A Combinatorial Strongly Subexponential Strategy Improvement Algorithm for Mean Payoff Games*

Henrik Björklund, Sven Sandberg, and Sergei Vorobyov

Uppsala University, Information Technology Department,
Box 337, 751 05 Uppsala, Sweden
vorobyov@csd.uu.se

Abstract. We suggest the first strongly subexponential and purely combinatorial algorithm for mean payoff games. It is based on solving a new "controlled" version of the shortest paths problem. By selecting exactly one outgoing edge in each of the controlled vertices we want to maximize the shortest distances to the unique sink. Mean payoff games easily reduce to this problem. To compute the 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, making attractive switches that seem to increase shortest paths (under the current evaluation). We show that this is a monotonic strategy improvement, and every locally optimal strategy is globally optimal. A careful choice of the next iterate results in a randomized algorithm of complexity $\min(poly(n) \cdot W,\ 2^{O(\sqrt{n \log n})})$, 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).

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) [18,8,22], also known as *cyclic games* [12,19]. 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 NP∩coNP. The well-known parity games, also in NP∩coNP, polynomial time equivalent to model-checking for μ-calculus [10,9], are polynomial time reducible to MPGs. Other well-known games

* Supported by grants from the Swedish Research Council (VR) and the Swedish Foundation for International Cooperation in Research and Higher Education (STINT).

J. Fiala et al. (Eds.): MFCS 2004, LNCS 3153, pp. 673–685, 2004.

with NP∩coNP decision problems, to which MPGs reduce, are *simple stochastic* [6] and *discounted payoff* [20,22] 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 [12,19], and a dynamic programming algorithm solving k-step games for big enough k by Zwick and Paterson [22]. 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 [22] and application of the algorithm from [15] gives subexponential complexity only if the game graph has bounded outdegree. The subexponential algorithms we suggested for simple stochastic games of arbitrary outdegree in [3] 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 in this paper, 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 and iteratively "improves" it with respect to a strategy evaluation function based on computing shortest paths. Iterative strategy improvement algorithms are known for the related simple stochastic [13,7], discounted payoff [20], and parity games [21,2]. 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 and requires only addition and comparison of integers in the same order of magnitude as occurring in the input. 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 [21,2], 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

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

The first bound matches those from [12,22,19], while the second part is an improvement when $n \log n = o(\log^2 W)$.

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 [21,13,7]. These include random single switch, all profitable switches, and random multiple switches [1]. They are simplex-type algorithms, very ef-

ficient 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 states the two main theorems guaranteeing correctness. Finally, Section 6 explains how to improve the cost per iteration and provides complexity results, generalizations, and applications. All proofs and extensions may be found in [4].

2 Mean Payoff Games and Associated Problems

A *mean payoff game* (MPG) 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 $e_1 e_2 e_3 \ldots$ 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 an integer threshold value p. We say that MAX *wins* a play if its value is $> p$, while MIN wins otherwise.

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 MPG 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 [18,8,12,19,5]. Moreover, both players have *uniform* positional strategies securing them optimal payoffs independently of the starting vertex. Accordingly, we consider 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 [8,12].

We address the following algorithmic problems for MPGs.

The Decision Problem. Given a distinguished start vertex and a threshold value p, can MAX guarantee a 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 (MIN, resp.) can secure a value $> p$ ($\leq p$, resp.) starting from every vertex in $G_{>p}$ (in $G_{\leq p}$, resp.)

Ergodic Partition. Compute the value of each vertex of the game. This gives an *ergodic* [12] partition of the vertices into subsets with the same value.

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 ones) smaller by p, and the problem reduces to 0-mean partition. The complexity remains the same for integer thresholds p, and changes slightly for rational ones; see Section 6. Clearly, the p-mean partition subsumes the decision problem. Other problems to which our algorithm extends are ergodic partition and finding optimal strategies [4].

3 A High-Level Description of the Algorithm

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

3.1 The Longest-Shortest Paths Problem (LSP)

The essential step in computing 0-mean partitions can be explained by using a "controlled" version of the well-known *single sink (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 each 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 equal $-\infty$). 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) a set of *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 in G_σ, 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 $+\infty$. For our purposes it suffices to consider a version of the problem above with the following additional input data.

Additionally Given: some strategy τ, which guarantees that in the graph G_τ there are no negative weight cycles.

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

For DAGs, the LSP problem can be solved in polynomial time using dynamic programming. Start by topologically sorting the 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 the LSP Problems

To find a 0-mean 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 in V_{MAX} to t. From now on, we assume G has undergone this transformation. Clearly, we have the following property.

Proposition 1. *Adding a retreat vertex t 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 introduce 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 the retreat t and consider the LSP problem for the resulting graph, with t being the unique sink. Vertices V_{MAX} become controlled, and the initial strategy (the "additionally given" clause on the previous page) selects t in every controlled vertex, guaranteeing that no vertex has distance $-\infty$.[1] We have the following equivalence.

Theorem 1. *The partition $G_{>0}$ in the MPG consists exactly of those vertices for which the longest-shortest path distance to t is $+\infty$.* □

Our algorithm computes the longest shortest paths by iterative improvement to be explained below. Surprisingly (to our knowledge), the longest-shortest path problem was not previously addressed in the literature and its relation to strategy improvement for the MPG problem was not exploited before.[2]

The evaluation of the shortest paths for a fixed positional strategy gives a useful quality measure on strategies that can be used in different iterative improvement schemes, discussed later.

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

[1] Actually, there may exist negative 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.

[2] Leonid Khachiyan (personal communication) considered the similar Blocking Nonpositive Cycles problem, polynomial time equivalent to the decision problem for MPGs [4]. The question is whether MAX can stop MIN from reaching a cycle with nonpositive weight in a graph without sinks. The authors would appreciate any further references on the LSP problem.

above), by making iterative strategy improvements. Once a strategy is fixed, all shortest paths are easily computable, using the Bellman-Ford algorithm. Since there are negative weight edges, the Dijkstra algorithm does not apply. However, in Section 6 we briefly discuss an improvement over the straightforward application of the BF-algorithm.

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 2 and 3, respectively):

1. every attractive switch really increases the shortest distances, i.e., attractive is improving;
2. once none of the switches is attractive, the longest shortest paths are found, i.e., stable is optimal.

Although our algorithm proceeds by making just one attractive switch at a time, other algorithms making many choices simultaneously are also possible and fit into our framework; see Theorem 2.

Another interpretation of our algorithm is game-theoretic. Max chooses 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 essential for the subexponential complexity bound; see [4] 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 provide for a shortest distance $> -\infty$ from each vertex to the sink.

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 [16,17]. When applied to the LSP and MPG problems, it gives a subexponential $2^{O(\sqrt{n \log n})}$ expected running time bound [16,3]. Another possibility would be to use the slightly more complicated randomization scheme of Kalai [14], 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, the same strategy (if any) secures MAX a value > 0 from a vertex in the original and modified games. 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 selects the retreat edge in every vertex, thus "losing only 0" and satisfying the "additionally given" clause of the LSP problem in Section 3.

Definition 1. *A strategy σ of* MAX *in G is* admissible *if all loops in G_σ are positive, except the loop over t.* □

Our algorithm iterates only through admissible strategies of MAX. This guarantees that the only losing (for MAX) loop ever constructible 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; see Section 6 and [4].

Given an admissible strategy σ, the best MIN can 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 negative cycles in G_σ. This is suggestive for defining values.

Definition 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. The value of a strategy is a vector of vertex values.* □

Note that for a fixed admissible strategy of MAX there is a positional counterstrategy of MIN (defined by the shortest paths forest) that guarantees the shortest paths from each vertex to the sink. The relative quality of two admissible strategies is defined componentwise in the strategy value.

Definition 3. *Let σ and σ' be two admissible strategies. Say that σ is* better than σ'*, formally $\sigma > \sigma'$, if $\mathrm{val}_\sigma(v) \geq \mathrm{val}_{\sigma'}(v)$ for all vertices $v \in V$, with strict inequality for at least one vertex. Define $\sigma \geq \sigma'$, if $\sigma > \sigma'$ or they have equal values.* □

The following notation will be useful for describing switches.

Notation. 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

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

We distinct between two kinds of switches later shown equivalent.

Definition 4. *Given an admissible strategy σ, a switch in vertex v to u is:*

1. attractive, *if $w(v, u) + \text{val}_\sigma(u) > \text{val}_\sigma(v)$;*
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. An admissible strategy that has no better (Definition 3) strategy is winning from all vertices in MAX's winning set $G_{>0}$. This follows from definitions.
2. If a strategy has no profitable switches then it is optimal. This is shown in two steps (the two main theorems of the following section).
   a) Every attractive switch is also profitable (Theorem 2).
   b) If an admissible strategy has no attractive switches, then there is no better strategy (Theorem 3).

Properties (2a) and (2b) give another advantage: 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. In addition, we prove the following property, allowing an algorithm to change the strategy in more than one vertex at a time.

3. If several switches in an admissible strategy σ are attractive at the same time, then making any subset of them results in a better strategy (Theorem 2).

5 Correctness of the Measure

In this section we state the two major theorems (proved in [4]), guaranteeing that every step is improving and that the final strategy is the best, respectively.

Theorem 2. *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 $\sigma' \stackrel{def}{=} \sigma[s_1 \mapsto t_1][s_2 \mapsto t_2] \cdots [s_r \mapsto t_r]$, then σ' is admissible and $\sigma' > \sigma$.* □

We also show that every strategy without attractive switches is optimal. This guarantees that the strategy computed by the algorithm is indeed optimal.

Theorem 3. *If σ is an admissible strategy with no attractive switches, then $\sigma \geq \sigma'$ for all admissible strategies σ'.* □

The following corollary equates attractiveness and profitability.

Corollary 1. *If σ is an admissible strategy and an admissible strategy σ' is obtained from σ by one or more* non-attractive *switches, then $\sigma' \leq \sigma$. In particular, a single switch is attractive iff it is profitable.* □

6 Complexity

6.1 Efficient Computation of the Measure

The measure can be straightforwardly computed in time $O(n \cdot |E|)$ by using the Bellman–Ford algorithm for single-sink shortest paths in graphs with possibly negative weights. Since every vertex can have its shortest path length improved at most $O(n \cdot W)$ times, and there are n vertices, the number of switches cannot exceed $O(n^2 \cdot W)$. Together with the $O(n \cdot |E|)$ cost per iteration this gives total time $O(n^3 \cdot |E| \cdot W)$. In [4] we show how to reuse the measure computed in a previous iteration to improve this upper bound to $O(n^2 \cdot |E| \cdot W)$. The idea is to first compute which vertices will change their value, by an efficient backward reachability algorithm, and then run the Bellman–Ford algorithm only on the induced subgraph. Single iteration steps may still need $O(n \cdot |E|)$ time but the amortized time over all iterations is improved. There are no known examples for which the algorithm makes many improvement steps, and therefore the time per iteration becomes a practically significant part of the complexity.

6.2 Worst-Case Analysis

Without changing the asymptotic running time, the basic algorithm of Section 3 solving the *0-mean partition* problem can be extended [4] to solve the p-mean partition, for integer p, as well as a slightly more general problem of *splitting into three sets* [22] around an integer threshold p, with vertices of value $< p$, $= p$, and $> p$, respectively. By 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 [4].

Theorem 4. *The decision, p-mean partition, and splitting into three sets problems for mean payoff games can be solved in time*

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

and space linear in the size of the input. □

6.3 Computing the Ergodic Partitions

By using the standard dichotomy and approximation techniques [12,22] together with the p-mean partitioning algorithm above (extended to deal with rational thresholds; see [4]), we construct an algorithm for finding ergodic partitions in mean payoff games. First, we proceed by dichotomy until the value of each vertex is contained within a unit interval. After that, the threshold in calls to the basic algorithm will be non-integral. The minimal difference between two possible vertex values is $1/n(n-1)$ [12,22]. Thus the dichotomy process can stop as soon as every vertex value is contained in an interval of this size. The thresholds considered will never have denominators larger than n^2, and no vertex can improve its value more than $n^2 \cdot W$ times during one call to the basic algorithm [4]. We thus obtain

Theorem 5. *The ergodic partition problem for MPGs can be solved in time*

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

Zwick's and Paterson's algorithm for this problem has complexity $O(n^3 \cdot |E| \cdot W)$ [22, Theorem 2.3], which is slightly better for small W, but worse for large W.

6.4 NP∩coNP-Membership

The decision version of the LSP problem is restricted to determine whether the longest shortest path from a distinguished vertex s to the sink is bigger than a given bound. Together with MPGs and related games, this is another example of a problem in NP∩coNP [4].

Proposition 2. *The decision version of the LSP problem is in* NP∩coNP. $\square$

6.5 Exponential Sequences of Attractive Switches

Since it is not so easy to come up with "hard" LSP examples, one might conjecture that any sequence of attractive switches converges fast on any LSP problem instance, and consequently MPGs are quickly solvable by polynomially many iterative improvements. However, in [4] we present a family of instances of the LSP and MPG problems, which together with one specific improvement policy (method for selecting an attractive switch in every step), leads to exponentially long sequences of strategy improvements. This shows that the problems are non-trivial, and the choice of the next attractive switch is crucial for the efficiency.

6.6 Variants of the Algorithm

Theorem 2 shows that any combination of attractive switches improves the strategy value, and thus any policy for selecting switches in each iteration will eventually find 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 [1]. In [4] we suggest two alternative ways of combining policies.

6.7 Application to Parity Games

The algorithm described in this paper immediately applies to parity games, after the usual translation [20]. 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 [11,5]. Our previous algorithm from [2] 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 number of colors. A direct analysis of the reduction from parity games to MPGs shows that our new algorithm improves on this bound by a factor n. An additional improvement is achieved by assigning smaller, compared to $(n/k+1)^k$, maximal weight translating a parity game to the MPG.

7 Conclusions

We defined the longest shortest paths (LSP) problem and applied it to create a discrete strategy measure and iterative improvement algorithms for mean payoff games. Similar measures were already known for parity [21,2], discounted payoff [20], and simple stochastic games [15], although not discrete for the last two classes of games. We showed that with our discrete measure any strategy iteration 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 our 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 has other applications. We showed that it belongs to NP∩coNP, and is solvable in expected subexponential randomized time.

The strategy measure presented does not apply to all strategies, only to admissible ones, which do not allow negative 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 [3].

The major open problem is still whether there is a polynomial time strategy improvement scheme for the games discussed.

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. This paper is based on [4].

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. Randomized subexponential algorithms for parity games. Technical Report 2003-019, Department of Information Technology, Uppsala University, April 2003. `http://www.it.uu.se/research/reports/`.
4. H. Bjorklund, S. Sandberg, and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. Technical Report DIMACS-2004-05, DIMACS: Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, NJ, March 2004. `http://dimacs.rutgers.edu/TechnicalReports/`.
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, January 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. A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journ. of Game Theory*, 8:109–113, 1979.
9. 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.
10. 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.
11. 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.
12. 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.
13. A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.
14. G. Kalai. A subexponential randomized simplex algorithm. In *24th ACM STOC*, pages 475–482, 1992.
15. W. Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117:151–155, 1995.
16. J. Matoušek, M. Sharir, and M. Welzl. A subexponential bound for linear programming. In *8th ACM Symp. on Computational Geometry*, pages 1–8, 1992.
17. J. Matoušek, M. Sharir, and M. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16:498–516, 1996.

18. H. Moulin. Extensions of two person zero sum games. *J. Math. Analysis and Applic.*, 55:490–508, 1976.
19. N. Pisaruk. Mean cost cyclical games. *Mathematics of Operations Research*, 24(4):817–828, 1999.
20. A. Puri. *Theory of hybrid systems and discrete events systems*. PhD thesis, EECS Univ. Berkeley, 1995.
21. 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.
22. U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158:343–359, 1996.