Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 155 (2007) 210–229
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games[☆]

Henrik Björklund, Sergei Vorobyov*

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

Received 30 September 2004; received in revised form 21 June 2005; accepted 20 April 2006
Available online 27 June 2006

Dedicated to the memory of Leonid Khachiyan, 1952–2005

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. The decision version of the problem (whether the shortest distance from a given vertex can be made bigger than a given bound?) 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 from 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).
© 2006 Elsevier B.V. All rights reserved.

Keywords: Mean payoff game; Cyclic game; Parity game; Ergodic partition; Combinatorial linear programming; Longest shortest parth; Iterative improvement; Randomized subexponential algorithm

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) [15,29,28,16,37], also known as *cyclic games* [20,30]. 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

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

* Corresponding author. Tel.: +46 18 471 10 55; fax: +46 18 55 02 25.
E-mail address: Sergei.Vorobyov@it.uu.se (S. Vorobyov).

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2006.04.029

formed. MPGs are *determined*, and every vertex has a *value*, which each player can secure by a *uniform positional* strategy. Deciding 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 [19,17], are polynomial time reducible to MPGs. Other well-known games with NP ∩ coNP decision problems, to which MPGs reduce, are *simple stochastic* [12] and *discounted payoff* [31,37] games. At present, despite substantial efforts, there are no known polynomial time algorithms for any of the games mentioned.

All previous algorithms for MPGs are either pseudopolynomial or exponential. These include a potential transformation method by Gurvich et al. [20] (see also Pisaruk [30]), and a dynamic programming algorithm solving k-step games for big enough k by Zwick and Paterson [37]. Both algorithms are *pseudopolynomial* of complexity $\mathrm{O}(poly(n) \cdot W)$, where n is the number of vertices and W is the maximal absolute edge weight. The algorithm [37] *always* makes $\Omega(poly(n) \cdot W)$ steps, and for the algorithm [20] there are known game instances on which it shows the worst case $\Omega(poly(n) \cdot W)$ behavior, where W may be exponential in n. Reduction to simple stochastic games [37] and application of the algorithm from [25] gives subexponential complexity (in the number of vertices n) only if the game graph has bounded outdegree (see the remark below). The subexponential algorithms we suggested for simple stochastic games of arbitrary outdegree in [4,5,11] make at most $2^{\mathrm{O}(\sqrt{n \log n})}$ iterations. However, in a reduction from MPGs [37], large weights result in long rational coefficients. This prevents each iteration, which requires solving a linear program, to run in strongly polynomial time, independent of the weights. Solving resulting linear programs by a combinatorial subexponential algorithm [22,26,27] *squares* the overall complexity. This drawback is overcome with the new techniques presented here, which avoid the detour over simple stochastic games and linear programming subroutines altogether.

We suggest the first *strongly subexponential strategy improvement* algorithm for MPGs, which starts with some strategy of the maximizing player[1] Max and iteratively "improves" it with respect to a simple strategy evaluation function, based on computing shortest paths in the residual graph. Iterative strategy improvement algorithms are known for the related simple stochastic [21,13], discounted payoff [31], and parity games [34,3]. Until the present paper, a direct combinatorial iterative strategy improvement for MPGs 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 the combinatorial model of computation, the subexponential running time bound is independent of the edge weights (strongly subexponential).

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

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

where n is the number of vertices and W is the largest absolute edge weight. The first bound matches those from [20,37,30], 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 [34,21,13]. These include random single switch, all profitable switches, and random multiple switches; see, e.g., [6,8]. They are simplex-type algorithms, very efficient in practice, but without currently known subexponential upper bounds, and no nontrivial lower bounds.

By a known simple reduction from parity games [18,19,31,17] to MPGs, our algorithm can be immediately applied for solving parity games. In contrast to the previous strategy improvement algorithms [34,3], our new algorithm is conceptually much simpler, more efficient, and easier to implement.

Remark (*On Ludwig's [25] algorithm*). The algorithm [25] is subexponential $2^{\mathrm{O}(\sqrt{n})}$ only for *binary* simple stochastic games, in which every vertex has outdegree *at most two* (more precisely, for games with outdegree bounded by a constant). Although a simple stochastic game, with *arbitrary* vertex outdegrees, reduces to a binary one, this may lead to a *quadratic* $\mathrm{O}(n^2)$ blow-up in the number of vertices, and the algorithm [25] becomes *exponential*: $2^{\mathrm{O}(\sqrt{n^2})} = 2^{\mathrm{O}(n)}$ in the number of vertices. One may weakly argue that the algorithm [25] is anyway *subexponential in the length of input*,

[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.

since the length of input for arbitrary outdegree games is $O(n^2)$, for representing edges. However, a straightforward brute-force algorithm systematically exploring all up to $2^{n \log n}$ positional strategies and selects the best one is also *subexponential in the length of input* in this case! So the algorithm [25] does not show drastic advantages over the brute-force one for arbitrary outdegree games. The first (really) subexponential $2^{O(\sqrt{n \log n})}$ algorithms (in the number of vertices n, which should be recognized as a the adequate measure of complexity) for simple stochastic games of arbitrary outdegree are based on different (compared to [25]) randomization schemes and were first suggested in [4,3,6,8,11]. Note how the real subexponential bound $2^{O(\sqrt{n \log n})}$ compares favorably to "subexponential" $2^{O(n)}$.

Outline: Section 2 defines MPGs and introduces the associated computational problems. Section 3 describes the longest shortest paths (LSP) problem and its relation to MPGs. 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 decision version of the LSP problem is in NP ∩ coNP. In Section 10 we give an example graph family for which the wrong choice of iterative improvement policy leads to an exponential number of iterations. Finally, in Section 11 we apply our algorithm to solving parity games.

2. Preliminaries

2.1. Mean payoff games

A *mean payoff game* (MPG for short) [29,28,16,20,37] is played by two adversaries, players MAX and MIN, on a finite, directed, edge-weighted, leafless[2] graph $G = (V, E, w)$, where $V = V_{\text{MAX}} \cup V_{\text{MIN}}$, $V_{\text{MAX}} \cap V_{\text{MIN}} = \emptyset$, $E \subseteq V \times V$, and $w : E \to \mathbb{Z}$ is the weight function. Starting from some designated initial vertex, the players move a pebble along edges of the graph. Whenever, the pebble comes to a vertex $v \in V_{\text{MAX}}$, player MAX makes a move by selecting some edge (v, u) and the pebble goes to vertex u. Similarly, MIN selects a move when the pebble is in a vertex from V_{MIN}. The duration of the game is infinite and the resulting infinite sequence of edges $e_1 e_2 e_3 \ldots$ is called a *play*. Player MAX wants to maximize the payoff

$$v_{\max} = \liminf_{k \to \infty} \frac{1}{k} \cdot \sum_{i=1}^{k} w(e_i),$$

whereas player MIN wants to minimize the payoff

$$v_{\min} = \limsup_{k \to \infty} \frac{1}{k} \cdot \sum_{i=1}^{k} w(e_i).$$

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. Using a positional strategy σ, the player always deterministically selects the same successor $\sigma(v)$ in each vertex v where he makes a move, independently of the history of the play. It turns out that in every MPG each vertex v has a *value* $v(v)$ such that whenever a play starts in v, MAX has an *optimal* positional strategy that secures him a payoff $v_{\max} \geqslant v(v)$ against any strategy of MIN, and, vice versa, MIN has an optimal positional strategy that secures him a payoff $v_{\min} \leqslant v(v)$.[3] Moreover, both players have *uniform* such strategies: the same optimal positional strategy may be used independently of the initial vertex of the game. Revealing this strategy before a play starts is not a disadvantage and does not lead to a suboptimal payoff. Accordingly, throughout the paper we restrict our attention to positional strategies only. Given a positional strategy σ for MAX, define $G_\sigma = (V, E')$, by deleting all MAX edges not used in σ, i.e., $E' = E \setminus \{(v, u) | v \in V_{\text{MAX}} \text{ and } \sigma(v) \neq u\}$. Note that if both players use positional strategies, then any play will follow a (possibly empty) simple path to a simple cycle, where it will stay forever. The value of this play is the average edge weight on this cycle. See [29,28,16,20,30,37,7] for details.

[2] i.e., every vertex has at least one outgoing edge.

[3] Thus, if both use optimal strategies, $v_{\max} = v_{\min} = v(v)$. Deviating from an optimal strategy may only lead to a suboptimal payoff for the player.

2.2. Algorithmic problems for MPGs

We will address several computational problems for MPGs.

The decision problem: Given an MPG, a distinguished start vertex, and a threshold value p, can MAX secure a payoff $> p$?

p-mean partition: Given p, partition the vertices of an MPG G into subsets $G_{\leqslant p}$ and $G_{>p}$ such that MAX can secure a payoff $> p$ starting from every vertex in $G_{>p}$, and MIN can secure a payoff $\leqslant p$ starting from every vertex in $G_{\leqslant p}$.

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* [20].

Clearly, the p-mean partitioning subsumes the decision problem. Whenever one player fixes a positional strategy, an optimal counterstrategy of the opponent can be computed in polynomial time, by using Karp's minimum mean weight cycle algorithm; see [14, p. 617]. Therefore, the decision problem is in NP ∩ CONP, and there is a straightforward exponential (in the number of vertices of one player) time algorithm exploring all MAX strategies. Known pseudopolynomial time algorithms were surveyed in the introduction.

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 cycles (in particular, of optimal cycles) 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. Until Section 7.2 we assume such thresholds to be integral. In Section 7.2 we extend the basic algorithm to solve the ergodic partition problem. Another problem to which our algorithm may be extended is finding optimal strategies in MPGs.

Some of our proofs rely on an equivalent *finite version* of MPGs [16], where a play stops as soon as some vertex is revisited and the mean value on the resulting cycle determines the payoff. Thus, for a play $e_1e_2\cdots e_re_{r+1}\cdots e_s$, where $e_r\cdots e_s$ is a simple cycle, the value is $\sum_{i=r}^{s} w(e_i)/(s-r+1)$. Ehrenfeucht and Mycielski [16] proved the following equivalence (see also [7]).

Theorem 2.1. *The value of every vertex in the finite duration version of MPGs equals its value in the infinite duration version.*

The next corollary will be used implicitly throughout the paper.

Corollary 2.2. *A positional strategy σ of* MAX *gives a value $> p$ in a vertex, iff all cycles reachable from it in G_σ have average value $> p$.*

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 algorithm for computing 0-mean partitions in MPGs is based on a new "*controlled*" version of the well-known *single source*[4] *shortest paths problem* on directed graphs. Suppose a given digraph has some distinguished set of *controlled* vertices, 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 outgoing edges from controlled vertices, except those used in σ.

[4] Actually, we use the symmetric and equivalent "single-sink" shortest paths problem as more adequate for our purposes. A sink is a vertex with no outgoing edges.

Formally, the problem is specified as follows.

THE LONGEST SHORTEST PATHS PROBLEM (LSP).

Given: (1) a directed edge-weighted graph G with a unique sink t,

(2) a distinguished set $U \subseteq V[G]$ of *controlled* vertices, with $t \notin U$.

Find: a positional strategy σ selecting exactly one outgoing edge from each vertex in U such that in the graph G_σ the length of the shortest path from every vertex to sink t is as large as possible (over all positional strategies).

Distances. As usual, the length of a finite simple path to the sink in G_σ equals the sum of edge weights on the path. In a cyclic G_σ the distances to the sink are defined as

(1) $+\infty$ for every vertex on a *positive* weight cycle;
(2) $-\infty$ for every vertex on a *negative* weight cycle;
(3) 0 for every vertex on a 0-weight cycle.

The last clause is motivated by our application to the 0-mean partition problem for MPGs, in which 0-weight loops do not contribute to the $G_{>0}$ partition. For the real LSP it would be more logical to postulate that every vertex on a 0-weight cycle has distance $+\infty$ to the sink, as we do in Section 9.

We make sure that all strategies σ generated by iterative improvement in every run of our algorithm are *admissible*, i.e., the graph G_σ has no *nonpositive* weight cycles. Thus, computing shortest paths in G_σ always yields finite or $+\infty$ distances, and the last two clauses in the distance definition above never need to apply.

For our purpose of finding 0-mean partitions in MPGs, it suffices to consider a version of the LSP problem with the following additional input data.

Additionally given: an admissible strategy σ_0, which guarantees that in the graph G_{σ_0} there are no cycles with *nonpositive* weights.

With this additionally supplied strategy σ_0 we know that the longest shortest distance from every vertex to the sink t is *at least finite*, $> -\infty$. It is not excluded that σ_0 or the optimal strategy will make some distances equal $+\infty$ (this is the goal of MAX, which he will try to figure out by iterative improvement starting with σ_0). We prove that our algorithm *never generates an inadmissible strategy*. Treating 0-weight cycles by maintaining the admissibility invariant for the algorithm (rather than just assuming wlog that 0-weight cycles are absent, as we do in Section 9) adds certain complications, but pays off with a better complexity, as explained below.

The simplifying additional input strategy is easy to provide in the reduction from MPGs by adding 0-weight edges from all MAX vertices to the sink. Actually, it is more difficult to obtain a subexponential iterative improvement algorithm without this "additionally given" strategy. Indeed, it is conceivable that there are no admissible strategies at all, and every strategy allows for reaching a negative cycle from every vertex (although we do not know that *a priori*). Then there is no apparent way to figure this out, except the full exponential enumeration and verification of the strategy space. This is because the neighborhood structure on this space is completely "flat" and moving between neighboring strategies provides no improvement.

Note that for DAGs, the LSP problem can be easily solved in polynomial time by dynamic programming. Start by topologically sorting the vertices and proceed backward from the sink (distance 0), using the known longest shortest distances for the preceding vertices.

3.2. Relating the 0-mean partition and LSP problems

The relation between finding 0-mean partitions and computing longest shortest paths is now easy to describe. To find a 0-mean partition in an MPG G, just add a *retreat* vertex t (belongs to MIN and will later become the sink) to the game graph with a self-cycle edge of weight 0, then add a 0-weight *retreat* edge from every MAX vertex to t. From now on, we assume that G has undergone this transformation. Clearly, we have the following property.

Proposition 3.1. *Adding the retreat does not change the 0-mean partition of the game, except that the new retreat vertex belongs to the* $G_{\leqslant 0}$ *part.*

This is because we do not add anything allowing MAX to create new positive cycles, or MIN to create new nonpositive cycles (except in retreat t). MAX will prefer playing to t only if all other positional strategies lead to negative cycles.

The key point is now as follows. Break the self-cycle in t and consider the LSP problem for the resulting graph, with

t being the unique sink. The set V_{MAX} becomes the set of controlled vertices, and the initial strategy (the "additionally given" clause in the LSP definition above) selects retreat t in every controlled vertex, guaranteeing that no vertex has distance $-\infty$.[5] 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 $+\infty$.*

Proof. The same MAX positional strategy simultaneously achieves positive weight cycles both in the MPG and the corresponding LSP instance. Recall that by convention 0-weight cycles yield 0 distances to the sink. □

3.3. Blocking nonpositive cycles

As early as in 1991 Leonid Khachiyan [23] considered the following variant of the LSP problem, stated here using our terminology.

BLOCKING NONPOSITIVE CYCLES.

Given: a directed edge-weighted leafless graph and a vertex subset, in which the controller must choose exactly one outgoing edge per vertex and delete all other outgoing edges.

Find: a subset B of vertices from which the controller can make unreachable any nonpositive weight cycles.

In the decision version the question is whether a given vertex belongs to B. As an immediate consequence of the preceding definitions we have the following.

Proposition 3.3.

(1) *The problems 0-mean partition in MPGs and blocking nonpositive cycles are polynomial time equivalent.*
(2) *The decision version of blocking nonpositive cycles is in* NP ∩ CONP.
(3) *Blocking nonpositive cycles is polynomial time reducible to the LSP problem.*

Note that in the LSP problem, except being interested in the $+\infty$ distances to the sink (which corresponds to positive cycles in blocking nonpositive cycles), we are additionally interested in computing *finite* distances. Our algorithm iteratively improves these finite distances, until, hopefully, improving them to $+\infty$. Our goal with the LSP problem was exactly inventing the optimization (rather than decision) problem appropriate for iterative improvement.

To our knowledge, there are no other mentions of the LSP problem and its relation to MPGs in the literature.[6]

3.4. The algorithm

Our 0-mean partition algorithm computes LSP in the graph resulting from an MPG (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, by using the Bellman–Ford algorithm. Since there are negative weight edges, the Dijkstra algorithm cannot be used. An improvement to the straightforward application of the Bellman–Ford algorithm is described in Section 6.

The main step of our iterative strategy improvement algorithm is the so-called *attractive switch*. 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*.

Switching is similar (but is the opposite) to the usual edge relaxation in the shortest paths algorithms. Suppose the current distance (using the current strategy σ) from a vertex v to the sink is $d_\sigma(v)$, but for an edge (v, u) not used by σ we have $d_\sigma(v) < w(v, u) + d_\sigma(u)$ (attractiveness). Then we switch the current edge $(v, \sigma(v))$ to (v, u), get new strategy σ, and recompute the shortest distances.

[5] Actually, there may exist negative cycles consisting only of vertices from V_{MIN}. Such cycles 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.

[6] The authors would appreciate any references.

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

(1) every such switch really increases the shortest distances (i.e., attractiveness is improving or profitable); this guarantees monotonic (acyclic) improvement, no backtracking is necessary;
(2) once none of the alternative possible choices is attractive, all possible positive weight cycles MAX can enforce are found (i.e., a stable strategy is optimal).[7]

Although our subexponential algorithm proceeds by making *just one* attractive switch at a time, 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. Other algorithms, making many switches simultaneously are also possible and fit into our framework. We discuss some such algorithms 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. This game always terminates in finitely many iterations with a stable strategy that cannot be improved. Randomization helps in obtaining a subexponential upper bound on the number of iterations.

3.5. Randomization scheme

The order in which attractive switches are made is crucial for the subexponential complexity bound. Section 10 gives an example of a poor switching policy with an exponentially long chain of attractive switches. The space of all positional strategies of MAX in an MPG/LSP instance 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. A facet corresponds to a subgame in which one of the choices of player MAX is fixed in one vertex, and choices in all other vertices are unconstrained. Edges of MAX are in one-to-one correspondence with game facets. In particular, deleting a MAX edge (without creating a sink) determines a subgame in which MAX has fewer choices, and corresponds to eliminating one facet. This allows us to define the algorithm recursively: a solution to a complicated game is determined using a solution to a simpler game with one facet/edge less. Operating in terms of facets is very convenient and makes the connections to combinatorial optimization transparent and explicit.

The algorithm for computing the LSP in an MPG/LSP instance G is as follows:

(1) Start from some strategy σ that guarantees for shortest distances $> -\infty$ in all vertices.[8]
(2) If σ is the only possible MAX strategy in G, return it as optimal.
(3) Otherwise, randomly and uniformly select some facet F of G *not containing* σ. Temporarily throw this facet away, and recursively find a best strategy σ^* on the remainder, $G \backslash F$. This corresponds to deleting a MAX edge not used by σ and finding a best strategy in the resulting subgame.
(4) If σ^* is optimal in G, return it as a result. Optimality is easily checked by computing shortest distances in G_{σ^*} from all vertices of the graph to the sink,[9] and testing whether there is an attractive switch from σ^* to the edge defining F.
(5) Otherwise, make an attractive switch to F, set $G = F$, denote the resulting strategy by σ, and repeat from step 2.

This algorithm incorporates the fairly well-known randomization scheme for combinatorial linear programming due to Matoušek et al. [26,27], although we use it here for a *nonconvex* optimization. Applied to the LSP and MPG 0-mean partition problems, it gives a subexponential $2^{O(\sqrt{n \log n})}$ expected running time bound, where n is the number of MAX

[7] The case when 0-weight cycles are interpreted as good (winning) for MAX is considered in Section 9.

[8] We may assume that G contains no 0-weight cycles. This is important for the LSP problem and can be done without loss of generality; see the proof of Theorem 5.2 and Section 9. This is not needed if we apply the algorithm for finding 0-mean partitions in MPGs, but we have to start with an admissible strategy.

[9] Note that the algorithm recurses by deleting edges/facets, but the set of vertices remains unchanged.

vertices. The essential properties needed for the analysis of [26,27] to work are as follows:

- For every strategy σ and (sub)game G in any run of the algorithm the set of facets F_i of G not containing σ define a *partial ordering* of subgames $G\backslash F_i$ by the value of their best strategies (this value is a vector of shortest distances to the sink under a strategy; see Section 4.1).
- The algorithm always finds a globally optimal strategy in a subgame $G\backslash F_i$ (F_i selected randomly) and makes only monotonic profitable switches (Theorems 5.2 and 5.1, respectively). Thus, after finding an optimal strategy in a subgame $G\backslash F_i$, the algorithm will never revisit any strategies in subgames $G\backslash F_j$, where $G\backslash F_j$ is lower than $G\backslash F_i$ or incomparable with it in the partial ordering above.

It follows that the so-called *hidden dimension* decreases randomly, and the subexponential analysis of [26,27] applies. Another possibility consists in using the slightly more complicated randomization scheme of Kalai [22], as we did in [3] for parity games, which leads to the same subexponential bound.

4. Retreats, admissible strategies, and strategy measure

As explained above, we modify an MPG by allowing MAX to "surrender" in every vertex. Add the retreat vertex t of MIN with a self-cycle of weight 0 and a retreat edge of weight 0 from every MAX vertex 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 (*Admissibility*). A strategy σ of MAX in G is *admissible* if all cycles in G_σ are *positive*, except the cycle over the retreat vertex t.

Our algorithm, starting from the initial "additionally given" admissible strategy, iterates through only admissible strategies σ of MAX (an important invariant). This guarantees that the only losing (for MAX, nonpositive) cycle in G_σ is the self-cycle over t. Once MAX has at least one admissible strategy providing for shortest distances $>-\infty$, he will never want to use any inadmissible strategies, which allow for distances $-\infty$ for some vertices (negative cycles). Hence optimal strategies can be searched for among admissible strategies only.

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-cycle over t. Any other reachable cycle 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 (*Vertex value wrt a strategy*). 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 for each admissible MAX strategy there is an optimal 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., [14].

The relative quality of two admissible strategies is defined componentwise.

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

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

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 shows that the two notions are in fact equivalent.

Definition 4.5 (*Attractiveness, stability, and profitability*). Given an admissible strategy σ, a switch $\sigma[v \mapsto u]$ is

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

A strategy without attractive switches is called *stable*.

Note that

- attractiveness is defined using the values with respect to the current strategy only (which is very easy to check), whereas
- profitability requires to recompute the shortest distances after making a switch and checking admissibility and distance improvements.

Our algorithm proceeds by making attractive switches only.

4.2. Requirements for the measure

The algorithm's correctness and efficiency rely 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, since optimal strategies are among admissible ones; see the explanation after Definition 4.1.
(2) Every attractive switch is profitable and preserves admissibility (Theorem 5.1).
(3) If an admissible strategy has no attractive switches, then there is no better admissible strategy (Theorem 5.2). In other words, every stable strategy is optimal.

Property (2) guarantees monotonicity, termination, and a pseudopolynomial upper bound. Subexponential analysis relies both on (2) and (3). Both properties allow the iterative improvement be guided by attractiveness, easily checkable as soon as the measure has been computed. Testing profitability would otherwise require recomputing the measure for every possible potential switch.

5. Correctness of the measure

In this section we state and prove the two major theorems, guaranteeing that every step of the algorithm (consisting in making an attractive switch) is improving and that the final strategy is the best, respectively. Afterward, we give two corollaries clarifying the relation between attractiveness and profitability.

5.1. Attractiveness implies profitability

Our first theorem states that any attractive switch 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 eventually terminate.

The number of iterations will be at most *pseudopolynomial*, because distances increase in integral steps and every vertex has a natural integral distance limit equal the number of vertices times the absolute weight of the heaviest edge. Either we terminate before reaching this limit, or the distance becomes $+\infty$. Recall that an admissible strategy does not permit any negative *or* 0-weight cycles.

Theorem 5.1 (*Attractive is profitable*). *If σ is an admissible strategy then any strategy σ' obtained by an attractive switch from σ is admissible and better*, *i.e.*, $\sigma' > \sigma$.

Proof. Let σ be admissible, $\sigma(v_i) = v_j$ be the current successor of v_i, and consider an attractive switch for v_i in σ resulting in a new strategy $\sigma' \equiv \sigma[v_i \mapsto v'_j]$, selecting another successor v'_j of v_i. Recall that the (v_i, t)-shortest path problem is expressible by the following linear program:

$$\begin{array}{rl} \text{maximize} & v_i \\ \text{subject to} & t = 0 \quad \text{for sink } t, \\ & v_i \leqslant w(i,j) + v_j \quad \text{for each edge } (v_i, v_j) \text{ of weight } w(i,j). \end{array}$$

Suppose $d_\sigma = (d_\sigma(v_1), \ldots, d_\sigma(v_n))$ is the vector of shortest distances from all vertices to the sink in G_σ. This vector is a feasible solution to the LP above (some components of d_σ may be $+\infty$, meaning that MAX enforces positive cycles, hence infinite distances to the sink; since σ is admissible, there are no nonpositive cycles in G_σ). Let $v_i \leqslant w(i,j) + v_j$ be the constraint corresponding to the edge (v_i, v_j) currently selected by σ, and $v_i \leqslant w(i,j') + v_{j'}$ be the constraint corresponding to an alternative edge $(v_i, v_{j'})$ selected by σ'. In order to speak about attractiveness, $d_\sigma(v_i)$ needs to be finite. Note that the constraint $v_i \leqslant w(i,j) + v_j$ corresponding to the current choice is satisfied by d_σ as equality; otherwise, $d_\sigma(v_i)$ is not the shortest path distance. By attractiveness, $d_\sigma(v_i) < w(i,j') + d_\sigma(v_{j'})$. Thus, replacing (switching) the old constraint $v_i \leqslant w(i,j) + v_i$ with the new $v_i \leqslant w(i,j') + v_{j'}$ results in a new LP, which keeps the old solution d_σ as feasible (thus we get $\sigma' \geqslant \sigma$), but also allows for *increasing* the value of v_i, maybe up to $+\infty$ (thus we get profitability). This holds because the new constraint is not tight for v_i after substituting d_σ. Therefore, increasing v_i may only increase the right-hand sides of other constraints. Note that this argument is based on the special monotone form of the linear constraints for the shortest path problem.

We also have to show that an attractive switch preserves strategy admissibility. By assumption, σ is admissible, therefore, G_σ has no *nonpositive* cycles. Let us prove that $G_{\sigma'}$ possesses the same property. Suppose, toward a contradiction, that $G_{\sigma'}$ has a nonpositive cycle C. This cycle should necessarily contain the vertex v_i in which the switch was made and the new edge $(v_i, v_{j'})$ used by σ'; otherwise, this cycle exists already in G_σ. Consider the path π from $v_{j'}$ to v_i defined by the cycle C. Since π also exists in G_σ (a switch in single vertex v_i was made), $d_\sigma(v_{j'}) \leqslant w(\pi) + d_\sigma(v_i)$, where $w(\pi)$ is the sum of edge weights in π, because d_σ is the vector of shortest distances. By attractiveness, $d_\sigma(v_i) < w(i,j') + d_\sigma(v_{j'})$ (strict!). But the last two inequalities imply $0 < w(\pi) + w(i,j') = w(C)$. Therefore, C is a positive cycle, a contradiction. □

By a similar argument, making simultaneously *several* (rather than just one at a time) attractive switches is profitable, but we do not need it here.

5.2. Stability implies optimality

Our second theorem shows that an admissible MAX strategy with no attractive switches is at least as good as any other admissible strategy, i.e., is winning from every vertex in $G_{>0}$. This means that if we are only looking for the vertices from which MAX can enforce positive weight cycles (when solving MPGs) we can stop as soon as we find an admissible stable strategy. Although there may be also inadmissible strategies that allow MAX to enforce 0-weight cycles, such cycles are of no interest for MAX, because they do not add vertices to $G_{>0}$. Section 9 deals with the case where 0-weight cycles are considered good for MAX.

Theorem 5.2 (*Stable is optimal*). *If σ is an admissible strategy with no attractive switches*, *then $\sigma \geqslant \sigma'$ for every other admissible strategy σ'.*

Proof. Suppose, σ is a stable strategy, and let d_σ be the vector of shortest distances in G_σ to the sink (one component per vertex). Suppose, toward a contradiction, that there is a better than σ admissible strategy σ', providing a longer distance $d_{\sigma'}(v_0) > d_\sigma(v_0)$, in $G_{\sigma'}$, for some vertex v_0. Note that $d_\sigma(v_0)$ should be finite (for a distance improvement to exist) and the shortest path in G_σ from v_0 leads to the sink. It is convenient to let $+\infty \bowtie +\infty + w$, for every relation $\bowtie \in \{\leqslant, =, \geqslant\}$ and finite w.

Since the components of d_σ are shortest distances, the relation $d_\sigma(u) \leqslant d_\sigma(v) + w$ holds for every edge $u \xrightarrow{w} v$ in G_σ, with at least one $\leqslant$ per vertex u satisfied as equality. Therefore, $d_\sigma(u) = d_\sigma(v) + w$ for every edge $u \xrightarrow{w} v$ of MAX in G_σ (tightness), because in G_σ MAX has exactly one edge per vertex. Moreover, $d_\sigma(u) \geqslant d_\sigma(v') + w'$ for every other edge $u \xrightarrow{w'} v'$ of MAX not in σ, by stability of σ (there are no attractive switches).

Let G_σ^f and $G_\sigma^{+\infty}$ be the partition of G_σ into the sets of vertices with finite and infinite shortest distances in d_σ. Thus, v_0 is in G_σ^f. Note that in G (full game graph, not just G_σ) MAX has no edges from G_σ^f to $G_\sigma^{+\infty}$. Otherwise, σ has an attractive switch to this edge (from a finite to infinite value), contradicting the stability assumption. Although MIN may have edges from G_σ^f to $G_\sigma^{+\infty}$, they are not parts of the shortest paths under σ, and we may delete them. Let us also throw away the part $G_\sigma^{+\infty}$, and in G_σ^f for each MIN vertex leave just one outgoing edge $u \xrightarrow{w} v$, for which $d_\sigma(u) = d_\sigma(v) + w$ (such edges exist!). On the contrary, we return back all edges of MAX to the partition G_σ^f. As explained above, they are not leaving G_σ^f. Denote the resulting graph G'.

In G', MAX has all strategies available, while MIN has just one. Nevertheless, we will show that in G' no admissible MAX strategy σ' can improve over $d_\sigma(v_0)$. Suppose the contrary, and make the following *potential transformation* [20] of G', by changing the weight of each edge according to the rule:

$$w'(u, v) = w(u, v) - d_\sigma(u) + d_\sigma(v).$$

After this transformation:

(1) all edges in σ become 0-weight (by tightness);
(2) all other MAX edges become nonpositive (by stability, as explained above);
(3) all edges of MIN become 0-weight;
(4) weights of all cycles remain *unchanged* (by telescoping);
(5) weights of finite paths $v_0, \ldots, v_l$ change by a constant $-d_\sigma(v_0) + d_\sigma(v_l)$ (also by telescoping), hence the relation $(<, =, >)$ between costs of two paths to the sink remains unchanged.

Let MAX use an allegedly better than σ admissible strategy σ' from v_0 in G' undergone the potential transformation above. Since MIN has just one available strategy, σ' defines a unique play, in which every edge traversed is *nonpositive* by the explanation above. This play may be either finite, or infinite (actually both possibilities will lead to contradictions).

Suppose this play is *finite*, terminating in the sink. Let π and π' be the paths from v_0 to the sink t in G' determined by σ and σ', respectively, and functions c, c' give paths costs before and after the potential transformation. We have $c'(\pi') = c(\pi') - d_\sigma(v_0) + d_\sigma(t) = c(\pi') - d_\sigma(v_0) + 0 = c(\pi') - d_\sigma(v_0) = c(\pi') - c(\pi)$. Hence, $c(\pi) + c'(\pi') = c(\pi')$, and $c(\pi) \geqslant c(\pi')$ whenever $c'(\pi') \leqslant 0$. But $c'(\pi') \leqslant 0$, because each edge traversed in π' after the potential transformation is nonpositive. Therefore, $d_\sigma(v_0) = c(\pi) \geqslant c(\pi') = d_{\sigma'}(v_0)$, and σ' provides no improvement over σ, a contradiction.

If the play is *infinite* (i.e., does not terminate in the sink), then the cycle C formed by σ' with the unique MIN strategy has all edges of *nonpositive weight*, i.e., C is a nonpositive weight cycle.[10] But C exists already in $G_{\sigma'}$, and σ' is admissible by assumption (i.e., has no nonpositive cycles), a contradiction. □

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* $G_{>0}$.

[10] Note that if G has no 0-weight cycles at all, then the cycle C is *negative*; thus σ' is no better than σ, a contradiction. We can achieve the absence of 0-weight cycles by multiplying the weight of each edge by $n + 1$ and subtracting 1. Thus, all 0-weight cycles become negative and all other cycles keep their signs. Although this is a possible and useful approach (see Section 9), it multiplies the maximal edge weight W by n, slightly deteriorating the pseudopolynomial upper bound. Instead, to finish the proof we rely on strategy admissibility, which is maintained invariant during attractive switching; cf., Theorem 5.1.

The following property is not necessary for correctness, but clarifies the relation between attractiveness and profitability.

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

Proof. Consider the game (V, E', w'), where $E' = E \backslash \{(u, v) : u \in V_{\mathrm{MAX}} \wedge \sigma(u) \neq v \wedge \sigma'(u) \neq v\}$ and w' is w restricted to E'. In this game σ is stable and both σ, σ' are admissible strategies. Hence, by Theorem 5.2, $\sigma \geqslant \sigma'$. □

Finally, by using 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 strategy 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., [14]. This algorithm runs in $\mathrm{O}(n \cdot |E|)$ time, where $n = |V|$. Every vertex can have its shortest path length improved at most $\mathrm{O}(n \cdot W)$ 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 $\mathrm{O}(n^2 \cdot W)$. Together with the $\mathrm{O}(n \cdot |E|)$ bound per iteration this gives total time $\mathrm{O}(n^3 \cdot |E| \cdot W)$. Here we show how to reuse the shortest distances computed in previous iteration to improve this upper bound by a linear factor to $\mathrm{O}(n^2 \cdot |E| \cdot W)$. 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 $\mathrm{O}(n_i)$ vertices and at most $|E|$ edges; hence it runs in time $\mathrm{O}(n_i \cdot |E|)$. 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 $\mathrm{O}(n^2 \cdot |E| \cdot W)$, 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 $d : V \rightarrow \mathbb{N} \cup \{\infty\}$ 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 ($G, U \subseteq V, d : V \rightarrow \mathbb{N} \cup \{\infty\}$)
(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
(6)                successors $x$ of $u$ in $G_{\sigma'}$
(7)              mark $u$
(8)              $U \leftarrow U \cup \{u\}$
```

Theorem 6.1. *If an attractive 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) $\Rightarrow$ (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 7. 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) $\Rightarrow$ (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) $\Rightarrow$ (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) $\Rightarrow$ (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. $\square$

We thus showed that Algorithm 1 does what it is supposed to. To finish the argument, we show that it runs in time $\mathrm{O}(|E|)$, 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* $\mathrm{O}(|E|)$.

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. $\square$

To improve the efficiency even further, instead of relying on Bellman–Ford's $\mathrm{O}(n|E|)$ algorithm (applies to arbitrary edge weights), we can use a faster Fredman–Tarjan's $\mathrm{O}(|E|+n\log(n))$ implementation of Dijkstra's algorithm (requires *nonnegative* weights) based on Fibonacci heaps. Nonnegative edge weights can be ensured by the following trick involving, once again, potential transformations. Recall that a *potential* for an edge-weighted DAG is a function p on its vertices such that $p(u) \leqslant p(v) + w(u, v)$ for every graph edge (u, v). *Knowing* a potential, we can transform edge weights by $w'(u, v) := w(u, v) - p(u) + p(v) \geqslant 0$ (by the potential property above), thus rendering all edge weights *nonnegative*. Since a potential transformation changes the lengths of all paths from u to v by $p(v) - p(u)$ (telescoping), the shortest paths are preserved under such transformations, and we can safely rely on Dijkstra's algorithm. The only question is how to get a potential. Luckily, it turns out that we implicitly maintain potentials through repeated computations of the shortest paths.

Indeed, let P be an instance of the linear program from the proof of Theorem 5.1 and d be its optimal solution. Recall from the proof that making an attractive switch keeps the vector of shortest distances d, previously optimal for P, as *feasible* for the new LP P' resulting from an attractive switch. But feasibility of d for P' means precisely that d represents the required potential for P'. Thus, shortest distances from one computation of the shortest paths are potentials for the next iteration.

Therefore, the above $\mathrm{O}(n^2 \cdot |E| \cdot W)$ bound with the Bellman–Ford's algorithm can be further improved to $\mathrm{O}(n \cdot (|E| + n \log n) \cdot W)$ with Dijkstra–Fredman–Tarjan's algorithm. This allows for improving pseudopolynomial bounds in Theorems 7.1 and 7.2, as well as in Section 11.

7. Complexity of the algorithm

Section 2 lists several computational problems for MPGs. 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

The 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 thresholds p. Zwick and Paterson [37] 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 $\leqslant p$ and $> p$, respectively. Second, invert the game by interchanging V_{MIN} and V_{MAX} and negating all edge weights, and solve the $(-p)$-mean partition problem. These two partitions correspond to the $< p$ and $\geqslant p$ 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 $\mathrm{O}(n^2 \cdot |E| \cdot W)$. This is the same complexity as for the algorithm by Zwick and Paterson for the splitting into three sets problem [37, Theorem 2.4]. If W is very big, the number of iterations can of course also be bounded by $\prod_{v\in V_{\text{MAX}}} \mathit{outdeg}(v)$, the total number of strategies for MAX.

Using the randomization scheme of Matoušek, Sharir, and Welzl from Section 3.5 we obtain the simultaneous bound $2^{\mathrm{O}(\sqrt{n\log n})}$, *independent* of W. Combining both bounds, we get the following theorem.

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

$$\min(\mathrm{O}(n^2 \cdot |E| \cdot W), \quad 2^{\mathrm{O}(\sqrt{n\log n})}). \tag{1}$$

The min in (1) means that both bounds apply simultaneously, for the same algorithm. One follows by monotonicity and boundedness of distance improvements, the other by randomized analysis. As W increases, the second bound becomes asymptotically better. Note that there are no nontrivial lower bounds for MPGs, and in our experiments with random games both bounds in (1) do not seem tight. After all, over years many people conjectured/claimed MPGs polynomial time solvable.

Note that a more precise estimation replaces n by $|V_{\text{MAX}}|$ in the subexponential bound, because we only consider strategies of MAX and only vertices in V_{MAX} matter. Also, $n \cdot W$ can be replaced by an upper bound on the length of the longest shortest path. 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 computing the values of all vertices of an MPG (ergodic partitioning) reduces to computing a series of p-mean partitions. We first describe the ergodic partition 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 largest edge weights, respectively, and $W = w^+ - w^-$. Then the average weight of any cycle (i.e., the value of any vertex in the MPG) is a rational number with denominator $\leqslant n$ in the interval $[w^-, w^+]$. We can find the value for each vertex by bisecting the interval, until each vertex value is contained in an interval of length $\leqslant 1/n^2$. There is at most one possible value inside each such interval (because the difference between two unequal mean cycle values is at least $1/n(n-1)$), and it can be found easily [20,37]. We start by solving the p-mean partition problems dichotomizing the interval $[w^-, w^+]$ with *integral* thresholds p while it is possible (making slightly uneven partitions if necessary). After that, for bisecting the subintervals of length $\leqslant 1$ we have to deal with rational thresholds p/q for $q \leqslant n^2$. We therefore have to solve the p-mean partition problem for rational nonintegral thresholds p. As can be readily verified, our p-mean partition algorithm directly applies to this case, with a slightly modified complexity analysis.

The combinatorial subexponential bound does not depend on thresholds being integers or rationals, but we need to analyze the depth of the measure, because it is important for the pseudopolynomial bound. Recall that p/q-mean partitioning reduces to 0-mean partitioning by subtracting p/q from edge weights. After subtracting p/q from each (integral) edge weight w, the weight of every path of length k to the sink has the form $\left(\sum_{i=1}^{k} w_i\right) - kp/q$. 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, compared with $n \cdot W$ in the integral case.[11] Thus, solving the 0-mean problem for rational thresholds takes at most n times longer.

During the dichotomy process, we consider subproblems (partitions) with different number of vertices and different thresholds in the range $[w^-, w^+]$. The largest absolute edge weight remains bounded by W, and the total number of vertices in all the subproblems remains n. To find a correct value we need to bisect at most $\mathrm{O}(\log(W \cdot n^2)) = \mathrm{O}(\log W + \log n)$ times. The complexity of one p-mean partitioning $T(n)$ is superlinear in n, so $T(i) + T(j) \leqslant T(i+j)$ (i.e., partitioning resulting in one vertex set empty is the worst case). Hence the total computation time does not exceed $\mathrm{O}((\log W + \log n) \cdot T(n))$. We summarize this in the following theorem.

Theorem 7.2. *The ergodic partition problem for MPGs can be solved in time*

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

Zwick–Paterson's algorithm for this problem has the worst case bound $\mathrm{O}(n^3 \cdot |E| \cdot W)$ [37, Theorem 2.3], which is slightly better for small W, but worse for large W. Note that the algorithm [37] *always* makes $\Theta(n^3 \cdot |E| \cdot W)$ iterations, on every game instance. We are not aware of any MPG instances on which our algorithm makes that many steps. The algorithm described in this section is not strongly subexponential. A strongly subexponential algorithm for the MPG ergodic partition problem is described in [38].

8. Variants of the algorithm

An extension of 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., [6]. In our experiments with large random and nonrandom game instances these algorithms witness extremely fast convergence. In this section we suggest two alternative ways of combining policies.

Initial random multiple switching: Start with the MAX strategy retreating to the sink from each vertex. Prior to running the randomized subexponential algorithm from Section 3.5, 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 subexponential algorithm of Section 3.5. There is a hypothesis due to Williamson Hoke [35] 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 LSP problem exhibits a structure similar to CU-functions (so the subexponential stage may never become necessary). 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, and N of all other vertices. In all vertices in $N \cap V_{\mathrm{MAX}}$, 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 3.5, 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.

[11] Note that a straightforward approach, multiplying all rational edge weights with denominator $q \leqslant n^2$ by q, thus reducing to the integral case, would lead to a worse bound $n^3 \cdot W$.

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.

In the definition of the LSP problem in Section 3.1 we postulated that a 0-weight cycle (which actually makes the sink unreachable) defines a 0 distance from every vertex on the cycle to the sink. This “unnatural” definition was motivated by simplicity and by the LSP application for computing 0-mean partitions in MPGs. This was convenient because 0-weight cycles are not interesting for Max for reaching a *positive* mean value. Moreover, 0-weight cycles are impossible in admissible strategies, and only such strategies are needed (visited) by our algorithms to compute 0-mean partitions in MPGs.

However, in the LSP problem it is more natural to postulate that whenever Max can enforce a 0-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 3.4, 3.5, to compute LSP, and to prove the NP ∩ coNP-membership.

The necessary modifications are 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 0-weight cycles.

This assumption may be done without any 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, all 0-weight cycles, if any, will disappear (become positive), and the lengths l, l' of all paths/simple cycles in G and the modified graph G' will only differ within the factor of $(n+1)$, i.e., $l(n+1) < l' \leqslant l(n+1)+n$. Consequently, all negative/positive cycles 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 of weight $-2\cdot n\cdot W-1$ to the sink. 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 $\leqslant D$. □

The absence of 0-weight cycles assumption is essential in the proof above. The example in Fig. 1 demonstrates a 0-weight cycle, and a stable strategy, which does not provide the optimal (in the LSP sense) solution.

For MPGs we were satisfied with the definition of optimality that only stipulates that Max can secure *positive* weight cycles whenever possible. Thus, in Fig. 1 the strategy “go to t” in the leftmost vertex is MPG-optimal, but not LSP-optimal. In contrast, the LSP-optimality makes 0-weight cycles 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.

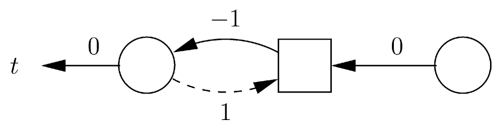

Fig. 1. Switching to the dotted edge is not attractive, but improves the values in all vertices to $+\infty$.

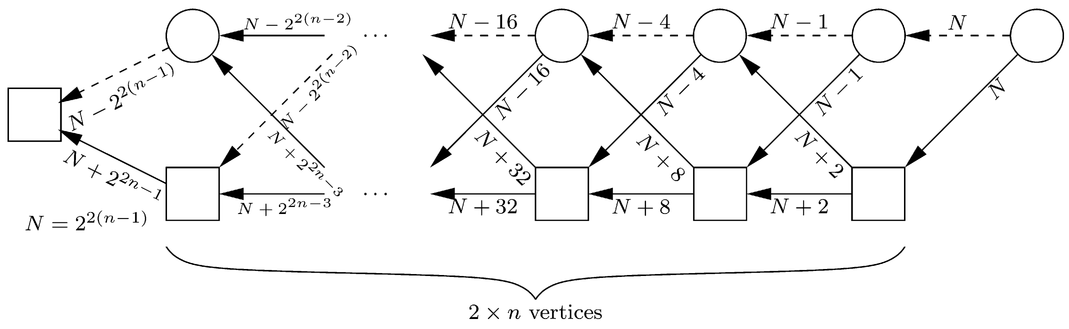

Fig. 2. LSP instances allowing for exponentially long chains of improving switches.

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 shown in Fig. 2, 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.

If the iterative improvement algorithm starts from the initial Max strategy σ_0 shown by solid lines in binary vertices and always makes the *rightmost* attractive single switch, it visits all 2^n possible strategies of Max, as can be readily verified using enough paper and patience.

The LSP instances above can be generated from MPGs as follows. In Fig. 2, add a self-cycle of weight 0 in the leftmost vertex. Add the retreat vertex and edges as explained in Section 3.2. If the algorithm starts by switching from the "retreat everywhere" strategy to the initial strategy σ_0 above, and then always chooses the rightmost attractive single switch, it follows exactly the same exponentially long chain of improving strategies, as in the LSP case. Adding a large N is now essential to keep all the paths nonnegative; otherwise, the initial "switch from the retreat" would be nonattractive.

Actually, the LSP-instances in Fig. 2 are "trivial", because the graphs are acyclic and longest shortest distances are easily computable by dynamic programming in polynomial time (in the left-to-right order), 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 Lebedev mentioned in [20], and kindly provided by Gurvich. Unlike the GKK-algorithm [20], 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 an obvious translation. Parity games are similar to MPGs, 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 [18,7].

Transform a parity game into a MPG by leaving the graph and the vertex partition between players unchanged, and by assigning every vertex[12] 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. Obviously, 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 [3], which has complexity

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

where n is the number of vertices, $|E|$ is the number of edges, and k is the number of colors. Thus, in the worst case the first component in the upper bound may be as high as $\mathrm{O}(|E| \cdot n^5 \cdot 2^n)$, when the number of colors equals the number of vertices. The MPG algorithm improves the first component in the upper bound to $\mathrm{O}(n^2 \cdot |E| \cdot (n/k+1)^k)$, thus gaining a factor of $n^2 \cdot k$. To prove this bound, we assign weights to vertices more sparingly than in the straightforward reduction mentioned above. 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 $W = \mathrm{O}((n/k+1)^k)$.

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 [3] (this is because the measure for every vertex in [3] is a triple, rather a single number, containing additionally the largest cycle 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 $\mathrm{O}(1.618\ldots^n)$, 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 paths (LSP) problem and showed how it can be used as a basis for a discrete (combinatorial) strategy evaluation for MPGs. Similar evaluations were already known for parity [34,3], discounted payoff [31], and simple stochastic games [25], although not discrete for the last two classes of games. Our result implies that any combinatorial strategy improvement policy may be applied to solve MPGs, thus avoiding the difficulties of high precision rational arithmetic involved in reductions to discounted payoff and simple stochastic games, and solving associated linear programs by either a nonstrongly polynomial, or a strongly subexponential subroutine. For comparison, our algorithm uses a strongly polynomial subroutine for shortest paths in edge-weighted directed graphs.

Combining the new strategy evaluation with the algorithm for combinatorial linear programming suggested by Matoušek, Sharir, and Welzl, allowed us to develop the first combinatorial subexponential $2^{\mathrm{O}(\sqrt{n \log n})}$ (and simultaneously pseudopolynomial) algorithm for solving MPGs.

The remaining major open problem is whether any strategy improvement scheme for the games discussed (or any other) has polynomial time complexity.

Since this paper was finished, there were several important developments. Khachiyan with colleagues [36,24] succeeded to prove that a version of the LSP problem restricted to *nonnegative* edge weights is polynomial time solvable. The case of arbitrary weights, needed for the reduction from MPGs, resists. The idea of "controlled" optimization problem, first implemented in the LSP, received a further development, in the form of a more general *controlled linear programming problem* [2,1,9]. A new efficient approach to solving mean payoff and simple stochastic games based on the *linear complementarity problem* has been introduced and investigated in [10,33]. An algorithm based

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

on representing an MPG as a linear program, solving it, and “tightening” a solution to a solution of the game was proposed in [32].

Acknowledgments

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

References

[1] H. Björklund, O. Nilsson, O. Svensson, S. Vorobyov, Controlled linear programming: boundedness and duality, Technical Report DIMACS-2004-56, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, NJ, December 2004 ⟨http://dimacs.rutgers.edu/TechnicalReports/⟩.

[2] H. Björklund, O. Nilsson, O. Svensson, S. Vorobyov, The controlled linear programming problem, Technical Report DIMACS-2004-41, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, NJ, September 2004 ⟨http://dimacs.rutgers.edu/TechnicalReports/⟩.

[3] H. Björklund, S. Sandberg, S. Vorobyov, A discrete subexponential algorithm for parity games, in: H. Alt, M. Habib (Eds.), 20th International Symposium on Theoretical Aspects of Computer Science, STACS’2003, Lecture Notes in Computer Science, vol. 2607, Springer, Berlin, 2003, pp. 663–674.

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

[5] H. Björklund, S. Sandberg, 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/⟩.

[6] H. Björklund, S. Sandberg, S. Vorobyov, Complexity of model checking by iterative improvement: the pseudo-Boolean framework, in: M. Broy, A. Zamulin (Eds.), Andrei Ershov Fifth International Conference “Perspectives of System Informatics”, Lecture Notes in Computer Science, vol. 2890, 2003, pp. 381–394.

[7] H. Björklund, S. Sandberg, S. Vorobyov, Memoryless determinacy of parity and mean payoff games: a simple proof, Theoret. Comput. Sci. 310 (1–3) (2004) 365–378.

[8] H. Björklund, S. Sandberg, S. Vorobyov, Randomized subexponential algorithms for infinite games, Technical Report DIMACS-2004-09, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, NJ, April 2004 ⟨http://dimacs.rutgers.edu/TechnicalReports/⟩.

[9] H. Björklund, O. Svensson, S. Vorobyov, Controlled linear programming for infinite games, Technical Report DIMACS-2005-13, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, NJ, April 2005 ⟨http://dimacs.rutgers.edu/TechnicalReports/⟩.

[10] H. Björklund, O. Svensson, S. Vorobyov, Linear complementarity algorithms for mean payoff games, Technical Report DIMACS-2005-05, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, NJ, February 2005 ⟨http://dimacs.rutgers.edu/TechnicalReports/⟩.

[11] H. Björklund, S. Vorobyov, Combinatorial structure and randomized subexponential algorithms for infinite games, Theoret. Comput. Sci. 349 (3) (2005) 347–360.

[12] A. Condon, The complexity of stochastic games, Inform. and Comput. 96 (1992) 203–224.

[13] A. Condon, On algorithms for simple stochastic games, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 13 (1993) 51–71.

[14] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., MIT Press and McGraw-Hill Book Company, Cambridge, MA, 2001.

[15] A. Ehrenfeucht, J. Mycielski, Positional games over a graph, Notices Amer. Math Soc. 20 (1973) A–334.

[16] A. Ehrenfeucht, J. Mycielski, Positional strategies for mean payoff games, Internat. J. Game Theory 8 (1979) 109–113.

[17] E.A. Emerson, Model checking and the Mu-calculus, in: N. Immerman, Ph.G. Kolaitis (Eds.), DIMACS Series in Discrete Mathematics, vol. 31, 1997, pp. 185–214.

[18] E.A. Emerson, C.S. Jutla, Tree automata, μ-calculus and determinacy, in: Annual IEEE Symposium on Foundations of Computer Science, 1991, pp. 368–377.

[19] E.A. Emerson, C. Jutla, A.P. Sistla, On model-checking for fragments of μ-calculus, in: C. Courcoubetis (Ed.), Computer Aided Verification, Proceedings of the Fifth International Conference, Lecture Notes in Computer Science, vol. 697, 1993, pp. 385–396.

[20] V.A. Gurvich, A.V. Karzanov, L.G. Khachiyan, Cyclic games and an algorithm to find minimax cycle means in directed graphs, USSR Comput. Math. Math. Phys. 28 (5) (1988) 85–91.

[21] A.J. Hoffman, R.M. Karp, On nonterminating stochastic games, Management Sci. 12 (5) (1966) 359–370.

[22] G. Kalai, A subexponential randomized simplex algorithm, in: 24th ACM STOC, 1992, pp. 475–482.

[23] L. Khachiyan, Private communication, DIMACS, April 2004.

[24] L. Khachiyan, V. Gurvich, J. Zhao, Extending Dijkstra’s algorithm to maximize the shortest path by node-wise limited arc interdiction, RUTCOR Research Report RRR 31-2005, RUTCOR, Rutgers Center of Operations Research, October 2005.

[25] W. Ludwig, A subexponential randomized algorithm for the simple stochastic game problem, Inform. and Comput. 117 (1995) 151–155.

[26] J. Matoušek, M. Sharir, M. Welzl, A subexponential bound for linear programming, in: Eighth ACM Symposium on Computational Geometry, 1992, pp. 1–8.

[27] J. Matoušek, M. Sharir, M. Welzl, A subexponential bound for linear programming, Algorithmica 16 (1996) 498–516.
[28] H. Moulin, Extensions of two person zero sum games, J. Math. Anal. Appl. 55 (1976) 490–508.
[29] H. Moulin, Prolongements des jeux à deux joueurs de somme nulle, Bull. Soc. Math. France 45 (1976).
[30] N. Pisaruk, Mean cost cyclical games, Math. Oper. Res. 24 (4) (1999) 817–828.
[31] A. Puri, Theory of hybrid systems and discrete events systems. Ph.D. Thesis, EECS University of Berkeley, 1995.
[32] O. Svensson, S. Vorobyov, LP-polytopes for mean payoff games, RUTCOR Research Report RRR 34-2005, RUTCOR, Rutgers Center of Operations Research, October 2005 ⟨http://rutcor.rutgers.edu/∼rrr⟩. See also: Linear programming polytope and algorithm for mean payoff games. Proc. 2nd Intern. Conference on Algorithmic Aspects in Information and Management (AAIM'06), Springer Lecture Notes in Computer Science, 2006, vol. 4041, pp. 64–78.
[33] O. Svensson, S. Vorobyov, A subexponential algorithm for a subclass of P-matrix generalized linear complementarity problems, Technical Report DIMACS-2005-20, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, NJ, June 2005. See also: Linear complementarity and P-matrices for stochastic Games, Proc. 6th Intern. Andrei Ershov Memorial Conference "Perspectives of System Informatics" (PSI'06), 2006, Springer, Lecture Notes in Computer Science, to appear.
[34] J. Vöge, M. Jurdziński, A discrete strategy improvement algorithm for solving parity games, in: E.A. Emerson, A.P. Sistla (Eds.), CAV'00: Computer-Aided Verification, Lecture Notes in Computer Science, vol. 1855, Springer, Berlin, 2000, pp. 202–215.
[35] K. Williamson Hoke, Completely unimodal numberings of a simple polytope, Discrete Appl. Math. 20 (1988) 69–81.
[36] J. Zhao, V. Gurvich, L. Khachiyan, Extending Dijkstra's algorithm to shortest paths with blocks, Technical Report DIMACS-2005-04, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, NJ, February 2005 ⟨http://dimacs.rutgers.edu/TechnicalReports/⟩.
[37] U. Zwick, M. Paterson, The complexity of mean payoff games on graphs, Theoret. Comput. Sci. 158 (1996) 343–359.
[38] D. Andersson, S. Vorobyov. Fast algorithms for monotonic discounted linear programs with two variables per inequality, Isaac Newton Institute for Mathematical Sciences, Cambridge, UK, May 2006, preprint NI06019-LAA.