A Discrete Subexponential Algorithm for Parity Games*

Henrik Björklund, Sven Sandberg, and Sergei Vorobyov

Computing Science Department, Uppsala University, Sweden

Abstract. We suggest a new randomized algorithm for solving parity games with worst case time complexity roughly

$$\min\left(O\left(n^3\cdot\left(\frac{n}{k}+1\right)^k\right),\ 2^{O(\sqrt{n\log n})}\right),$$

where n is the number of vertices and k the number of colors of the game. This is comparable with the previously known algorithms when the number of colors is small. However, the subexponential bound is an advantage when the number of colors is large, $k=\Omega(n^{1/2+\varepsilon})$.

1 Introduction

Parity games are infinite games played on finite directed bipartite leafless graphs, with vertices colored by integers. Two players alternate moving a pebble along edges. The goal of Player 0 is to ensure that the biggest color visited by the pebble infinitely often is even, whereas Player 1 tries to make it odd. The complexity of determining a winner in parity games, equivalent to the Rabin chain tree automata non-emptiness, as well as to the μ-calculus[1] model checking [5,3], is a fundamental open problem in complexity theory [11]. The problem belongs to NP∩coNP, but its PTIME-membership status remains widely open. All known algorithms for the problem are exponential, with an exception of [12] when the number of colors is large and games are binary.

In this paper we present a new discrete, randomized, subexponential algorithm for parity games. It combines ideas from iterative strategy improvement based on randomized techniques of Kalai [9] for Linear Programming and of Ludwig [10] for simple stochastic games, with discrete strategy evaluation similar to that of Vöge and Jurdziński [15]. Generally, algorithms for parity games are *exponential* in the number of colors k, which may be as big as the number n of vertices. For most, exponentially hard input instances are known [4,3,2,14, 8]. Our algorithm is subexponential in n. Earlier we suggested a subexponential algorithm [12], similar to [10], but based on graph optimization rather than linear programming subroutines. Both algorithms [10,12] become exponential

* Supported by Swedish Research Council Grants "Infinite Games: Algorithms and Complexity", "Interior-Point Methods for Infinite Games".

[1] One of the most expressive temporal logics of programs [3].

H. Alt and M. Habib (Eds.): STACS 2003, LNCS 2607, pp. 663–674, 2003.

for graphs with unbounded vertex outdegree. The present paper eliminates this drawback. There is a well-known reduction from parity to mean payoff games, but the best known algorithms for the latter [7,16,13] are known to be exponential (pseudopolynomial). Reducing parity to simple stochastic games [16] leads to manipulating high-precision arithmetic and to algorithms invariably subexponential in the number of vertices, which is worse than an exponential dependence on colors when colors are few.

A recent iterative strategy improvement algorithm [15] uses a discrete strategy evaluation involving game graph characteristics like colors, sets of vertices, and path lengths. Despite a reportedly good practical behavior, the only known worst-case bound for this algorithm is exponential in the number of vertices, independently of the number of colors.

Our new algorithm avoids any reductions and directly applies to parity games of arbitrary outdegree. We use a discrete strategy evaluation measure similar to, but more economical than the one used in [15]. Combined with Kalai's and Ludwig's randomization schemes this provides for a worst case bound that is simultaneously subexponential in the number of vertices and exponential in the number of colors. This is an advantage when the colors are few.

Outline. After preliminaries on parity games, we start by presenting a simpler, Ludwig-style randomized algorithm in combination with an abstract discrete measure on strategies. This simplifies motivation, exposition, and definitions for the specific tight discrete measure we build upon. We then proceed to a more involved Kalai-style randomized algorithm allowing for arbitrary vertex outdegrees. All proofs can be found in [1].

2 Parity Games

Definition 1 (Parity Games). *A* parity game *is an infinite game played on a finite directed bipartite leafless graph* $G[n,k] = (V_0, V_1, E, c)$*, where* $n = |V_0 \cup V_1|$*,* $E \subseteq (V_0 \times V_1) \cup (V_1 \times V_0)$*,* $k \in \mathbb{N}$*, and* $c : V_0 \cup V_1 \to \{1, \ldots, k\}$ *is a coloring function. The sizes of* V_0 *and* V_1 *are denoted by* n_0 *and* n_1*, respectively. Starting from a vertex, Player 0 and 1 alternate moves constructing an infinite sequence of vertices; Player* i *moves from a vertex in* V_i *by selecting one of its successors. Player 0 wins if the highest color encountered infinitely often in this sequence is even, while Player 1 wins otherwise.*[2] □

Parity games are known to be *determined*: from each vertex exactly one player has a winning positional strategy, selecting a unique successor to every vertex [5]. All our results straightforwardly generalize to the non-bipartite case.

A *binary* parity game is a game where the vertex outdegree is at most two.

[2] We systematically use n for the number of vertices and k for the number of colors; consequently we usually skip $[n,k]$ in $G[n,k]$.

3 Ludwig-Style Algorithm with a Well-Behaved Measure

Every positional strategy of Player 0 in a binary parity game can be associated with a corner of the n_0-dimensional boolean hypercube. If there is an appropriate way of assigning values to strategies, then we can apply an algorithm similar to [10] to find the best strategy as follows.

1. Start with some strategy σ_0 of Player 0.
2. Randomly choose a facet F of the hypercube, containing σ_0.
3. Recursively find the best strategy σ' on F.
4. Let σ'' be the neighbor of σ' on the opposite facet $\overline{F}$. If σ' is better than σ'', then return σ'. Else recursively find the optimum on $\overline{F}$, starting from σ''.

To guarantee correctness and subexponentiality, the assignment cannot be completely unstructured. Also, evaluating strategies is costly, so a full evaluation should only be performed for strategies that are really better than the current one. In subsequent sections, we present a function Evaluate that given a strategy σ returns an assignment ν_σ of values to vertices of the game that meets the following criteria (where $\prec$ is a comparison operator on the values).

Stability. Let $\sigma(v)$ be the successor of vertex v selected by strategy σ and let $\overline{\sigma}(v)$ be the other successor of v. If $\nu_\sigma(\sigma(v)) \succeq \nu_\sigma(\overline{\sigma}(v))$ for all vertices v of Player 0, then σ is optimal (maximizes the winning set of Player 0).

Uniqueness of optimal values. All optimal strategies have the same valuation. (This is essential for a subexponential bound.)

Profitability. Suppose that $\nu_\sigma(\sigma(u)) \succeq \nu_\sigma(\overline{\sigma}(u))$ for every vertex $u \in V_0 \setminus v$ and $\nu_\sigma(\sigma(v)) \prec \nu_\sigma(\overline{\sigma}(v))$ (attractiveness). Let σ' be the strategy obtained by changing σ only at v (single switch), and let $\nu_{\sigma'}$ be its valuation. Then $\nu_\sigma(v) \prec \nu_{\sigma'}(v)$ and $\nu_\sigma(u) \preceq \nu_{\sigma'}(u)$ for all other vertices u (profitability).

The Ludwig-style algorithm with Evaluate applies to solving binary parity games. The evaluation function has the benefit that in step 4 of the algorithm, σ'' does not have to be evaluated, unless the recursive call is needed.

Ludwig [10] shows that his algorithm for simple stochastic games has a $2^{O(\sqrt{n_0})}$ upper bound on the expected number of improvement steps. With only minor modifications, the same proof shows that the Ludwig-style algorithm together with our Evaluate function has the same bound for parity games.

The value space of Evaluate allows at most $O(n^3 \cdot (n/k+1)^k)$ improvement steps. Since the algorithm makes only improving switches, the upper bound on the number of switches of the combined approach is

$$\min\left(O\left(n^3 \cdot \left(\frac{n}{k}+1\right)^k\right), 2^{O(\sqrt{n_0})}\right).$$

Any parity game reduces to a binary one. This allows for a subexponential algorithm for games with subquadratic total outdegree. For arbitrary games the reduction gives a quadratic explosion in the number of vertices and the Ludwig-style algorithm becomes exponential. In Section 10 we achieve a subexponential bound by employing a more involved randomization scheme from [9].

4 Strategies and Values

For technical reasons, each vertex is assigned a unique value, called a *tint*.

Definition 2 (Tints). *A bijection* $\mathbf{t} : V \to \{1, \ldots, n\}$ *such that* $c(u) \leq c(v) \Rightarrow \mathbf{t}(u) \leq \mathbf{t}(v)$ *assigns tints to vertices. The color of a tint* $s \in \{1, \ldots, n\}$ *equals* $c(\mathbf{t}^{-1}(s))$. □

Note that tints of vertices of the same color form a consecutive segment of natural numbers. Subsequently we identify vertices with their tints, and slightly abuse notation by writing $c(t)$ for the color of the vertex with tint t.

Definition 3 (Winning and Losing Colors and Tints). *Color* i *is* winning for Player 0 *(Player 1 resp.) if it is even (odd resp.). Tint* t *is* winning for Player 0 *(Player 1 resp.) if its color* $c(t)$ *is. A color or tint is* losing *for a player if it is winning for his adversary.* □

Note that tints of different colors are ordered as these colors. Within the same winning (resp. losing) color the bigger (resp. smaller) tint is better for Player 0.

In this section we define the 'value' of a strategy – the target to be iteratively improved. An elementary improvement step is as follows: given a strategy σ of Player 0, its value is a vector of values of all vertices of the game, assuming that the adversary Player 1 applies an 'optimal' response counterstrategy τ against σ. The value of each vertex is computed with respect to the pair of strategies (σ, τ), where the optimality of τ is essential for guiding Player 0 in improving σ. We delay the issue of constructing optimal counterstrategies until Section 9, assuming for now that Player 1 always responds with an optimal counterstrategy.

Definition 4. *A* positional strategy *for Player 0 is a function* $\sigma : V_0 \to V_1$, *such that if* $\sigma(v) = v'$, *then* $(v, v') \in E$. *Saying that Player 0 fixes his positional strategy means that he deterministically chooses the successor* $\sigma(v)$ *each time the play comes to* v, *independently of the history of the play. Positional strategies for Player 1 are defined symmetrically.* □

Assumption. From now on we restrict our attention to positional strategies only. The iterative improvement proceeds by improving positional strategies for Player 0, and this is justified by Profitability, Stability, and Uniqueness Theorems 20, 22, and 23 below. The fact that Player 1 may also restrict himself to positional strategies is demonstrated in Section 9.

Definition 5 (Single Switch). *A* single switch *in a positional strategy* σ *of Player 0 is a change of successor assignment of* σ *in exactly one vertex.* □

When the players fix their positional strategies, the trace of any play is a simple path leading to a simple loop. Roughly speaking, the value of a vertex with respect to a pair of positional strategies consists of a loop value (major tint) and a path value (a record of the numbers of more significant colors on the path to the major, plus the length of this path), as defined below.

Notation 6 *Denote by* V^i *the set of vertices of* color i *and by* $V^{>t}$ *the set of vertices with* tints *numerically bigger than* t. □

Definition 7 (Traces, Values). *Suppose the players fix positional strategies σ and τ, respectively. Then from every vertex u_0 the* trace of the play *takes a simple δ-shape form: an initial simple path (of length $q \geq 0$, possibly empty) ending in a loop:*

$$u_0, u_1, \ldots u_q, \ldots, u_r, \ldots u_s = u_q, \tag{1}$$

where all vertices u_i are distinct, except $u_q = u_s$. The vertex u_r with the maximal tint t on the loop $u_q, \ldots, u_r, \ldots, u_s = u_q$ in (1) is called principal *or* major.

Values for Non-principal Vertices. *If the vertex u_0 is non-principal, then its* value $\nu_{\sigma,\tau}(u_0)$ *with respect to the pair of strategies (σ, τ) has the form (L, P, p) and consists of:*

Loop Value (Tint) *L equal to the principal tint t;*
Path Color Hit Record Relative to t *defined as a vector*

$$P = (m_k, m_{k-1}, \ldots, m_l, \underbrace{0, \ldots, 0}_{l-1 \text{ times}}),$$

where $l = c(t)$ is the color of the principal tint t, and

$$m_i = \left|\{u_0, u_1, \ldots u_{r-1}\} \cap V^i \cap V^{>t}\right|$$

is the number of vertices of color $i \geq l$ on the path to from u_0 to the major u_r (except that for the color l of the major we account only for the vertices with tint bigger than t.)
Path Length $p = r$.

Values for Principal Vertices. *If the vertex u_0 is principal (case $q = r = 0$ in (1)) then its value $\nu_{\sigma,\tau}(u_0)$ with respect to the pair of strategies (σ, τ) is defined as $(t, \bar{0}, s)$, where $\bar{0}$ is a k-dimensional vector of zeros.*

Path Value *is a pair (P, p), where (t, P, p) is a vertex value.* □

The reason of the complexity of this definition is to meet the criteria enumerated in Section 3 and simultaneously obtain the 'tightest possible' bound on the number of iterative improvements. It is clear that such a bound imposed by the value measure from Definition 7 is $O(n^3 \cdot (n/k+1)^k)$.

5 Value Comparison and Attractive Switches

Definition 8 (Preference Orders). *The* preference order *on colors (as seen by Player 0) is as follows: $c \prec c'$ iff $(-1)^c \cdot c < (-1)^{c'} \cdot c'$.*

The preference order *on tints (as seen by Player 0) is as follows:*

$$t \prec t' \text{ iff } (-1)^{c(t)} \cdot t < (-1)^{c(t')} \cdot t'. \qquad \square$$

We thus have $\ldots \prec 5 \prec 3 \prec 1 \prec 0 \prec 2 \prec 4 \prec \ldots$ on colors.

Definition 9 ('Lexicographic' Ordering). *Given two vectors (indexed in descending order from the maximal color k to some $l \geq 1$)*

$$P = (m_k, m_{k-1}, \ldots, m_{l+1}, m_l),$$
$$P' = (m'_k, m'_{k-1}, \ldots, m'_{l+1}, m'_l),$$

define $P \prec P'$ if the vector

$$\big((-1)^k \cdot m_k, (-1)^{k-1} \cdot m_{k-1}, \ldots, (-1)^{l+1} \cdot m_{l+1}, (-1)^l \cdot m_l\big)$$

is lexicographically smaller (assuming the usual ordering of integers) than the vector $\big((-1)^k \cdot m'_k, (-1)^{k-1} \cdot m'_{k-1}, \ldots, (-1)^{l+1} \cdot m'_{l+1}, (-1)^l \cdot m'_l\big)$. □

Definition 10 (Path Attractiveness). *For two vertex values (t, P_1, p_1) and (t, P_2, p_2), where t is a tint, $l = c(t)$ is its color, and*

$$P_1 = (m_k, m_{k-1}, \ldots, m_{l+1}, m_l, \ldots, m_1),$$
$$P_2 = (m'_k, m'_{k-1}, \ldots, m'_{l+1}, m'_l, \ldots, m'_1),$$

say that the path value (P_2, p_2) is more attractive[3] modulo t *than the path value (P_1, p_1), symbolically $(P_1, p_1) \prec_t (P_2, p_2)$, if:*

1. *either $(m_k, m_{k-1}, \ldots, m_{l+1}, m_l) \prec (m'_k, m'_{k-1}, \ldots, m'_{l+1}, m'_l)$,*
2. *or $(m_k, m_{k-1}, \ldots, m_{l+1}, m_l) = (m'_k, m'_{k-1}, \ldots, m'_{l+1}, m'_l)$ and*

$$(-1)^l \cdot p_1 > (-1)^l \cdot p_2. \tag{2}$$

Remark 11. Note that (2) means that shorter (resp. longer) paths are better for Player 0 when the loop tint t is winning (resp. losing) for him. □

Definition 12 (Value Comparison). *For two vertex values define $(t_1, P_1, p_1) \prec (t_2, P_2, p_2)$ if*

1. *either $t_1 \prec t_2$,*
2. *or $t_1 = t_2 = t$, and $(P_1, p_1) \prec_t (P_2, p_2)$.* □

Definition 13 (Vertex Values). *The value $\nu_\sigma(v)$ of a vertex v with respect to a strategy σ of Player 0 is the minimum of the values $\nu_{\sigma,\tau}(v)$, taken over all strategies τ of Player 1.* □

In Section 9 we show that the 'minimum' in this definition can be achieved in all vertices simultaneously by a positional strategy τ of Player 1.

Definition 14. *The value of a strategy σ of Player 0 is a vector of values of all vertices with respect to the pair of strategies (σ, τ), where τ is an optimal response counterstrategy of Player 1 against σ; see Section 9.* □

[3] In the sequel, when saying "attractive", "better", "worse", etc., we consistently take the viewpoint of Player 0.

Definition 15. *A strategy* σ' improves σ*, symbolically* $\sigma \prec \sigma'$*, if* $\nu_\sigma(v) \preceq \nu_{\sigma'}(v)$ *for all vertices* v *and there is at least one vertex* u *with* $\nu_\sigma(u) \prec \nu_{\sigma'}(u)$*.* □

Proposition 16. *The relations* $\prec$ *on colors, tints, values, and strategies, and* $\prec_t$ *on path values (for each* t*) are transitive.* □

Our algorithms proceed by single attractive switches only.

Definition 17 (Attractive Switch). *Let* (t_1, P_1, p_1) *and* (t_2, P_2, p_2) *be the values with respect to* σ *of vertices* v_1 *and* v_2*, respectively. Consider a single switch in strategy* σ *of Player 0, consisting in changing the successor of* v *with respect to* σ *from* v_1 *to* v_2*. The switch is* attractive *if* $(t_1, P_1, p_1) \prec (t_2, P_2, p_2)$*.* □

Remark 18. Note that deciding whether a switch is attractive (when comparing values of its successors) we do not directly account for the color/tint of the current vertex. However, this color/tint may be eventually included in the values of successors possibly dependent on the current vertex.

6 Profitability of Attractive Switches

Our algorithms proceed by making single attractive switches. Attractiveness is established locally, by comparing values of a vertex successors with respect to a current strategy; see Definition 17.

Definition 19. *Say that a single switch from* σ *to* σ' *is* profitable *if* $\sigma \prec \sigma'$*.* □

Profitability of attractive switches is crucial for the efficiency, correctness, and termination of our algorithms, as explained in Sections 3 and 10. Profitability is a consequence of the the preceding complicated definitions of values, value comparison, and strategy evaluation.

Theorem 20 (Profitability). *Every attractive switch is profitable:*

1. *it increases the value of the vertex where it is made, and*
2. *all other vertices either preserve or increase their values,*

i.e., the switch operator is monotone. □

7 Stability Implies Optimality

The Main Theorem 22 of this section guarantees that iterative improvement can terminate once a strategy with no attractive switches is found. In more general terms it states that every local optimum is global. This is one of the main motivations for the complex strategy evaluation definitions.

Definition 21. *Say that a strategy* σ *is* stable *if it does not have attractive switches with respect to* $\tau(\sigma)$*, an optimal counterstrategy of Player 1.* □

In Section 9 we show that all optimal counterstrategies provide for the *same* values. Thus stability of σ in the previous definition may be checked after computing any optimal counterstrategy $\tau(\sigma)$.

Theorem 22 (Stability). *Any stable strategy of Player 0 is optimal: vertices with loop values of even colors form the winning set of Player 0.* □

8 Uniqueness of Optimal Values

Theorem 22 does not guarantee that different stable (hence optimal) strategies provide for the same (or even comparable) vectors of values for the game vertices. The uniqueness of optimal values is however crucial for the subexponential complexity analysis of Sections 3 and 10, and is provided by the following

Theorem 23 (Uniqueness). *Any two stable strategies of Player 0 give the same values for all vertices of the game.* □

9 Computing Optimal Counterstrategies

Let G_σ be the game graph induced by a positional strategy σ of Player 0 (delete all edges of Player 0 not used by σ). Partition vertices of G_σ into classes L_t containing the vertices from which Player 1 can ensure the loop tint t, but cannot guarantee any worse loop tint. This can be done by using finite reachability in G_σ as follows. For each tint t in $\prec$-ascending order, check whether t can be reached from itself without passing any tint $t' > t$. If so, Player 1 can form a loop with t as major. Since the tints are considered in $\prec$-ascending order, t will be the best loop value Player 1 can achieve for all vertices from which t is reachable. Remove them from the graph, place them in class L_t, and proceed with the next tint.

For each class L_t, use dynamic programming to calculate the values of 1-optimal paths of different lengths from each vertex to t. For each vertex, the algorithm first computes the optimal color hit record (abbreviated *chr* in the algorithm) over all paths of length 0 to the loop major (∞ for each vertex except t). Then it calculates the color hit record of optimal paths of length one, length two, and so forth. It uses the values from previous steps in each step except the initial one.[4]

Algorithm 1: Computing path values within a class L_t.

PathValues(L_t)

(1) $t.chr[0] \leftarrow (0, \ldots, 0)$
(2) **foreach** vertex $v \in L_t$ except t
(3) $v.chr[0] \leftarrow \infty$
(4) **for** $i \leftarrow 1$ **to** $|L_t| - 1$
(5) **foreach** vertex $v \in L_t$ except t
(6) $v.chr[i] \leftarrow \min_{\prec_t}\{\textsc{AddColor}(t, v'.chr[i-1], \mathbf{t}(v)) : v' \in L_t \text{ is a successor of } v)\}$
(7) **foreach** vertex $v \in L_t$ except t
(8) $v.pathvalue \leftarrow \min_{\prec_t}\{(v.chr[i], i) : 0 \leq i < |L_t|\}$
(9) $t.pathvalue \leftarrow \min_{\prec_t}\{v.pathvalue : v \in L_t \text{ is a successor of } t\}$
(10) $t.pathvalue.pathlength \leftarrow t.pathvalue.pathlength + 1$

[4] The algorithm assumes the game is bipartite; in particular, t in line (9) cannot be a successor of itself. It can be straightforwardly generalized for the non-bipartite case.

The function AddColor takes a tint, a color hit record, and a second tint. If the second tint is bigger than the first one, then AddColor increases the position in the vector representing the color of the second tint. The function always returns ∞ when the second argument has value ∞.

The algorithm also handles non-binary games.

Lemma 24 (Algorithm Correctness). *The algorithm correctly computes values of optimal paths. Moreover:*

1. *optimal paths are simple;*
2. *the values computed are consistent with an actual positional strategy that guarantees loop value t.* □

Lemma 25 (Algorithm Complexity). *The algorithm for computing an optimal counterstrategy runs in time $O(|V| \cdot |E| \cdot k)$, where $|V|$ is the number of vertices of the graph, $|E|$ is the number of edges, and k is the number of colors.*

10 Kalai-Style Randomization for Games with Unbounded Outdegree

As discussed in Section 3, any non-binary parity game reduces to a binary one, and the Ludwig-style algorithm applies. However, the resulting complexity gets worse and may become exponential (rather than subexponential) due to a possibly quadratic blow-up in the number of vertices. In this section we describe a different approach relying on the randomization scheme of Kalai [9,6] used for Linear Programming. This results in a subexponential randomized algorithm directly applicable to parity games of arbitrary outdegree, without any preliminary translations. When compared with reducing to the binary case combined with the Ludwig-style algorithm of Section 3, the algorithm of this section provides for a better complexity when the total number of edges is roughly $\Omega(n \log n)$.

Games, Subgames, and Facets. Let $\mathcal{G}(d, m)$ be the class of parity games with vertices of Player 0 partitioned into two sets U_1 of outdegree one and U_2 of an arbitrary outdegree $\delta(v) \geq 1$, with $|U_2| = d$ and $m \geq \sum_{v \in U_2} \delta(v)$. Informally, d is the dimension (number of variables to determine), and m is a bound on the number of edges (constraints) to choose from. The numbers of vertices and edges of Player 1 are unrestricted.

Given a game $G \in \mathcal{G}(d, m)$, a vertex $v \in U_2$ of Player 0, and an edge e leaving v, consider the (sub)game F obtained by fixing e and deleting all other edges leaving v. Obviously, $F \in \mathcal{G}(d-1, m-\delta(v))$ and also, by definition, $F \in \mathcal{G}(d, m)$, which is convenient when we need to consider a strategy in the subgame F as a strategy in the full game G in the sequel. Call the game F a *facet* of G.

If σ is a positional strategy and e is an edge leaving a vertex v of Player 0, then we define $\sigma[e]$ as the strategy coinciding with σ in all vertices, except possibly v, where the choice is e. If σ is a strategy in $G \in \mathcal{G}(d, m)$, then a facet F is *σ-improving* if some *witness* strategy σ' in the game F (considered as a member of $\mathcal{G}(d, m)$) satisfies $\sigma \prec \sigma'$.

The Algorithm takes a game $G \in \mathcal{G}(d, m)$ and an initial strategy σ_0 as inputs, and proceeds in three steps.

1. Collect a set M containing r pairs (F, σ) of σ_0-improving facets F of G and corresponding witness strategies $\sigma \succ \sigma_0$.
   (The parameter r specified later depends on d and m. Different choices of r give different algorithms. The subroutine to find σ_0-improving facets is described below. This subroutine may find an optimal strategy in G, in which case the algorithm returns it immediately.)
2. Select one pair $(F, \sigma_1) \in M$ uniformly at random. Find an optimal strategy σ in F by applying the algorithm recursively, taking σ_1 as the initial strategy.[5]
3. If σ is an optimal strategy also in G, then return σ. Otherwise, let σ' be a strategy differing from σ by an attractive switch. Restart from step 1 using the new strategy σ' and the same game $G \in \mathcal{G}(d, m)$.

The algorithm terminates because each solved subproblem starts from a strictly better strategy. It is correct because it can only terminate by returning an optimal strategy.

How to Find Many Improving Facets. In step 1 the algorithm above needs to find either r different σ_0-improving facets or an optimal strategy in G. To this end we construct a sequence $(G^0, G^1, \ldots, G^{r-d})$ of games, with $G^i \in \mathcal{G}(d, d+i)$. All the $d + i$ facets of G^i are σ_0-improving; we simultaneously determine the corresponding witness strategies σ^j optimal in G^j. The subroutine returns r facets of G, each one obtained by fixing one of the r edges in $G^{r-d} \in \mathcal{G}(d, r)$. All these are σ_0-improving by construction.

Let e be the target edge of an attractive switch from σ_0. (If no attractive switch exists, then σ_0 is optimal in G and we are done.) Set G^0 to the game where all choices are fixed as in $\sigma_0[e]$, and all other edges of Player 0 in G are deleted. Let σ^0 be the unique, hence optimal, strategy $\sigma_0[e]$ in G^0. Fixing any of the d edges of σ^0 in G defines a σ_0-improving facet of G with σ^0 as a witness.

To construct G^{i+1} from G^i, let e be the target edge of an attractive switch from σ^i in G. (Note that σ^i is optimal in G^i but not necessarily in the full game G. If it is, we terminate.) Let G^{i+1} be the game G^i with e added, and compute σ^{i+1} as an optimal strategy in G^{i+1}, by a recursive application of the algorithm above. Note that fixing any of the $i+1$ added target edges defines a σ_0-improving facet of G. Therefore, upon termination we have r such facets.

Complexity Analysis. The following recurrence bounds the expected number of calls to the algorithm solving a game in $\mathcal{G}(d, m)$ in the worst case:

$$T(d, m) \leq \sum_{i=d}^{r} T(d, i) + T(d-1, m-2) + \frac{1}{r} \sum_{i=1}^{r} T(d, m-i) + 1$$

[5] Rather than computing all of M, we may select a random number $x \in \{1, \ldots, r\}$ before step 1 and compute only x improving facets. This is crucial in order for the computed sequence of strategies to be strictly improving, and saves some work.

The first term represents the work of finding r different σ_0-improving facets in step 1. The second term comes from the recursive call in step 2. [6] The last term accounts for step 3 and can be understood as an average over the r equiprobable choices made in step 2, as follows. All facets of G are partially ordered by the values of their optimal strategies (although this order is unknown to the algorithm). Optimal values in facets are unique by Theorem 23, and the algorithm visits only improving strategies. It follows that all facets possessing optimal strategies that are worse, equal, or incomparable to the strategy σ of step 2 will never be visited in the rest of the algorithm. In the *worst* case, the algorithm selects the r worst possible facets in step 1. Thus, in the worst case, in step 3 it solves a game in $\mathcal{G}(d, m-i)$ for $i = 1, \ldots, r$, with probability $1/r$. This justifies the last term.

Kalai uses $r = \max(d, m/2)$ in step 1 to get the best solution of the recurrence. The result is subexponential, $m^{O(\sqrt{d/\log d})}$. By symmetry, we can choose to optimize a strategy of the player possessing fewer vertices.

Let n_i denote the number of vertices of player i. Since m is bounded above by the maximal number of edges, $(n_0 + n_1)^2$, and $d \leq \min(n_0, n_1)$, we get

$$\min\left\{2^{O\left((\log n_1)\cdot\sqrt{n_0/\log n_0}\right)}, 2^{O\left((\log n_0)\cdot\sqrt{n_1/\log n_1}\right)}\right\}$$

as the bound on the number of calls to the algorithm. Combining it with the bound on the maximal number of improving steps allowed by our measure yields

$$\min\left\{2^{O\left((\log n_1)\cdot\sqrt{n_0/\log n_0}\right)}, 2^{O\left((\log n_0)\cdot\sqrt{n_1/\log n_1}\right)}, O\left(n^3\cdot\left(\frac{n}{k}+1\right)^k\right)\right\}.$$

If $n_0 = O(poly(n_1))$ and $n_1 = O(poly(n_0))$ then this reduces to

$$\min\left\{2^{O\left(\sqrt{n_0\log n_0}\right)}, 2^{O\left(\sqrt{n_1\log n_1}\right)}, O\left(n^3\cdot\left(\frac{n}{k}+1\right)^k\right)\right\}.$$

These are the bounds on the number of recursive calls to the algorithm. Within each recursive call, the auxiliary work is dominated by time to compute a strategy value, multiplying the running time by $O(n \cdot |E| \cdot k)$.

Acknowledgements. We thank anonymous referees for valuable remarks and suggestions.

References

1. H. Björklund, S. Sandberg, and S. Vorobyov. A discrete subexponential algorithm for parity games. Technical Report 2002-026, Department of Information Technology, Uppsala University, September 2002. `http://www.it.uu.se/research/reports/`.

[6] Actually, if δ is the outdegree in the vertex where we fix an edge, then the second term is $T(d-1, m-\delta)$. We consider the worst case of $\delta = 2$.

2. A. Browne, E. M. Clarke, S. Jha, D. E Long, and W Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theor. Comput. Sci.*, 178:237–255, 1997. Preliminary version in CAV'94, LNCS'818.
3. 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.
4. E. A. Emerson, C. Jutla, and A. P. Sistla. On model-checking for fragments of μ-calculus. In *Computer Aided Verification, Proc. 5th Int. Conference*, volume 697, pages 385–396. Lect. Notes Comput. Sci., 1993.
5. 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.
6. Goldwasser. A survey of linear programming in randomized subexponential time. *SIGACTN: SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 26:96–104, 1995.
7. 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.
8. M. Jurdzinski. Small progress measures for solving parity games. In *17th STACS*, volume 1770 of *Lect. Notes Comput. Sci.*, pages 290–301. Springer-Verlag, 2000.
9. G. Kalai. A subexponential randomized simplex algorithm. In *24th ACM STOC*, pages 475–482, 1992.
10. W. Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117:151–155, 1995.
11. C. Papadimitriou. Algorithms, games, and the internet. In *ACM Annual Symposium on Theory of Computing*, pages 749–753. ACM, July 2001.
12. V. Petersson and S. Vorobyov. A randomized subexponential algorithm for parity games. *Nordic Journal of Computing*, 8:324–345, 2001.
13. N. Pisaruk. Mean cost cyclical games. *Mathematics of Operations Research*, 24(4):817–828, 1999.
14. H Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(3):303–308, 1996.
15. J. Vöge and M. Jurdzinski. A discrete strategy improvement algorithm for solving parity games. In *CAV'00: Computer-Aided Verification*, volume 1855 of *Lect. Notes Comput. Sci.*, pages 202–215. Springer-Verlag, 2000.
16. U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158:343–359, 1996.