(v) \succeq f(v') \ominus w(v,v')$ for *all* $v' \in \mathit{Out}(v)$. Let $v \in V$ then $\mathit{lift}(\cdot, v) : [V \mapsto C_\Gamma] \mapsto [V \mapsto C_\Gamma]$ is defined by $\mathit{lift}(f, v) = g$ where:

$$g(u) = \begin{cases} f(u) & \text{if } u \neq v, \\ \min\{f(v') \ominus w(v,v') \mid (v,v') \in E\} & \text{if } u = v \in V_1, \\ \max\{f(v') \ominus w(v,v') \mid (v,v') \in E\} & \text{if } u = v \in V_2. \end{cases}$$

Static Algorithm The static algorithm in [5] is an iterative algorithm which maintains and returns a progress measure f and a list L of vertices which are not consistent. The initial progress measure of every vertex is set to zero. Also, $w(e)$ is set to $w(e) - \nu$ for all edges e in E. The list L is initialized with the vertices which are not consistent considering the initial progress measure. Then the following steps are executed in a while-loop:

1. Take out a vertex v of L.
2. Perform the *lift*-operation on the vertex, i.e., $f \leftarrow \mathit{lift}(f, v)$.
3. If a vertex v' in $\mathit{In}(v)$ is not consistent, put v' into L.
4. If L is empty, return f else proceed to the next iteration.

If every vertex is consistent, i.e., the list L is empty, the winning region of player 1 is the set of vertices which are not set to $\top$ in f, i.e., $W_1(\nu) = \{v \in V \mid f(v) \neq \top\}$.

Decremental input/output. Let Γ be a mean-payoff game with threshold ν. The *input* to the decremental algorithm is a sequence of sets $A_1, A_2, \ldots, A_k$, such that each A_i is a player-2 attractor of a set X_i in the game $\Gamma_i = \Gamma \upharpoonright (V \setminus \bigcup_{j<i} A_j)$. The *output requirement* is the player-1 winning set after the deletion of $\bigcup_{j<i} A_j$ for $i = 1, \ldots, k$, i.e., the output requirement is the sequence $Z_1, Z_2, \ldots, Z_k$, where $Z_i = W_1(\phi)$ in $\Gamma_i = \Gamma \upharpoonright (V \setminus \bigcup_{j<i} A_j)$, where $\phi = \mathit{MeanPayoff}_{\Gamma_i}(\nu)$ is the threshold mean-payoff objective. In other words, we

repeatedly delete a vertex set X_i along with its player-2 attractor A_i from the current game graph Γ_i, and require the winning set for player 1 as an output after each deletion.

Decremental algorithm. We maintain a progress measure f_i, $1 \leq i \leq k$, during the whole sequence of deletions. The initial progress measure f_1 for the mean-payoff game Γ with threshold mean-payoff objective ϕ is calculated using the static algorithm. For all edges e in E, we set $w(e) = w(e) - \nu$. In iteration i with input A_i, in the game Γ_i with its corresponding vertex set V_i the following steps are executed:

1. If a vertex in the set $\{v \in V_i \setminus A_i \mid \exists v' : v' \in Out(v) \wedge v' \in A_i\}$ is not consistent in f_i without the set A_i, put it in a list L_i.
2. Delete the set A_i from Γ_i to receive Γ_{i+1} (and thus V_{i+1}).
3. Execute steps (1)-(4) of the above described iterative algorithm from [5] initialized with Γ_{i+1}, L_i and f_i restricted to the vertices in V_{i+1}.
4. Finally the winning region of player 1 can be extracted from the obtained progress measure f_{i+1}, i.e., $W_1(\phi) = \{v \in V_{i+1} \mid f(v) \neq \top\}$.

Correctness. Let Γ be a game graph, ϕ a threshold objective and $A_1, A_2, \ldots, A_k$ a sequence of sets, such that each A_i is a player-2 attractor in the game $\Gamma_i = \Gamma \upharpoonright (V \setminus \bigcup_{j<i} A_j)$. To show the correctness of the decremental algorithm we need to show that the condition that the list L contains all vertices which are not consistent is an invariant of the decremental algorithm at line 3. This property was proved for the static algorithm in [5].

▶ **Lemma 1.** *The condition that L_i contains all vertices which are not consistent with the progress measure f_i restricted to V_{i+1} in Γ_{i+1} is an invariant of the static algorithm called in step 3 of the decremental algorithm for $1 \leq i \leq k-1$.*

Proof. The fact that the static algorithm correctly returns a progress measure with only consistent vertices when the invariant holds was shown in [5]. It was also shown in [5] that the invariant is maintained in the loop. It remains to show that the condition holds when we call the static algorithm at step 3. For the base case, let $i = 1$. In the initial progress measure f_1 and the initial game graph Γ_1, every vertex is consistent. By the definition of a player-2 attractor, deleting the set A_1 potentially removes edges (v, v') where v is a player-1 vertex in $V \setminus A_1$ and v' is in A_1. (Note that v cannot be a player-2 vertex.) All of the vertices not consistent anymore are added to L_i in step 1 of the decremental algorithm. For the inductive step let $i = j$. By induction hypothesis, all vertices which were not consistent with the progress measures f_{h-1} restricted to V_h for $2 \leq h \leq j$ were added to the corresponding lists. Thus by the correctness of the static algorithm, it correctly computes the new progress measure f_h for the game graph Γ_h where every vertex is consistent. Thus also every vertex in the progress measure f_j restricted to V_j is consistent. Again the player-2 attractor is removed and vertices which are not consistent with progress measure f_j restricted to V_{j+1} are put into L_j by step 1 of the algorithm. ◀

Thus we proved that the static algorithm always correctly updates to the new progress measure in each iteration. The winning region of player-1 is obtained by the returned progress measure (step 4). The decremental algorithm thus correctly computes the sequence $Z_1, Z_2, \ldots Z_k$, where $Z_i = W_1(\phi)$ in Γ_i.

Running Time. The calculation of the initial progress measure for the mean-payoff game Γ with threshold ν is in time $\mathcal{O}(n \cdot m \cdot W)$. The vertices which are not consistent anymore after the deletion of A_i can be found in time $\mathcal{O}(m)$ (step 1). As at most n such sets A_i exist, the running time is $\mathcal{O}(mn)$. In step 3 the static algorithm is executed with our current progress measure f_i: Every time a vertex v is picked from the list L_i it costs $\mathcal{O}(|Out(v) + In(v)|)$ time to use *lift* on it and to look for vertices in $In(v)$ which are not consistent anymore (steps 1-3

in the static algorithm). This cost is charged to its incident edges. Note that deleting a set of vertices and their corresponding player-2 attractor will only potentially *increase* the progress measure of some player-1 vertices. As we can increase the progress measure of every vertex only nW times before it is set to $\top$ where it is always consistent, we get the desired bound of $\mathcal{O}(m \cdot n \cdot W)$.
Thus our decremental algorithm for threshold mean-payoff games works as desired and we obtain the following result:

▶ **Theorem 2.** *Given a game graph Γ, a threshold mean-payoff objective ϕ and a sequence of sets $A_1, A_2, \dots, A_k$ such that each A_i is a player-2 attractor of a set X_i in the game $\Gamma_i = \Gamma \upharpoonright (V \setminus \bigcup_{j<i} A_j)$, the sequence $Z_1, Z_2, \dots, Z_k$, where $Z_i = W_1(\phi)$ in Γ_i can be computed in $\mathcal{O}(n \cdot m \cdot W)$ time.*

▶ Remark. Note that the running time analysis of our decremental algorithm crucially depends on the monotonicity property of the progress measure. If edges are both added and deleted, then the monotonicity property does not hold. Hence obtaining a fully dynamic algorithm that supports both addition/deletion of vertices/edges with running time $\mathcal{O}(n \cdot m \cdot W)$ is an interesting open problem. However, we will show that for solving mean-payoff parity games, the decremental algorithm plays a crucial part.

4 Threshold Mean-Payoff Parity Games

In this section we present algorithms for threshold mean-payoff parity games. Our most interesting contributions are for the base case of mean-payoff Büchi- and mean-payoff coBüchi objectives, and the general case follows a standard recursive argument.

4.1 Threshold Mean-Payoff Büchi Games

In this section we consider threshold mean-payoff Büchi games.

Algorithm for threshold mean-payoff Büchi games. The basic algorithm is an iterative algorithm that deletes player-2 attractors. The algorithm proceeds in iterations. In iteration i, let D_i be the set of vertices already deleted. Consider the subgame $\Gamma_i = \Gamma \upharpoonright (V \setminus D_i)$. Then the following steps are executed:

1. Let $V^i = V \setminus D_i$ and B_i denote the set of Büchi vertices (or vertices with priority 0) in Γ_i. Compute $Y_i = Attr_1(B_i)$ the player-1 attractor to B_i in Γ_i.
2. Let $X_i = V^i \setminus Y_i$. If X_i is non-empty, remove $A_i = Attr_2(X_i)$ from the game graph, and proceed to the next iteration.
3. Else $V^i = Y_i$. Let $U_i = W_1(\phi)$ in Γ_i, where $\phi = \mathit{MeanPayoff}(\nu)$, be the winning region for the threshold mean-payoff objective in Γ_i. Let $X_i = V^i \setminus U_i$. If X_i is non-empty, remove $A_i = Attr_2(X_i)$ from the game graph, and proceed to the next iteration. If X_i is empty, then the algorithm stops and all the remaining vertices are winning for player 1 for the threshold mean-payoff Büchi objective.

Correctness. Since the correctness argument has been used before [13], we only present a brief sketch: The basic correctness argument is to show that all vertices removed over all iterations do not belong to the winning set for player 1. In the end, for the remaining vertices, player 1 can ensure to reach the Büchi vertices, and ensures the threshold mean-payoff objectives. A strategy that plays for the threshold mean-payoff objectives longer and longer, and in between visits the Büchi vertices, ensures that the threshold mean-payoff Büchi objective is satisfied.

Running time analysis. We observe that the total running time to compute all attractors is at most $\mathcal{O}(n \cdot m)$, since the algorithm runs for $\mathcal{O}(n)$ iterations and each attractor computation is linear time. In step 3, the algorithm needs to compute the winning region for threshold mean-payoff objective. The algorithm always removes a set X_i and its player-2 attractor A_i, and requires the winning set for player 1. Thus we can use the decremental algorithm from Section 3, which precisely supports these operations. Hence using Theorem 2 in the algorithm for threshold mean-payoff Büchi games, we obtain the following result.

▶ **Theorem 3.** *Given a game graph Γ and a threshold mean-payoff Büchi objective ϕ, the winning set $W_1(\phi)$ can be computed in $\mathcal{O}(m \cdot n \cdot W)$ time.*

4.2 Threshold Mean-Payoff coBüchi Games

In this section we will present an $\mathcal{O}(n \cdot m \cdot W)$-time algorithm for threshold mean-payoff coBüchi games. We start with the description of the basic algorithm for threshold mean-payoff coBüchi games.

Algorithm for threshold mean-payoff coBüchi games. The basic algorithm is an iterative algorithm that deletes player-1 attractors. The algorithm proceeds in iteration. In iteration i, let D_i be the set of vertices already deleted. Consider the subgame $\Gamma_i = \Gamma \restriction (V \setminus D_i)$. Then the following steps are executed:

1. Let $V^i = V \setminus D_i$ and C_i denote the set of coBüchi vertices (or vertices with priority 1) in Γ_i. Compute $Y_i = \mathit{Attr}_2(C_i)$ the player-2 attractor to C_i in Γ_i.
2. Let $X_i = V^i \setminus Y_i$. Consider the subgame $\widehat{\Gamma}_i = \Gamma_i \restriction X_i$. Compute the winning region $Z_i = W_1(\phi)$ for player 1 in $\widehat{\Gamma}_i$, where $\phi = \mathit{MeanPayoff}(\nu)$ is the threshold mean-payoff objective.
3. If Z_i is non-empty, remove $\mathit{Attr}_1(Z_i)$ from Γ_i, and proceed to the next iteration. Else if Z_i is empty, then all remaining vertices are winning for player 2.

Correctness argument. Consider the subgame Γ_i. In each subgame $\widehat{\Gamma}_i$ of Γ_i all edges of player 2 are intact, since it is obtained after removing a player-2 attractor Y_i. Moreover, there is no priority-1 vertex in $\widehat{\Gamma}_i$. Hence ensuring the threshold mean-payoff objective in $\widehat{\Gamma}_i$ for player 1 ensures satisfying the threshold mean-payoff coBüchi objective. Hence the set Z_i and its player-1 attractor belongs to the winning set of player 1 and can be removed. Thus all vertices removed are part of the winning region for player 1. Upon termination, in $\widehat{\Gamma}_i$, player 1 cannot satisfy the threshold mean-payoff condition from any vertex. Consider a player-2 strategy, where in $\widehat{\Gamma}_i$ player 2 falsifies the threshold mean-payoff condition, and in Y_i plays an attractor strategy to reach C_i (priority-1 vertices). Given such a strategy, either (a) Y_i is visited infinitely often, and then the coBüchi objective is violated; or (b) from some point on the play stays in $\widehat{\Gamma}_i$ forever, and then the threshold mean-payoff objective is violated. This shows the correctness of the algorithm. However, the running time of this algorithm is not $\mathcal{O}(n \cdot m \cdot W)$. We now present the key ideas to obtain an $\mathcal{O}(n \cdot m \cdot W)$-time algorithm.

First intuition. Our first intuition is as follows. In step 2 of the above algorithm, instead of obtaining the whole winning region $W_1(\phi)$ in $\widehat{\Gamma}_i$ it suffices to identify a subset X_i of the winning region (if it is non-empty) and remove its player-1 attractor. We call this the modified algorithm for threshold mean-payoff coBüchi games. We first describe why we cannot use the decremental approach in the following remark.

▶ Remark. Consider the subgames for which the threshold mean-payoff objective must be solved. Consider Figure 1. The first player-2 attractor removal induces subgame $\widehat{\Gamma}_1$. After identifying a winning region X_1 of $\widehat{\Gamma}_1$ we remove its player-1 attractor A_1. After removal of A_1, we consider the second player-2 attractor to the priority-1 vertices. The removal of this attractor induces $\widehat{\Gamma}_2$. We observe comparing $\widehat{\Gamma}_1$ and $\widehat{\Gamma}_2$ that certain vertices are removed, whereas other vertices are added. Thus the subgames to be solved for threshold mean-payoff objectives do not satisfy the condition of decremental or incremental algorithms (see Remark 3).

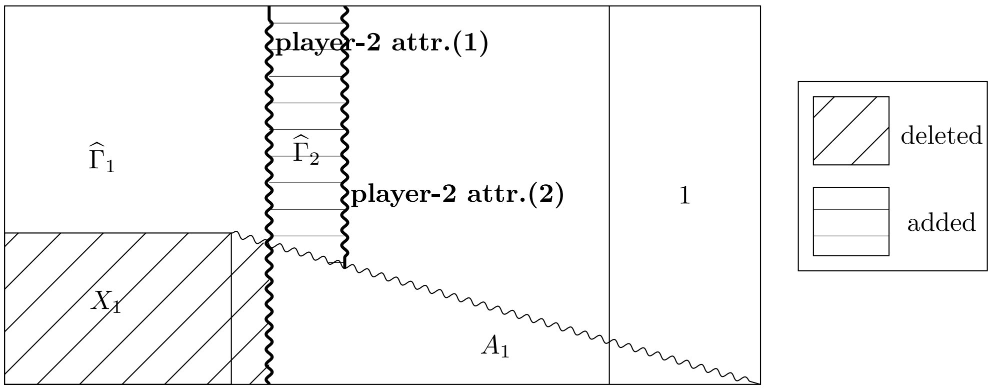

Figure 1 Pictorial illustration of threshold mean-payoff coBüchi games. The subgames $\widehat{\Gamma}_1$ and $\widehat{\Gamma}_2$ are shown. We observe that $\widehat{\Gamma}_2$ is obtained both by addition and deletion of game parts to $\widehat{\Gamma}_1$.

Second intuition. While we cannot use the decremental algorithm, we can solve the problem in $\mathcal{O}(n \cdot m \cdot W)$ time, if we have a modified static algorithm for threshold mean-payoff games, with the following property: (a) it identifies a subset of the winning region X for player 1, if the winning region is non-empty, in time $\mathcal{O}(|X| \cdot m \cdot W)$; (b) if the winning region is empty, it returns the empty set, and then it takes time $\mathcal{O}(n \cdot m \cdot W)$. With such an algorithm we analyze the running time of the above modified algorithm for threshold mean-payoff coBüchi games. The total time required for all attractor computations is again $\mathcal{O}(n \cdot m)$. Otherwise, we use the modified static algorithm to remove vertices of player-1 and to remove set of size X we take $\mathcal{O}(|X| \cdot m \cdot W)$ time, and thus we can charge each vertex $\mathcal{O}(m \cdot W)$ time. Hence the total time required is $\mathcal{O}(n \cdot m \cdot W)$. In the rest of the section we present this modified static algorithm for threshold mean-payoff games.

Problem Statement.

Input:	Mean-payoff game Γ with threshold ν.
Question:	If $W_1(\mathit{MeanPayoff}(\nu))$ is non-empty, return a nonempty set $X \subseteq W_1(\mathit{MeanPayoff}(\nu))$ in time $\mathcal{O}(\|X\| \cdot m \cdot W)$, else return $\emptyset$ in time $\mathcal{O}(n \cdot m \cdot W)$.

Modified static algorithm for threshold mean-payoff games. The basic algorithm for threshold mean-payoff games computes a progress measure, with a defined top element value $\top$. If the progress measure has the value $\top$ for a vertex, then the vertex is declared as winning for player 2. With value $\top = n \cdot W$, the correct winning region for both players can be identified. Moreover, for a given value α for $\top$, the progress measure algorithm requires $\mathcal{O}(\alpha \cdot m)$ time. Our modified static algorithm is based on the following idea:

1. Consider a value $\alpha \leq n \cdot W$ for the top element. With this reduced value for the top element, if a winning region is identified for player 1, then it is a subset of the whole winning region for player 1.
2. We will iteratively double the value for the top element.

Given the above ideas our algorithm is an iterative algorithm defined as follows: Initialize top value $\top_0 = W$. The i-th iteration is as follows:

1. Run the progress measure algorithm with top value $\top_i$.
2. If a winning region X for player is identified, return X.
3. Else $\top_{i+1} = 2 \cdot \top_i$ (i.e., the top value is doubled).
4. If $\top_{i+1} \geq 2 \cdot n \cdot W$, stop the algorithm and return $\emptyset$, else proceed to the next iteration.

Details can be found in Appendix A.

Correctness and running time analysis. The key steps of the correctness argument and the running time analysis are as follows:

1. The above algorithm is correct, since if it returns a set X then it is a subset of the winning set for player 1.
2. If the algorithm returns a winning set with top value α, then the total running time till this iteration is $m \cdot (\alpha + \alpha/2 + \alpha/4 + \cdots)$, because the progress with top value α requires time $\mathcal{O}(\alpha \cdot m)$. Hence the total running time if a set X is returned with top value α is $\mathcal{O}(\alpha \cdot m)$.
3. Let Z be a set of vertices such that no player-2 vertex in Z has an edge out of Z, and the whole subgame $\Gamma \restriction Z$ is winning for player 1. Then a winning strategy in Z ensures that a progress measure with top value $|Z| \cdot W$ would identify the set Z as a winning set.
4. From above it follows that if the winning set X is identified at top value α, but no winning set was identified with top value $\alpha/2$, then the size of the winning set is at least $\alpha/(2W)$.
5. It follows from above that if a set X is identified, then the total running time to obtain set X is $\mathcal{O}(|X| \cdot m \cdot W)$.
6. Moreover, the total running time of the algorithm when no set X is identified is in $\mathcal{O}(n \cdot m \cdot W)$, and in this case, the winning region is empty.

Thus we solved the modified static algorithm for threshold mean-payoff games as desired and obtain the following result.

▶ **Theorem 4.** *Given a mean-payoff game Γ and a threshold ν, let $Z = W_1(\mathit{MeanPayoff}(\nu))$. If $Z \neq \emptyset$, then a non-empty set $X \subseteq Z$ can be computed in time $\mathcal{O}(|X| \cdot m \cdot W)$, else an empty set is returned if $Z = \emptyset$, which takes time $\mathcal{O}(n \cdot m \cdot W)$.*

Using the above algorithm to compute the winning set for player 1 in the subgames, we obtain an algorithm for threshold mean-payoff coBüchi games in time $\mathcal{O}(n \cdot m \cdot W)$. Details can be found in Appendix B.

▶ **Theorem 5.** *Given a game graph Γ and a threshold mean-payoff coBüchi objective ϕ, the winning set $W_1(\phi)$ can be computed in $\mathcal{O}(n \cdot m \cdot W)$ time.*

4.3 Threshold Mean-Payoff Parity Games

The algorithm for threshold mean-payoff parity games is the standard recursive algorithm [13] (classical parity game-style algorithm) that generalizes the Büchi and coBüchi cases (which are the base cases). The running time recurrence is as follows: $T(n, d, m, w) = n(T(n, d - 1, m) + \mathcal{O}(m)) + \mathcal{O}(nmW)$. Using our approach we obtain the following result (details in Appendix).

▶ **Theorem 6.** *Given a game graph Γ and a threshold mean-payoff parity objective ϕ, the winning set $W_1(\phi)$ can be computed in $\mathcal{O}(n^{d-1} \cdot m \cdot W)$ time.*

5 Optimal Values for Mean-payoff Parity Games

In this section we present an algorithm which computes the value function for mean-payoff parity games. For mean-payoff games a dichotomic search approach was presented in [5]. We show that such an approach can be generalized to mean-payoff parity games.

Range of Values for the Dichotomic Search. To describe the algorithm we recall a lemma about the possible range of optimal values of a mean-payoff parity game. The lemma is an easy consequence of the characterization of [13] that the mean-payoff parity value coincide with the mean-payoff value, and the possible range of value for mean-payoff games.

▶ **Lemma 7** ([13, 15, 16]). *Let Γ be a mean-payoff parity game. For each vertex $v \in V$, the optimal value $val_\Gamma(MPP)(v)$ is a rational number $\frac{y}{z}$ such that $1 \le z \le n$ and $|y| \le z \cdot W$.*

By Lemma 7 the value of each vertex $v \in V$, is contained in the following set of rationals

$$S^\Gamma = \left\{ \frac{y}{z} \;\middle|\; y, z \in \mathbb{Z}, 1 \le z \le n \wedge -z \cdot W \le y \le z \cdot W \right\}.$$

▶ **Definition 8.** Let Γ be a mean-payoff parity game. We denote the set of vertices $v \in V$ such that $val_\Gamma(MPP)(v) \circ \mu$ where $\circ \in \{<, \le, =, \ge, >\}$ with $V_\Gamma^{\circ\mu}$.

Key Observation. Let $\Gamma = (V, E, \langle V_1, V_2 \rangle, w, p)$ be a mean-payoff parity game. Let $\mu \in [-W, W]$. The sets $V_\Gamma^{>\mu}, V_\Gamma^{=\mu}$ and $V_\Gamma^{<\mu}$ can be computed using any algorithm for threshold mean-payoff parity games twice (for example using Theorem 6). To calculate $V_\Gamma^{\ge\mu}$ and $V_\Gamma^{<\mu}$ use the algorithm on Γ with the mean-payoff parity objective $\phi = Parity_\Gamma(p) \cap MeanPayoff_\Gamma(\mu)$. Consider $\Gamma' = (V, E, \langle V_2, V_1 \rangle, w', p)$, where $w'(e) = -w(e)$ for all edges $e \in E$ and player-1 and player-2 vertices are swapped. To calculate $V_\Gamma^{\le\mu}$ and $V_\Gamma^{>\mu}$ use the algorithm on Γ' with mean-payoff parity objective $\phi = Parity_{\Gamma'}(p) \cap MeanPayoff_{\Gamma'}(-\mu)$. Given the sets $V_\Gamma^{\le\mu}$, $V_\Gamma^{>\mu}, V_\Gamma^{\ge\mu}$ and $V_\Gamma^{<\mu}$ we can extract the sets $V_\Gamma^{>\mu}, V_\Gamma^{=\mu}$ and $V_\Gamma^{<\mu}$. All values μ' in S^Γ are of the form $\frac{y}{z}$. For those values we can determine whether $v \in V_\Gamma^{\ge\mu'}$ by applying the algorithm for threshold mean-payoff parity games on $\Gamma' = (V, E, \langle V_2, V_1 \rangle, w', p)$ where $w'(e) = w(e) \cdot z$ for all $e \in E$ with the mean-payoff parity objectives $\phi = Parity_\Gamma(p) \cap MeanPayoff_\Gamma(y)$. Note that in the worst case, the weight function w' of Γ' is in $\mathcal{O}(nW)$.

Dichotomic Search. Let Γ be a mean-payoff parity game. The dichotomic search algorithm is recursive algorithm initialized with $\Gamma_0 = \Gamma$ and $S_0 = S^\Gamma$. In recursive call i the following steps are executed:

1. Let $r_i = \min(S_i)$ and $s_i = max(S_i)$.
2. Determine a_1, the largest element in S_i less than or equal to $\frac{r_i+s_i}{2}$ and a_2, the smallest element in S_i greater than or equal to $\frac{r_i+s_i}{2}$.
3. Determine the partitions $V_{\Gamma_i}^{<a_1}$, $V_{\Gamma_i}^{=a_1}$, $V_{\Gamma_i}^{=a_2}$, $V_{\Gamma_i}^{>a_2}$ using the key observation.
4. For all $v \in V_{\Gamma_i}^{=a_1}$ set the value to a_1, for all $v \in V_{\Gamma_i}^{=a_2}$ set the value to a_2 and set the value to $-\infty$ for all vertices v which are not in any set calculated in step 3.
5. Recurse upon $\Gamma_i \restriction V_{\Gamma_i}^{<a_1}$ and $\Gamma_i \restriction V_{\Gamma_i}^{>a_2}$.

Correctness. Let Γ be a mean-payoff parity game. We prove that the dichotomic search algorithm correctly calculates $val_\Gamma(MPP)(v)$ for all $v \in V$. The algorithm is initialized with Γ and S^Γ. By Lemma 7 the values of the vertices $v \in V$ are in the set S^Γ. Because we perform a binary search over the set S^Γ we can guarantee the termination of the algorithm. Notice that we need to show that the values calculated in the subgames constructed in step 4 are identical to the values in the original game. Then correctness follows immediately by our key observation and because we perform a binary search over the set S^Γ.

▶ **Lemma 9.** *Given a mean-payoff parity game Γ and $\mu \in \mathbb{Q}$, let $\Gamma' = \Gamma \restriction V_\Gamma^{>\mu}$ and $\Gamma'' = \Gamma \restriction V_\Gamma^{<\mu}$. For all $v \in V_\Gamma^{>\mu}$, we have $val_{\Gamma'}(MPP)(v) = val_\Gamma(MPP)(v)$ and for all $v \in V_\Gamma^{<\mu}$, we have $val_{\Gamma''}(MPP)(v) = val_\Gamma(MPP)(v)$.*

Proof. Let $v \in V_\Gamma^{>\mu}$ be arbitrary. We will prove $val_{\Gamma'}(MPP)(v) = val_\Gamma(MPP)(v)$ by showing the following two cases:

- $val_{\Gamma'}(MPP)(v) \leq val_{\Gamma}(MPP)(v)$: Note that there can be no player-2 vertex in $V_{\Gamma}^{>\mu}$ with an edge to $V_{\Gamma}^{\leq\mu}$. Thus we cut away only edges of player-1 vertices in Γ'. Consequently player-1 has less choices in Γ' than in Γ at each of her vertices. Thus $val_{\Gamma'}(MPP)(v) \leq val_{\Gamma}(MPP)(v)$ holds.
- $val_{\Gamma'}(MPP)(v) \geq val_{\Gamma}(MPP)(v)$: Let σ_1 be an optimal strategy for player 1 and let σ_2 be an optimal strategy for player 2 which both exist by [13]. We will show that σ_1 produces plays with vertices in $V_{\Gamma}^{>\mu}$ only, if it starts from v. For the sake of contradiction assume that a play $\rho = outcome(v, \sigma_1, \sigma_2)$ contains a vertex $v^* \in V_{\Gamma}^{\leq\mu}$. Notice that there are no player-2 vertices in $V_{\Gamma}^{>\mu}$ with edges to $V_{\Gamma}^{\leq\mu}$. Thus σ_1 chose a successor vertex in $V_{\Gamma}^{\leq\mu}$. But when ρ ends up in $V_{\Gamma}^{\leq\mu}$ the optimal player-2 strategy σ_2 can guarantee that $MPP_{\Gamma}(w, p, \rho) \leq \mu$ by the definition of $V_{\Gamma}^{\leq\mu}$. There is a strategy to keep the value of the play starting at v greater than μ by the definition of $V_{\Gamma}^{>\mu}$. Thus any play ρ leading to $V_{\Gamma}^{\leq\mu}$ by σ_1 is not optimal which is a contradiction to our assumption. Consequently $val_{\Gamma'}(MPP)(v) \geq val_{\Gamma}(MPP)(v)$ follows.

The fact that for all $v \in V_{\Gamma}^{<\mu}$, we have $val_{\Gamma''}(MPP)(v) = val_{\Gamma}(MPP)(v)$ follows by a symmetric argument.

◀

Running Time. The running time of the dichotomic search is $\mathcal{O}(n \cdot \log(nW) \cdot \mathsf{TH})$ where TH is the running time of an algorithm for the threshold mean-payoff parity problem. The additional factor n comes from rescaling the weights of the mean-payoff parity game Γ which is described in the key observation. The factor $\mathcal{O}(\log(nW))$ is from using binary search on S as $|S| = \mathcal{O}(n^2 \cdot W)$.

▶ **Theorem 10.** *Given a game graph Γ and an algorithm that solves the threshold mean-payoff parity problem in $\mathcal{O}(\mathsf{TH})$, the value function of Γ can be computed in time $\mathcal{O}(n \cdot \log(nW) \cdot \mathsf{TH})$.*

As a corollary of the above theorem and Theorem 6, the value function for mean-payoff parity games can be computed in $\mathcal{O}(n^d \cdot m \cdot W \cdot \log(nW))$ time.

6 Conclusion

In this paper we present faster algorithms for mean-payoff parity games. Our most interesting results are for mean-payoff Büchi and mean-payoff coBüchi games, which are the base cases. For threshold mean-payoff Büchi and mean-payoff coBüchi games, our bound $\mathcal{O}(n \cdot m \cdot W)$ matches the current best-known bound for mean-payoff games. For the value problem, we show the dichotomic search approach of [5] for mean-payoff games can be generalized to mean-payoff parity games. This gives an additional multiplicative factor of $n \cdot \log(nW)$ as compared to the threshold problem. A recent work [14] shows that the value problem for mean-payoff objective can be solved with a multiplicative factor n compared to the threshold objective (i.e., it shaves of the log factor). An interesting question is whether the approach of [14] can be generalized to mean-payoff parity games.

Acknowledgements The authors are partially supported by the Vienna Science and Technology Fund (WWTF) grant ICT15-003. For M. H. and A. S. the research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506. K.C. is supported by the Austrian Science Fund (FWF) NFN Grant No S11407-N23 (RiSE/SHiNE) and an ERC Start grant (279307: Graph Games).

References

1 R. Bloem, K. Chatterjee, K. Greimel, T. A. Henzinger, G. Hofferek, B. Jobstmann, B. Könighofer, and R. Könighofer. Synthesizing robust systems. *Acta Inf.*, 51(3-4):193–220, 2014.

2 R. Bloem, K. Chatterjee, T. A. Henzinger, and B. Jobstmann. Better quality in synthesis through quantitative objectives. In *Proc. of CAV*, LNCS 5643, pages 140–156. Springer, 2009.

3 P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In *Proc. of FORMATS*, LNCS 5215, pages 33–47. Springer, 2008.

4 P. Bouyer, N. Markey, J. Olschewski, and M. Ummels. Measuring permissiveness in parity games: Mean-payoff parity games revisited. In *Proc. of ATVA*, LNCS 6996, pages 135–149. Springer, 2011.

5 L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J. F. Raskin. Faster algorithms for mean-payoff games. *Form. Methods Syst. Des.*, 38(2):97–118, Apr. 2011.

6 J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the AMS*, 138:295–311, 1969.

7 P. Cerný, K. Chatterjee, T. A. Henzinger, A. Radhakrishna, and R. Singh. Quantitative synthesis for concurrent programs. In *Proc. of CAV*, LNCS 6806, pages 243–259. Springer, 2011.

8 A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource interfaces. In *Proc. of EMSOFT*, LNCS 2855, pages 117–133. Springer, 2003.

9 K. Chatterjee and L. Doyen. Energy parity games. In *Proc. of ICALP: Automata, Languages and Programming (B)*, LNCS 6199, pages 599–610. Springer, 2010.

10 K. Chatterjee and L. Doyen. Energy and mean-payoff parity Markov decision processes. In *Proc. of MFCS*, LNCS 6907, pages 206–218. Springer, 2011.

11 K. Chatterjee and L. Doyen. Games and markov decision processes with mean-payoff parity and energy parity objectives. In *MEMICS*, pages 37–46, 2011.

12 K. Chatterjee, L. Doyen, H. Gimbert, and Y. Oualhadj. Perfect-information stochastic mean-payoff parity games. In *FOSSACS*, pages 210–225, 2014.

13 K. Chatterjee, T. A. Henzinger, and M. Jurdziński. Mean-payoff parity games. In *Proc. of LICS*, pages 178–187. IEEE Computer Society, 2005.

14 C. Comin and R. Rizzi. Improved pseudo-polynomial bound for the value problem and optimal strategy synthesis in mean payoff games. *Algorithmica*, 77(4):995–1021, 2017.

15 A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.

16 Y. M. Lifshits and D. S. Pavlov. Potential theory for mean payoff games. *Journal of Mathematical Sciences*, 145(3):4967–4974, 2007.

17 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of POPL*, pages 179–190. ACM Press, 1989.

18 P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.

19 W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3, Beyond Words, chapter 7, pages 389–455. Springer, 1997.

A Details of the modified static algorithm for threshold mean-payoff games

The formal description of the winning set algorithm is as follows:

Algorithm 1: Calculating a winning set of a game mean-payoff game Γ

```
Input: A mean-payoff game $\Gamma = ((V, E, \langle V_1, V_2\rangle), w)$ with mean-payoff objective $\phi$.
Output: A set of winning vertices $X \subseteq W_1(\phi)$ or $\emptyset$
1  begin
2      i=1;
3      while $i \le n$ do
4          Define $\ominus_i : C_\Gamma \times \mathbb{Z} \mapsto C_\Gamma$, where $C_\Gamma = \{j \in \mathbb{N} \mid j \le iW\} \cup \{\top\}$
               $$a \ominus_i b = \begin{cases} \max(0, a-b) & \text{if } a \neq \top \text{ and } a - b \le iW \\ \top & \text{otherwise} \end{cases}$$
5          Use the static algorithm of [5] described in Section 3 (replacing every
            occurrence of the original $\ominus$ with $\ominus_i$) on $\Gamma$ and $\phi$.
6          The algorithm will return a progress measure $f$ where every vertex is consistent
            and we can obtain the winning set: $X = \{v \mid f(v) \neq \top\}$.
7          if $X \neq \emptyset$ then
8              return $X$;
9          end
10         $i = \min(i \cdot 2, n)$;
11     end
12     return $\emptyset$;
13 end
```

▶ **Lemma 11** (Correctness). *Given a game graph Γ and mean-payoff objectives $\phi = MeanPayoff(\nu)$, Algorithm 1 returns a winning set $X \subseteq W_1(\phi)$ or $\emptyset$ if no such set exists.*

Proof. Let Algorithm 1 return a progress measure f for Γ at line 5 and the set X is not empty. By the correctness (shown in [5]) of the static algorithm used in step 5, $X \subseteq W_1(\phi)$. Assume now that Algorithm 1 returns $\emptyset$. Because i will at some point be greater or equal to n, note that the original static algorithm is then executed at line 5. Again by its correctness we get that there are no winning vertices and thus $\emptyset$ is the correct result. ◀

▶ **Lemma 12** (Running Time). *Algorithm 1 returns a winning set X in $\mathcal{O}(|X| \cdot m \cdot W)$ or $\emptyset$ in $\mathcal{O}(n \cdot m \cdot W)$.*

Proof. If Algorithm 1 terminates at line 8 in iteration $i \le n$ returning X, the total running time until this iteration is $m \cdot W \cdot (i + i/2 + i/4 + \cdots)$ because using the static algorithm with $\ominus_i$ requires time $\mathcal{O}(i \cdot m \cdot W)$. Thus, when a set X is returned in iteration i, it requires time $\mathcal{O}(i \cdot m \cdot W)$. Let Z be a set of vertices such that no player-2 vertex in Z has an edge out of Z, and the whole subgame $\Gamma \restriction Z$ is winning for player 1. Then a winning strategy in Z ensures that a progress measure with $i = |Z|$ would identify the set Z as a winning set. From our assumption that Algorithm 1 terminates at $i \le n$ we know that no winning set was identified when i had value $i/2$. Thus the returned set X had size greater than $i/2$. Therefore when a set X is returned in iteration i, it requires time $\mathcal{O}(|X| \cdot m \cdot W)$. If Algorithm 1 terminates at line 12 returning $\emptyset$ we have a runtime $\mathcal{O}(n \cdot m \cdot W)$ as i was n in the last iteration. ◀

The last two lemmas yield Theorem 4.

B Details of the Mean-Payoff coBüchi Algorithm

Algorithm 2 is the new algorithm for threshold mean-payoff coBüchi games, whose correctness is the same as the correctness of the basic algorithm for threshold mean-payoff coBüchi games. Using Theorem 4 for line 6 we obtain that the running time is $\mathcal{O}(n \cdot m \cdot W)$ and hence obtain Theorem 5.

Algorithm 2: SolveMeanPayoffcoBüchi

```
   Input: A mean-payoff coBüchi game $\Gamma = ((V, E, \langle V_1, V_2\rangle, w, p)$ and objectives
          $\phi = \mathit{MeanPayoff}(\nu) \cap \mathit{Parity}(p)$
   Output: The winning region of player 1 in $\Gamma$, i.e., $W_1(\phi)$.
 1 begin
 2     $i \leftarrow 0$, $B_i \leftarrow V$;
 3     repeat
 4         $Y_i \leftarrow \mathit{Attr}_2(B_i \cap p^{-1}(1))$;
 5         $X_i \leftarrow B_i \setminus Y_i$;
 6         Compute a subset of the winning region $Z_i \subseteq W_1(\phi)$ for player 1 in $\Gamma \restriction X_i$
            where $\phi = \mathit{MeanPayoff}(\nu)$ is the threshold mean-payoff objective.;
 7         $B_{i+1} \leftarrow B_i \setminus \mathit{Attr}_1(Z_i)$;
 8         $i \leftarrow i + 1$
 9     until $B_i = B_{i+1}$;
10     return $V \setminus B_i$
```

C Details of the Mean-Payoff Parity Algorithm

We recall the algorithm for mean-payoff parity games [13], and present the relevant details for the sake of being self-contained. Algorithm 3 is the detailed pseudocode, and we present a succinct correctness proof.

▶ **Lemma 13** (Correctness). *Given game graph $\Gamma = (V, E, \langle V_1, V_2\rangle, w, p)$ with objectives $\phi = \mathit{MeanPayoff}(\nu) \cap \mathit{Parity}(p)$, Algorithm 3 correctly computes the set $W_1(\phi)$ for Γ.*

Proof. We proceed by induction on the number of priorities d. For the base cases, i.e. when $d = 2$ we can use Theorem 3 and Theorem 5 to receive the winning set $W_1(\phi)$ for Γ. Assume now that for $d \leq k$ Algorithm 3 correctly returns $W_1(\phi)$ for Γ. For the induction step, assume that we have $d = k + 1$ priorities. We need to show that we will correctly return $W_1(\phi)$ for Γ. Therefore we make a case distinction whether the smallest priority is even or odd.

- Assume the smallest priority k in Γ is even and thus v is in the set returned by line 18. By the construction of the algorithm, we have two cases:
  - Either v is in Y_i (line 7),
  - or v is in the mean-payoff parity winning set of the game $\Gamma \restriction C_i$ for some i (line 9).

  Note that if v is in Y_i we can ensure $\mathit{MeanPayoff}(\nu)$ because we remove every vertex not sufficing the objective in lines 13-15. We thus have a strategy σ_{mp} ensuring $\mathit{MeanPayoff}(\nu)$. It remains to argue why we can in both cases win the mean-payoff parity game. If v is in the mean-payoff parity winning set obtained by the recursive call (line 9) we have a strategy by the induction hypothesis. If v is in Y_i we will propose a strategy for player-1 starting from v. The strategy will be played in rounds $1, 2, \ldots$. In round i we will play the following strategy: Because v is in the player-1 attractor of $p^{-1}(k)$ we can visit k. This could mean that we accumulate (in the worst case) up to $-(n-1)W$ credits for the

Algorithm 3: SolveMeanPayoffParity

```
Input: A mean-payoff parity game $\Gamma = ((V, E, \langle V_1, V_2\rangle, w, p)$ and objectives
       $\phi = \mathit{MeanPayoff}(\nu) \cap \mathit{Parity}(p)$
Output: The winning region of player 1 in $\Gamma$, i.e., $W_1(\phi)$.
begin
    if $V = \emptyset$ then return $\emptyset$;
    $k \leftarrow \min\{p(v) \mid v \in V\}$, $i \leftarrow 0$, $A_0 \leftarrow V$;
    if $k$ is even then
        if $p$ has two priorities then solve the mean-payoff Büchi game $\Gamma$ with
         objectives $\phi$.;
        repeat
            $Y_i \leftarrow \mathit{Attr}_1(A_i \cap p^{-1}(k))$;
            $C_i \leftarrow (A_i \setminus Y_i)$;
            $X_i \leftarrow C_i \setminus \mathit{SolveMeanPayoffParity}(\Gamma \upharpoonright C_i, \phi))$;
            if $X_i \neq \emptyset$ then
                $A_{i+1} \leftarrow A_i \setminus \mathit{Attr}_2(X_i)$;
            else
                $U_i \leftarrow W_1(\varphi)$ in $\Gamma \upharpoonright A_i$, where $\varphi = \mathit{MeanPayoff}(\nu)$
                $X_i \leftarrow A_i \setminus U_i$.
                $A_{i+1} = A_i \setminus \mathit{Attr}_2(X_i)$
            $i \leftarrow i + 1$;
        until $A_i = A_{i+1}$;
        return $A_i$;
    else if $k$ is odd then
        if $p$ has two priorities then solve the mean-payoff Büchi game $\Gamma$ with
         objectives $\phi$.;
        $B_0 \leftarrow V$;
        repeat
            $Y_i \leftarrow \mathit{Attr}_2(B_i \cap p^{-1}(k))$;
            $X_i \leftarrow B_i \setminus Y_i$;
            $Z_i \leftarrow \mathit{SolveMeanPayoffParity}(\Gamma \upharpoonright X_i, \phi)$;
            $B_{i+1} \leftarrow B_i \setminus \mathit{Attr}_1(Z_1)$;
            $i \leftarrow i + 1$;
        until $B_i = B_{i+1}$;
        return $V \setminus B_i$;
```

mean-payoff objective. If we end up in the mean-payoff parity winning set, we win by the induction hypothesis. If we are still in the player-1 attractor set Y_i, we play σ_{mp} for i steps which will ensure the mean-payoff conditions as $i \to \infty$. After playing σ_{mp} for i steps we can end up (i) again in the player-1 attractor set Y_i, enabling us to visit a vertex of priority k, or (ii) in the mean-payoff parity winning set where we win by induction hypothesis.

- Assume the smallest priority k in Γ is odd. Thus v must have been in the set returned by line 29. It must be that v is in some Z_i or some player-1 attractor to it. If v is in Z_i we win the game by the induction hypothesis. Otherwise if we are in a player-1 attractor to Z_i we will use the strategy induced by the attractor to reach Z_i.

◀

▶ **Lemma 14** (Running Time)**.** *The worst case complexity of Algorithm 3 is $\mathcal{O}(n^{d-1} \cdot m \cdot W)$.*

Proof. Let $T(n, d, m, w)$ be the complexity of Algorithm 3. Since every recursive call removes at least one state from A_i and since the number of priorities decrease in a recursive call we get the following recurrence relation: $T(n, d, m, w) = n(T(n, d-1, m, w) + \mathcal{O}(m)) + \mathcal{O}(nmW) = n(T, n, d-1, m, w) + \mathcal{O}(nm) + \mathcal{O}(nmW)$. Note that $\mathcal{O}(m)$ is used to calculate the attractors. We only get $\mathcal{O}(nmW)$ once every iteration, because we can use the decremental algorithm introduced in Section 3 to calculate the mean-payoff objectives in line 13. Note that this is particularly possible because only player-2 attractors get removed, thus ensuring the input condition for the algorithm. We can thus simplify to $T(n, d, m, w) = n(T, n, d-1, m, w) + \mathcal{O}(nmW)$. Also note that when we have the base case $T(n, 2, m, w)$ we can solve the problem using Theorem 3 and Theorem 5 in $\mathcal{O}(n \cdot m \cdot w)$. Solving the recurrence relation yields $T(n, d, m, w) = n^{d-1} + (d-1) \cdot \mathcal{O}(n \cdot m \cdot W)$ which concludes the proof. ◀

The last two lemmas yield Theorem 6.