A Combinatorial Strongly Subexponential Strategy Improvement Algorithm for Mean Payoff Games

Henrik Björklund, Sven Sandberg, Sergei Vorobyov

Uppsala University, Information Technology Department, Box 337, 751 05 Uppsala, Sweden

Abstract

We suggest the first strongly subexponential and purely combinatorial algorithm for solving the mean payoff games problem. It is based on iteratively improving the longest shortest distances to a sink in a possibly cyclic directed graph. We identify a new "controlled" version of the shortest paths problem. By selecting exactly one outgoing edge in each of the controlled vertices we want to make the shortest distances from all vertices to the unique sink as long as possible. Under reasonable assumptions the problem belongs to the complexity class NP∩coNP. Mean payoff games are easily reducible to this problem. We suggest an algorithm for computing longest shortest paths. Player Max selects a strategy (one edge in each controlled vertex) and player Min responds by evaluating shortest paths to the sink in the remaining graph. Then Max locally changes choices in controlled vertices looking at attractive switches that seem to increase shortest paths lengths (under the current evaluation). We show that this is a monotonic strategy improvement, and every locally optimal strategy is globally optimal. This allows us to construct a randomized algorithm of complexity $\min(poly \cdot W,\ 2^{O(\sqrt{n \log n})})$, which is simultaneously pseudopolynomial (W is the maximal absolute edge weight) and subexponential in the number of vertices n. All previous algorithms for mean payoff games were either exponential or pseudopolynomial (which is purely exponential for exponentially large edge weights).

* Supported by Swedish Research Council grants "*Infinite Games: Algorithms and Complexity*", "*Interior-Point Methods for Infinite Games*", and by an Institutional grant from the Swedish Foundation for International Cooperation in Research and Higher Education.

**Extended abstract in MFCS'2004, ©Springer-Verlag.

1 Introduction

Infinite games on finite graphs play a fundamental role in model checking, automata theory, logic, and complexity theory. We consider the problem of solving *mean payoff games* (MPGs) [9,21,20,10,26], also known as *cyclic games* [14,22]. In these games, two players take turns moving a pebble along edges of a directed edge-weighted graph. Player MAX wants to maximize and player MIN to minimize (in the limit) the average edge weight of the infinite path thus formed. Mean payoff games are determined, and every vertex has a *value*, which each player can secure by a uniform positional strategy. Determining whether the value is above (below) a certain threshold belongs to the complexity class NP∩coNP. The well-known parity games, also in NP∩coNP, polynomial time equivalent to model-checking for the μ-calculus [12,11], are polynomial time reducible to MPGs. Other well-known games with NP∩coNP decision problems, to which MPGs reduce, are *simple stochastic* [6] and *discounted payoff* [23,26] games. At present, despite substantial efforts, there are no known polynomial time algorithms for the games mentioned.

All previous algorithms for mean payoff games are either pseudopolynomial or exponential. These include a potential transformation method by Gurvich, Karzanov, and Khachiyan [14] (see also Pisaruk [22]), and a dynamic programming algorithm solving k-step games for big enough k by Zwick and Paterson [26]. Both algorithms are *pseudopolynomial* of complexity $O(poly(n) \cdot W)$, where n is the number of vertices and W is the maximal absolute edge weight. For both algorithms there are known game instances on which they show a worst-case $\Omega(poly(n) \cdot W)$ behavior, where W may be exponential in n. Reduction to simple stochastic games [26] and application of the algorithm from [17] gives subexponential complexity (in the number of vertices n) only if the game graph has bounded outdegree. The subexponential algorithms we suggested for simple stochastic games of arbitrary outdegree in [3,4] make $2^{O(\sqrt{n \log n})}$ iterations, but when reducing from mean payoff games, the weights may not allow each iteration (requiring solving a linear program) to run in strongly polynomial time, independent of the weights. This drawback is overcome with the new techniques presented here, which avoid the detour over simple stochastic games altogether.

We suggest a *strongly subexponential strategy improvement* algorithm, which starts with some strategy of the maximizing player[1] MAX and iteratively "improves" it with respect to some strategy evaluation function. Iterative strategy improvement algorithms are known for the related simple stochas-

[1] The games are symmetric, and the algorithm can also be applied to optimize for the minimizing player. This is an advantage when the minimizing player has fewer choices.

tic [15,7], discounted payoff [23], and parity games [24,2]. Until the present paper, a direct combinatorial iterative strategy improvement for mean payoff games appeared to be elusive. Reductions to discounted payoff games and simple stochastic games (with known iterative strategy improvement) lead to numerically unstable computations with long rationals and solving linear programs. The algorithms suggested in this paper are free of these drawbacks. Our method is discrete, requires only addition and comparison of integers in the same order of magnitude as occurring in the input. In a combinatorial model of computation, the subexponential running time bound is independent of the edge weights. There is also a simple reduction from parity games to MPGs, and thus our method can be used to solve parity games. Contrasted to the strategy improvement algorithms of [24,2], the new method is conceptually much simpler, more efficient, and easier to implement.

We present a simple and discrete randomized subexponential strategy improvement scheme for MPGs, and show that for any integer p, the set of vertices from which MAX can secure a value $> p$ can be found in time

$$\min(O(n^2 \cdot |E| \cdot W),\ 2^{O(\sqrt{n \log n})}),$$

where n is the number of vertices and W is the largest absolute edge weight. The first bound matches those from [14,26,22], while the second part is an improvement when, roughly, $n \log n < \log^2 W$.

The new strategy evaluation for MPGs may be used in several other iterative improvement algorithms, which are also applicable to parity and simple stochastic games [24,15,7]. These include random single switch, all profitable switches, and random multiple switches; see, e.g., [1]. They are simplex-type algorithms, very efficient in practice, but without currently known subexponential upper bounds, and no nontrivial lower bounds.

Outline. Section 2 defines mean payoff games and introduces the associated computational problems. Section 3 describes the longest-shortest paths problem and its relation to mean payoff games. In addition, it gives an intuitive explanation of our algorithm and the particular randomization scheme that achieves subexponential complexity. Section 4 describes the algorithm in detail and Section 5 proves the two main theorems guaranteeing correctness. In Section 6 we explain how to improve the cost per iteration, while detailed complexity analysis is given in Section 7. Possible variants of the algorithm are discussed in Section 8. Section 9 shows that the longest-shortest path problem is in NP∩coNP. In Section 10 an example graph family is given for which the wrong choice of iterative improvement policy leads to an exponential number of iterations. Finally, Section 11 discusses the application of our algorithm to solving parity games.

2 Preliminaries

2.1 Mean Payoff Games

A *mean payoff game* (MPG) [21,20,10,14,26] is played by two adversaries, Max and Min, on a finite, directed, edge-weighted, leafless graph $G = (V = V_{\text{Max}} \uplus V_{\text{Min}}, E, w)$, where $w : E \to \mathbb{Z}$ is the weight function. The players move a pebble along edges of the graph, starting in some designated initial vertex. If the current vertex belongs to V_{Max}, Max chooses the next move, otherwise Min does. The duration of the game is infinite. The resulting infinite sequence of edges is called a *play*. The *value* of a play $e_1e_2e_3\dots$ is defined as $\liminf_{k\to\infty} 1/k \cdot \sum_{i=1}^{k} w(e_i)$. The goal of Max is to maximize the value of the play, while Min tries to minimize it. In the decision version, the game also has a threshold value p. We say that Max *wins* a play if its value is $> p$, while Min wins otherwise. Until Section 7.2 we assume such thresholds to be integral.

A *positional strategy* for Max is a function $\sigma : V_{\text{Max}} \to V$ such that $(v, \sigma(v)) \in E$ for all $v \in V_{\text{Max}}$. Positional strategies for Min are defined symmetrically. Every mean payoff game has a value and is *memoryless determined*, which means that for every vertex v there is a value $\nu(v)$ and positional strategies of Max and Min that secure them payoffs $\geq \nu(v)$ and $\leq \nu(v)$, respectively, when a play starts in v, against any strategy of the adversary [21,20,10,14,22,5]. Moreover, both players have *uniform* positional strategies securing them optimal payoffs independently of the starting vertex. Accordingly, throughout the paper we restrict our attention to positional strategies only. Given a positional strategy σ for Max, define $G_\sigma = (V, E')$, where $E' = E \setminus \{(v,u) | v \in V_{\text{Max}} \text{ and } \sigma(v) \neq u\}$, i.e., G_σ results from G by deleting all edges leaving vertices in V_{Max} except those selected by σ. Note that if both players use positional strategies, the play will follow a (possibly empty) path to a simple loop, where it will stay forever. The value of the play is the average edge weight on this loop [10,14].

2.2 Algorithmic Problems for MPGs

We will address several computational problems for mean payoff games.

The Decision Problem. Given a distinguished start vertex and a threshold value p, can Max guarantee value $> p$?

p-Mean Partition. Given p, partition the vertices of an MPG G into subsets $G_{\leq p}$ and $G_{>p}$ such that Max can guarantee a value $> p$ starting from every vertex in $G_{>p}$, and Min can secure a value $\leq p$ starting from every vertex in $G_{\leq p}$.

Ergodic Partition. Compute the value of each vertex of the game. This gives a partition of the vertices into subsets with the same value. Such a partition is called *ergodic* [14].

Our basic algorithm solves the *0-mean partition* problem, which subsumes the p-mean partition. Indeed, subtracting p from the weight of every edge makes the mean value of all loops (in particular, of optimal loops) smaller by p, and the problem reduces to 0-mean partitioning. The complexity remains the same for integer thresholds p, and changes slightly for rational ones; see Section 7. Clearly, the p-mean partitioning subsumes the decision problem. Section 7.2 extends the basic algorithm to solve the ergodic partition problem. Another problem to which our algorithm may be extended is finding optimal strategies in MPGs.

Our proofs rely on a finite variant of mean payoff games, where the play stops as soon as some vertex is revisited and the mean value on the resulting cycle determines the payoff. Thus, for a play $e_1e_2 \dots e_r e_{r+1} \dots e_s$, where $e_r \dots e_s$ is a loop, the value is $\sum_{i=r}^{s} w(e_i)/(s-r+1)$. Ehrenfeucht and Mycielski [10] proved the following (see also [5]).

Theorem 2.1 *The value of every vertex in the finite-duration version of mean payoff games equals its value in the infinite-duration version.* □

The next corollary will be used implicitly throughout the paper.

Proposition 2.2 *A positional strategy σ of* Max *gives value $> p$ in a vertex, if and only if all loops reachable from it in G_σ have average value $> p$.* □

In particular, Max can ensure value > 0 if and only if all reachable loops are positive. Since the partition threshold 0 has a special role in our exposition, we call the $G_{>0}$ partition the *winning set* of Max.

3 A High-Level Description of the Algorithm

We start by informally describing the essential ingredients of our algorithm.

3.1 The Longest-Shortest Paths Problem

The key step in computing 0-mean partitions can be explained by using a "controlled" version of the well-known *single source (target) shortest paths problem* on directed graphs. Suppose in a given digraph some set of *controlled* vertices is distinguished, and we can select *exactly one edge* leaving every controlled vertex, deleting all other edges from these vertices. Such a selection is called a *positional strategy.* We want to find a positional strategy that maximizes the shortest paths from all vertices to the distinguished sink (also avoiding negative cycles that make the sink unreachable and the distances $-\infty$). For a strategy σ denote by G_σ the graph obtained from G by deleting all edges from controlled vertices except those in σ. Formally, the problem is specified as follows.

The Longest-Shortest Paths Problem (LSP).
Given:

(1) a directed weighted graph G with unique sink t,
(2) some distinguished *controlled* vertices U of G, with $t \notin U$.

Find:

- a positional strategy σ such that in the graph G_σ the shortest simple path from every vertex to t is as long as possible (over all positional strategies).

If a negative weight loop is reachable from a vertex, the length of the shortest path is $-\infty$, which Max does not want. If only positive loops are reachable, and t is not, then the shortest path distance is $+\infty$.[2]

For our purposes it suffices to consider a version of the LSP problem with the following additional input data.

Additionally Given:

- some strategy σ_0, which guarantees that in the graph G_{σ_0} there are no cycles with nonpositive weights.

This additionally supplied strategy σ_0 guarantees that the longest shortest distance from every vertex to the sink t is not $-\infty$; it is not excluded that σ_0 or the optimal strategy will make some distances equal $+\infty$. We make sure that our algorithm never comes to a strategy that allows for nonpositive cycles. The simplifying additional input strategy is easy to provide in the reduction from MPGs, as we show below.

Note that for DAGs, the longest-shortest path problem can be solved in polynomial time using dynamic programming. Start by topologically sorting the

[2] The case of zero weight loops is inconsequential for the application to mean payoff games, and we only need to consider it when proving that the LSP problem is in NP∩coNP in Section 9.

vertices and proceed backwards from the sink (distance 0), using the known longest-shortest distances for the preceding vertices.

3.2 Relating the 0-Mean Partition and Longest-Shortest Paths Problems

The relation between computing 0-mean partitions and computing longest shortest paths is now easy to describe. To find such a partition in an MPG G, add a *retreat* vertex t to the game graph with a self-loop edge of weight 0, plus a 0-weight *retreat* edge from every vertex of Max to t. From now on, we assume G has undergone this transformation. Clearly, we have the following property.

Proposition 3.1 *Adding a retreat does not change the 0-mean partition of the game, except that t is added to the $G_{\leq 0}$ part.* □

This is because we do not create any new loops allowing Max to create positive cycles, or Min to create new nonpositive cycles. Max will prefer playing to t only if all other positional strategies lead to negative loops.

The key point is now as follows. Break the self-loop in t and consider the LSP problem for the resulting graph, with t being the unique sink. The set V_{Max} becomes the controlled vertices, and the initial strategy (the "additionally given" clause in the LSP definition above) selects t in every controlled vertex, guaranteeing that no vertex has distance $-\infty$.[3] We have the following equivalence:

Theorem 3.2 *The partition $G_{>0}$ consists exactly of those vertices for which the longest-shortest path distance to t is $+\infty$.* □

As early as in 1991 Leonid Khachiyan (private communication) considered the following variant of Longest-Shortest Paths.

Blocking Nonpositive Cycles. Given a directed edge-weighted leafless graph G, a vertex v, and a set of controlled vertices, where the controller has to choose exactly one outgoing edge, does he have a selection such that in the resulting graph (obtained after deleting all unselected edges from controlled vertices) there are no nonpositive weight cycles reachable from v? □

As an immediate consequence one has the following

[3] Actually, there may exist negative value loops consisting only of vertices from V_{Min}. Such loops are easy to identify and eliminate in a preprocessing step, using the Bellman–Ford algorithm. In the sequel we assume that this is already done.

Proposition 3.3 *1. The problems 0-Mean Partition in Mean Payoff Games*[4] *and Blocking Nonpositive Cycles are polynomial time equivalent.*

2. Blocking Nonpositive Cycles is in NP∩coNP.

3. Blocking Nonpositive Cycles is polynomial time reducible to Longest-Shortest Paths. □

Note that in Longest-Shortest Paths, except being interested in the $+\infty$ distances to the sink (which corresponds to positive loops in Blocking Nonpositive Cycles) we are additionally interested in computing *finite* distances. Our algorithm iteratively improves these finite distances (until hopefully improving them to $+\infty$).

To our knowledge, there are no other mentions of the Longest-Shortest Paths problem and its relation to mean payoff games in the literature.[5]

Actually, the evaluation of the shortest paths for a fixed positional strategy gives a useful quality measure on strategies that can be used in other iterative improvement schemes. We discuss some possibilities in Section 8.

3.3 The Algorithm

Our algorithm computes longest-shortest paths in the graph resulting from a mean payoff game (after adding the retreat vertex and edges, as explained above), by making iterative strategy improvements. Once a strategy is fixed, all shortest paths are easily computable, using the Bellman-Ford algorithm. Note that there are negative weight edges, so the Dijkstra algorithm does not apply. An improvement to the straightforward application of the BF-algorithm is described in Section 6. Comparing a current choice made by the strategy with alternative choices, a possible improvement can be decided locally as follows. If changing the choice in a controlled vertex to another successor seems to give a longer distance (seems attractive), we make this change. Such a change is called a *switch*.

We prove two crucial properties (Theorems 5.1 and 5.2, respectively):

(1) every such switch really increases the shortest distances (i.e., attractive is improving or profitable);

[4] Recall that this problem consists in finding the set of the game vertices from which the maximizing player can secure a *positive* mean value.
[5] The authors would appreciate any references.

(2) once none of the alternative possible choices is attractive, all possible positive-weight loops Max can enforce are found (i.e., stable is optimal). [6]

Although our subexponential algorithm proceeds by making just one attractive switch at a time, other algorithms making many switches simultaneously are also possible and fit into our framework. Such algorithms are discussed in Section 8.

Another interpretation of our algorithm is game-theoretic. Max makes choices in the controlled vertices, and the choices in all other vertices belong to Min. For every strategy of Max, Min responds with an optimal counterstrategy, computing the shortest paths from every vertex to the sink. After that, the algorithm improves Max's strategy by making an attractive switch, etc.

3.4 Randomization Scheme

The order in which attractive switches are made is crucial for the subexponential complexity bound; see Section 10 for an example of an exponentially long chain of switches. The space of all positional strategies of Max can be identified with the Cartesian product of sets of edges leaving the controlled vertices. Fixing any edge in this set and letting others vary determines a *facet* in this space.

Now the algorithm for computing the longest-shortest paths in G looks as follows, starting from some strategy σ assumed to guarantee for shortest distances $> -\infty$ in all vertices.

(1) Randomly and uniformly select some facet F of G not containing σ. Throw this facet away, and recursively find a best strategy σ^* on what remains. This corresponds to deleting an edge not selected by σ and finding the best strategy in the resulting subgame.
(2) If σ^* is optimal in G, stop (this is easily checked by testing whether there is an attractive switch from σ^* to F). The resulting strategy is globally optimal, providing for the longest-shortest distances.
(3) Otherwise, switch to F, set $G = F$, denote the resulting strategy by σ, and repeat from step 1.

This is the well-known randomization scheme for linear programming due to Matoušek, Sharir, and Welzl [18,19]. When applied to the LSP and MPG problems, it gives a subexponential $2^{O(\sqrt{n \log n})}$ expected running time bound

[6] The case when zero-weight loops are interpreted as good (winning) for Max is considered in Section 9.

[18,4]. An essential condition for the analysis to work is as follows. The strategy evaluation we use satisfies the property that facets are partially ordered by the values of their best strategies. After finding the optimum on one facet, the algorithm will never visit a facet with an optimum that is not strictly better in the partial order. It follows that the so-called *hidden dimension* decreases randomly, and the subexponential analysis of [18,19] applies. Another possibility would be to use the slightly more complicated randomization scheme of Kalai [16], as we did in [2] for parity games, which leads to the same subexponential complexity bound.

4 Retreats, Admissible Strategies, and Strategy Measure

As explained above, we modify an MPG by allowing Max to "surrender" in every vertex. Add a retreat vertex t of Min with a self-loop of weight 0 and a retreat edge of weight 0 from every vertex of Max to t. Clearly, Max secures a value > 0 from a vertex in the original MPG iff the same strategy does it in the modified game. Assume from now on that the retreat has been added to G. Intuitively, the "add retreats" transformation is useful because Max can start by a strategy that chooses the retreat edge in every vertex, thus "losing only 0". We call strategies "losing at most 0" *admissible.*

Definition 4.1 *A strategy σ of* Max *in G is* admissible *if all loops in G_σ are positive, except the loop over the retreat vertex t.* □

Our algorithm iterates only through admissible strategies of Max. This guarantees that the only losing (for Max) loop in G_σ is the one over t.

4.1 Measuring the Quality of Strategies

We now define a measure that evaluates the "quality" of an admissible strategy. It can be computed in strongly polynomial time, as shown in Section 6.

Given an admissible strategy σ, the best Min can hope to do is to reach the 0-mean self-loop over t. Any other reachable loop will be positive, by the definition of an admissible strategy. The shortest path from every vertex v to t is well-defined, because there are no nonpositive cycles in G_σ (except over t). Therefore, we define the value of a strategy in a vertex as follows.

Definition 4.2 *For an admissible strategy σ of* Max*, the value* $\mathrm{val}_\sigma(v)$ *of vertex v is defined as the shortest path distance from v to t in G_σ, or $+\infty$ if t is not reachable.* □

It follows that for admissible strategies finite values may only result from shortest paths leading to the sink (retreat) t.

Note that there is a positional counterstrategy of MIN that guarantees the shortest paths are taken in each vertex, namely the strategy defined by the shortest path forest; see, e.g., [8]. The relative quality of two admissible strategies is defined componentwise.

Definition 4.3 *Let σ and σ' be two admissible strategies. Say that σ is* better than *σ', formally denoted $\sigma > \sigma'$, if $\mathrm{val}_\sigma(v) \geq \mathrm{val}_{\sigma'}(v)$ for all vertices $v \in V$, with strict inequality for at least one vertex. Say that $\sigma \geq \sigma'$, if $\sigma > \sigma'$ or they have equal values in all vertices.* □

The notation below will be useful for describing switches.

Notation 4.4 If σ is a strategy of MAX, $x \in V_{\text{MAX}}$, and $(x, y) \in E$, then the *switch in x to y* changes σ to the new strategy $\sigma[x \mapsto y]$, defined as

$$\sigma[x \mapsto y](v) \overset{\text{def}}{=} \begin{cases} y, & \text{if } v = x; \\ \sigma(v), & \text{otherwise.} \end{cases} \qquad \square$$

The following definition makes a distinction between switches that improve the strategy value, and switches that merely look like they do. Later (Corollary 5.5) we will prove that the two notions are equivalent.

Definition 4.5 *Given an admissible strategy σ, a switch $\sigma[v \mapsto u]$ is:*

(1) attractive*, if $w(v, u) + \mathrm{val}_\sigma(u) > \mathrm{val}_\sigma(v)$;*
(2) profitable*, if $\sigma[v \mapsto u]$ is admissible and $\sigma[v \mapsto u] > \sigma$.* □

4.2 Requirements for the Measure

The algorithm relies on the following properties.

(1) If σ is an admissible strategy, and there is no better admissible strategy, then σ is winning from all vertices in MAX's winning set $G_{>0}$. This is evident from the definitions.
(2) Every combination of attractive switches is profitable (Theorem 5.1).
(3) If an admissible strategy has no attractive switches, then there is no better admissible strategy (Theorem 5.2).

Property (2) guarantees monotonicity, termination, and a pseudopolynomial upper bound. Another advantage of (2) is as follows. To find profitable switches, we only need to test attractivity, which is efficient as soon as the measure has been computed. Testing profitability would otherwise require recomputing the measure for every possible switch.

5 Correctness of the Measure

In this section we state the two major theorems, guaranteeing that every step is improving and that the final strategy is the best, respectively. Afterwards, we give two corollaries that are not strictly necessary for the algorithm to work, but which with little extra effort give an additional insight into the problem.

5.1 Attractiveness Implies Profitability

Our first theorem states that any combination of attractive switches is profitable. This means that we never have to actually evaluate other strategies before selecting the next iterate. Instead we can let the improvement scheme be guided by attractiveness. Monotonicity, guaranteed by Theorem 5.1, implies that every sequence of attractive switches will always terminate. Recall that an admissible strategy does not permit any negative *or* zero value loops.

Theorem 5.1 *If σ is an admissible strategy then any strategy obtained by one or more attractive switches is admissible and better. Formally, if the switches in s_i to t_i are attractive for $1 \leq i \leq r$ and $\sigma' \overset{def}{=} \sigma[s_1 \mapsto t_1][s_2 \mapsto t_2] \cdots [s_r \mapsto t_r]$, then σ' is admissible and $\sigma' > \sigma$.*

Proof. It is enough to prove that all loops in $G_{\sigma'}$ are positive and the value does not decrease in any vertex. Then it follows that σ' is admissible and the value in every s_i increases strictly, hence $\sigma' > \sigma$, because

$$\begin{aligned} \mathrm{val}_{\sigma'}(s_i) &= w(s_i, t_i) + \mathrm{val}_{\sigma'}(t_i) && \text{[by definition]} \\ &\geq w(s_i, t_i) + \mathrm{val}_{\sigma}(t_i) && \text{[t_i's value does not decrease (to be shown)]} \\ &> \mathrm{val}_{\sigma}(s_i). && \text{[the switch in s_i to t_i is attractive]} \end{aligned}$$

First, we prove that every loop present in $G_{\sigma'}$, but not in G_σ, is positive. Second, we prove that for every path from some vertex v to t present in $G_{\sigma'}$, but not in G_σ, there is a shorter path in G_σ.

1. New loops are positive. Consider an arbitrary loop present in $G_{\sigma'}$, but not in G_σ. Some of the switching vertices s_i must be on the loop; denote them by $\{v_0, \ldots, v_{p-1}\} \subseteq \{s_1, \ldots, s_r\}$, in cyclic order; see Figure 1.

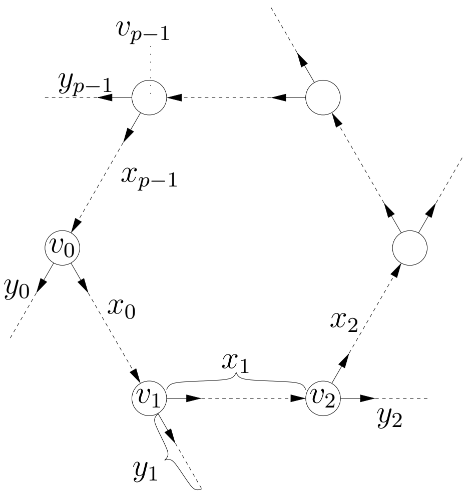

Figure 1. A loop in $G_{\sigma'}$ not in G_σ is depicted. Strategy σ breaks the cycle in vertices $v_0, \dots, v_{p-1}$, and instead follows the dashed paths to t, each of total edge weight y_i. The weight of the segments between two adjacent v_i's is x_i.

To simplify notation, let $v_p \overset{\text{def}}{=} v_0$. Since the switch in v_i ($0 \le i \le p-1$) is attractive, $\text{val}_\sigma(v_i)$ is finite, and there is a path in G_σ from v_i to t. Denote by x_i the sum of weights on the shortest path from v_i to v_{i+1} under σ' and let $y_i = \text{val}_\sigma(v_i)$, i.e., y_i is the sum of weights on the path from v_i to t under σ. Moreover, $x_p \overset{\text{def}}{=} x_0$ and $y_p \overset{\text{def}}{=} y_0$. Note that

$$y_i = \text{val}_\sigma(v_i) < w(v_i, \sigma'(v_i)) + \text{val}_\sigma(\sigma'(v_i)) \le x_i + y_{i+1},$$

where the first inequality holds because the switch in v_i to $\sigma'(v_i)$ is attractive and the second because $\text{val}_\sigma(\sigma'(v_i))$ is the length of a shortest path from $\sigma'(v_i)$ to t and $x_i - w(v_i, \sigma'(v_i)) + y_{i+1}$ is the length of another path. Combining these p equalities for every i, we get

$$\begin{aligned} y_0 &< x_0 + y_1 \\ &< x_0 + x_1 + y_2 \\ &< x_0 + x_1 + x_2 + y_3 \\ &\;\vdots \\ &< x_0 + x_1 + \cdots + x_{p-1} + y_0. \end{aligned}$$

Therefore, $x_0 + \cdots + x_{p-1} > 0$, hence the loop is positive.

2. New paths to the sink are longer. Consider an arbitrary shortest path from a vertex v to t, present in $G_{\sigma'}$ but not in G_σ. It must contain one or more switching vertices, say $\{v_1, \dots, v_p\} \subseteq \{s_1, \dots, s_r\}$, in order indicated; see Figure 2.

Under strategy σ', denote by x_0 the sum of edge weights on the path from v to

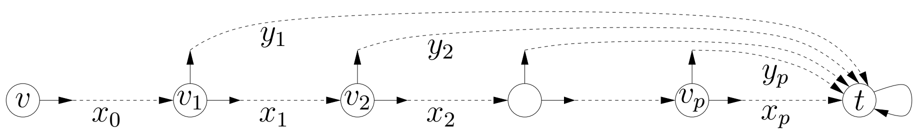

Figure 2. The path going right occurs in $G_{\sigma'}$ but not in G_σ. Strategy σ breaks the path in vertices $v_1, \ldots, v_p$ by going up and following the curved paths to t, each of total edge weight y_i. The edge weight of the segments between two adjacent v_i's is x_i.

v_1, by x_p the sum of weights on the path from v_p to t, and by x_i $(1 \leq i \leq p-1)$ the sum of weights on the path from v_i to v_{i+1}. Let $y_i = \mathrm{val}_\sigma(v_i)$ (attractiveness of switches in v_i implies that y_i are finite and determined by paths to the sink). As above, note that, for $1 \leq i \leq p-1$ one has

$$y_i < w(v_i, \sigma'(v_i)) + \mathrm{val}_\sigma(\sigma'(v_i)) \leq x_i + y_{i+1},$$

for the same reason as in the previous case, and if we let $y_{p+1} = 0$ it holds for $i = p$ as well. Combining these p inequalities we obtain

$$\begin{aligned} x_0 + y_1 &< x_0 + x_1 + y_2 \\ &< x_0 + x_1 + x_2 + y_3 \\ &< x_0 + x_1 + x_2 + x_3 + y_4 \\ &\vdots \\ &< x_0 + \cdots + x_p. \end{aligned}$$

Thus the path from v to t in G_σ taking value $x_0 + y_1$ is shorter than the new path in $G_{\sigma'}$. □

5.2 Stability Implies Optimality

Our second theorem shows that an admissible strategy with no attractive switches is at least as good as any other admissible strategy. This means that if we are only looking for the vertices where Max can enforce positive weight loops (when solving mean payoff games) we can stop as soon as we find an admissible stable strategy. It also follows that if there are no zero-weight loops in G, then any stable admissible strategy is optimal. Section 9 deals with the case where zero-weight loops are considered good for Max.

Theorem 5.2 *If σ is an admissible strategy with no attractive switches, then $\sigma \geq \sigma'$ for all admissible strategies σ'.*

Proof. The proof is in two steps: first we prove the special case when Min does not have any choices, and then we extend the result to general games.

1. One-player games. Assume Min does not have any choices, i.e., the out-degree of every vertex in V_{Min} is one. Let σ be an admissible strategy with no attractive switches. We claim that every admissible strategy σ' is no better than σ, i.e., $\sigma' \leq \sigma$.

1a. Finite values cannot become infinite. First, we prove that if $\text{val}_\sigma(v) < \infty$ then $\text{val}_{\sigma'}(v) < \infty$. Consider, toward a contradiction, an arbitrary loop formed in $G_{\sigma'}$, not formed in G_σ (recall that a positive loop is the only way to create an infinite value). There is at least one vertex on this loop where σ and σ' make different choices; assume they differ in the vertices $v_0, \ldots, v_{p-1}$ in cyclic order. Figure 1 shows the situation and again let $v_p \overset{\text{def}}{=} v_0$. Denote by x_i the sum of edge weights on the path from v_i to v_{i+1} under strategy σ', and let $y_i = \text{val}_\sigma(v_i)$, i.e., y_i is the sum of edge weights on the path from v_i to t under strategy σ. Let $x_p \overset{\text{def}}{=} x_0$ and $y_p \overset{\text{def}}{=} y_0$. The condition "no switch is attractive for σ" says exactly that $y_i \geq x_i + y_{i+1}$. Combining these p inequalities, we obtain

$$\begin{aligned}
y_0 &\geq x_0 + y_1 && \text{[by non-attractiveness in } v_0\text{]} \\
&\geq x_0 + x_1 + y_2 && \text{[by non-attractiveness in } v_1\text{]} \\
&\geq x_0 + x_1 + x_2 + y_3 && \text{[by non-attractiveness in } v_2\text{]} \\
&\;\vdots \\
&\geq x_0 + x_1 + \cdots + x_{p-1} + y_0. && \text{[by non-attractiveness in } v_{p-1}\text{]}
\end{aligned}$$

Thus $x_0 + x_1 + \cdots + x_{p-1} \leq 0$. Since σ' is admissible, there can be no such (nonpositive) loops, a contradiction.

1b. Finite values do not improve finitely. Assume $\text{val}_\sigma(v) < \text{val}_{\sigma'}(v) < \infty$.[7] As in the previous proof, consider the path from v to t under σ'. The strategies must make different choices in one or more vertices on this path, say in $v_1, \ldots, v_p$, in order; Figure 2 applies here as well.

Under strategy σ', denote by x_0 the sum of weights on the path from v to v_1, by x_p the sum of weights on the path from v_p to t, and by x_i $(1 \leq i \leq p-1)$ the sum of weights on the path from v_i to v_{i+1}. Let $y_i = \text{val}_\sigma(v_i)$, i.e., y_i is the sum of weights on the path from v_i to t under σ. The condition "no switch is attractive for σ" says exactly that $y_i \geq x_i + y_{i+1}$, for $1 \leq i \leq p-1$, and

[7] Recall that σ and σ' are admissible, so finite values may only result from paths leading to t.

$y_p \geq x_p$. Combining these p inequalities, we obtain

$$\begin{aligned}
y_1 &\geq x_1 + y_2 && \text{[by non-attractiveness in } v_1\text{]}\\
&\geq x_1 + x_2 + y_3 && \text{[by non-attractiveness in } v_2\text{]}\\
&\geq x_1 + x_2 + x_3 + y_4 && \text{[by non-attractiveness in } v_3\text{]}\\
&\vdots\\
&\geq x_1 + x_2 + \cdots + x_{p-1} + y_p && \text{[by non-attractiveness in } v_{p-1}\text{]}\\
&\geq x_1 + x_2 + \cdots + x_{p-1} + x_p, && \text{[by non-attractiveness in } v_p\text{]}
\end{aligned}$$

and in particular $x_0+y_1 \geq x_0+\cdots+x_p$. But $x_0+y_1 = \mathrm{val}_\sigma(v)$ and $x_0+\cdots+x_p = \mathrm{val}_{\sigma'}(v)$, so indeed we cannot have better finite values under σ' than under σ.

2. Two-player games. Finally, we prove the claim in full generality, where Min may have choices. The simple observation is that Min does not need to use these choices, and the situation reduces to the one-player game we already considered. Specifically, let σ be as before and let τ be an optimal counterstrategy of Min, obtained from the shortest-path forest for vertices with value $< \infty$ and defined arbitrarily in other vertices. Clearly, in the game G_τ, the values of all vertices under σ are the same as in G, and σ does not have any attractive switches. By Case 1, in G_τ we also have $\sigma \geq \sigma'$. Finally, σ' cannot be better in G than in G_τ, since the latter game restricts choices for Min. □

As a consequence, we obtain the following

Corollary 5.3 *In any MPG G every admissible stable strategy is winning for player* Max *from all vertices in $G_{>0}$.* □

The following property is not necessary for correctness, but completes the comparison of attractiveness versus profitability.

Corollary 5.4 *If an admissible strategy σ' is obtained from another admissible strategy σ by one or more* non-attractive *switches, then $\sigma' \leq \sigma$.*

Proof. Consider the game (V, E', w'), where $E' = E \setminus \{(u,v) : u \in V_{\mathrm{Max}} \wedge \sigma(u) \neq v \wedge \sigma'(u) \neq v\}$ and w' is w restricted to E'. In this game, σ has no attractive switches, and both σ and σ' are admissible strategies in the game, hence by Theorem 5.2, $\sigma \geq \sigma'$. □

As a special case (together with Theorem 5.1), we obtain the following equivalence.

Corollary 5.5 *A single switch (between two admissible strategies) is attractive if and only if it is profitable.* □

6 Efficient Computation of the Measure

For an admissible strategy σ the shortest paths in G_σ (strategy measure) can be computed by using the Bellman–Ford algorithm for single-sink shortest paths in graphs with possibly negative edge weights; see, e.g., [8]. This algorithm runs in $O(n \cdot |E|)$ time. Every vertex can have its shortest path length improved at most $O(n \cdot W)$ times (W is the largest absolute edge weight; distances are increased in integral steps). Since there are n vertices, the number of switches cannot exceed $O(n^2 \cdot W)$. Together with the $O(n \cdot |E|)$ bound per iteration this gives total time $O(n^3 \cdot |E| \cdot W)$. Here we show how to reuse the shortest distances computed in previous iteration to improve this upper bound by a linear factor to $O(n^2 \cdot |E| \cdot W)$. Since there are no known nontrivial lower bounds on the number of improvement steps, it is practically important to reduce the cost of each iteration.

We first compute the set of vertices that have different values under the old and the new strategies σ and σ', respectively, and then recompute the values only in these vertices, using the Bellman–Ford algorithm. If the algorithm improves the value of n_i vertices in iteration i, we only need to apply the Bellman–Ford algorithm to a subgraph with $O(n_i)$ vertices and at most $|E|$ edges; hence it runs in time $O(n_i \cdot |E|)$. Since the maximum possible number of integral distance improvements in n vertices is $n^2 \cdot W$, the sum of all n_i's does not exceed $n^2 \cdot W$, so the total running time becomes at most $O(n^2 \cdot |E| \cdot W)$, saving a factor of n. It remains to compute, in each iteration, which vertices need to change their values. Algorithm 1 does this, taking as arguments a game G, the shortest distances $d : V \to \mathbb{N} \cup \{\infty\}$ computed with respect to the old strategy σ, the new strategy σ', and the set of switched vertices $U \subseteq V$, where σ' differs from σ.

Algorithm 1. Mark all vertices v for which $\mathrm{val}_\sigma(v) \neq \mathrm{val}_{\sigma'}(v)$.

```
MARK-CHANGING-VERTICES($G, U \subseteq V, d : V \to \mathbb{N} \cup \{\infty\}$)
(1)  mark all vertices in $U$
(2)  **while** $U \neq \emptyset$
(3)     remove some vertex $v$ from $U$
(4)     **foreach** unmarked predecessor $u$ of $v$ in $G_{\sigma'}$
(5)        **if** $w(u, x) + d[x] > d[u]$ for all unmarked successors $x$ of $u$ in $G_{\sigma'}$
(6)           mark $u$
(7)           $U \leftarrow U \cup \{u\}$
```

Theorem 6.1 *If an attractive (multiple) switch changes an admissible strategy σ to σ', then for every vertex $v \in V$, the following claims are equivalent.*

(1) Algorithm 1 marks v.
(2) Every shortest path from v to t in G_σ passes through some switch vertex.

(3) $\mathrm{val}_\sigma(v) \neq \mathrm{val}_{\sigma'}(v)$.

Proof. (1) $\implies$ (2). By induction on the set of marked vertices, which expands as the algorithm proceeds. The base case holds since the vertices marked before the while loop are the switch vertices; these clearly satisfy (2). For the induction step, assume the claim holds for every marked vertex and that vertex u is about to be marked on line 6. Let x be any successor of u included in some shortest path from u to t in G_σ. Since $w(u,x) + d[x] = d[u]$, and by the condition on line 5, x must be already marked. Hence, by the induction hypothesis, every shortest path through x passes through U. This completes the induction step.

(2) $\implies$ (1). For an arbitrary vertex v, consider all its shortest paths in G_σ. Denote by $\overline{v}$ the maximal number of edges passed by such a path before a vertex in U is reached (so $\overline{v}$ is the unweighted length of an initial segment). The proof is by induction on $\overline{v}$. The base case is clear: $\overline{v} = 0$ iff $v \in U$, and all vertices in U are marked. Assume that the claim holds for all vertices u with $\overline{u} < k$ and consider an arbitrary vertex v with $\overline{v} = k$. By the inductive hypothesis, all successors of v that occur on a shortest path are marked. Hence, when the algorithm removes the last of them from U, the condition on line 5 is triggered and v is marked.

(3) $\implies$ (2). If some shortest path from v to t in G_σ does not pass through a switch vertex, then the same path is available also in $G_{\sigma'}$, hence $\mathrm{val}_\sigma(v) = \mathrm{val}_{\sigma'}(v)$.

(2) $\implies$ (3). Assume (2) and consider an arbitrary shortest path from v to t in $G_{\sigma'}$. If it contains any switch vertices, let u be the first of them. The same path from v to u, followed by the path in G_σ from u to t, gives a shorter path in G_σ, since the length of shortest paths strictly increase in switch vertices. If the path does not contain any switch vertices, then by (2) it is longer than every shortest path in G_σ. □

We thus showed that Algorithm 1 does what it is supposed to. To finish the argument, we show that it runs in time $O(|E|)$, so it is dominated by the time used by the Bellman–Ford algorithm.

Proposition 6.2 *Algorithm 1 can be implemented to run in time $O(|E|)$.*

Proof. Every vertex can be added to U and analyzed in the body of the **while** loop at most once. The condition on line 5 can be tested in constant time if we keep, for each vertex u, the number of unmarked successors x of u with $w(u,x) + d[x] = d[u]$. Thus, the time taken by the **foreach** loop is linear in the number of predecessors of v (equivalently, in the number of edges entering v), and the claim follows. □

7 Complexity of the Algorithm

Section 2 lists several computational problems for mean payoff games. We first show that our basic 0-mean partition algorithm with small modifications also solves the p-mean partition and the splitting into three sets problems with the same asymptotic running time bound. In Section 7.2, we show how to solve the ergodic partition problem, which introduces a small extra polynomial factor in the complexity.

7.1 Complexity of Partitioning with Integer Thresholds

Our basic algorithm in Section 3 solves the *0-mean partition* problem for MPGs. The *p-mean partition* problem with an integer threshold p can be solved by subtracting p from all edge weights and solving the 0-mean partition problem. As a consequence, we also solve the *decision* problem for integer p. Zwick and Paterson [26] consider a slightly more general problem of *splitting into three sets* around an integer threshold p, with vertices of value $< p$, $= p$, and $> p$, respectively. We can solve this by two passes of the p-mean partition algorithm. First, partition the vertices into two sets with values $\leq p$ and $> p$, respectively. Second, invert the game by interchanging $V_{\textsc{Min}}$ and $V_{\textsc{Max}}$ and negating all edge weights, and solve the $(-p)$-mean partition problem. These two partitions correspond to the $< p$ and $\geq p$ partitions of the original game, and combining the two solutions we get the desired three-partition for only twice the effort.

We now analyze the running time of our algorithms, asymptotically the same for all versions of the problem mentioned in the previous paragraph. The complexity of a strategy improvement algorithm consists of two parts: the cost of computing the measure times the number of iterations necessary. Section 6 demonstrates that this combined cost is at most $O(n^2 \cdot |E| \cdot W)$. This is the same complexity as for the algorithm by Zwick and Paterson for the splitting into three sets problem [26, Theorem 2.4]. If W is very big, the number of iterations can of course also be bounded by $\prod_{v \in V_{\textsc{Max}}} outdeg(v)$, the total number of strategies for Max.

Using the randomization scheme of Matoušek, Sharir, and Welzl from Section 3.4 we obtain the simultaneous bound $2^{O(\sqrt{n \log n})}$, *independent* of W. Combining the bounds, we get the following.

Theorem 7.1 *The decision, p-mean partition, and splitting into three sets problems for mean payoff games can be solved in time*

$$\min\left(O(n^2 \cdot |E| \cdot W),\ 2^{O(\sqrt{n \log n})}\right). \qquad \square$$

Note that a more precise estimation replaces n by $|V_{\text{MAX}}|$ in the subexponential bound, since we only consider strategies of MAX. Also, $n \cdot W$ can be replaced by the length of the longest shortest path, or any upper estimate of it. For instance, one such bound is the sum, over all vertices, of the maximal positive outgoing edge weights.

7.2 Computing the Ergodic Partition

We now explain how to use the solution to the p-mean partition problem to compute the ergodic partition. We first describe the algorithm, which uses an algorithm for the p-mean partition problem with rational thresholds as a subroutine. We then analyze our algorithm for the case of rational thresholds, and finally bound the total running time, including all calls to the p-mean partition algorithm.

Denote by w^- and w^+ the smallest and biggest edge weights, respectively. Then the average weight on any loop (i.e., the value of any vertex in the MPG) is a rational number with denominator $\leq n$ in the interval $[w^-, w^+]$. We can find the value for each vertex by dichotomy of the interval, until each vertex has a value contained in an interval of length $\leq 1/n^2$. There is at most one possible value inside each such interval (the difference between two unequal mean loop values is at least $1/n(n-1)$), and it can be found easily [14,26]. The interval is first partitioned into parts of integer size. After that we deal with rational thresholds p/q, where $q \leq n^2$. We therefore have to solve the p-mean partition problem when the threshold p is rational and not integral. As is readily verified, our algorithm directly applies to this case: only the complexity analysis needs to be slightly changed.

The subexponential bound does not depend on thresholds being integers, but we need to analyze the depth of the measure. After subtracting p/q from each (integral) weight w, it can be written in the form $(qw - p)/q$, so all weights become rationals with the same denominator. The weight of every path of length k to t has the form $(\sum_{i=1}^{k} w_i) - kp/q$. The sum $\sum_{i=1}^{k} w_i$ can take at most $n \cdot W$ different values, and k can take at most n values, so each vertex can improve its value at most $n^2 \cdot W$ times. Thus, solving the 0-mean problem for rational thresholds takes at most n times longer.

During the dichotomy process, we consider subproblems with different thresholds. The thresholds are always in the range $[w^-, w^+]$, so the largest absolute edge weight is linear in W. We will bisect the intervals at most $O(\log(W \cdot n^2))$ times. The total number of vertices in the subproblems solved after bisecting k times is at most n. The complexity function T is superlinear in n, so that $T(i) + T(j) \leq T(i + j)$. Hence the total time for the subproblems after k bisections does not exceed that of solving one n-vertex game. This shows that the whole computation takes time $O(\log(W \cdot n) \cdot T)$, where T is the time for solving the p-mean partition problem. We summarize this in the following theorem.

Theorem 7.2 *The ergodic partition problem for mean payoff games can be solved in time*

$$\min\Big(O(n^3 \cdot |E| \cdot W \cdot \log(n \cdot W)), \quad (\log W) \cdot 2^{O(\sqrt{n \log n})}\Big). \qquad \square$$

Zwick's and Paterson's algorithm for this problem has the worst-case bound $O(n^3 \cdot |E| \cdot W)$ [26, Theorem 2.3], which is slightly better for small W, but worse for large W. Note that the algorithm [26] exhibits its worst case behavior on simple game instances.

8 Variants of the Algorithm

Theorem 5.1 shows that any combination of attractive switches improves the strategy value, and thus any policy for selecting switches in each iteration will eventually lead to an optimal strategy. In particular, all policies that have been suggested for parity and simple stochastic games apply. These include the all profitable, random single, and random multiple switch algorithms; see, e.g., [1]. In our experiments with large random and non-random game instances these algorithms witness extremely fast convergence. In this section we suggest two alternative ways of combining policies.

Initial Multiple Switching. We can begin with the strategy where MAX goes to the sink t in each vertex and rather than starting the randomized algorithm described in Section 3.4 immediately, instead make a polynomially long sequence of random (multiple) attractive switches, selecting them at each step uniformly at random. Use the last strategy obtained, if not yet optimal, as an initial one in the randomized algorithm above. There is a hypothesis due to Williamson Hoke [25] that every *completely unimodal* function (possessing a unique local maximum on every boolean subcube) can be optimized by the random single switch algorithm in polynomially many steps. The longest-shortest paths problem is closely related to CU-functions. Investigating these problems may shed light on possibilities of polynomial time optimization for both.

Proceeding in Stages. Another version of the algorithm starts as described in the previous subsection, and afterward always maintains a partition of the vertices (from which the longest shortest distance is not yet $+\infty$) into two sets: R of vertices where MAX is still using the conservative strategy of retreating immediately to the sink t, and N of all other vertices. In all vertices in $N \cap V_{\text{MAX}}$, MAX already switched away from retreating. Since the value of a vertex can only increase, MAX will never change back playing to retreats in those vertices (so retreat edges from vertices in N may be safely removed without influencing the 0-mean partition). We fix the choices in R and proceed as in Section 3.4, to find the best strategy in controlled vertices in N. If the resulting strategy is globally optimal (contains no attractive switches in R), we stop. Otherwise we make some or all attractive switches in vertices in R. Each stage of this version of the algorithm is subexponential, and there are only linearly many such stages, because a vertex leaving R never returns back.

9 The LSP Problem is in NP∩coNP

The decision version of the LSP problem, restricted to determine whether the longest shortest path from a distinguished vertex s to the sink is bigger than a given bound D is another example of a problem in NP∩coNP.

Recall that in the definition of the LSP problem (Section 3.1) we have not stated how the shortest distance to the sink is defined when the MAX player can enforce a zero-weight loop. This was unneeded because such loops are uninteresting for MAX in mean payoff games for reaching a *positive* mean value. Zero-weight loops are impossible in admissible strategies, and only such strategies are needed (visited) by our algorithms to compute zero-mean partitions in games. However, in the LSP problem it is natural to postulate that whenever MAX can enforce a zero-weight cycle, the distance to the (unreachable) sink becomes $+\infty$. We show here that a minor modification allows us to reuse Theorems 5.1 (attractiveness implies profitability) and 5.2 (stability implies optimality), as well as subexponential algorithms from Sections 3.3, 3.4, to compute longest shortest paths, and to prove the NP∩coNP-membership.

The necessary modification is achieved by making the following simplifying assumption about the instances of the LSP problem.

Assumption. *A graph in an LSP problem instance does not contain* zero-*weight loops.* □

This assumption may be done without loss of generality. Indeed, if the graph G has n vertices, we can multiply all edge weights by $n + 1$ and add 1. As a result, zero loops will disappear (become positive), and the lengths l, l' of all paths/simple loops in G and the modified graph G' will only differ within the factor of $(n + 1)$, i.e., $l(n + 1) < l' \leq l(n + 1) + n$. Consequently, all negative/positive loops in the original graph will preserve their signs.

Proposition 9.1 *The decision version of the LSP problem (subject to the assumption above) is in* NP∩coNP.

Proof. First note that we can add retreat edges to any LSP problem instance similarly as to MPGs: from any controlled vertex, make an extra edge to the sink with value $-2 \cdot n \cdot W - 1$. Thus, we guarantee that there is an admissible strategy, namely the one always using the retreat edge. In a solution to the transformed LSP problem, a vertex has value $< -n \cdot W$, iff it can only reach the sink through a retreat edge, iff it has value $-\infty$ in the original problem.

Both for YES- and NO-instances, the short witness is an optimal (stable) positional strategy σ in controlled vertices of the transformed problem. By computing the shortest paths to the sink in G_σ, i.e., computing the strategy measure, it can be verified in polynomial time that no switch is attractive and thus that the strategy is optimal by Theorem 5.2. This can be used as a witness for YES-instances by testing if the value is $> D$, and for NO-instances by testing whether the value is $\leq D$. □

The absence of zero-weight loops is essential in the proof above. The example in Figure 3 demonstrates a zero-weight loop, and a stable strategy, which does not provide the optimal (in the LSP sense) solution.

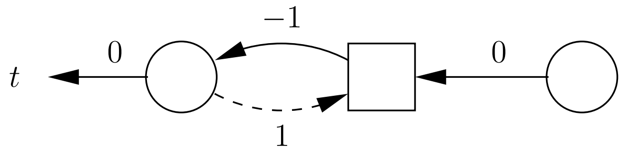

Figure 3. Switching to the dotted edge is not attractive, but improves the values in all vertices to $+\infty$.

For mean payoff games we were satisfied with the definition of optimality that only stipulates that Max can secure *positive-weight* loops whenever possible. Thus, in Figure 3 the strategy "go to t" in the leftmost vertex is MPG-optimal, but not LSP-optimal. In contrast, the LSP-optimality makes zero-weight loops attractive (shortest distance to the sink equals $+\infty$). This was essentially used in the proof of Proposition 9.1 and can always be achieved by making the initial transformation to satisfy the Assumption.

10 LSP: Exponential Sequences of Attractive Switches

One might conjecture that any sequence of attractive switches converges fast on any LSP problem instance, and consequently MPGs are easily solvable by iterative improvement. It is not so easy to come up with “hard” LSP examples. In this section we present a set of instances of the LSP problem and an improvement policy, selecting an attractive switch in every step, leading to exponentially long sequences of strategy improvements. This shows that the LSP problem is nontrivial, and the choice of the next attractive switch is crucial for the efficiency.

Consider the $(2n+2)$-vertex graph in Figure 4, where round vertices belong to Max, square ones to Min, and the leftmost vertex is the sink. The optimal strategy of Max is marked by the dashed edges. Adding N is unnecessary here (N may be set to 0 for simplicity), but will be needed later.

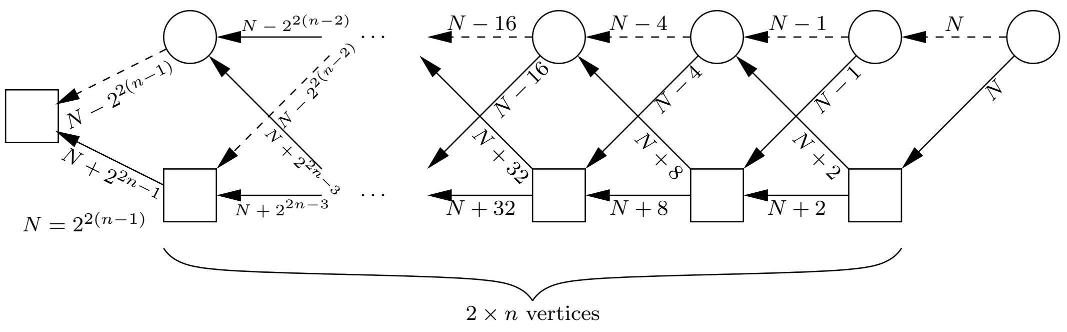

Figure 4. LSP instances allowing for exponentially long chains of improving switches.

If the iterative improvement algorithm starts from the strategy “go left everywhere” and always makes the *rightmost* attractive single switch, it visits all 2^n possible strategies of Max, as can be readily verified.

The LSP instances above can be generated from MPGs as follows. In Figure 4, add a self-loop of weight zero in the leftmost vertex. Add the retreat vertex and edges as explained in Section 3.2. If the algorithm initially switches from the “go to the retreat everywhere” strategy to the “go left everywhere”, and then always chooses the rightmost attractive single switch, it follows exactly the exponentially long chain of improving strategies, as in the LSP case. Adding N now is essential to keep all the paths nonnegative; otherwise, the initial “switches from the retreat” would be non-attractive.

Actually, the LSP-instances in Figure 4 are “trivial”, because the graphs are acyclic and longest-shortest distances are easily computable by dynamic programming in polynomial time, as mentioned in the end of Section 3.1. These LSP-instances are also easily solvable by the “random single switches” policy, selecting an attractive switch uniformly at random. The reason is as follows. The second from the left dashed edge remains always attractive, and once the algorithm switches to this edge, it will never switch back. Such an edge defines a so-called *absorbing facet.* Obviously, the random single switch algorithm is expected to switch to the absorbing facet in polynomially many, namely $O(n)$, steps. The problem that remains has one dimension less, and the preceding dashed edge determines an absorbing facet. The algorithm converges in $O(n^2)$ expected iterations. Currently we are unaware of any LSP instances that require superpolynomially many steps of the single random, multiple random, or all profitable (attractive) switches algorithms.

The example of this section is inspired by the one due to V. Lebedev mentioned in [14], and kindly provided by V. Gurvich. Unlike the GKK-algorithm [14], which is deterministic and bound to perform the exponential number of iterations, corresponding exactly to the exponential sequence determined by the rightmost attractive switches above, our randomized algorithms quickly solve the examples from this section.

11 Application to Parity Games

The algorithm described in this paper immediately applies to parity games, after the usual translation; see, e.g., [23]. Parity games are similar to mean payoff games, but instead of weighted edges they have vertices colored in nonnegative integer colors. Player Even (Max) wants to ensure that in every infinite play the largest color appearing infinitely often is *even*, and player Odd (Min) tries to make it *odd.* Parity games are determined in positional strategies [13,5].

Transform a parity game into a mean payoff game by leaving the graph and the vertex partition between players unchanged, and by assigning every vertex[8] of color c the weight $(-n)^c$, where n is the total number of vertices in the game. Apply the algorithm described in the preceding sections to find a 0-mean partition. The vertices in the partition with value > 0 are winning for Even and all other are winning for Odd in the parity game.

Actually, the new MPG algorithm, together with a more economic translation and more careful analysis, gives a considerable improvement for parity games over our previous algorithm from [2], which has complexity

$$\min\left(O(n^4 \cdot |E| \cdot k \cdot (n/k+1)^k),\ 2^{O(\sqrt{n \log n})}\right),$$

where n is the number of vertices, $|E|$ is the number of edges, and k is the

[8] Formally, we have to assign this weight to every edge leaving a vertex, but it makes no difference.

number of colors. Thus, in the worst case the first component in the upper bound may be as high as $O(|E| \cdot n^5 \cdot 2^n)$, when the number of colors equals the number of vertices. The MPG algorithm improves the first component in the upper bound to $O(n^2 \cdot |E| \cdot (n/k+1)^k)$, thus gaining a factor of $n^2 \cdot k$. To prove this bound, we assign weights to vertices more sparingly than in the mentioned reduction. It is readily verified that we may equally well give a vertex of color c the weight $(-1)^c \cdot \prod_{i=0}^{c-1}(n_i + 1)$, where n_i is the number of vertices of color i. In the worst case (when all n_i are roughly equal), this gives $W = O((n/k + 1)^k)$.

There are two reasons for this improvement. First, each vertex can improve its value at most $n \cdot (n/k+1)^k$ times, compared with $n^2 \cdot (n/k+1)^k$ in [2] (this is because the measure for every vertex in [2] is a triple, rather a single number, containing additionally the loop color and the path length, now both unneeded; however the shortest distance may be as big as the sum of all positive vertices). Second, we now apply the simpler and more efficient counterstrategy algorithm from Section 6, which saves an extra factor nk.

Moreover, translating a parity game into an MPG allows us to assign weights even more sparingly, and this additionally improves over $(n/k + 1)^k$. Indeed, if we want to assign a minimum possible weight to a vertex of color i, we may select this weight (with an appropriate sign) equal to the total absolute weight of all vertices of preceding colors of opposite parity, plus one. This results in a sequence $w_0 = 1$, $w_1 = -(n_0 \cdot w_0 + 1)$, $w_{i+2} = -(n_{i+1} \cdot w_{i+1} + n_{i-1} \cdot w_{i-1} + \cdots + 1) = -n_{i+1} \cdot w_{i+1} + w_i$. In the case of $k = n$ (one vertex per color) it results in the Fibonacci sequence with alternating signs: $w_0 = 1$, $w_1 = -2$, $w_{i+2} = -w_{i+1} + w_i$, which is, in absolute value, asymptotically $O(1.618^n)$, better than 2^n.

Finally, the new MPG-based algorithm for parity games is easier to explain, justify, and implement.

12 Conclusions

We defined the longest shortest path (LSP) problem and showed how it can be applied to create a discrete strategy evaluation for mean payoff games. Similar evaluations were already known for parity [24,2], discounted payoff [23], and simple stochastic games [17], although not discrete for the last two classes of games. The result implies that any strategy improvement policy may be applied to solve mean payoff games, thus avoiding the difficulties of high precision rational arithmetic involved in reductions to discounted payoff and simple stochastic games, and solving associated linear programs.

Combining the strategy evaluation with the algorithm for combinatorial linear programming suggested by Matoušek, Sharir, and Welzl, yields a $2^{O(\sqrt{n \log n})}$ algorithm for the mean payoff game decision problem.

An interesting open question is whether the LSP problem is more general than mean payoff games, and if it can find other applications. We showed that it belongs to NP $\cap$ coNP, and is solvable in expected subexponential randomized time.

The strategy measure presented does not apply to all strategies, only admissible ones, which do not allow nonpositive weight loops. This is enough for the algorithm, but it would be interesting to know if the measure can be modified or extended to the whole strategy space, and in this case if it would be completely local-global, like the measures for parity and simple stochastic games investigated in [4].

The major open problem is still whether any strategy improvement scheme for the games discussed has polynomial time complexity.

Acknowledgments. We thank DIMACS for providing a creative working environment. We are grateful to Leonid Khachiyan, Vladimir Gurvich, and Endre Boros for inspiring discussions and illuminating ideas.

References

[1] H. Björklund, S. Sandberg, and S. Vorobyov. Complexity of model checking by iterative improvement: the pseudo-Boolean framework. In M. Broy and A. Zamulin, editors, *Andrei Ershov Fifth International Conference "Perspectives of System Informatics"*, volume 2890 of *Lecture Notes in Computer Science*, pages 381–394, 2003.

[2] H. Björklund, S. Sandberg, and S. Vorobyov. A discrete subexponential algorithm for parity games. In H. Alt and M. Habib, editors, *20th International Symposium on Theoretical Aspects of Computer Science, STACS'2003*, volume 2607 of *Lecture Notes in Computer Science*, pages 663–674, Berlin, 2003. Springer-Verlag.

[3] H. Björklund, S. Sandberg, and S. Vorobyov. On combinatorial structure and algorithms for parity games. Technical Report 2003-002, Department of Information Technology, Uppsala University, 2003. `http://www.it.uu.se/research/reports/`.

[4] H. Björklund, S. Sandberg, and S. Vorobyov. Randomized subexponential algorithms for parity games.

Technical Report 2003-019, Department of Information Technology, Uppsala University, 2003. `http://www.it.uu.se/research/reports/`.

[5] H. Björklund, S. Sandberg, and S. Vorobyov. Memoryless determinacy of parity and mean payoff games: A simple proof. *Theoretical Computer Science*, 310(1-3):365–378, 2004.

[6] A. Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.

[7] A. Condon. On algorithms for simple stochastic games. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 13:51–71, 1993.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press and McGraw-Hill Book Company, Cambridge, MA, 2nd edition, 2001.

[9] A. Ehrenfeucht and J. Mycielski. Positional games over a graph. *Notices Amer. Math Soc.*, 20:A–334, 1973.

[10] A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journ. of Game Theory*, 8:109–113, 1979.

[11] E. A. Emerson. Model checking and the Mu-calculus. In N. Immerman and Ph. G. Kolaitis, editors, *DIMACS Series in Discrete Mathematics*, volume 31, pages 185–214, 1997.

[12] E. A. Emerson, C. Jutla, and A. P. Sistla. On model-checking for fragments of μ-calculus. In C. Courcoubetis, editor, *Computer Aided Verification, Proc. 5th Int. Conference*, volume 697, pages 385–396. Lect. Notes Comput. Sci., 1993.

[13] E. A. Emerson and C. S. Jutla. Tree automata, μ-calculus and determinacy. In *Annual IEEE Symp. on Foundations of Computer Science*, pages 368–377, 1991.

[14] V. A. Gurvich, A. V. Karzanov, and L. G. Khachiyan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 28(5):85–91, 1988.

[15] A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.

[16] G. Kalai. A subexponential randomized simplex algorithm. In *24th ACM STOC*, pages 475–482, 1992.

[17] W. Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117:151–155, 1995.

[18] J. Matoušek, M. Sharir, and M. Welzl. A subexponential bound for linear programming. In *8th ACM Symp. on Computational Geometry*, pages 1–8, 1992.

[19] J. Matoušek, M. Sharir, and M. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.

[20] H. Moulin. Extensions of two person zero sum games. *J. Math. Analysis and Applic.*, 55:490–508, 1976.

[21] H. Moulin. Prolongements des jeux de deux joueurs de somme nulle. *Bull. Soc. Math. France*, 45, 1976.

[22] N. Pisaruk. Mean cost cyclical games. *Mathematics of Operations Research*, 24(4):817–828, 1999.

[23] A. Puri. *Theory of hybrid systems and discrete events systems.* PhD thesis, EECS Univ. Berkeley, 1995.

[24] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In E. A. Emerson and A. P. Sistla, editors, *CAV'00: Computer-Aided Verification*, volume 1855 of *Lect. Notes Comput. Sci.*, pages 202–215. Springer-Verlag, 2000.

[25] K. Williamson Hoke. Completely unimodal numberings of a simple polytope. *Discrete Applied Mathematics*, 20:69–81, 1988.

[26] U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158:343–359, 1996.