An Improved Subexponential Algorithm for Parity Games

Henrik Björklund, Sven Sandberg, and Sergei Vorobyov

Department of Information Technology
Uppsala University
Box 337, SE-751 05 Uppsala, Sweden

Technical report 2003-017
March 2003
ISSN 1404-3203

Abstract

We suggest a new algorithm for deciding parity games, a fundamental problem of unknown complexity (in NP $\cap$ coNP, not known to belong to P) in game, automata, complexity theories, combinatorial optimization, temporal logic of programs, computer-aided verification. The novelty of the algorithm consists in exploiting a special form of parity games with *retreats*, where optimal retreat edges define *absorbing facets* (with better values than their neighbors on complementary facets) in the strategy space. A superset of such absorbing facets can be found by standard random iterative improvement algorithms in expected polynomial time. Additional dual techniques are used to minimize this superset. As a result, the dimension of the problem shrinks, to which we finally apply the Kalai-Matoušek-Sharir-Welzl-Ludwig-style randomization techniques we recently adapted for games [5, 6].

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

Contents

1 Introduction

Classifying the exact computational complexity of *parity games* is an exciting open problem on the boundary of P and NP. The problem consists in determining a winner in an infinite full information game played by two adversaries alternating moves on a finite leafless graph with vertices painted in integer colors. One player tries to ensure that in any infinite play the maximum color occurring infinitely often is *even*, while the other wants to make it *odd*. The problem belongs to NP $\cap$ coNP (this follows immediately from memoryless determinacy – the winner can always win by using a positional strategy; it suffices to guess a winner and his strategy), but despite extensive effort, the P-time membership of the problem is open. The practical importance of the problem stems from the fact that it is polynomial time equivalent to the so-called Mu-Calculus Model Checking problem [11]. Mu-calculus is one of the most expressive temporal logics of programs, expressively subsuming virtually all other known such logics [10].

We were the first to suggest *subexponential* randomized algorithms for parity games [19, 5, 6]. These algorithms combine ideas from iterative strategy improvement, randomization techniques of Kalai [14] and Matoušek-Sharir-Welzl [17] for Linear Programming, of Ludwig [16] for simple stochastic games, with discrete strategy evaluation of Vöge and Jurdziński [22]. Previous 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 of these algorithms, exponentially hard input instances are known [11, 10, 7, 20, 13]. Our algorithms from [5, 6] are simultaneously *subexponential* in n and exponential in k, independently of the game outdegree, unlike the algorithms [16, 19], which become exponential for games of unbounded outdegree.

In this paper we suggest a further improvement. A partial drawback of subexponential algorithms [16, 19, 5, 6] is that they always start randomization on the full space of strategies, exponential in the number of vertices n. Consequently, as our extensive experiments [1, 4, 2] demonstrate, despite the best known bounds for these algorithms, in practice they do not outperform standard Greedy and Random local search improvement algorithms, for which the best known upper bounds are only exponential.

The algorithm we present in this paper is more clever, and applies the more expensive subexponential randomization as a last resort, when simpler polynomial tools fail to completely solve the game. As a result, the dimension of the problem reduces, and although the algorithm remains subexponential, it is subexponential in a smaller dimension. We first translate a parity game into a special form with *retreats*. Our strategy cost evaluation function is carefully selected in such a way that in some vertices where the player loses, he minimizes the loss by going directly to a retreat. It turns out that precisely knowing such vertices is enough to solve the game completely. We also prove that such 'retreat' vertices define the so-called *absorbing* facets, possessing better cost values than their neighbors on complementary facets. An old result of Kelly [15] shows that standard random local search algorithms find (and stay forever in) absorbing facets in expected low-degree polynomial time. This allows us to easily guess a superset of the real absorbing retreat facets. We use additional, dual techniques to find as close upper approximation to this set as possible. Finally we resort to the Kalai-Matoušek-Sharir-Welzl-Ludwig-style randomization to decide real absorbing retreat facets in this superset. Actually, the algorithm may well stop before reaching the last expensive phase. There is a long-standing conjecture due to Williamson Hoke [23] that the standard random local search optimizes any *completely unimodal* pseudo-Boolean function in expected polynomial time (if true, our algorithm will always stop in the preprocessing stage!). We showed that parity games with our strategy cost evaluation give rise to completely unimodal function [6].

We extensively refer for backgrounds to our recent papers [5, 6]. Detailed proofs and complete complexity analysis will appear in subsequent publications.

2 Preliminaries

Definition 2.1 *A* parity game *is played by two adversaries, Player* 0 *and Player* 1*, on a finite directed bi-partite*[1] *graph without sinks*[2]*. Each vertex of the game graph is assigned a non-negative integer weight, called a* color*. A play starts in some initial vertex, and the adversaries alternate moves by selecting outgoing edges and traveling from vertex to vertex. An infinite sequence of vertices is formed in this way, and the winner is determined by the parity of the largest vertex color occurring in this sequence. If this color is* even*, then Player* 0 *wins. Otherwise, Player* 1 *wins.*

Parity games are known to be *determined* and solvable in *positional (memoryless) strategies*. A positional strategy is a selection by a player of exactly one outgoing edge from every vertex where he moves. When a play comes to a vertex the player deterministically follows his preselected edge. It turns out that for any parity game starting from every vertex exactly one of the player wins (against any strategy of the opponent). Moreover, the winner always has a positional winning strategy, and revealing this strategy in advance is no disadvantage. Memoryless determinacy of parity games was first established by Emerson [9, 12], but a particularly simple proof follows from an earlier result by Ehrenfeucht and Mycielski [8] for mean payoff games; see [3].

By W_i we denote the *winning set* of Player i, i.e., the set of vertices starting from which he can secure an even or odd (respectively) winning color.

For the purposes of this paper we need to consider a slightly more general version of parity games, in which players not only care about parity of the winning color but the winner tries to make this color as big as possible, and the loser tries to make it as small as possible. More formally, consider the following ordering of colors:

$$\cdots \prec 9 \prec 7 \prec 5 \prec 3 \prec 1 \prec 0 \prec 2 \prec 4 \prec 6 \prec 8 \prec 10 \prec \ldots,$$

i.e., $i \prec j$ iff $(-1)^i \cdot i < (-1)^j \cdot j$

Now the goals of the players is as follows. Player 0 tries to maximize wrt $\prec$ and Player 1 to minimize the largest color appearing in any infinite play,

[1]bi-partiteness is no loss of generality for parity games; it can always be achieved by inserting auxiliary vertices of outdegree one.

[2]i.e., there is an outgoing edge from every vertex.

assuming the opponent plays optimally. It follows from the general result of [3] that this version of parity games is solvable in positional strategies for both players.

For a positional strategy σ and vertex v of Player i, and edge e leaving v, denote $\sigma[v \to e]$ the positional strategy obtained from σ by selecting e from v.

2.1 Strategy Space, Facets

Consider a game in which a player[3] has n vertices where he makes choices, and m_i is the number of possible choices at vertex v_i, for $i \in \{1, \dots, n\}$. The space of all positional strategies of the player in such a game may be identified with the Cartesian product $\mathcal{S} = \prod_{i=1}^{n}\{1, \dots, m_i\}$, of dimension n. In particular, when the are exactly two choices in every vertex $\mathcal{S}$ is a boolean hypercube.

The following geometric terminology is convenient and used below in describing the algorithm.

Positional strategies are just vectors in this space $\mathcal{S}$. Two strategies are *neighbors* if they differ in exactly one coordinate. A *facet* of $\mathcal{S}$ is a subset of $\mathcal{S}$ obtained by fixing the value of some coordinate, and letting all other coordinates vary. The dimension of the facet remains the same (n components), but counting the number of *nontrivial coordinates* (vertices where the player has at least two choices), a proper facet has a one smaller dimension compared to the full space.

2.2 Strategy Evaluation

It is possible to equip the positional strategy space $\mathcal{S}$ of a parity game with a well-behaved cost function allowing for efficient optimization by iterative improvement [5] (another possibility is to use the cost from [22]). Without going deep into technical details and repeating [5] here we summarize the main properties of this cost function, essential for the algorithm we are going to describe.

1. The strategies are assigned values in a partially ordered set.

[3] 0 or 1, we can take the viewpoint of any player, actually we will optimize simultaneously for both players

2. The costs of any two neighboring strategies (differing in exactly one vertex) are comparable.

3. Any strategy better or equal to its neighbors (locally optimal) is globally optimal. (It follows that any two optimal strategies, not necessarily neighbors, have the same value.)

Roughly (for details see [5]) this cost function is defined as follows.

1. Suppose players 0 and 1 fix their positional strategies σ and τ. In this case the play from every vertex v takes a simple δ-shaped form: a path leading to a loop.

2. Define the value of v wrt σ and τ as $((-1)^l \cdot l, P, d)$, where l is the maximal color on the loop, P records the number of occurrences of more significant colors on the path from v to the vertex with the largest color on the loop, and d is the length of this path.

3. The values of the form above are ordered lexicographically. The overall value of a strategy σ is the vector of values of all vertices, assuming the adversary uses an *optimal counterstrategy* τ against σ. An optimal counterstrategy against a *fixed* strategy can be found in polynomial time.

It is demonstrated in [5] that this cost function on strategies satisfies all the required properties formulated above. The following two local search iterative improvement algorithms are well known.

Random Single Switch Algorithm (RSSA) proceeds from a current strategy to one of its better (possessing better cost) neighbors, chosen uniformly at random (if there are several better neighbors).

Greedy Single Switch Algorithm (GSSA) proceeds from a current strategy to one of the best neighbors (using some extra rule to break ties, when there are several best neighbors with the same cost).

The term 'single switch' is motivated by locality of the change – every iteration changes a strategy in just one vertex.

It follows immediately that with our strategy evaluation function we can run the RSSA or GSSA to correctly find optimal strategies. Another algorithm, whose correctness does not follows from the abstract properties we formulated above, but nevertheless correct for our strategy evaluation function [6] is the following

Random Multiple Switch Algorithm (RMSA) that proceeds from a current strategy by selecting a random subset of its better neighbors and simultaneously making switches towards these neighbors (see the details in [6]).

All these algorithms perform excellently in practice [4, 1, 2]. However, currently there are no known subexponential upper bounds for these algorithms and nontrivial cost functions arising from game strategy evaluation, like the one described above. In [5, 6] we described different subexponential algorithms, and below present a new one.

3 Description of the Algorithm

The algorithm works on a special preprocessed form of a parity game, which we describe in Section 3.1. The algorithm starts by running in parallel the Random Single or Multiple Switch algorithms on strategy spaces of *both* players, for polynomially many steps. The goal of this stage consists in diminishing the dimension of the problem by finding an essential small subset of facets. Characterizing facets in this subset allows to solve the game, as shows Lemma 3.10.

3.1 Transforming into a Game with Retreats

Every parity game can be transformed into a special form, we call a *game with retreats*, described below.

Without loss of generality we may assume that colors 0 and 1 do not occur in the initial game subject to the transformation. Otherwise, we can always make a shift by adding 2 to each color. This will not change winning sets of the players, only the optimal loop colors will increase by 2. Colors 0 and 1 will play special roles in games with retreats.

We also make a technical assumption that in any infinite play from any vertex each of the players eventually gets a turn to move (this is automatically guaranteed for games on bi-partite graphs).

Definition 3.1 (Games with Retreats) *Given an arbitrary parity game G with no colors 0 and 1, transform it into the* game with retreats G' *as follows:*

1. *Add two vertices v_0 of color 0 and v_1 of color 1, called* retreats*. For $i = 0, 1$, the only edge leaving v_i leads to v_i (self-loop).*

2. *For $i = 0, 1$, from every vertex of Player i of non-unary outdegree add an edge, called the* retreat edge*, to the retreat v_{1-i}.* □

There is a subtlety here. To achieve bi-partiteness of the game graph, we have to introduce artificial vertices of outdegree one. Adding retreat edges from these trivial vertices would increase the dimension of the problem (i.e., the number of vertices where there is a choice), which we want to avoid in our definition. Obviously, non-increasing the dimension is crucial for efficiency.

Avoiding the addition of retreat edges from unary vertices is possible for the following reason. Assume Player 0 has a strategy that does not allow Player 1 moving to the retreat. This is easy to determine by computing strongly connected components on the graph generated by the vertices of Player 0 and unary vertices of Player 1. Suppose that there is a loop in this graph winning for Player 0. This can be easily verified by a modification of the Karp optimal loop algorithm. Then we do not need to apply any further machinery, the game immediately simplifies. We simply throw away unary vertices losing for Player 1 and all vertices from which Player 0 can force the game to such vertices. If, on the other hand, all such loops are winning for Player 1, he does not need any retreat edges. This justifies that retreat edges from unary edges need not be added, and transforming a game to the retreat form does not increase the dimension.

As a consequence, we have the following

Proposition 3.2 *The loop value of each vertex in any game with retreats is either 0 or 1. If the game with retreats G' is obtained from a game G, then for $i = 0, 1$ the vertices with loop value i in G' (except retreats) form the winning set for Player i in G.* □

This allows us to concentrate on games with retreats, provided we are only interested in the parity of the optimal loop color.

Given a game, we can take the viewpoint of one of the players, consider the space of his positional strategies, and search for the best one by iterative improvement, using the RSSA, GSSA, or RMSA described above.

3.2 Properties of Games with Retreats

Games with retreats possess interesting and useful properties. We need several technical definitions.

Definition 3.3 (Retreat Facet) *Given a game G with retreats, call a* retreat facet *a subgame F of G obtained by fixing some retreat edge.* □

Informally, an absorbing facet (defined below) is better than all other facets obtained by fixing an edge from the same vertex.

Definition 3.4 (Absorbing Facet) *Let $\mathcal{S}$ be the strategy space of a game equipped with the strategy evaluation function, and F be a facet of $\mathcal{S}$ defined* *by an edge e from vertex v. Say that F is an* absorbing facet *if every strategy σ is superseded by its projection $\sigma[v \to e]$ on F, i.e., $\sigma \preceq \sigma[v \to e]$.* □

The following theorem is one of the key ingredients of our algorithm.

Theorem 3.5 *Suppose $i \in \{0,1\}$ and the winning set of Player $1-i$ is nonempty. Then there is a vertex w_j of Player i such that the edge from w_j to the retreat v_{1-i}:*

1. *belongs to an optimal strategy of player i,*

2. *determines an absorbing facet on the the set of strategies of player i.*

Proof. The proof depends on the properties of the strategy evaluation function [5], as described above in Section 2.2, and will appear in the full paper. Roughly, suppose that Player 0 has a loop through his losing vertices, never taking a retreat. Than he loses with a loop value worse than 1. If he is interested in minimizing his loss, he should take a retreat edge. Note that *not every* (just some) losing vertex should play to a retreat. □

Another useful property of retreats is also due to the properties of the strategy evaluation measure.

Lemma 3.6 *Let $\sigma_0, \sigma_1, \ldots,$ be an improving sequence of strategies for Player i ($i \in \{0,1\}$) obtained by a Single or Multiple Random Switch Algorithms on a strategy hyperstructure determined by a parity game with retreats.*

If σ_{j+1} is obtained from σ_j by making a switch in vertex w_p from the retreat edge to another edge, then no strategy σ_k with $k > j$ contains the retreat edge from w_p.

(Informally, if the player in w_p decides to give up a retreat edge, he never considers taking it again. Eventually, he may get into the same retreat vertex, but following a different, better path.)

Proof. This follows from the partial order property (no cycles) for our strategy improvement measure [5]. □

Random Switch Algorithms are very good in optimizing fully absorbing functions on boolean hypercubes, as was shown by [15, 23] (see also [21, p. 76.]) for the Single Random Switch Algorithm.

Definition 3.7 (Fully Absorbing Boolean Hypercube) *Consider an injective function f on a d-dimensional boolean hypercube H^d. Call a facet ($(d-1)$-dimensional face) of H^d* absorbing *if all its vertices have bigger values than their neighbors on the complementary facet. Say that $f : H^d \to \mathcal{D}$ is* fully absorbing *if either $d = 1$ or H^d has a an absorbing facet that is fully absorbing.* □

Theorem 3.8 ([15, 23]) *The expected number of iterations of the Random Single Switch Algorithm, from any starting point to the global optimum, is at most $\binom{d+1}{2}$ for any fully absorbing hypercube.* □

We extend it to hyperstructures (as opposed to hypercubes) and to the Random Multiple Switch Algorithm as well. We only need to care about retreat absorbing facets, and the theorem below guarantees us to reach them easily.

Theorem 3.9 *The expected number of iterations from any starting point until reaching all absorbing facets determined by retreat edges is:*

1. *$O(d^3)$ for the Random Single Switch Algorithm, and*

2. *$O(d^2)$ for the Random Multiple Switch Algorithm.*

Proof. Let F be an absorbing facet determined by a retreat edge e from some vertex w_p. Since for any strategy τ on any other face F' determined by another edge leaving w_p the switch from τ to $\tau[w_p \to e]$ is *improving* (this relies on our strategy evaluation [5]), we switch from F to F' at each improving step with probability at least $1/d(d-1)$. This is because we have at most d vertices to switch, in each one to at most $d-1$ edges. Therefore, the expected time to forever switch to an improving facet F is $O(d^2)$. Consequently, the overall expected time to reach all absorbing facets is at most $O(d^3)$. The latter estimate holds for the Random Single Switch Algorithm. Random Multiple Switch Algorithm fairs slightly better in $O(d^2)$ expected steps until discovering (reaching) all absorbing facets. Indeed, at every iteration the probability to switch to an absorbing face F is at least $1/2(d-1)$, because a vertex where an improvement is possible is selected with probability $1/2$, and the absorbing edge/facet is selected with probability $1/(d-1)$. □

By restarting the algorithms several times and using Markov's inequality[4], we may guarantee that after making polynomially many iterations, all absorbing facets will be reached with high probability.

3.3 First Stage: Finding Candidate Retreat Absorbing Facets

Suppose first that we are optimizing for Player 0. Run the Single or Multiple Random Switch Algorithm from any initial strategy for $O(d^3)$ or $O(d^2)$ number of steps. Theorem 3.9 implies that after that many iterations all absorbing retreat facets will be reached with high probability, and will never be left by further iterations. In particular, all absorbing facets determined by retreat edges, if any, will be reached, as guaranteed by Theorem 3.5. Symmetrically, and in parallel, we are optimizing for Player 1, and after $O(d^3)$ or $O(d^2)$ steps with high probability reach all absorbing facets, if any. During the runs of both algorithms and afterwards we are looking for and most interested in the *absorbing facets candidates* among retreat facets. The next section explains how real absorbing facets among those candidates allows to completely solve the game.

What is an Absorbing Retreat Facet Candidate? After running for polynomially many steps, the RSSA or RMSA produce a polynomially long sequence of strategies $\bar{\sigma} \equiv \sigma_0, \ldots, \sigma_r$. Looking at this sequence we want to determine, which among retreat facets are *possibly absorbing.* Obviously, for such candidates the final strategy σ_r should contain the corresponding retreat edge. Additionally, to diminish the set of spurious candidates, we verify the following necessary conditions:

1. *every* strategy in the sequence $\bar{\sigma}$ on the way to an absorbing retreat facet candidate has a possible improving switch to the corresponding retreat edge,

2. *every* strategy in the sequence $\bar{\sigma}$ after an absorbing retreat facet candidate has been reached has no improving switches leaving the retreat edge (facet).

[4]The probability that a random variable takes a value k times bigger than its expectation does not exceed $1/k$.

If any of the conditions above is violated, a retreat edge *cannot* define an absorbing facet, by definition. Note that these conditions are *necessary*, but not sufficient. As soon as we run the algorithm only for polynomially many steps, it is conceivable that the facet we believe absorbing will actually be left after a more significant number of improvement steps.

Running the RSSA or RMSA several times may be helpful in diminishing the size of the set of potential absorbing facet candidates. If two different runs give two different resulting sets, it is safe to take their intersection as a result. If a run is started from the strategy that goes to the retreat in every vertex, the computed set of candidates is guaranteed to contain all absorbing retreat facets.

3.4 Absorbing Retreat Facets Decide the Game

It turns out that knowing all *really* absorbing retreat facets (as opposed to the superset of candidates determined in the previous section) is as good as being able to completely solve the game. Indeed, deciding a winner in all other vertices amounts to solving a finite reachability game.

Lemma 3.10 *Let R_i, $i = 0, 1$ be the sets of game vertices where Player i takes retreat edges (assuming both play optimally). Then all other vertices of the game partition into the winning sets of of the players determined by solving a finite reachability game with final winning sets R_i.*

Proof. The sets of absorbing retreat facets cannot be empty for both players simultaneously. The opposite would be a contradiction. Indeed, if nobody ever takes a retreat edge, the loop value of the game is never 0 or 1. Therefore, either Player 0 secures a win with an even loop color > 0, or Player 1 wins with an odd loop color > 1. But this contradicts the fact that Player i secures a loss with a color at most $1 - i$.

Now we know that some vertices of the game are immediately decided as winning for one of the players. Namely, the vertices in R_i that take retreat edges are winning for the opponent Player $1 - i$. Similarly, vertices of Player $1 - i$ with edges leading to R_i are winning for Player $1 - i$. We claim that all the other vertices are decided in this manner, i.e., by solving a finite reachability game with sets R_i. Suppose not, and for some vertices both players have optimal strategies allowing them to play never reaching R_i's. But in such plays, for one of the players the losing loop value would be worse than 0 or 1 respectively. Therefore, one of the players will certainly prefer to take a retreat edge, reaching one of the R_i, a contradiction. $\square$

It follows that once we know which of the candidates are really absorbing (rather than just spurious), we know everything about the game, i.e., can solve who wins in each vertex. Actually, it gives a polynomial time reducibility.

Performing this stage is very useful, it allows us to concentrate on an essential, smaller, sets of vertices. All other vertices will be easy to solve.

Imagine in particular, that for one of the players there are no candidate absorbing facets at all. In this case we terminate: all vertices are winning for that player. Of course, in general, we are not always that lucky. We therefore need to proceed to the next stage.

3.5 Second Stage: Finding Absorbing Facets by Dual Iterations

Now let us compare the outcomes of the two parallel runs of the RSSA or RMSA, one for each player, making polynomially many iteration each in an attempt to figure out absorbing retreat facets.

Suppose the algorithm for Player 0 reports vertices $u_1, \ldots, u_p$ as potential candidates to determine absorbing retreat facets. This means that the final strategy after polynomially many iterations of the algorithm contains edges to the 1-retreat, and the necessary conditions from Section 3.3 are satisfied.

Symmetrically the algorithm for Player 1 reports vertices $v_1, \ldots, v_p$ as potential candidates to determine absorbing retreat facets. (Note that u_j's are vertices of Player 0, and v_j's are vertices of Player 1.)

Consider the following table summarizing all possible cases of outcomes of the two algorithms.

	a	b	c	d
Player 0	1	0	0	1
Player 1	1	0	1	0

We interpret this table case by case.

Case a. Vertex u_j is reported by the algorithm for Player 0 as determining a potential absorbing retreat facet (with loop value 1). Simultaneously, the iterations of the dual algorithm for Player 1 confirm that u_j is winning for Player 1 with loop color 1. This is a clear case, the vertex u_j indeed determines the absorbing facet (losing for Player 0), and the dimension of the problem (game) diminishes.

Case b. Symmetrically, vertex v_j is reported by the algorithm for Player 1 as determining a potential absorbing retreat facet (with loop value 0). Simultaneously, the iterations of the dual algorithm for Player 0 confirm that v_j is winning for Player 0 with loop color 0. This is a clear case, the vertex v_j indeed determines the absorbing facet (losing for Player 1), and the dimension of the problem (game) diminishes.

Case c. This case is impossible. The iterations of the algorithms for both players report contradictory results: one says that it is winning for Player 0, the other that it is winning for Player 1.[5]

Case d. This is the only nontrivial case. The algorithm for each player reports that the vertex is *possibly losing* for this player.

Consequently, after comparing the outcomes of the RSSA (or RMSA) run in parallel on polynomially many steps, optimizing for both players simultaneously, we are left with nontrivial sets of potential absorbing retreat facets candidates. The advantage of the work done so far consists in diminishing the dimension of the problem. The next stage we are going to describe is more expensive (no polynomial upper bound is currently known), therefore, diminishing the dimension is crucial for the overall performance.

It may happen, that all of the candidates for one of the players are eliminated during that stage. In this case we terminate: the game is completely solved.

3.6 Third Stage: Eliminating Spurious Absorbing Facet Candidates

We proceed to the final, most expensive stage of the algorithm. At that point all trivial cases are eliminated by the preceding stages, and we are left with the strictly smaller subsets of vertices we need to classify as real/spurious absorbing retreat facets.

Let $v_1, \dots v_p$ be all vertices of Player i corresponding to candidate retreat absorbing facets $F_1, \dots, F_p$ (determined by taking the retreat edges from v_i's).

[5]Our algorithms have no false positives. When they say 'the vertex is winning' it is always correct. They can make mistakes saying 'the vertex is losing', which should be interpreted as 'the vertex is *possibly* losing'.

1. some of these F_i's are indeed *real retreat absorbing facets*
2. some are not, i.e., *spurious*.

Suppose, without loss of generality, that the facets F_i (vertices v_i) are ordered in such a way that

$$\nu(F_i) < \nu(F_j) \Rightarrow j < i,$$

where $\nu(F)$ is the value of the best strategy on F, i.e., best facets come first.

This can be done by changing the order of indices and topological sorting. Note that we actually *do not know this ordering*.

There are several important observations to be made:

1. $p < n$ — this is a clear advantage, since the dimension of the problem strictly diminishes by the previous stages;
2. denote by $s(F_i)$ the number of facets among $F_1, \dots, F_p$ that have *bigger* value than $\nu(F_i)$; note that $s(F_i) \in \{0, \dots, p-1\}$ and $s(F_i) < i$;
3. for really absorbing facets $s(F_i) = 0$; for spurious candidates $s(F_i) > 0$; for a non-absorbing facet $s(F_i) = 0$ only if all F_i are non-absorbing, hence Player i wins in all vertices (see Theorem 3.5).

Suppose we sample uniformly at random one of the facets F_i with the purpose to explore it and to find an optimal strategy on that facet.

If we are lucky and $s(F_i) = 0$ then after this search we obtain an optimal strategy on F_i, which is also an optimal strategy in the full game. If (by magic) we were always lucky we would have the recurrence $T(p) = T(p-1) + poly(n)$ with a *polynomial* solution.

However, since we do not know anything about the actual ordering of absorbing facet candidates, we may only rely on chance. If we are unlucky, and choose a facet F_i with $s(F_i) = r > 0$, then after finding the best strategy on F_i, the algorithm, proceeding by improvement steps only, will never ever visit any facets F_j with $s(F_j) > r$. Therefore, the actual (hidden) dimension of the remaining problem will be r rather than $p - 1$. As a consequence, we obtain the following recurrence on the expected number of switches of the algorithm:

$$T(p) = T(p-1) + 1 + \frac{1}{p}\sum_{i=0}^{p-1} T(i) \tag{1}$$

This recurrence is similar to the one obtained by Ludwig [16] for his subexponential randomized algorithm for *binary* simple stochastic games. In [5, 6] we showed two randomized algorithms, based on [14] and [18], that apply for games of *arbitrary outdegrees* and remain subexponential. Here, by concentrating on the retreat facets, a binary analysis is applies: each vertex either goes to the retreat or not. Therefore, the recurrence (1) holds. It has solution $2^{O(\sqrt{p})}$ [16]. Actually, the analysis is slightly more involved. Starting with p candidate retreat facets we unfold the recursion until reaching the bottom. At this point we have $n-p$ remaining vertices not taking retreats. We need to define the best possible strategy in those vertices. This can be done by our Kalai-Matoušek-Sharir-Welzl-Ludwig-style algorithms [5, 6] in expected time $2^{O(\sqrt{n-p})}$. The overall complexity thus becomes $2^{O(\max\{\sqrt{p},\sqrt{n-p}\})}$

The apparent advantages of our algorithm are:

1. the previous phases, searching for potential absorbing facets, and finding simple ones, reduce the dimension of the problem;

2. while the first recursive call is processed, finding an optimal strategy on the facet, we may simultaneously watch and exclude candidate absorbing facets which do not satisfy the necessary conditions from Section 3.3; this in turn implies that the actual recurrence is better than (1), because we sample only from the remaining candidates, and the algorithm actually converges faster.

3.7 The Algorithm: Summary

We summarize all steps of the algorithm described above.

1. Transform an input parity game into a game with retreats.

2. Determine the set of candidate absorbing facets $\mathcal{F}$ defined by retreat edges, by running the RSSA of the RMSA for polynomially many steps (cf., Theorem 3.9), and simultaneously verifying the necessary conditions from Section 3.3. If one of the players has no candidate absorbing retreat facets, terminate, all vertices are winning for that player.

3. By comparing and analyzing the outcomes of both algorithms, as described in Section 3.5, figure out facets that are definitely absorbing retreat facets.

If one of the players has no candidate absorbing retreat facets, remaining at this point, terminate, all remaining vertices are winning for that player.

4. Pick uniformly at random one of the remaining candidate facets $F \in \mathcal{F}$ of the game G generated by a retreat edge e from a vertex u, and recursively, starting from step 4, find an optimal strategy σ on F. During this search watch all the strategies visited, and keep the set $\mathcal{F}$ up-to-date, by verifying the conditions from Section 3.3, and excluding the facets not satisfying them.

5. If the strategy σ, extended to the full game G by adding the retreat edge e from u turns out optimal in G (this is checked by verifying profitability of a switch from e to some other edge leaving u), terminate returning $\sigma' \equiv \sigma \cup \{u \to e\}$.

6. Otherwise, make a profitable switch in u (possible since σ' is non-optimal), and recursively repeat the algorithm starting from step 4.

3.8 Additional Improvements

To optimize the algorithm we can employ the following additional heuristics.

1. In order to diminish the number of candidate absorbing retreat facets, we can restart the RSSA (RMSA) several times from random initial strategies, and perform First and Second Stages, as described in Sections 3.3 and 3.5. In the first stage it is safe to take the intersection of all candidate sets. In the second stage multiple restarts may allow for deciding more definitive absorbing facets.

2. Additionally, it is always useful to start from the strategy where the player takes retreat edges from all vertices. The advantage is that according to Lemma 3.6, once a player switches from a retreat edge, he will never switch back, and this edge is forever excluded from the set of potential retreat absorbing facets candidates. (This holds even if he has an incentive to switch, at some point.)

The detailed complexity analysis (in addition to a sketch in Section 3.6) of the algorithm will appear in the full paper. In conclusion we mention that:

1. Our strategy evaluation measure from [5] we use in the algorithm allows for at most $O\left(n^3 \cdot \left(\frac{n}{k}+1\right)^k\right)$ improving iterations. This imposes an independent restriction on the number of iterations, which may be better than the subexponential bound when the number of colors is small. So our algorithm may terminate earlier.

2. The decrease in the problem dimension can be made precise as follows. If we start with the 'all go to a retreat' strategy, then any potential switch from a retreat edge in the run of the RSSA (RMSA) will forever exclude one potential retreat facet from the set of candidates. If there are q current vertices where non-retreat edges are taken, then we can easily estimate the maximal number of iterations until one more retreat edge is given up (or the algorithm terminates). Although it does not guarantee a substantial decrease in subexponential time, we feel that the decrease should be substantial on practical problem instances, and believe a stronger analysis will be possible. This requires further investigations and will be reported in later publications.

References

[1] H. Björklund, V. Petersson, and S. Vorobyov. Experiments with iterative improvement algorithms on completely unimodal hypercubes. Technical Report 2001-017, Information Technology/Uppsala University, September 2001. http://www.it.uu.se/research/reports/.

[2] H. Björklund, S. Sandberg, and S. Vorobyov. An experimental study of algorithms for completely unimodal optimization. Technical Report 2002-030, Department of Information Technology, Uppsala University, October 2002. http://www.it.uu.se/research/reports/.

[3] H. Björklund, S. Sandberg, and S. Vorobyov. Memoryless determinacy of parity games: A simple proof. Technical Report 2002-033, Department of Information Technology, Uppsala University, October 2002. http://www.it.uu.se/research/reports/, submitted for publication.

[4] H. Björklund, S. Sandberg, and S. Vorobyov. Optimization on completely unimodal hypercubes. Technical Report 018, Uppsala University / Information Technology, May 2002. http://www.it.uu.se/research/reports/.

[5] 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. Full preliminary version: TR-2002-026, Department of Information Technology, Uppsala University, September 2002.

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

[7] 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.

[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. Automata, tableaux, and temporal logics. In *Proc. Workshop on Logic of Programs*, volume 193 of *Lect. Notes Comput. Sci.*, pages 79–87. Springer-Verlag, 1985.

[10] 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.

[11] 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.

[12] 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.

[13] M. Jurdziński. Small progress measures for solving parity games. In *17th STACS*, volume 1770 of *Lect. Notes Comput. Sci.*, pages 290–301. Springer-Verlag, 2000.

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

[15] D. G. Kelly. Some results on random linear programs. *Methods of Operations Research*, pages 351–355, 1981.

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

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

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

[19] V. Petersson and S. Vorobyov. A randomized subexponential algorithm for parity games. *Nordic Journal of Computing*, 8:324–345, 2001.

[20] H Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(3):303–308, 1996.

[21] C. A. Tovey. Local improvement on discrete structures. In E. Aarts and Lenstra J. K., editors, *Local Search in Combinatorial Optimization*, pages 57–89. John Wiley & Sons, 1997.

[22] J. Vöge and M. Jurdziński. 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.

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

Recent technical reports from the Department of Information Technology

2002-038 Parosh Aziz Abdulla and Alexander Rabinovich: *Verification of Probabilistic Systems with Faulty Communication*

2002-039 R. Blaheta, S. Margenov, and M. Neytcheva: *Uniform estimate of the constant in the strengthened CBS inequality for anisotropic non-conforming FEM systems*

2002-040 Torsten Söderström: *Why are errors-in-variables problems often tricky?*

2002-041 Per Lötstedt and Martin Nilsson: *A Minimum Residual Interpolation Method for Linear Equations with Multiple Right Hand Sides*

2003-001 Parosh Abdulla, Johann Deneux, Pritha Mahata, and Aletta Nylén: *Downward Closed Language Generators*

2003-002 Henrik Björklund, Sven Sandberg, and Sergei Vorobyov: *On Combinatorial Structure and Algorithms for Parity Games*

2003-003 Magnus Svärd and Jan Nordström: *A Stable and Accurate Summation-by-Parts Finite Volume Formulation of the Laplacian Operator*

2003-004 Kaushik Mahata and Torsten Söderström: *Subspace estimation of real-valued sine wave frequencies*

2003-005 Samuel Sundberg: *Solving the linearized Navier-Stokes equations using semi-Toeplitz preconditioning*

2003-006 Henrik Brandén and Per Sundqvist: *An Algorithm for Computing Fundamental Solutions of Difference Operators*

2003-007 Henrik Brandén, Sverker Holmgren, and Per Sundqvist: *Discrete Fundamental Solution Preconditioning for Hyperbolic Systems of PDE*

2003-008 Julian Richardson and Pierre Flener: *Program Schemas as Proof Methods*

2003-009 Alexandre David, M. Oliver Möller, and Wang Yi: *Verification of UML Statecharts with Real-Time Extensions*

2003-010 Alexandre David, Johann Deneux, and Julien d'Orso: *A Formal Semantics for UML Statecharts*

2003-011 Alexandre David, Gerd Behrmann, Kim G. Larsen, and Wang Yi: *A Tool Architecture for the Next Generation of UPPAAL*

2003-012 Ken Mattsson and Jan Nordström: *Finite Difference Approximations of Second Derivatives on Summation by Parts*

2003-013 Ken Mattsson, Magnus Svärd, and Jan Nordström: *Stable Artificial Dissipation Operators for High Order Finite Difference Schemes*

2003-014 Brahim Hnich, Julian Richardson, and Pierre Flener: *Towards Automatic Generation and Evaluation of Implied Constraints*

2003-015 Henrik Björklund and Sven Sandberg: *Algorithms for Combinatorial Optimization and Games Adapted from Linear Programming*

2003-016 Bertil Gustafsson and Eva Mossberg: *Time Compact High Order Difference Methods for Wave Propagation*

UPPSALA
UNIVERSITET

March 2003
ISSN 1404-3203
http://www.it.uu.se/