A DETERMINISTIC SUBEXPONENTIAL ALGORITHM FOR SOLVING PARITY GAMES*

MARCIN JURDZIŃSKI†, MIKE PATERSON‡, AND URI ZWICK§

Abstract. The existence of polynomial time algorithms for the solution of parity games is a major open problem. The fastest known algorithms for the problem are *randomized* algorithms that run in subexponential time. These algorithms are all ultimately based on the randomized subexponential simplex algorithms of Kalai and of Matoušek, Sharir and Welzl. Randomness seems to play an essential role in these algorithms. We use a completely different, and elementary, approach to obtain a *deterministic* subexponential algorithm for the solution of parity games. The new algorithm, like the existing randomized subexponential algorithms, uses only polynomial space, and it is almost as fast as the randomized subexponential algorithms mentioned above.

Key words. analysis of algorithms and problem complexity, specification and verification, 2-person games, games involving graphs, discrete-time games

AMS subject classifications. 68Q25, 68Q60, 91A05, 91A43, 91A50.

1. Introduction. A parity game [11, 15] is played on a directed graph (V, E) by two players, *Even* and *Odd*, who move a token from vertex to vertex along the edges of the graph so that an infinite path is formed. A partition (V_0, V_1) is given of the set V of vertices: player Even moves if the token is at a vertex of V_0 and player Odd moves if the token is at a vertex of V_1. Finally, a priority function $p : V \to \{ 1, 2, \ldots, d \}$ is given. The players compete for the parity of the highest priority occurring infinitely often: player Even wins if $\limsup_{i\to\infty} p(v_i)$ is even while player Odd wins if it is odd, where $v_0, v_1, v_2, \ldots$ is the infinite path formed by the players.

The algorithmic problem of solving parity games is, given a parity game $G = (V_0, V_1, E, p)$ and an initial vertex $v_0 \in V$, to determine whether player Even has a winning strategy in the game if the token is initially placed on vertex v_0. Algorithms for solving parity games [30, 20, 29, 15, 1] usually compute the winning sets win_0 and win_1, i.e., the sets of vertices from which players Even and Odd, respectively, have a winning strategy. By the Determinacy Theorem for parity games [11, 15] the winning sets win_0 and win_1 form a partition of the set of vertices V. None of these algorithms is known to run in polynomial time and the existence of a polynomial time algorithm for the solution of parity games is a long-standing open problem [12, 15].

The original motivation for the study of parity games comes from the area of formal verification of systems by temporal logic model checking [5, 15]. The problem of solving parity games is polynomial time equivalent to the non-emptiness problem of ω-automata on infinite trees with Rabin-chain acceptance conditions [12], and to the the model checking problem of the modal μ-calculus (modal fixpoint logic) which is a specification formalism of choice in formal specification and validation [10, 15]. The model checking problem is a fundamental algorithmic problem in automated hardware and software verification [10, 5].

*This paper is an updated and extended version of the SODA'06 paper [21]. The work was partially supported by the London Mathematical Society.

†Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom (mju@dcs.warwick.ac.uk).

‡Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom (msp@dcs.warwick.ac.uk).

§School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (zwick@cs.tau.ac.il).

Another important motivation to study the problem of solving parity games is its intriguing complexity theoretic status: the problem is known to be in NP ∩ co-NP [12] and even in UP ∩ co-UP [19] but, as mentioned, despite considerable efforts of the community [12, 20, 29, 15, 1, 27] no polynomial time algorithm has been found so far. Moreover, parity games are polynomial time reducible to mean payoff games [31], simple stochastic games [6], and discounted games [6, 31]. A stochastic generalization of parity games was also studied [9, 3]. The problems of solving all those games are in UP ∩ co-UP as well [19, 3]. Condon has shown that simple stochastic games are log-space complete in the class of log-space randomized alternating Turing machines [6].

The task of solving parity, mean payoff, discounted, and simple stochastic games can be also viewed as a search problem: given a game graph, compute optimal strategies for both players. The value functions used in strategy improvement algorithms [7, 24, 29, 1] witness membership of all those optimal strategies search problems in PLS (i.e., the class of polynomial local search problems) [17]. On the other hand, the problem of computing optimal strategies in simple stochastic games can be reduced in polynomial time to solving a P-matrix linear complementarity problem [14] and to finding a Brouwer fixpoint [18] and hence it is also in PPAD [28]. It follows that there are polynomial time reductions from the problems of computing optimal strategies in parity, mean payoff, discounted, and simple stochastic games to the problem of finding Nash equlibria in bimatrix games [8, 4].

Let $n = |V|$ and $m = |E|$ be the numbers of vertices and edges of a parity game graph and let d be the largest priority assigned to vertices by the priority function $p : V \to \{ 1, 2, \ldots, d \}$. For parity games with a small number of priorities, more specifically if $d = O(n^{1/2})$, the progress-measure lifting algorithm [20] gives currently the best time complexity of $O(dm \cdot (2n/d)^{d/2})$. If $d = \Omega(n^{1/2+\varepsilon})$ then the randomized algorithm of Björklund et al. [1] has a better (expected) running time bound of $n^{O(\sqrt{n/\log n})}$.

The main contribution of this paper is a *deterministic* algorithm for solving parity games which achieves roughly the same complexity as the randomized algorithm of Björklund et al. [1]: the complexity of our algorithm is $n^{O(\sqrt{n/\log n})}$ if the out-degree of all vertices is bounded, and it is $n^{O(\sqrt{n})}$ otherwise. The new algorithm uses only polynomial space.

The randomized algorithm of Björklund et al. [1] is based on the randomized algorithm of Ludwig [24] for simple stochastic games, which in turn is inspired by the subexponential randomized simplex algorithms for linear programming and LP-type problems by Kalai and by Matoušek et al. [22, 25]. For games with out-degree two these algorithms are instantiations of the Random-Facet algorithm for finding the unique sink in an acyclic unique sink orientation (AUSO) of a hypercube [13]. The nodes of a hypercube correspond to positional strategies for one of the players and the orientation of an edge connecting two positional strategies that differ at exactly one vertex is determined by which of the two strategies has a better value when her opponent plays a best-response strategy [7, 24, 29, 1].

In contrast, our deterministic algorithm for parity games is obtained by a modification of a more elementary algorithm of McNaughton and Zielonka for parity games [30, 15]. The methods we use are thus very different from those of Ludwig and Björklund et al. Our method is applicable, so it seems, only to parity games, while the randomized algorithms for finding the unique sink in an AUSO [13] and for solving an LP-type problem [25] can be applied to a number of problems including computing the values of parity, mean payoff, discounted, and simple stochastic

games [1, 2, 16].

2. Preliminaries. A *parity game* $G = (V_0, V_1, E, p)$ is composed of two disjoint sets of vertices V_0 and V_1, a directed edge set $E \subseteq V \times V$, where $V = V_0 \cup V_1$, and a *priority* function $p : V_0 \cup V_1 \to \{1, 2, \ldots, d\}$, for some integer d, defined on its vertices. Every vertex $u \in V$ has at least one outgoing edge $(u, v) \in E$. The game is played by two players: *Even*, also referred to as Player 0, and *Odd*, also referred to as Player 1.

The game starts at some vertex $v_0 \in V$. The players construct an infinite path as follows. Let u be the last vertex added so far to the path. If $u \in V_0$, then Player 0 chooses an edge $(u, v) \in E$. Otherwise, if $u \in V_1$, then Player 1 chooses an edge $(u, v) \in E$. In either case, vertex v is added to the path, and a new edge is then chosen by either Player 0 or Player 1. As each vertex has at least one outgoing edge, the path constructed can always be continued.

Let $v_0, v_1, v_2, \ldots$ be the infinite path constructed by the two players, and let $p(v_0), p(v_1), p(v_2), \ldots$ be the sequence of the priorities of the vertices on the path. Player 0 wins the game if the largest priority seen infinitely many times is even, and Player 1 wins otherwise.

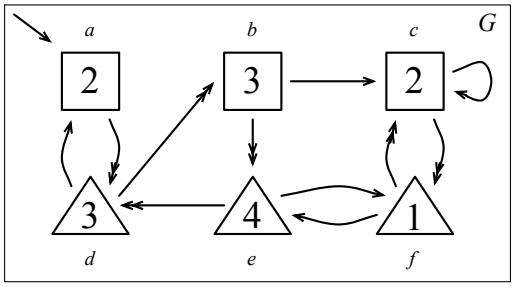

FIG. 2.1. *A parity game G. Double-headed arrows show a positional strategy for each player.*

A *strategy* for Player i in a game G specifies, for every finite path $v_0, v_1, \ldots, v_k$ in G that ends in a vertex $v_k \in V_i$, an edge $(v_k, v_{k+1}) \in E$. A strategy is said to be a *positional* strategy if the edge $(v_k, v_{k+1}) \in E$ chosen depends only on v_k, the last vertex visited. A strategy for player i is said to be a *winning* strategy if using this strategy ensures a win for Player i, no matter which strategy is used by the other player. The Determinacy Theorem for parity games [11, 15] says that for any parity game G and any start vertex v_0, either Player 0 has a winning strategy or Player 1 has a winning strategy. (This claim is not immediate as the games considered are infinite.) Furthermore, if a player has a winning strategy from a vertex in a parity game then she also has a winning positional strategy from this vertex.

In the parity game G illustrated in Figure 2.1, the initial vertex is labelled a, *Even*'s vertices are represented as squares (even number of sides) and *Odd*'s as triangles. As an example, if each player were to choose the double-headed arrow out of each of their vertices then the infinite path would be $a, d, b, e, d, b, e, \ldots$, and the largest priority seen infinitely often would be 4 at vertex e. So *Even* would win this play.

The winning set for Player i, denoted by $win_i(G)$, is the set of vertices of the game from which Player i has a winning strategy. By the Determinacy Theorem for parity games [11, 15] we have that $win_0(G) \cup win_1(G) = V$.

A set $B \subseteq V$ is said to be *i-closed*, where $i \in \{0, 1\}$, if for every $u \in B \cap V_i$, there

is some $(u, v) \in E$, such that $v \in B$, and if for every $(u, v) \in E$ with $u \in B \cap V_{\neg i}$, we have $v \in B$. (We use $\neg i$ for the complement of i in $\{0, 1\}$.) In other words, a set B is i-closed if Player i can always choose to stay in B while Player $\neg i$ cannot escape from it, i.e., B is a "trap" for Player $\neg i$.

LEMMA 2.1. *For every $i \in \{0, 1\}$, the set $win_i(G)$ is i-closed.*

Proof. Let $j = \neg i$, and note that $V \setminus win_i(G) = win_j(G)$ by the Determinacy Theorem. Let u be an arbitrary vertex in $win_i(G)$. If $u \in V_i$ then Player i's strategy uses some $(u, v) \in E$ such that $v \in win_i(G)$. If $u \in V_j$ then $v \in win_i(G)$ for all $(u, v) \in E$, since otherwise there would be a move for Player j from u to a vertex of $v \in win_j(G)$, contradicting the assumption of $u \in win_i(G)$. So, $win_i(G)$ is i-closed. □

Let $A \subseteq V$ be an arbitrary set. The *i-reachability set* of A, denoted $reach_i(A)$, contains all vertices in A and all vertices from which Player i has a strategy to enter the set A at least once; we call it an *i-reachability strategy* to set A. (See Figure 2.2 for a simple example.)

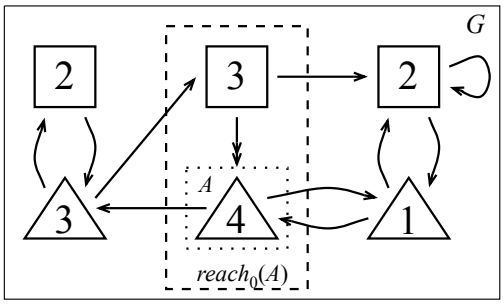

FIG. 2.2. *The 0-reachability set of A and a positional 0-reachability strategy to set A.*

LEMMA 2.2. *For any set $A \subseteq V$ and $i \in \{0, 1\}$, the set $reach_i(A)$ can be computed in $O(m)$ time, where $m = |E|$ is the number of edges in the game.*

Proof. The vertices of A are in $reach_i(A)$ so we initialize $B \leftarrow A$. We then iteratively add to B every vertex of V_i that has at least one edge going into B, and every vertex of $V_{\neg i}$ all of whose edges go into B. We stop when no new vertices can be added to B. It is easy to see that this process can be performed in $O(m)$ time, and that when it ends we have $B = reach_i(A)$, as required. □

LEMMA 2.3. *For any set $A \subseteq V$ and $i \in \{0, 1\}$, the set $V \setminus reach_i(A)$ is $(\neg i)$-closed.*

Proof. Let $u \in V \setminus reach_i(A)$. If $u \in V_{\neg i}$ then there must be an edge $(u, v) \in E$ from vertex u into the set $V \setminus reach_i(A)$, i.e., such that $v \notin reach_i(A)$, since otherwise the iterative procedure from the proof of Lemma 2.2 would add vertex u to $reach_i(A)$. Similarly, if $u \in V_i$ then all edges from vertex u must go into the set $V \setminus reach_i(A)$. Therefore, the set $V \setminus reach_i(A)$ is $(\neg i)$-closed. □

If $B \subseteq V$ is such that for every vertex u in $V \setminus B$ there is an edge (u, v) with v in $V \setminus B$, then the game $G \setminus B$ is the game obtained from G by removing the vertices of B and all the edges that touch them.

LEMMA 2.4. *Let G be a parity game and let $i \in \{0, 1\}$ and $j = \neg i$. If $U \subseteq win_i(G)$ and $\overline{U} = reach_i(U)$, then $win_i(G) = win_i(G \setminus \overline{U}) \cup \overline{U}$ and $win_j(G) = win_j(G \setminus \overline{U})$.*

Proof. It suffices to exhibit strategies for Players j and i that are winning for them in the game G, from the sets $win_j(G \setminus \overline{U})$ and $win_i(G \setminus \overline{U}) \cup \overline{U}$ respectively. We claim that a winning strategy for Player j from the set $win_j(G \setminus \overline{U})$ in the subgame

$G \setminus \overline{U}$ is also winning for her from the same set in the original game G. It suffices to establish that Player i cannot escape from the set $win_j(G \setminus \overline{U})$ in the game G. This follows from Lemma 2.3 (the set $V \setminus \overline{U}$ is j-closed in G, and hence Player i cannot escape to $\overline{U}$) and from Lemma 2.1 (the set $win_j(G \setminus \overline{U})$ is j-closed in $G \setminus \overline{U}$, and hence Player i cannot escape to $win_i(G \setminus \overline{U})$).

Now we exhibit a winning strategy for Player i in the game G from the set $win_i(G \setminus \overline{U}) \cup \overline{U}$. By the assumption that $U \subseteq win_i(G)$, there is a strategy σ for Player i in the game G which is winning for her from all vertices in U. Let τ be a winning strategy for Player i from the set $win_i(G \setminus \overline{U})$ in the subgame $G \setminus \overline{U}$. We construct a strategy π for Player i in the game G by composing strategies τ and σ in the following way: if the play so far is contained in the set $win_i(G \setminus \overline{U})$ then follow strategy τ, otherwise use the i-reachability strategy to the set U and "restart" the play following the strategy σ thenceforth. The strategy π is well-defined because, by Lemma 2.1, Player j can escape from $win_i(G \setminus \overline{U})$ only into the set $\overline{U}$. It is a winning strategy for Player i because if strategy π ever switches from following τ into following σ then an infinite suffix of the play is winning for Player i, and in parity games removing an arbitrary finite prefix of a play does not change the winner. □

LEMMA 2.5. *Let G be a parity game. Let $d = d(G)$ be the highest priority and let $A = A_d(G)$ be the set of vertices of highest priority. Let $i = d \bmod 2$ and $j = \neg i$. Let $G' = G \setminus reach_i(A)$. Then, we have $win_j(G') \subseteq win_j(G)$. Also, if $win_j(G') = \emptyset$ then $win_i(G) = V(G)$, i.e., Player i wins from every vertex of G.*

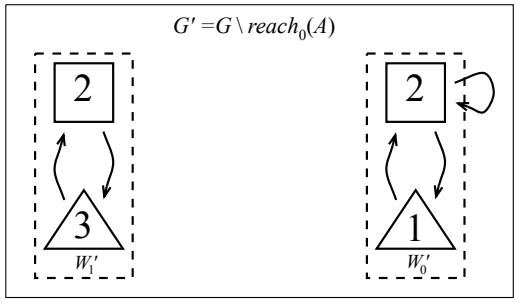

FIG. 2.3. *The game $G' = G \setminus reach_0(A)$ and winning sets $W'_i = win_i(G')$ for $i = 0, 1$.*

Proof. We claim that a winning strategy for Player j from vertices in the set $win_j(G')$ in the subgame G' is also winning for her in the whole game G. Indeed, by Lemma 2.1 Player i cannot escape from $win_j(G')$ to $win_i(G')$, and by Lemma 2.3 Player i cannot escape from $win_j(G')$ to $reach_i(A)$. (As an example, consider Figures 2.2 and 2.3, with $i = 0$ and $j = 1$.)

Suppose now that $win_j(G') = \emptyset$. Let τ be a winning strategy for Player i from $win_i(G')$ (which, by determinacy, is equal to $V \setminus reach_i(A)$) in the subgame G'. We construct a strategy π for Player i in the following way: if a play so far is contained in the set $win_i(G')$ then follow strategy τ; otherwise the current vertex is in $reach_i(A)$ so follow the i-reachability strategy to the set A; moreover, each time the play re-enters the set $win_i(G')$ "restart" the play and follow strategy τ henceforth, etc. If a play following the strategy π eventually never leaves the set $win_i(G')$ then it has an infinite suffix played according to strategy τ and hence it is winning for Player i. Otherwise it leaves the set $win_i(G')$ infinitely often, so it visits a vertex of the maximal priority d infinitely often, and hence it is winning for Player i because $i = d \bmod 2$. □

3. An exponential algorithm. A simple exponential-time algorithm for the solution of parity games is given in Figure 3.1. This algorithm originates from the work of McNaughton [26] and was first presented for parity games by Zielonka [30, 15]. Algorithm $\mathbf{win}(G)$ receives a parity game G and returns the pair of winning sets $(win_0(G), win_1(G))$ for the two players.

```
algorithm win(G)
if V(G) = ∅ then return (∅, ∅)
d ← d(G) ; A ← A_d(G)
i ← d mod 2 ; j ← ¬i
(W'_0, W'_1) ← win(G \ reach_i(A))
if W'_j = ∅ then
   (W_i, W_j) ← (V(G), ∅)
else
   (W''_0, W''_1) ← win(G \ reach_j(W'_j))
   (W_i, W_j) ← (W''_i, V(G) \ W''_i)
endif
return (W_0, W_1)
```

FIG. 3.1. *An exponential algorithm for solving parity games.*

Algorithm $\mathbf{win}(G)$ is based on Lemmas 2.4 and 2.5. It starts by letting d be the largest priority in G and by letting A be the set of vertices having this highest priority. Suppose $i = d \bmod 2$, the index of the player associated with the parity of this highest priority, and $j = \neg i$, the index of the other player. The algorithm first finds the winning sets (W'_0, W'_1) of the smaller game $G' = G \setminus reach_i(A)$, using a recursive call. By Lemma 2.5, if $W'_j = \emptyset$ then Player i wins from all vertices of G and we are done. Otherwise, again by Lemma 2.5, we know that $W'_j \subseteq win_j(G)$. The algorithm then finds the winning sets (W''_0, W''_1) of the smaller game $G'' = G \setminus reach_j(W'_j)$ by a second recursive call. By Lemma 2.4, we then know that $win_i(G) = W''_i$ and thus $win_j(G) = V(G) \setminus W''_i$.

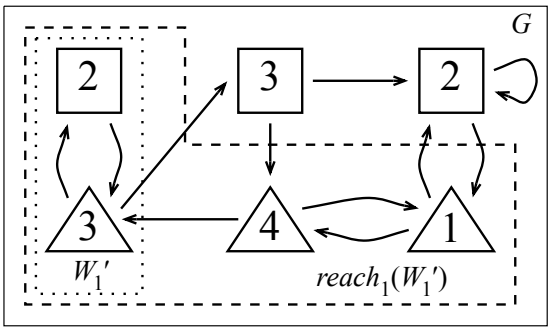

FIG. 3.2. *The* 1*-reachability set of the set* W'_1.

A small detailed illustration is given in Figures 2.2, 2.3, 3.2, and 3.3.

THEOREM 3.1. *Algorithm* $\mathbf{win}(G)$ *correctly finds the winning sets of the parity game* G*. Its running time is* $O(2^n)$*, where* $n = |V(G)|$ *is the number of vertices in* G*.*

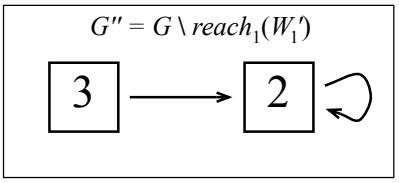

FIG. 3.3. *The game* $G'' = G \setminus reach_1(W_1')$.

Proof. The correctness of the algorithm follows from Lemmas 2.4 and 2.5, as argued above. Let $T(n)$ be the maximum running time of algorithm $\mathbf{win}(G)$ on a game on at most n vertices. Algorithm $\mathbf{win}(G)$ makes two recursive calls $\mathbf{win}(G')$ and $\mathbf{win}(G'')$ on games with at most $n-1$ vertices. Other than that, it performs only $O(n^2)$ operations. (The most time consuming operations are the computations of the sets $reach_i(A)$ and $reach_j(W_j')$.) Thus $T(n) \leq 2T(n-1) + O(n^2)$. It is easy to see then that $T(n) = O(2^n)$. ∎

4. Finding small dominions. A set $D \subseteq V(G)$ is said to be an *i-dominion* if Player i can win from every vertex of D without ever leaving D. Note, in particular, that an i-dominion must be i-closed. A set $D \subseteq V(G)$ is said to be a *dominion* if it is either a 0-dominion or a 1-dominion. Clearly, the winning set $win_i(G)$ of Player i is an i-dominion.

LEMMA 4.1. *Let G be a parity game on n vertices and let $\ell \leq n/3$. There is an $O(n^\ell)$-time algorithm that finds a non-empty dominion in G of size at most ℓ, or determines that no such dominion exists.*

Proof. If $\ell \leq n/3$ then for all $i \leq \ell$, we have that $\binom{n}{i}/\binom{n}{i-1} > 2$. The number $\sum_{i=1}^{\ell}\binom{n}{i}$ of subsets of V of size at most ℓ is therefore at most $2\binom{n}{\ell}$. For each such subset U of V we check, in $O(\ell^2)$ time, whether it is 0-closed or 1-closed. If both tests fail, then U is clearly not a dominion. If U is i-closed, for some $i \in \{0,1\}$, we form the game $G[U]$ which is the game G restricted to U. This is well-defined since U is i-closed. We now apply the exponential algorithm of the previous section to $G[U]$ and find out, in $O(2^\ell)$ time, whether Player i can win from all the vertices of $G[U]$. If so, then U is an i-dominion, otherwise it is not. The total running time of the algorithm is therefore $O(\binom{n}{\ell}2^\ell) = O(n^\ell)$, as required. ∎

In a game with bounded out-degrees we can find small dominions even faster. For simplicity, the lemma below and the analysis in Section 6 are stated for games in which the out-degree of every vertex is exactly two. Note, however, that for every constant b, any game on n vertices with out-degrees at most b can be easily converted into an equivalent game on at most $n(b-1)$ vertices and out-degrees exactly two.

LEMMA 4.2. *Let G be a parity game on n vertices in which the out-degree of each vertex is two. There is an $O(n2^\ell \ell \log \ell)$-time algorithm that finds a non-empty dominion in G of size at most ℓ, or determines that no such dominion exists.*

Proof. Assume, without loss of generality, that the vertices of G are numbered from 1 to n. Let $u \in V$ be vertex of G and let (u, v_0) and (u, v_1) be the two edges emanating from v, where $v_0 \leq v_1$. We say that (u, v_0) is the 0-th outgoing edge of v, while (u, v_1) is the 1-st outgoing edge of v.

The algorithm generates at most $O(n2^\ell)$ 0-closed sets of size at most ℓ that are candidates for being 0-dominions. For every vertex $v \in V$ and a binary sequence $\langle a_1, \ldots, a_\ell \rangle \in \{0,1\}^\ell$, construct a set $U \subset V$ as follows. Start with $U = \{v\}$ and

$r = 1$. Vertices added to U are initially unmarked. As long as there is still an unmarked vertex in U, pick the smallest such vertex $u \in U$ and mark it. If $u \in V_0$, then add the endpoint of the a_r-th outgoing edge of u to U, if it is not already there, and increment r. If $u \in V_1$, then add the endpoints of all the outgoing edges of u to U. If at some stage $|U| > \ell$, then discard the set U and restart the construction with the next binary sequence.

If the process above ends with $|U| \leq \ell$, then a 0-closed set of size at most ℓ has been found. Furthermore, for every vertex $u \in U \cap V_0$, one of the outgoing edges of u was selected. This corresponds to a suggested strategy for Player 0 in the game $G[U]$. Using an algorithm of King et al. [23] we can check, in $O(\ell \log \ell)$ time, whether this is indeed a winning strategy for Player 0 from all the vertices of U. It is easy to see that if there is a 0-dominion of size at most ℓ in G, then the algorithm will find one.

Finding 1-dominions of size at most ℓ can be done in an analogous manner. □

The algorithm described in Lemma 4.1 finds some i-dominion D if there is a dominion of size at most ℓ. We denote this algorithm by **dominion**(G, ℓ), and suppose that it returns either the pair (D, i) if successful, or $(\emptyset, -1)$ if not.

```
algorithm new-win(G)
n ← |V(G)| ; ℓ ← ⌈√(2n)⌉
if V(G) = ∅ then return (∅, ∅)
(D, i) ← dominion(G, ℓ); j ← ¬i
if D ≠ ∅ then
    (W'_0, W'_1) ← new-win(G \ reach_i(D))
    (W_j, W_i) ← (W'_j, V(G) \ W'_j)
else
    (W_0, W_1) ← old-win(G)
endif
return(W_0, W_1)
```

```
algorithm old-win(G)
d ← d(G) ; A ← A_d(G)
i ← d mod 2 ; j ← ¬i
(W'_0, W'_1) ← new-win(G \ reach_i(A))
if W'_j = ∅ then
    (W_i, W_j) ← (V(G), ∅)
else
    (W''_0, W''_1) ← new-win(G \ reach_j(W'_j))
    (W_i, W_j) ← (W''_i, V(G) \ W''_i)
endif
return (W_0, W_1)
```

FIG. 5.1. *The new subexponential algorithm for solving parity games.*

5. The new subexponential algorithm. The new algorithm for solving parity games is given in Figure 5.1. The algorithm **new-win** starts by trying to find a dominion of size at most ℓ, where $\ell = \lceil \sqrt{2n} \rceil$ (and $\ell = \lceil \sqrt{n \log n} \rceil$ for games with bounded out-degree) is a parameter chosen to minimize the running time of the whole

algorithm. If such a small i-dominion is found, then it is easy to remove it, and its i-reachability set, from the game and recurse on what is left over. If no small dominion is found, then **new-win**(G) simply calls algorithm **old-win**(G) which is almost identical to the exponential algorithm **win**(G) of Section 3. The only difference between **old-win**(G) and **win**(G) is that the recursive calls are made to **new-win**(G) and not to **win**(G).

THEOREM 5.1. *Algorithm* **new-win**(G) *correctly finds the winning sets of a parity game G. Its running time on a game with n vertices is $n^{O(\sqrt{n})}$.*

Proof. The correctness of the algorithm is immediate. We next analyse its running time. Let $T(n)$ be the maximum running time of **new-win**(G) on a game with at most n vertices.

Algorithm **new-win**(G) tries to find dominions of size at most $\ell = \lceil \sqrt{2n} \rceil$. By Lemma 4.1 this takes $O(n^\ell)$ time. If a non-empty dominion is found, then the algorithm simply proceeds on the remaining game, which has at most $n-1$ vertices, and the remaining running time is therefore at most $T(n-1)$. Otherwise, a call to **old-win**(G) is made. This results in a call to **new-win**$(G \setminus \mathit{reach}_i(A))$, which takes at most $T(n-1)$ time. If the set W'_j returned by the call is empty, then we are done. Otherwise, as W'_j is a j-dominion of G, we know that $|W'_j| > \ell$, and so the second recursive call **new-win**$(G \setminus \mathit{reach}_j(W'_j))$ takes at most $T(n-\ell)$ time. Thus we get

$$T(n) \;\leq\; O(n^\ell) + T(n-1) + T(n-\ell)\;.$$

This recurrence relation, with $\ell = \lceil \sqrt{2n} \rceil$, is analysed in the next section where it is shown that $T(n) = n^{O(\sqrt{n})}$. ∎

A slightly better bound is achieved for graphs with out-degree two.

THEOREM 5.2. *Consider the algorithm* **new-win**(G) *in which the variable ℓ is set to $\lceil \sqrt{n \log n} \rceil$. If the game G has n vertices and the out-degree of each of them is two, then the running time of the modified algorithm is $n^{O(\sqrt{n/\log n})}$.*

Proof. Note that if $\ell = \lceil \sqrt{n \log n} \rceil$ then $O(n 2^\ell \ell \log \ell) = n^{O(\sqrt{n/\log n})}$. Therefore, by Lemma 4.2 and by the analysis in the proof of the previous theorem, the time complexity $T(n)$ satisfies the following recurrence:

$$T(n) \;\leq\; n^{O(\sqrt{n/\log n})} + T(n-1) + T(n-\ell).$$

This recurrence is analysed in the next section. It is shown there that $T(n) = n^{O(\sqrt{n/\log n})}$. ∎

6. Solving the recurrence relations. In this section we analyze the recurrence relations used in the previous section to bound the running time of the new algorithm.

We start by analyzing the recurrence relation used to bound the running time of the algorithm for game graphs with arbitrary out-degrees.

THEOREM 6.1. *If $T(n)$ is a positive function such that, for every $n > 3$,*

$$T(n) \;\leq\; O(n^\ell) + T(n-1) + T(n-\ell)\;,$$

where $\ell = \lceil \sqrt{2n} \rceil$, then $T(n) = n^{O(\sqrt{n})}$.

Proof. For every integer n we construct a binary tree $\mathcal{T}_n$ in the following way. The root of $\mathcal{T}_n$ is labelled by n. A node labelled by a number $k > 3$ has two children: a left child labelled by $k-1$ and a right child labelled by $k - \lceil \sqrt{2k} \rceil$. Nodes labelled by the numbers 1,2 and 3 are leaves. A node labelled by k has a cost of $k^{O(\sqrt{2k})}$

associated with it. It is easy to see that the sum of the costs of the nodes of $\mathcal{T}_n$ is an upper bound on $T(n)$.

Clearly, the length of every path in $\mathcal{T}_n$ from the root to a leaf is most n. We say that such a path makes a *right turn* when it descends from a vertex to its right child. We next claim that each such path makes at most $\lfloor\sqrt{2n}\rfloor$ right turns. This follows immediately from the observation that the function $f(n) = n - \lceil\sqrt{2n}\rceil$ can be iterated on n at most $\lfloor\sqrt{2n}\rfloor$ times before reaching the value of 3 or less. This observation is based on the fact that if $\frac{1}{2}k^2 < n \leq \frac{1}{2}(k+1)^2$, then $n - \lceil\sqrt{2n}\rceil \leq \frac{1}{2}k^2$.

As each leaf of $\mathcal{T}_n$ is determined by the positions of the right turns on the path leading to it from the root, we get that the number of leaves in $\mathcal{T}_n$ is at most $\binom{n}{\lfloor\sqrt{2n}\rfloor}$. The total number of nodes in $\mathcal{T}_n$ is therefore at most $2\binom{n}{\lfloor\sqrt{2n}\rfloor}$. As the cost of each node is at most $n^{O(\sqrt{2n})}$, we immediately get that

$$T(n) \;\leq\; 2\binom{n}{\lfloor\sqrt{2n}\rfloor} n^{O(\sqrt{2n})} \;=\; n^{O(\sqrt{n})}\,,$$

as claimed. ∎

A more careful analysis, in which the $O(n^\ell)$ term in the recurrence relation is replaced by $O(\binom{n}{\ell}2^\ell)$, can be used to show that $T(n) = O((cn)^{\sqrt{n/2}})$, for some constant $c > 0$, and that the choice $\ell = \lceil\sqrt{2n}\,\rceil$ is essentially optimal.

The running time of the algorithm for graphs with out-degree two satisfies a tighter recurrence relation, which is analysed similarly in the next theorem.

THEOREM 6.2. *If $T(n)$ is a positive function such that, for every $n > 2$,*

$$T(n) \leq n^{O(\sqrt{n/\log n}\,)} + T(n-1) + T(n-\ell),$$

where $\ell = \lceil\sqrt{n \log n}\,\rceil$, then $T(n) = n^{O(\sqrt{n/\log n}\,)}$.

Proof. The proof is similar to the proof of Theorem 6.1. For every integer n we again construct a tree $\mathcal{T}_n$. A node labelled by a number $k > 2$ now has a left child labelled by $k-1$ and a right child labelled by $k - \lceil\sqrt{k \log k}\,\rceil$. The cost of a node labelled by k is now $k^{O(k/\log k)}$. Every root to leaf path in $\mathcal{T}_n$ is again of length at most n, and it can now make at most $O(\sqrt{n/\log n})$ right turns. Thus, the number of nodes in $\mathcal{T}_n$ is at most $n^{O(\sqrt{n/\log n})}$. As the cost of each node is also $n^{O(\sqrt{n/\log n})}$, we get that $T(n) = n^{O(\sqrt{n/\log n})}$, as claimed. ∎

7. Concluding remarks. We have obtained the first deterministic subexponential algorithm for solving parity games. Our algorithm does not seem to extend in an obvious way to the solution of the more general mean payoff games and simple stochastic games.

Acknowledgements. We thank an anonymous SODA'06 referee for suggestions that resulted in a significant simplification of the proofs of Theorems 6.1 and 6.2. We also thank Reto Spöhel for pointing out inaccuracies in an earlier version of the proof of Lemma 4.1.

REFERENCES

[1] H. BJÖRKLUND, S. SANDBERG, AND S. VOROBYOV, *A discrete subexponential algorithm for parity games*, in Symposium on Theoretical Aspects of Computer Science, STACS 2003, vol. 2607 of LNCS, Springer, 2003, pp. 663–674.

[2] H. Bjorklund, S. Sandberg, and S. Vorobyov, *Randomized subexponential algorithms for infinite games*, Tech. Rep. 2004-09, DIMACS, 2004.
[3] K. Chatterjee, M. Jurdziński, and T. A. Henzinger, *Quantitative stochastic parity games*, in Symposium on Discrete Algorithms, SODA 2004, ACM/SIAM, 2004, pp. 114–123.
[4] X. Chen and X. Deng, *Settling the complexity of two-player Nash equilibrium*, in FOCS 2006, IEEE, 2006, pp. 261–272.
[5] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 1999.
[6] A. Condon, *The complexity of stochastic games*, Information and Computation, 96 (1992), pp. 203–224.
[7] A. Condon, *On algorithms for simple stochastic games*, in Advances in Computational Complexity Theory, American Mathematical Society, 1993, pp. 51–73.
[8] C. Daskalakis, P. W. Goldberg, and C. H. Papadimitriou, *The complexity of computing a Nash equilibrium*, in STOC 2006, ACM Press, 2006, pp. 71–78.
[9] L. de Alfaro and R. Majumdar, *Quantitative solution of omega-regular games*, Journal of Computer and System Sciences, 68 (2004), pp. 374–397.
[10] E. A. Emerson, *Model checking and mu-calculus*, in Descriptive Complexity and Finite Models, N. Immerman and P. G. Kolaitis, eds., vol. 31 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996, pp. 185–214.
[11] E. A. Emerson and C. Jutla, *Tree automata, μ-calculus and determinacy*, in Symposium on Foundations of Computer Science, FOCS'91, IEEE Computer Society Press, 1991, pp. 368–377.
[12] E. A. Emerson, C. S. Jutla, and A. P. Sistla, *On model-checking for fragments of μ-calculus*, in International Conference on Computer-Aided Verification, CAV'93, vol. 697 of LNCS, Springer, 1993, pp. 385–396.
[13] B. Gärtner, *The Random-Facet simplex algorithm on combinatorial cubes*, Random Structures and Algorithms, 20 (2002), pp. 353–381.
[14] B. Gärtner and L. Rüst, *Simple stochastic games and P-matrix generalized linear complementarity problems*, in FCT 2005, vol. 3623 of LNCS, 2005, pp. 209–220.
[15] E. Grädel, W. Thomas, and T. Wilke, eds., *Automata, Logics, and Infinite Games. A Guide to Current Research*, vol. 2500 of LNCS, Springer, 2002.
[16] N. Halman, *Discrete and Lexicographic Helly Theorems and Their Relations to LP-type Problems*, PhD thesis, Tel-Aviv University, 2004.
[17] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, *How easy is local search?*, J. Comput. Syst. Sci., 37 (1988), pp. 79–100.
[18] B. Juba, *On the hardness of simple stochastic games.* Manuscript, 2004.
[19] M. Jurdziński, *Deciding the winner in parity games is in UP $\cap$ co-UP*, Information Processing Letters, 68 (1998), pp. 119–124.
[20] M. Jurdziński, *Small progress measures for solving parity games*, in Symposium on Theoretical Aspects of Computer Science, STACS 2000, vol. 1770 of LNCS, Springer, 2000, pp. 290–301.
[21] M. Jurdziński, M. Paterson, and U. Zwick, *A deterministic subexponential algorithm for solving parity games*, in Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, ACM/SIAM, 2006, pp. 117–123.
[22] G. Kalai, *A subexponential randomized simplex algorithm (Extended abstract)*, in Symposium on Theory of Computing, STOC'92, ACM Press, 1992, pp. 475–482.
[23] V. King, O. Kupferman, and M. Y. Vardi, *On the complexity of parity word automata*, in Foundations of Software Science and Computation Structures, FoSSaCS 2001, vol. 2030 of LNCS, Springer, 2001, pp. 276–286.
[24] W. Ludwig, *A subexponential randomized algorithm for the simple stochastic game problem*, Information and Computation, 117 (1995), pp. 151–155.
[25] J. Matoušek, M. Sharir, and E. Welzl, *A subexponential bound for linear programming*, Algorithmica, 16 (1996), pp. 498–516.
[26] R. McNaughton, *Infinite games played on finite graphs*, Annals of Pure and Applied Logic, 65 (1993), pp. 149–184.
[27] J. Obdržálek, *Fast mu-calculus model checking when tree-width is bounded*, in International Conference on Computer-Aided Verification, CAV 2003, vol. 2725 of LNCS, Springer, 2003, pp. 80–92.
[28] C. H. Papadimitriou, *On the complexity of the parity argument and other inefficient proofs existence*, J. Comput. Syst. Sci., 48 (1994), pp. 498–532.
[29] J. Vöge and M. Jurdziński, *A discrete strategy improvement algorithm for solving parity games (Extended abstract)*, in International Conference on Computer-Aided Verification, CAV 2000, vol. 1855 of LNCS, Springer, 2000, pp. 202–215.

[30] W. Zielonka, *Infinite games on finitely coloured graphs with applications to automata on infinite trees*, Theoretical Computer Science, 200 (1998), pp. 135–183.
[31] U. Zwick and M. Paterson, *The complexity of mean payoff games on graphs*, Theoretical Computer Science, 158 (1996), pp. 343–359.