A Faster Deterministic Exponential Time Algorithm for Energy Games and Mean Payoff Games

Dani Dorfman
Blavatnik School of Computer Science, Tel Aviv University, Israel
dannatand@mail.tau.ac.il

Haim Kaplan
Blavatnik School of Computer Science, Tel Aviv University, Israel
haimk@post.tau.ac.il

Uri Zwick
Blavatnik School of Computer Science, Tel Aviv University, Israel
zwick@tau.ac.il

Abstract

We present an improved exponential time algorithm for Energy Games, and hence also for Mean Payoff Games. The running time of the new algorithm is $O\left(\min\left(mnW, mn2^{n/2}\log W\right)\right)$, where n is the number of vertices, m is the number of edges, and when the edge weights are integers of absolute value at most W. For small values of W, the algorithm matches the performance of the pseudopolynomial time algorithm of Brim et al. on which it is based. For $W \geq n2^{n/2}$, the new algorithm is faster than the algorithm of Brim et al. and is currently the fastest *deterministic* algorithm for Energy Games and Mean Payoff Games. The new algorithm is obtained by introducing a technique of forecasting repetitive actions performed by the algorithm of Brim et al., along with the use of an edge-weight *scaling* technique.

2012 ACM Subject Classification Computing methodologies → Stochastic games

Keywords and phrases Energy Games, Mean Payoff Games, Scaling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.114

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Funding Haim Kaplan was partially supported by the Israel Science Foundation, grant 1841/14, by the German-Israeli Foundation for Scientific Research and Development, grant 1367/2017, and by the Blavatnik Research Foundation. Part of this research was carried out while Uri Zwick was visiting BARC, Copenhagen, funded by the VILLUM Foundation, grant 16582, and while he was visiting IRIF, Paris, with support from the FSMP.

1 Introduction

Energy Games (EGs) and Mean Payoff Games (MPGs) are simple and natural infinite-duration games played on graphs that can be used to model quantitative properties of interactive systems. They are also interesting as they are perhaps the most natural combinatorial problems that are in $NP \cap \text{co-}NP$ and yet not known to be in P or in BPP. Mean Payoff Games (MPGs) were introduced by Ehrenfeucht and Mycielski [9]. Energy Games (EGs) were introduced by Chakrabarti et al. [7] and later by Bouyer et al. [4] who also showed their equivalence to MPGs.

Energy Games are games played by two players, player 0 and player 1, on a weighted directed graph whose vertices are partitioned among the two players. The two players construct an infinite path, that starts at a designated start vertex, in the following way. The player controlling the end-point u of the path constructed so far extends the path by

46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 114; pp. 114:1–114:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

choosing an edge emanating from u. Let $w_1, w_2, \ldots$ be the weights of the edges on the path constructed. Player 0 wins this play if $\liminf_{n\to\infty} \sum_{i=1}^{n} w_i > -\infty$, i.e., if there exists an initial finite *energy level* c such that $c + \sum_{i=1}^{n} w_i \geq 0$, for every $n \geq 1$. Player 1 wins otherwise. Player 0 wins the game from an initial vertex u if she can ensure a winning play from u, no matter what player 1 does. It is known that if player 0 can win from a certain vertex, then she can also do it using a *positional strategy*, i.e., a deterministic strategy in which the edge chosen depends only on the current vertex. Furthermore, she has a single positional strategy using which she wins from all the vertices from which she can win. Solving an EGs amounts to finding the winner from each vertex, and possibly an optimal positional strategy and the minimal energy level required from every winning vertex.

Parity Games (PGs) form a very special sub-class of MPGs. In a recent breakthrough, Calude et al. [6] obtained a deterministic quasipolynomial $n^{O(\log n)}$-time algorithm for PGs, where n is the number of vertices. (Variants of their algorithm were obtained by [3, 10, 13, 18, 21].) Unfortunately, these techniques do not seem applicable to MPGs and EGs. (See [11].) The currently fastest algorithm for these games, as well as the more general (turn-based) Stochastic Games (SGs), is a sub-exponential $2^{\tilde{O}(\sqrt{n})}$ ([1, 2, 16, 17, 23]). These sub-exponential algorithms are based on randomized pivoting rules for the simplex algorithm devised by Kalai [19, 20] and Matoušek, Sharir and Welzl [24]. The fastest known deterministic algorithms for EGs and MPGs are the exponential $O(mn2^n \log W)$-time algorithm of Lifshits and Pavlov [22],[1] and a pseudo-polynomial $O(mnW)$-time algorithm of Brim et al. [5].[2] Polynomial time algorithms for EGs with very special weight structures were obtained by Chatterjee et al. [8].

The simple and elegant $O(mnW)$-time algorithm of Brim et al. [5], henceforth referred to as the BCDGR algorithm, is a *progress measure lifting* algorithm for solving EGs. MPGs are essentially equivalent to EGs ([4]), hence the algorithm can also be used to solve MPGs. The lifting technique used by Brim et al. is similar to the *value iteration* technique used by Zwick and Paterson [26] on MPGs.

We present an improvement of the BCDGR algorithm that runs in $O(\min\{mnW, mn2^{n/2} \log W\})$-time. The new algorithm is always as fast as the BCDGR algorithm and strictly faster when $W = \omega(n2^{n/2})$. The running time of the new algorithm can be made to be $O(poly(n)2^{n/2})$, without any dependence on W, assuming that arithmetic operations on integers of absolute value $O(nW)$ take constant time. (Details will appear in the full version of the paper.) The new algorithm is currently the fastest *deterministic* algorithm for EGs and MPGs when $W \geq n2^{n/2}$.

The new algorithm uses two new ideas. The first is a technique for predicting sequences of update steps that are performed repetitively by the BCDGR algorithm, and achieving the net effect of these repetitions much more quickly. To make this approach work, a second idea, that of *scaling*, needs to be used. Scaling is a well-known technique used in various combinatorial optimization problems such as shortest paths, flow problems, matching problems etc. (See, e.g., [12, 14, 15].) A scaling algorithm first divides all edge weights by 2, rounds them up so that they remain integers, solves the reduced problem recursively, and then converts the solution of the reduced problem to a solution of the original algorithm. It is quite natural to try to use the scaling technique on EGs or MPGs. However, naïve or

[1] For solving EGs, and for deciding whether the values of a MPG are non-negative, the $\log W$ factor in the running time of [22] is not needed.

[2] Recently, Fijalkow et al. [11] gave an $O(mn(nW)^{1-1/n})$-time algorithm for solving MPGs. This, however, is never asymptotically better than $O(\min\{mnW, mn2^n\})$, as $W^{1-1/n} < \frac{1}{2}W$ only if $W \geq 2^n$.

direct approaches do not seem to give any improvement. To the best of our knowledge, the algorithm presented here is the first algorithm that successfully uses scaling for speeding up the solution of EGs and MPGs.

An EG, MPG or SG is said to be *binary* if the outdegree of each vertex is 2. It is known that binary EGs, MPGs and SGs can be modeled as *Acyclic Unique Sink Orientations* (AUSOs) (see, e.g., [23, 25]). Solving a game is then equivalent to finding the *sink* of the associated AUSO. The fastest deterministic sink-finding algorithm runs in $O(1.606^n)$-time. Our new algorithm is faster than this algorithm and works for all games, not only binary.

The rest of the paper is organized as follows. In the next section we provide some definitions and basic results and briefly review the algorithm of Brim et al. [5] on which our new algorithm is based. In Section 3 we present our new algorithm. In the full version of the paper we describe energy games on which the new algorithm requires $\Omega(2^{n/2})$ time, showing that our analysis is essentially tight. We end in Section 4 with concluding remarks and open problems.

2 Preliminaries

A **game graph** is a tuple $\Gamma = (V_0, V_1, E, w)$, where $V = V_0 \cup V_1$ is the set of vertices, $E \subseteq V \times V$ is the set of edges, and $w : E \to \mathbb{Z}$ is a weight function. We assume that $V_0 \cap V_1 = \emptyset$ and that each vertex has at least one outgoing edge. The sets V_0 and V_1 are the sets of vertices controlled by player 0 and player 1. A *positional strategy* of player i is a mapping $\sigma : V_i \to E$ such that for every $v \in V_i$ we have $(v, \sigma(v)) \in E$. Given *positional strategies* σ_0, σ_1 of player 0 and player 1 and an initial vertex v_0, $\mathit{play}(v_0, \sigma_0, \sigma_1) = v_0, v_1, \ldots, v_i, \ldots$ is the infinite walk resulting from σ_0 and σ_1 starting at v_0.

An **Energy-Game** is an infinite game on a game graph Γ. Player 0 wins from an initial vertex $v_0 \in V$ if and only if there exists a positional strategy σ_0, and a finite *energy level* $c = c(v_0)$, such that for every positional strategy σ_1 of player 1, we have $c + \sum_{i=0}^{n-1} w(v_i, v_{i+1}) \geq 0$, for every $n \geq 1$, where $\mathit{play}(v, \sigma_0, \sigma_1) = v_0, v_1, \ldots$.

We shall refer to a function $f : V \to \mathbb{N} \cup \{\infty\}$ as a *potential* function.

▶ **Definition 2.1.** *Let $\Gamma = (V_0, V_1, E, w)$ be an energy-game. A function $f : V \to \mathbb{N} \cup \{\infty\}$ is a feasible potential iff for every $v \in V$:*

- *if $v \in V_0$, then $f(v) + w(v, v') \geq f(v')$ for some $(v, v') \in E$.*
- *if $v \in V_1$, then $f(v) + w(v, v') \geq f(v')$ for all $(v, v') \in E$.*

We call the potential function $g(v) = \min\{f(v) \mid f \text{ feasible potential}\}$ the solution *of Γ.*

Brim et al. [5] proved that g is a *feasible* potential and that player 0 wins from v if and only if $g(v) < \infty$, in which case $g(v)$ is the minimal required initial energy.

Let $\Gamma = (V_0, V_1, E, w)$ be an energy-game and let $f : V \to \mathbb{N} \cup \{\infty\}$ be a *potential* function. We denote by $w_f(u, v) = w(u, v) + f(u) - f(v)$ the *modified* weight of (u, v). An edge (u, v) is *valid* with respect to f if $w_f(u, v) \geq 0$. A vertex $v \in V_0$ (V_1) is *valid* with respect to f if (v, v') is *valid* with respect to f for some (all) $(v, v') \in E$, otherwise we say that v is *invalid* with respect to f (we say just *valid* when f is clear from the context). An edge (v, v') is *tight* if $w_f(v, v') = 0$. A path p is *tight* if all its edges are tight. A vertex $v \in V_0$ is *tight* if it is *valid* and $w_f(v, v') \leq 0$ for all $(v, v') \in E$. A vertex $v \in V_1$ is *tight* if it is *valid* and $w_f(v, v') = 0$ for some $(v, v') \in E$. We denote by $\mathit{in}(u)$ and $\mathit{out}(u)$ the sets of incoming and outgoing edges from u, respectively.

2.1 The algorithm of Brim et al.

Brim et al. [5] suggested the following algorithm: maintain $f : V \to \mathbb{N} \cup \{\infty\}$, starting with $f \equiv 0$. As long as there are *invalid* vertices, pick some *invalid* vertex v and increase $f(v)$ to the minimal value that would make v *valid*. It is known that if player 0 can win from a certain vertex, then she can win with an initial energy of at most nW. Thus, if $f(v)$ reaches nW, we know that v is a losing vertex for player 0, and we can let $f(v) \leftarrow \infty$.

To efficiently find an invalid vertex, the algorithm maintains a list L of *invalid* vertices. When $f(v)$ of some *invalid* vertex $v \in V$ is updated, the algorithm checks for every edge $(v', v) \in in(v)$ that became *invalid* whether v' is now also *invalid.* If $v' \in V_1$, then this is the case, and v' is added to L, if it is not already there. If $v' \in V_0$, then increasing $f(v)$ does not necessarily make v' *invalid*, as v' may have had other valid edges. The algorithm maintains $count[v']$, the number of valid edges in $out(v')$. If (v', v) was valid, then $count[v']$ is decremented. If $count[v']$ becomes 0, then v' is now *invalid* and it is added to L. It is not hard to check that the running time of the resulting algorithm is $O(mnW)$, which is also known to be tight.

2.2 A reduction to games with finite values

The description and the correctness proof of algorithms for solving EGs are often simplified if it assumed that all vertices have finite values, i.e., are all winning for player 0. (This does not trivialize the problem, as we still want to find the minimum energy level needed from each vertex, and corresponding optimal positional strategies for the two players.) We describe a simple reduction, inspired by a reduction of Björklund et al. [2], that shows that the solution of a general EG can be reduced to the solution of an EG with finite values.

Let $\Gamma = (V_0, V_1, E, w)$ be an EG, and let $n = |V|$ and $W = \max_{e \in E} |w(e)|$. Let f be the solution of Γ. For every $v \in V$, we know that either $0 \leq f(v) < nW$, or $f(v) = \infty$. To convert Γ into a game Γ' in which all values are finite, we add a *sink* vertex s, with a self-loop of weight 0, and add an edge (v, s) of weight $-2nW$ for every $v \in V_0$. This ensures that the values of all vertices in V_0 are finite. (In particular, their value is at most $2nW$.)

To ensure that the values of all vertices in V_1 are also finite, we need to perform a simple preprocessing step. If $u \in V_1$ and player 0 has a strategy for reaching a vertex of V_0, starting at u, then the value of u is also finite. We are thus left with vertices of V_1 from which player 1 can win the game, i.e., reach a negative cycle, without leaving V_1. It is easy to identify these vertices and remove them from the game. The value of all remaining vertices is now finite.

If it is to player 0's advantage to escape to the sink, she might as well do it without closing any cycles. Player 0 can therefore gain at most $(n-1)W$ units of energy by following original edges before deciding to take an edge to the sink. The energy needed in such a case is therefore at least nW. We thus have:

▶ **Lemma 2.2.** *Let $\Gamma = (V_0, V_1, E, w)$ be an EG and let Γ' be the EG obtained by the reduction above. Let f and f' be the solutions of Γ and Γ'. Then, for every $u \in V = V_0 \cup V_1$, we have*

$$f(u) \;=\; \begin{cases} f'(u) & \text{if } f'(u) < nW, \\ \infty & \text{otherwise.} \end{cases}$$

Note that the reduction introduces only one new vertex which is important if we want to use it in conjunction with exponential time algorithms. The maximal edge weight is increased from W to $2nW$, but this is not an issue if the running time of the algorithm depends only logarithmically on W.

```
Compute-Energy (V0, V1, E, w).
if w ≥ 0 then
    return f ≡ 0
w' ← ⌈w/2⌉
f ← Compute-Energy(V0, V1, E, w')
f, w' ← 2f, 2w'
foreach v ∈ V do
    foreach e ∈ out(v) do
        if w'(e) > w(e) then  w'(e) ← w'(e) − 1
    Update-Energy (V0, V1, E, w', f, v)
return f
```

```
Update-Energy (V0, V1, E, w, f, v).
if v ∈ V0 and ∀(v, u) ∈ E : w_f(v, u) < 0 then L ← {v}
if v ∈ V1 and ∃(v, u) ∈ E : w_f(v, u) < 0 then L ← {v}
foreach u ∈ V0 do
    count[u] ← |{u' | (u, u') ∈ E,  w_f(u, u') ≥ 0}|
while L = {v} do
    B ← {v}
    Update(v, L, B)
    while L \ {v} ≠ ∅ do
        pick u ∈ L \ {v}
        Update(u, L, B)
    Δ ← Delta(B)
    foreach u ∈ B do f(u) ← f(u) + Δ
```

Figure 1 The main two functions of the new $O\left(\min\left(mnW, mn2^{n/2}\log W\right)\right)$-time algorithm.

3 The new algorithm

We now describe our new algorithm. For simplicity, we assume that all the values in the input game are finite. This can be achieved, for example, using the simple reduction above. In the full version we show that the algorithm presented actually works, as is, even if some values are infinite, but the correctness proof becomes slightly more complicated.

Given an EG $\Gamma = (V_0, V_1, E, w)$, we construct a scaled down version $\Gamma' = (V_0, V_1, E, w' = \lceil \frac{w}{2} \rceil)$, where $\lceil \frac{w}{2} \rceil(e) = \lceil \frac{w(e)}{2} \rceil$, for every $e \in E$, and solve it recursively, obtaining the solution f' of Γ'. (Note that Γ' is "easier" for player 0 because of the rounding up. In particular, if all values in Γ are finite, so are all the values in Γ'.) We now scale Γ' back up to $\Gamma'' = (V_0, V_1, E, 2\lceil \frac{w}{2} \rceil)$. Note that $f'' \equiv 2f'$ is the solution of Γ'' and that Γ'' is very close to Γ: $2\lceil \frac{w}{2} \rceil$ and w only differ, by 1, on edges e for which $w(e)$ is *odd*. To convert f'' into a solution of Γ, we consider each vertex $v \in V$ with odd outgoing edges, decrease the corresponding edge weights in $2\lceil \frac{w}{2} \rceil$ by 1, and update the solution f'' accordingly. (This is, of course, the hardest part of the algorithm.) These operations are carried out by algorithms Compute-Energy and Update-Energy given in Figure 1.

UPDATE(u, L, B).

$L \leftarrow L \setminus \{u\}$
$f(u) \leftarrow f(u) + 1$
if $u \in V_0$ **then** $count[u] \leftarrow |\{(u,u') \in E \mid \ w_f(u,u') \geq 0\}|$
foreach $u' \in in(u)$ ***such that*** $w_f(u',u) < 0$ **do**
  if $u' \in V_0$ **then**
    if $w_f(u',u) = -1$ **then** $count[u'] \leftarrow count[u'] - 1$
    if $count[u'] = 0$ **then** $L \leftarrow L \cup \{u'\}, B \leftarrow B \cup \{u'\}$
  if $u' \in V_1$ **then** $L \leftarrow L \cup \{u'\}, B \leftarrow B \cup \{u'\}$

DELTA(B).

$p_1 \leftarrow \min\left\{-w_f(u,u') \mid (u,u') \in E(B \cap V_0, \bar{B})\right\}$
$p_2 \leftarrow \min\left\{\gamma(u) \mid u \in \bar{B} \cap V_0, \forall u' \in \bar{B} \ \ w_f(u,u') < 0\right\}$
$p_3 \leftarrow \min\left\{w_f(u,u') \mid (u,u') \in E(\bar{B} \cap V_1, B)\right\}$
return $\min\{p_1, p_2, p_3\}$

Figure 2 The remaining two function of the new $O\left(\min\left(mnW, mn2^{n/2}\log W\right)\right)$-time algorithm.

UPDATE-ENERGY updates the solution f after the decrease of the weights of some of the edges emanating from v by 1. As in [5], UPDATE-ENERGY maintains a list L of all vertices that are currently *invalid*. Initially only v may be *invalid*. To quickly determine whether a vertex $u \in V_0$ becomes *invalid*, we maintain in $count[u]$ the number of *valid* edges from u. A vertex $u \in V_0$ is thus *invalid* iff $count[u] = 0$.

Lines 1–2 of UPDATE-ENERGY determine whether v is *invalid*. If v is *valid*, we are done. Otherwise, we add v to L and *fix* v by increasing $f(v)$ by 1. This makes v *valid*, but vertices u with edges (u, v) may become *invalid* and need to be added to L. Updating v's potential and checking for new *invalid* vertices is carried out by UPDATE given in Figure 2. If we fix *invalid* vertices from L in an arbitrary order, as done by [5], we get the running time of [5].

The main point in which our algorithm differs from the algorithm of [5], in addition to the use of scaling, is that we fix at first only *invalid* vertices *different* from v, delaying an additional fixing of v, if required, by as much as possible. Lemma 3.2 below shows that no vertex needs to be fixed twice, before v is fixed again.

If v does not become *invalid* again, then UPDATE-ENERGY is done. Otherwise, let B be the set of vertices, including v, updated until v is the only *invalid* vertex. (Lemma 3.2 shows that this must eventually happen.) When we update v again, it could be that the *same* set of vertices B will eventually become *invalid*, and hence updated, again. Furthermore, in worst-case examples of [5], the same sequences of update operations may be repeated many times. Instead of carrying out these updates again and again, we *predict* how many times the same sequence of updates will be repeated and perform all these updates at once. This approach seems to work only when the weights of edges are decreased by 1, which is why the scaling idea needs to be used.

The computation of Δ, the number of repetitions of the current sequence, carried out by DELTA(B) in Figure 2, is based on the following observation. Let B be the set of vertices that became *invalid* after updating v and let B' be the set of vertices that became *invalid* after updating v again. Assume $B' \neq B$. If $B' \setminus B \neq \emptyset$, let $u \in B' \setminus B$ be the first vertex in $B' \setminus B$ that became *invalid*. It must be the case that at least one of u's *valid* edges to B became *invalid*. If $u \in V_1$ this could be any edge from u, and if $u \in V_0$ this edge is the edge with maximal *modified* weight from u and u had no *valid* edges to $\bar{B} \equiv V \setminus B$.

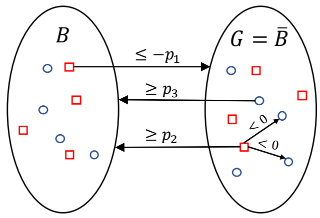

Figure 3 Calculating DELTA(B): Vertices in V_0 are red squares; Vertices in V_1 are blue circles.

If $B \setminus B' \neq \emptyset$, then $(B \setminus B') \cap V_0 \neq \emptyset$. To see this, let $v_0 = v, v_1, \ldots$ be the vertices of B in the order in which they were updated. Let v_j be the first vertex in this order which is not in B'. Vertex v_j must have an *invalid* edge (v_j, v_ℓ) such that $\ell < j$. This edge became *invalid* when we updated v_ℓ and caused the addition of v_j to B. Since v_j is the first vertex which is not in B', $v_\ell \in B'$ and therefore the edge (v_j, v_ℓ) becomes *invalid* when we collect B' following the second update of v. So $v_j \in V_0$, as otherwise it should have been added to B'. Vertex v_j is not added to L since it has an edge (v_j, w), $w \notin B$ that had a *modified* weight of -1 that became 0 (i.e., *valid*) following the update of v_j.

Therefore, to compute Δ, we must consider all *valid* edges from $\bar{B}$ to B (to detect new vertices that might become *invalid*) and all *invalid* edges from vertices in $V_0 \cap B$ to $\bar{B}$, see Figure 3. We refer to minimum and maximum of an empty set as ∞ and $-\infty$, respectively. We let $\Delta = \min\{p_1, p_2, p_3\}$ were p_1, p_2 and p_3 are defined as follows. The value p_1 is minus the maximum modified edge weight of an edge from $B \cap V_0$ to $\bar{B}$. Note that $p_1 \geq 0$. To define p_2 consider every vertex $u \in V_0 \cap \bar{B}$ that does not have a valid edge (u, w) to $w \in \bar{B}$. For every such u let $\gamma(u)$ be the maximum modified weight of an edge (u, w), $w \in B$. Note that $\gamma(u) \geq 0$. We define p_2 to be the minimum value of $\gamma(u)$ over all such vertices u. The value p_3 is the minimum modified edge weight of an edge from $V_1 \cap \bar{B}$ to B. Note that p_2 and p_3 are nonnegative. Pseudo-code of DELTA(B) is given in Figure 2.

3.1 Correctness

As explained, we assume for simplicity that all values are finite. This assumption is removed in the full version of the paper. The correctness of the new algorithm follows from the fact that the potential function kept by the algorithm is always a lower bound on the values of the vertices, and hence the updates performed are justified, as in the correctness proof of the BCDGR algorithm. As the new algorithm predicts sequence of updates that are going to be performed repeatedly, and performs all these repetitions at once, what remains to be shown is that the predictions of the algorithm are correct.

▶ **Lemma 3.1.** *Let $\Gamma^1 = \left(V_0, V_1, E, w^1\right)$, $\Gamma^2 = \left(V_0, V_1, E, w^2\right)$ be two games graphs with solutions f^1 and f^2, respectively. If $w^1 \leq w^2$ then $f^1 \geq f^2$ (coordinate-wise).*

It follows Lemma 3.1 that the solution of Γ'' is a lower bound on the solution of the original game Γ. All that remains, therefore, is to show that the updates performed by UPDATE-ENERGY are justified.

Let UPDATE($v_1 \equiv v$), UPDATE(v_2), …, UPDATE(v_k) be the sequence of vertex updates performed by UPDATE-ENERGY($\ldots, f, v$). A *round* is one iteration of the outer while loop of UPDATE-ENERGY, i.e., all vertex updates starting with UPDATE(v) until and not including the next UPDATE(v). We number the rounds starting from 1 and let f^r be f at the end of round r. For convenience, we define $f^0 \equiv f$. We let B^r be the set of vertices that were updated during round r ("bad" vertices). Thus, B^r is B at the end of round r of the outer while loop of UPDATE-ENERGY. Let $G^r = \overline{B^r} = V \setminus B^r$ be the set of "good" vertices.

▶ **Lemma 3.2.** *In each round, each vertex is updated at most once.*

Proof. By contradiction, let u be the first vertex that joined L for the second time during a round. We have that $u \neq v$ by the definition of a *round*. Assume $u \in V_0$. Since $u \neq v$ (i.e., u is *valid* at the beginning of the round), $w_f(u, u') \geq 0$ for some vertex u' at the beginning of the round. From the beginning of the round until u's second update, u' is updated at most once and u is updated exactly once so we have that $w_f(u, u') \geq 0$ right before u's second update which is a contradiction (u is *valid*). A similar argument works when $u \in V_1$. ◀

▶ **Lemma 3.3.** *During* Update-Energy*, $u \in L$ if and only if u is* invalid*.*

Proof. By induction on the iterations of the algorithm. ◀

Note that vertices u that were never updated (in any call to Update-Energy) are those with $f(u) = 0$. Also, note that every *tight* vertex u has at least one *tight* edge.

▶ **Lemma 3.4.** *During* Update-Energy*, if $u \notin L$ and $f(u) > 0$, then u is tight.*

Proof. Following an update, u becomes *tight* and it remains *tight* as long as it is *valid*. Since $f(u) > 0$, u was updated and since $u \notin L$, u is *valid* (Lemma 3.3). ◀

▶ **Lemma 3.5.** *At the end of round r, every $u \in (G^r \setminus \{v \mid f(v) = 0\}) \cap V_0$ is tight and for every tight edge (u, u') it holds that $u' \in G^r$. Also, there exists $u' \in G^r$ such that $(u, u') \in E$ is* tight *during round r.*

Proof. We prove the first part by contradiction. Let (u, u') be a *tight* edge at the end of round r such that $u' \in B^r$. Since $u' \in B^r$, u' was updated during round r and therefore at the beginning of round r it holds that $w_f(u, u') > 0$, so u was not *tight* at the beginning of the round. This contradicts Lemma 3.4 that implies that u is always *tight* during round r.

We now prove the second part. Since u is *tight* during the round and in particular at the end of the round, $(u, u') \in E$ is *tight* for some u' at the end of the round. By the first part of the Lemma $u' \in G^r$. Thus, (u, u') is *tight* during the entire round (since both $f(u)$ and $f(u')$ remain unchanged during the round). ◀

▶ **Lemma 3.6.** *At the end of round r, if $u \in (B^r \setminus \{v\}) \cap V_1$ and $(u, u') \in E$ is* tight*, then $u' \in B^r$.*

Proof. If $u' \in G^r$ then (u, u') was *invalid* at the beginning of round r (since $f(u)$ but not $f(u')$ was increased during the round), and therefore u was *invalid* at the beginning of round r, but only v is invalid at the beginning of each round, a contradiction. ◀

▶ Remark. Note that we cannot guarantee that u has a *tight* edge during the round (as in Lemma 3.5). This is because when u becomes *invalid*, it might be the case that all of its edges became *invalid* (u is ensured to have a *tight* edge only when it is *valid*).

The proof of the following lemma is given in the full version of the paper.

▶ **Lemma 3.7.** *Consider round r. If we would have performed* Update-Energy *without lines* 11–12*, then in the following Δ rounds $r + 1, ..., r + \Delta$ we would have $B^r = B^{r+i}$, for $1 \leq i \leq \Delta$, where Δ is the value returned by* Delta(B)*. Furthermore, if $r + \Delta$ is not the last round then $B^{r+\Delta+1} \neq B^r$.*

As a consequence we get:

▶ **Theorem 3.8.** Update-Energy (V_0, V_1, E, w, f, v) *updates f correctly.*

The rest of the lemmas in this section are used in the next section to bound the complexity of the algorithm.

Let u be a *valid* vertex such that $f^r(u) > f^0(u)$ (i.e., the potential of u was changed at least once before the end of round r). We let $t_r(u)$ be the time right after the last $\text{UPDATE}(u)$ that occurred before the end of round r. We remove the subscript r when it is clear from the context.

▶ **Lemma 3.9.** *At the end of round r, for any $u \in G^r$ with $f^r(u) > f^0(u)$:*

1. *If $u \in V_0$ then for any* tight *edge (u, u') with $u' \in G^r$, either $f^r(u') = f^0(u')$ or $t_r(u') < t_r(u)$.*
2. *If $u \in V_1$ then there exists a* tight *edge (u, u') such that $u' \in G^r$ and $t_r(u') < t_r(u)$.*

Proof. Note that since $u \in G^r$, $t_r(u)$ is before round r begins. We begin with the first part. Let (u, u') be such an edge. If $f^r(u') = f^0(u')$ then we are done. Otherwise, u' must have been updated at least once (and therefore $t(u')$ is defined). Assume by contradiction that $t(u') > t(u)$. Since (u, u') is *tight* at the end of round r then $w_f(u, u') > 0$ at $t_r(u)$. This contradicts Lemma 3.4 since at $t_r(u)$ it holds that u is *valid* and not *tight*.

We now prove the second part. Since u must be *tight* at $t(u)$ there exists $u' \in V$ such that (u, u') is *tight* at $t(u)$. Note that u' must be *valid* from $t(u)$ until the end of round r, since otherwise u will become *invalid* after $t(u')$ which is a contradiction. Thus, $u' \in G^r$ and $t(u') < t(u)$. Since u and u' remain *valid* from $t(u)$ until the end of round r, (u, u') is *tight* at the end of round r. ◀

▶ **Lemma 3.10.** *At the end of round r the following holds for all $u \in V$:*

1. *If $u \in B^r$ then u has a* tight *path of vertices in B^r to v.*
2. *If $u \in G^r$ then u has a* tight *path of vertices in G^r to a vertex u' with $f^r(u') = f^0(u')$.*

Proof. We begin by proving the first claim. Every vertex $u \in B^r$, $u \neq v$ joins L because of some edge (u, u') which is *invalid* after we update u'. This edge must be *tight* at the end of the round. So each vertex u has a *tight* edge to a vertex u' which was updated before u during round r. This implies the first part.

We now prove the second claim. If $f^r(u) = f^0(u)$ then we are done. Otherwise, assume $f^r(u) > f^0(u)$. We continue the proof by induction on $t(u)$. Base case, $t(u)$ is minimal (i.e., u was updated first). By Lemma 3.9, u has a *tight* edge (u, u') with $u' \in G^r$ such that $f^r(u') = f^0(u')$ (since $t(u)$ is minimal) and we are done. Assume that the claim follows for all vertices u' with $t(u') < t(u)$. By Lemma 3.9, u has a *tight* edge (u, u') with $u' \in G^r$ such that either $f^r(u') = f^0(u')$ or $t(u') < t(u)$. If $f^r(u') = f^0(u')$ then we are done. Otherwise, $t(u') < t(u)$ and therefore by the induction hypothesis, u' has a *tight* path to some vertex u'' with $f^r(u'') = f^0(u'')$. ◀

3.2 Complexity

Recall that $|V| = n, |E| = m$ and W is the maximal absolute value weight.

▶ **Theorem 3.11.** *The running time of compute–energy is $O\big(\min\big(mnW, mn \cdot 2^{n/2} \log W\big)\big)$.*

The $O(mnW)$ bound follows immediately since each vertex u is updated at most $O(|V| \cdot W)$ times and each such update takes $O(|in(u)| + |out(u)|)$ time, where $in(u)$ and $out(u)$ are the sets of ingoing and outgoing edges of u, respectively. To prove the latter bound we must have a better understanding of the relation between B^r and G^r. In the rest of this section we prove the $O\left(mn \cdot 2^{n/2} \log W\right)$ bound.

For this we define a potential function that maps rounds (as defined in Section 3.1) into integers. The *good anchor* of round r is defined as the set $GA^r = V \setminus \bigcup_{i=1}^{r} B^i$, i.e., vertices whose potential was not changed yet ($f^r(u) = f^0(u)$). Following each round the *good anchor* can only lose vertices. The *bad anchor* of round r is defined as the set $BA^r = \{u \in B^r \mid \exists \textit{tight}$ path of vertices in $V_0 \cap B^r$ from u to $v\}$, i.e., BA^r contains v and all vertices in $V_0 \cap B^r$ that have a *tight* path of vertices in $V_0 \cap B^r$ to v. Note that if $GA^r = \emptyset$, then no vertex is winning for player 0: To see this, note that in this case $f(u) > 0$ for all $u \in V$. Therefore, the potential $f'(v) \equiv f(v) - 1$, for all $v \in V$ is *feasible*, a contradiction (since f is the *solution*).

We partition B^r and G^r into layers BL_i^r, GL_i^r, $i = 0, 1 \ldots$, respectively, see Figure 4. The layer BL_i^r/GL_i^r is called the i'th layer of B^r/G^r, respectively. The 0'th layers are the anchors, i.e., $BL_0^r = BA^r, GL_0^r = GA^r$. The layers are defined inductively as follows.

$$\begin{aligned} BL_i^r &= \{u \in B^r \cap V_p \mid \ u \text{ has a tight path of vertices in } u \in B^r \cap V_p \text{ to } BL_{i-1}^r\} \\ GL_i^r &= \{u \in G^r \cap V_{1-p} \mid \ u \text{ has a tight path of vertices in } u \in G^r \cap V_{1-p} \text{ to } GL_{i-1}^r\}. \end{aligned} \tag{1}$$

where $p = i \ (mod\ 2)$.

The following lemmas prove that only the first layers have tight edges to the anchors.

▶ **Lemma 3.12.** *At the end of round r, if $(u, u') \in E$ is a tight edge s.t $u \in G^r \setminus GA^r$ and $u' \in GA^r$, then $u \in GL_1^r$.*

Proof. It suffices to show that $u \in V_0$. By contradiction, assume that $u \in V_1$. Since $u \notin GA^r$, u was in $B^{r'}$ at some round $r' < r$. Let $\textsc{Update}(z)$ be the update that added u into L in round r' and let f_1 be f right before this $\textsc{Update}(z)$. Clearly, $z \neq u'$ (since $u' \in GA^r$). Since u is was *valid* before $\textsc{Update}(z)$ in round r', and $u \in V_1$, we have that $f_1(u) + w(u, u') \geq f_1(u')$. Therefore, since $f(u')$ did not change until the end of round r and u must have been updated following the update of z and before the end of round r, we have that $f(u) + w(u, u') > f_1(u) + w(u, u') \geq f_1(u') = f(u')$ at the end of round r, a contradiction to the assumption that (u, u') is *tight* at the end of round r. ◀

▶ **Lemma 3.13.** *At the end of round r, if $(u, u') \in E$ is a tight edge s.t $u \in B^r \setminus BA^r$ and $u' \in BA^r$, then $u \in BL_1^r$.*

Proof. Immediate from the definition of BA^r. ◀

The following lemma, which follows immediately from Lemma 3.10, shows that every vertex belongs either to an *anchor* or to some layer of B^r or G^r.

▶ **Lemma 3.14.** *For any round r,* $B^r = \bigcup_{i=0}^{n} BL_i^r, \ G^r = \bigcup_{i=0}^{n} GL_i^r$.

We associate with B^r and G^r binary numbers b^r and g^r, respectively of length $n + 1$ defined as follows. Let k be maximal such that $|BL_k^r| > 0$. Then, b^r is:

$$b^r = \begin{cases} \underbrace{1.....1}_{|BL_1^r|}\underbrace{0.....0}_{|BL_2^r|}.....\underbrace{1.....1}_{|BL_k^r|}\ \underbrace{10..........0}_{n+1-|\bigcup_i BL_i^r|} & \text{if } k \text{ is odd} \\ \underbrace{1.....1}_{|BL_1^r|}\underbrace{0.....0}_{|BL_2^r|}.....\underbrace{0.....0}_{|BL_k^r|}\ \underbrace{10..........0}_{n+1-|\bigcup_i BL_i^r|} & \text{if } k \text{ is even.} \end{cases}$$

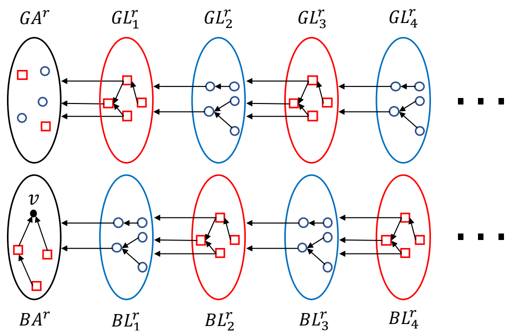

Figure 4 The layer graph of round r. Red vertices refer to V_0 and blue vertices refer to V_1. All drawn edges are *tight* at the end of round r. By Definition (1), each layer is either contained in V_0 or in V_1.

That is, for an odd layer we add a sequence of 1's whose length is the size of the layer. Similarly, for even layers we add sequences of 0's. At the end we pad the number with a single 1 followed by zeros. The number g^r is defined similarly with respect to the layers of G^r. Finally, the potential ϕ^r of round r is defined as $\phi^r = b^r + g^r$. Clearly $\phi^r \leq 2 \cdot 2^{n+1}$. In Lemma 3.18 we prove that for every round r, under certain conditions (that can be violated in at most $|V|^2$ rounds), $\phi^{r+1} \geq \phi^r + 2^{n/2}$, yielding the desired runtime.

The following lemmas consider Update-Energy at the end of round r.

▶ **Lemma 3.15.** *For every r, $GA^{r+1} \subseteq GA^r$ and if $GA^{r+1} = GA^r$, then $BA^{r+1} \subseteq BA^r$.*

Proof. Since f only grows the first claim follows directly from the definition of the algorithm. We prove the second claim by contradiction. Assume $GA^{r+1} = GA^r$ and let $u \in BA^{r+1} \setminus BA^r$. By definition of *bad anchor* $u \in V_0$ and at the end of round $r+1$ there exists a *tight* path $p = v_0 v_1 ... v_k$ from $u = v_0$ to $v = v_k$ such that $v_i \in V_0 \cap B^{r+1}$ for all $i < k$. Let j be maximal such that $v_j \in BA^{r+1} \setminus BA^r$ (therefore $v_{j+1} \in BA^{r+1} \cap BA^r$, j is well defined since $u \in BA^{r+1} \setminus BA^r$). If $v_j \in B^r$ then (v_j, v_{j+1}) was tight also at the end of round r (since both $v_j, v_{j+1} \in B^r \cap B^{r+1}$) and thus $v_j \in BA^r$, a contradiction. So we have that $v_j \in G^r$. Therefore, at the beginning of round r it must hold that $f(v_j) + w(v_j, v_{j+1}) > f(v_{j+1})$ (i.e., v_j is not *tight* at the beginning of round r). By Lemma 3.3, $v_j \in GA^r$, a contradiction to the assumption that $GA^r = GA^{r+1}$. ◀

The following lemma is similar to Lemma 3.7. Its proof is given in the full version of the paper.

▶ **Lemma 3.16.** *For every r, $B^{r+1} \neq B^r$.*

▶ **Lemma 3.17.** *Suppose that $GA^{r+1} = GA^r$ and $BA^{r+1} = BA^r$.*

1. *Let i be the smallest such that $GL_i^{r+1} \neq GL_i^r$. Then, if i is odd then $GL_i^r \subset GL_i^{r+1}$, and if i is even then $GL_i^{r+1} \subset GL_i^r$.*
2. *Let i be the smallest such that $BL_i^{r+1} \neq BL_i^r$. Then, if i is odd then $BL_i^r \subset BL_i^{r+1}$, and if i is even then $BL_i^{r+1} \subset BL_i^r$.*

Proof. We prove only the first claim as the latter is similar. We divide the proof into cases according to the parity of i.

i is *odd*: We show that $GL_i^r \subset GL_i^{r+1}$. By contradiction, assume that $\exists u \in GL_i^r \setminus GL_i^{r+1}$. By Definition (1), $u \in V_0$ and at the end of round r there exists a tight path $p = u_0, u_1, ... u_k$ from $u_0 = u \in GL_i^r$ to $u_k \in GA^r = GL_0^r$ that traverses the "good" layers in non-increasing order. Let j be the maximal such that $u_j \in GL_i^r \setminus GL_i^{r+1}$. Thus, either $u_{j+1} \in GL_{i-1}^r$ or $u_{j+1} \in GL_i^r \cap GL_i^{r+1}$. Note that in both cases $u_{j+1} \in G^{r+1}$ and therefore we have that also $u_j \in G^{r+1}$ (since (u_j, u_{j+1}) was tight at the end of round r and remains tight during round $r+1$). Assume $u_{j+1} \in GL_{i-1}^r$. Since $GL_{i-1}^r = GL_{i-1}^{r+1}$ and since (u_j, u_{j+1}) is *tight* at the end of round $r+1$, we have that $u_j \in GL_\ell^{r+1}$ for some $\ell \leq i$, a contradiction (since $u_j \notin GL_i^{r+1}$ and because lower layers are equal in both rounds by our assumption). Assume now that $u_{j+1} \in GL_i^r \cap GL_i^{r+1}$. We get a contradiction since $u_j \in GL_\ell^{r+1}$ for some $\ell \leq i$.

i is *even*: We that show $GL_i^{r+1} \subset GL_i^r$. By contradiction, assume that $\exists u \in GL_i^{r+1} \setminus GL_i^r$. By Definition (1), $u \in V_1$ and at the end of round $r+1$ there exists a tight path $p = u_0, u_1, ...u_k$ from $u_0 = u \in GL_i^{r+1}$ to $u_k \in GA^{r+1} = GL_0^{r+1}$ that traverses the "good" layers in non-increasing order. Assume $u \in B^r$. By Lemma 3.6, at the end of round r, all of u's *tight* edges are directed to B^r. Let ℓ be minimal such that $u_\ell \in V_0$. Hence, by Lemma 3.6 $u_\ell \in B^r$. Therefore $u_\ell \in GL_m^{r+1}$ for some $m < i$ (since $u_\ell \in V_0$ and $GL_i^{r+1} \subset V_1$), this contradicts our assumption $GL_m^{r+1} = GL_m^r$. Thus, $u \in G^r$.

Let j be maximal such that $u_j \in GL_i^{r+1} \setminus GL_i^r$. Hence, either $u_{j+1} \in GL_{i-1}^r$ or $u_{j+1} \in GL_i^r \cap GL_i^{r+1}$. In both cases $u_j, u_{j+1} \in G^r \cap G^{r+1}$ and thus (u_j, u_{j+1}) is *tight* in both rounds. Therefore $u_j \in GL_\ell^r$ for some $\ell \leq i$. Note that $\ell > i-1$ since otherwise $GL_\ell^r \neq GL_\ell^{r+1}$ which contradicts our assumption. Therefore $\ell = i$ and this contradict the assumption $u_j \in GL_i^{r+1} \setminus GL_i^r$. ◀

Note that if the conditions of Lemma 3.17 are satisfied, then by Lemma 3.17 and by the definition of b^r and g^r we have that $b^{r+1} \geq b^r$ and $g^{r+1} \geq g^r$.

▶ **Lemma 3.18.** *For every r, If $GA^{r+1} = GA^r$ and $BA^{r+1} = BA^r$, then $\phi^{r+1} \geq \phi^r + 2^{(n+k)/2}$, where $k = |GA^r| + |BA^r|$.*

Proof. Assume $|B^r \setminus BA^r| \geq |G^r \setminus GA^r|$, so $|G^r \setminus GA^r| \leq (n-k)/2$ and therefore g^r contains at least $(n+k)/2$ "padding bits". Hence, by Lemma 3.17 we have that $g^{r+1} \geq g^r + 2^{(n+k)/2}$ and $b^{r+1} \geq b^r$. Thus, $\phi^{r+1} \geq \phi^r + 2^{(n+k)/2}$ and we are done.

The case $|B^r \setminus BA^r| < |G^r \setminus GA^r|$ is identical. ◀

We are now ready to present the proof of our main result.

Proof of Theorem 3.11. By Lemma 3.18, there can be at most $2^{(n-k)/2}$ consecutive rounds satisfying $GA^{r+1} = GA^r$ and $BA^{r+1} = BA^r$, where $k = |GA^r| + |BA^r|$. Thus, by Lemma 3.15 we get that the following bounds the number of rounds during UPDATE-ENERGY

$$\sum_{i_1=1}^{n} \sum_{i_1=1}^{n-i_1} 2^{(n-(i_1+i_2))/2} = O(2^{n/2}) ,$$

where i_1 and i_2 represent $|GA^r|$ and $|BA^r|$, respectively. Hence, since a round takes $O(m)$ time and COMPUTE-ENERGY calls UPDATE-ENERGY at most $n \cdot \log W$ times, we get that COMPUTE-ENERGY terminates in $O\left(mn \cdot 2^{n/2} \log W\right)$ time. ◀

4 Concluding remarks and open problems

We presented an $O\left(\min\left(mnW, mn2^{n/2} \log W\right)\right)$-time algorithm for solving EGs and MPGs. The algorithm is always at least as fast as the algorithm of Brim et al. [5], and is the fastest known deterministic algorithm when $W \geq n2^{n/2}$. (As mentioned the $\log W$ factor can be replaced by a $poly(n)$ factor.) The exponential running time of the new algorithm is still far from what we would wish for. We hope, however, that the techniques used in our paper may lead to further improvements. The ultimate goal of using scaling is to obtain an algorithm whose running time is $O(poly(n)\log W)$. We are, of course, still extremely far from achieving this goal.

Many open problems remain: (1) Improve the pseudopolynomial running time to $O(mnf(W))$, where $f(W) = o(W)$. A more ambitious open problem is: (2) Obtain a deterministic sub-exponential time algorithm for solving EGs and MPGs, matching the running time of the fastest randomized algorithms. Even more ambitious open problem is: (3) obtain a quasipolynomial time algorithm for EGs and MPGs, matching the running time of the fastest algorithm for solving PGs. The most ambitious problem, of course, is: (4) obtain a polynomial time algorithm for PGs, EGs and MPGs.

References

1 Henrik Björklund and Sergei G. Vorobyov. Combinatorial structure and randomized subexponential algorithms for infinite games. *Theor. Comput. Sci.*, 349(3):347–360, 2005. doi:10.1016/j.tcs.2005.07.041.

2 Henrik Björklund and Sergei G. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155(2):210–229, 2007.

3 Mikołaj Bojanczyk and Wojciech Czerwiński. An Automata Toolbox, February 2018. URL: https://www.mimuw.edu.pl/~bojan/papers/toolbox.pdf.

4 Patricia Bouyer, Uli Fahrenberg, Kim G Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 33–47. Springer, 2008.

5 Lubos Brim, Jakub Chaloupka, Laurent Doyen, Raffaella Gentilini, and Jean-François Raskin. Faster algorithms for mean-payoff games. *Formal methods in system design*, 38(2):97–118, 2011.

6 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proc. of 49th STOC*, pages 252–263, 2017. doi:10.1145/3055399.3055409.

7 Arindam Chakrabarti, Luca De Alfaro, Thomas A Henzinger, and Mariëlle Stoelinga. Resource interfaces. In *International Workshop on Embedded Software*, pages 117–133. Springer, 2003.

8 Krishnendu Chatterjee, Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Polynomial-time algorithms for energy games with special weight structures. *Algorithmica*, 70(3):457–492, 2014.

9 A. Ehrenfeucht and J. Mycielski. Positional Strategies for Mean Payoff Games. *International Journal of Game Theory*, 8:109–113, 1979.

10 John Fearnley, Sanjay Jain, Sven Schewe, Frank Stephan, and Dominik Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software*, pages 112–121. ACM, 2017.

11 Nathanaël Fijalkow, Paweł Gawrychowski, and Pierre Ohlmann. The complexity of mean payoff games using universal graphs. *CoRR*, abs/1812.07072, 2018. arXiv:1812.07072.

12 Harold N. Gabow and Robert Endre Tarjan. Faster Scaling Algorithms for General Graph-Matching Problems. *J. ACM*, 38(4):815–853, 1991. doi:10.1145/115234.115366.

13 Hugo Gimbert and Rasmus Ibsen-Jensen. A short proof of correctness of the quasi-polynomial time algorithm for parity games. *CoRR*, abs/1702.01953, 2017. arXiv:1702.01953.

14 Andrew V. Goldberg. Scaling Algorithms for the Shortest Paths Problem. *SIAM J. Comput.*, 24(3):494–504, 1995. doi:10.1137/S0097539792231179.

15 Andrew V. Goldberg and Satish Rao. Beyond the Flow Decomposition Barrier. *J. ACM*, 45(5):783–797, 1998. doi:10.1145/290179.290181.

16 Nir Halman. Simple stochastic games, parity games, mean payoff games and discounted payoff games are all LP-type problems. *Algorithmica*, 49(1):37–50, 2007.
17 Thomas Dueholm Hansen and Uri Zwick. An Improved Version of the Random-Facet Pivoting Rule for the Simplex Algorithm. In *Proc. of 47th STOC*, pages 209–218, 2015. doi:10.1145/2746539.2746557.
18 Marcin Jurdziński and Ranko Lazić. Succinct progress measures for solving parity games. In *Proc. of 32nd LICS*, pages 1–9, 2017. doi:10.1109/LICS.2017.8005092.
19 Gil Kalai. A Subexponential Randomized Simplex Algorithm (Extended Abstract). In *Proc. of 24th STOC*, pages 475–482, 1992. doi:10.1145/129712.129759.
20 Gil Kalai. Linear programming, the simplex algorithm and simple polytopes. *Math. Program.*, 79:217–233, 1997. doi:10.1007/BF02614318.
21 Karoliina Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 639–648. ACM, 2018.
22 Yuri M Lifshits and Dmitri S Pavlov. Potential theory for mean payoff games. *Journal of Mathematical Sciences*, 145(3):4967–4974, 2007.
23 Walter Ludwig. A Subexponential Randomized Algorithm for the Simple Stochastic Game Problem. *Inf. Comput.*, 117(1):151–155, 1995. doi:10.1006/inco.1995.1035.
24 Jiří Matoušek, Micha Sharir, and Emo Welzl. A Subexponential Bound for Linear Programming. *Algorithmica*, 16(4/5):498–516, 1996. doi:10.1007/BF01940877.
25 Tibor Szabó and Emo Welzl. Unique Sink Orientations of Cubes. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 547–555, 2001. doi:10.1109/SFCS.2001.959931.
26 Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.