Is Randomized Gurvich-Karzanov-Khachiyan's Algorithm for Parity Games Polynomial?

Emmanuel Beffara (ENS Lyon),
Sergei Vorobyov (IT UU)

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

Technical report 2001-025
October 2001
ISSN 1404-3203

Abstract

We report on the experimental study of the Gurvich-Karzanov-Khachiyan (GKK) algorithm for cyclic games adapted for *parity games* (equivalent to the μ-calculus model checking), one of the major open problems in complexity and automata theories, computer-aided verification. The algorithms demonstrates excellent polynomial (actually, a sublinear number of iterations) behavior in a substantial (millions) number of experiments with games of sizes up to 20.000 and 6 – 7 colors. It also allows for a natural randomization. We conducted extensive experiments of the randomized GKK algorithm on the 'hard' Lebedev-Gurvich's game instances, which force the deterministic version of the algorithm to make *exponentially* many iterations. With high probability the algorithm converges after just a few hundred iterations (compared with 2^{50} – 2^{60} for deterministic version). This allows for giving up computations that converge slowly and restart with a fresh initial potential transformation. While it remains to be theoretically justified and proved, we present convincing experimental data on dependency of the average and maximal number of iterations on game sizes, outdegrees, colors, initial randomizations.

Topics: automata, game theory, computational complexity, logic in computer science, verification.

Note: The web page `http://www.csd.uu.se/ vorobyov/eb-gkk/` contains extensive experimental data, Maple worksheets, histograms, supporting claims contained in this report, which could fit only a small amount of experimental material. The information on the web page is rapidly and regularly updated.

Acknowledgments. Supported by TFR Grant 221-2000-103 "Games for Computer-Aided Verification: Algorithms and Complexity". Many thanks to Vladimir Gurvich for providing examples forcing the deterministic GKK algorithm behave exponentially.

Correspondence to: `vorobyov@csd.uu.se`

Contents

List of Figures

List of Tables

1 Introduction

In this paper we report on the implementation and experimental results with the new (at least in the model-checking and automata theory community) algorithm for solving the Parity Games problem based on the adaptation of the method due to Gurvich, Karzanov & Khachiyan (1988). The Parity Games problem is known to be equivalent to the so-called *propositional μ-calculus model checking*, a fundamental problem in program verification. The modal μ-calculus is a powerful logic expressively subsuming virtually all known temporal logics of programs (Emerson 1997, Emerson 1990). Therefore, settling this problem represents both theoretical and practical interests.

Solving parity games means finding the so-called *optimal strategies* (or *Nash equilibria*), a problem whose complexity is a fundamental open question in complexity theory, as Papadimitriou (2001) stressed in a recent lecture: *"The most interesting aspect of the Nash equilibrium concept to our community is that it is a most fundamental computational problem whose complexity is wide open. Suppose that two players have finitely many strategies each. Is there a polynomial algorithm for computing a (mixed) Nash equilibrium in such a game? [...] Together with factoring, the complexity of finding a Nash equilibrium is in my opinion the most important concrete open question on the boundary of P today."* Not surprisingly, the precise complexity characterization of Parity Games remains open.

Several approaches to solving parity games were extensively studied in the literature. The first one is based on the equivalence with the μ-calculus and computing fixpoints of monotonic operators in lattices, represented by Browne, Clarke, Jha, Long & Marrero (1997), Browne et al. (1997), Seidl (1996), Jurdzinski (2000); see also the survey (Emerson 1997). The second approach uses the iterative discrete strategy improvement algorithms (Vöge & Jurdzinski 2000). The third approach is based on a reduction to Mean Payoff Games problem (Puri 1995, Jurdziński 1998) and using a pseudopolynomial dynamic programming algorithm of Zwick & Paterson (1996). Other approaches are based on randomized simplex-like subexponential algorithms (Petersson & Vorobyov 2001*b*), on interior-point optimization schemes (Petersson & Vorobyov 2001*a*), or on pseudo-boolean optimization of completely unimodal functions (Björklund, Petersson & Vorobyov 2001). In (Beffara & Vorobyov 2001) and in this paper we try yet another approach, less familiar and traditional in the computer-aided verification and model-checking community.

Gurvich et al. (1988) suggested an algorithm (later referred to as GKK) for solving the so-called Mean Payoff Games problem, a slightly more general problem than Parity Games. However, their solution, quite different from known fixed-point algorithms for parity games, was ignored for a long time by the model-checking community. Recently (Pisaruk 1999) demonstrated that the algorithm is actually *pseudo-polynomial*, a result previously known for Mean Payoff Games from Zwick & Paterson (1996), using another, dynamic programming algorithm. Moreover, the behavior of the GKK algorithm is somewhat similar to the good practical behavior of the Simplex method. We were unable to observe or construct any instances of parity games that would require exponentially many iterations of the randomized version of the algorithm. The advantages of the GKK algorithm, compared to other known algorithms for *Parity Games*, are as follows. It is a discrete graph algorithm, unlike (Petersson & Vorobyov 2001*a*) based on continuous optimization techniques. Another discrete graph-theoretic algorithm of (Vöge & Jurdzinski 2000) has an *exponential* worst-case bound, while GKK is pseudopolynomial (which guarantees its termination in a low degree polynomial time for a small number of colors). Another known pseudopolynomial algorithm of Zwick & Paterson (1996) starts to demonstrate its worst-case behavior on trivial examples, where just one player has choices (which is not the case for GKK). The algorithms of Browne et al. (1997), Seidl (1996) are exponential in time and space, with known examples of exponential behavior. The algorithm of Jurdzinski (2000) (randomized or not) also performs poorly on some known examples. In contrast, the GKK algorithm naturally randomizes. All this makes the GKK an attractive alternative to investigate. We hope in the later studies, and its was one of the main motivations of this work, to be able to improve the known pseudopolynomial upper bound of the randomized GKK algorithm in the particular case of PARITY GAMES.

In this work, we continue the experimental study of the behavior of the GKK algorithm adapted to PARITY GAMES, started in (Beffara & Vorobyov 2001). An implementation was carried out (Beffara & Vorobyov 2001) in the *literate programming* paradigm due to Knuth (1984), Knuth (1992), in which: *"Computer programs are fun to write, and well-written computer programs are fun to read. One of the life's greatest pleasures can be the composition of a computer program that you know will be a pleasure for other people to read and for yourself to read."*

The extensive experiments on a large number (millions of games of sizes 1000 – 20.000) of randomly generated games demonstrate that, on the average, the number of steps required to solve parity games using GKK seems sub-linear, and that the most complicated examples are generated for small mean vertex outdegrees. In our experiments the algorithm never made more that 4500 iterations and in any case always less than half the number of vertices in the game graph.

Vladimir Gurvich (2001) has kindly provided us with an example (possibly the same as mentioned by Gurvich et al. (1988, p. 90)) of the exponential behavior of the GKK algorithm. We discuss it in Section 6. It turns out that a simple randomization we already implemented and investigated on random instances of games shows excellent results on those exponentially hard Lebedev-Gurvich instances of games, and the randomized GKK converges extremely fast (a few hundred iterations versus $O(2^n)$ for deterministic version).

After preliminaries on infinite games we proceed to the description of the adapted algorithm and then to the implementation details and experimental results. Details about this work with more experimental data can be found in (Beffara & Vorobyov 2001). Fresh results on ongoing experiments, with larger data, different distributions, will be posted on the associated web site `http://www.csd.uu.se/~vorobyov/eb-gkk`.

2 Preliminaries

We consider *parity games* (PGs for short) and *mean-payoff games* (MPGs for short) (Gurvich et al. 1988, Zwick & Paterson 1996, Emerson 1997, Jurdziński 1998). These are infinite games on finite graphs with two antagonistic players, **1** and **2**.

Definition 2.1 (Parity Games) *Let $G = (V, E)$ be a finite directed graph without sinks, $\{V_{\mathbf{1}}, V_{\mathbf{2}}\}$ be a partition of its vertex set, and p be a function from V to $\mathbb{N}$. Then the tuple $\mathcal{N} = (V_{\mathbf{1}}, V_{\mathbf{2}}, E, p)$ is called a* parity game *with the* priority function *p.* $\square$

(See also (Emerson, Jutla & Sistla 1993, Emerson 1997, Jurdziński 1998, Jurdzinski 2000).)

A play in a PG consists in moving a token along the edges of the graph. First, an initial position v_0 is given in V. Then, at each step i, one of the players pushes the token along an edge starting at v_i. If $v_i \in V_{\mathbf{1}}$, then the edge is chosen by player **1**, otherwise it is chosen by **2**, which defines the next position v_{i+1}. This process defines an infinite sequence $\mathcal{V} = (v_i)_{i \geqslant 0}$ in V, and the priority function assigns a value to each node in this sequence. Denote by $\mathit{Inf}(\mathcal{V})$ the set of priority values appearing an infinite number of times in $\mathcal{V}$: $\mathit{Inf}(\mathcal{V}) = \big\{ p \mid p \in \mathbb{N}, \big|\{ i \mid p(v_i) = p \}\big| = \infty \big\}$. Then player **1** is declared to be the winner of the play if the largest integer in $\mathit{Inf}(\mathcal{V})$ is *even*, and **2** is the winner if this number is *odd.*

In **Mean-Payoff Games** the valuation is determined by the average cost of the edges traversed during the play; see (Gurvich et al. 1988, Zwick & Paterson 1996).

Definition 2.2 (Mean Payoff Games) *Let $G = (V, E)$ be a finite directed graph without sinks, $\{V_{\mathbf{1}}, V_{\mathbf{2}}\}$ be a partition of its vertex set, and c be a function from E to $\mathbb{Z}$. Then the tuple $\mathcal{N} = (V_{\mathbf{1}}, V_{\mathbf{2}}, E, c)$ is called a* mean payoff game *with the* cost function *c.* $\square$

The players construct infinite sequences $\mathcal{V} = (v_i)_{i \geqslant 0}$ in the same way as in parity games. To decide the winner, consider the following *mean payoff* function for player **1**:

$$f_{\mathbf{1}}(c, \mathcal{V}) = \liminf_{n \to \infty} \frac{1}{n} \sum_{i=0}^{n-1} c(v_i, v_{i+1})$$

and $f_{\mathbf{2}}$ is defined the same way for player **2** using an upper limit. Then $f_{\mathbf{1}}(c, \mathcal{V})$ is considered to be the reward of the game for player **1**, and $-f_{\mathbf{2}}(c, \mathcal{V})$ to be the reward for player **2**; thus **1** wants to maximize the mean cost while **2** wants to minimize it. Say that player **1** is the *winner* if $f_{\mathbf{1}}(c, v) > 0$, and player **2** is the *winner* if $f_{\mathbf{1}}(c, v) \leqslant 0$. This definition of a winner is enough for our purposes. Moreover, we will construct MPGs in a way the value 0 cannot be obtained no mater how the players move, so determining a winner amounts to determining the sign of an optimal payoff.

Decision Problems for Games. The main problem in both kinds of games is the following. Given a game and an initial position, we want to know if one of the players has a *winning strategy*, i.e., a way to choose his moves so that he is assured to win the game, no matter how his opponent plays.

Stationary Strategies. For the games considered, without loss of generality, we may restrict attention to the so-called *stationary strategies*, also known as *memoryless* strategies.

Definition 2.3 *(Stationary Strategy) Let* $\mathcal{N}$ *be a game and* $G = (V_{\mathbf{1}} \cup V_{\mathbf{2}}, E)$ *be its graph. A* stationary strategy *for player* $\boldsymbol{X}$ *is any function* s *associating to each vertex in* $V_{\boldsymbol{X}}$ *a successor vertex. A play* $(v_i)_{i\in\mathbb{N}}$ *is* consistent *with a stationary strategy* s *if, for each step* i *such that* v_i *is in* $V_{\boldsymbol{X}}$*, one has* $v_{i+1} = s(v_i)$*.* □

Memoryless Determinacy. A fundamental fact is that both kinds of games are always solvable in stationary strategies. In MPGs, for each starting position, there exists a constant p (called the *value of the game*) such that player **1** can force the game to have a payoff at least p (by following a stationary strategy) and player **2** can force it to have a payoff at most p (also by a stationary strategy), independently of the strategy the opponent applies; see (Moulin 1976, Ehrenfeucht & Mycielski 1979, Gurvich et al. 1988).

For PGs, similarly, there is a partition of vertices into two sets and stationary strategies on those sets that guarantee both players to win, when the game starts in a respective set, independently of the strategy applied by the opponent.

2.1 Economical Reduction from PGs to MPGs

It is well known that PGs can be reduced to MPGs. We give an economical such reduction (crucial for practical applicability!) and then study how the algorithm for MPGs due to Gurvich et al. (1988) and Pisaruk (1999) adapts to the resulting MPGs. There is a usual transformation of PGs into MPGs, which keeps the graph of the game unchanged and assigns initial costs to edges depending on the priority of their source vertex. The method we use here is an optimized version of (Puri 1995, Jurdziński 1998).

Consider a PG $\mathcal{N} = (V_{\mathbf{1}}, V_{\mathbf{2}}, E, p)$, with p taking values in $\{0, \ldots, d\}$. Construct the MPG $\tilde{\mathcal{N}} = (V_{\mathbf{1}}, V_{\mathbf{2}}, E, c)$ by defining for every edge $(u, v) \in E$ its cost $c(u, v) = w_{p(u)}$, for some sequence of weights (w_i) growing in absolute value, to be defined later. The idea is to define the sequence (w_i) in such a way that for any (eventually cyclic) play $\mathcal{V}$, the sign of the mean payoff is determined only by the largest w_i on the period. Considering an eventually cyclic path $\mathcal{V} = (v_i)_{i\geqslant 0}$ in the game (as obtained if both players conform to stationary strategies), the mean cost of the edges along the path is

$$f(c, \mathcal{V}) = \lim_{n\to\infty} \sum_{k=0}^{n-1} c(v_k, v_{k+1}) = \frac{1}{|C|} \sum_{(u,v)\in C} c(u, v) = \frac{1}{|C|} \sum_{(u,v)\in C} w_{p(u)}.$$

We want $f(c, \mathcal{V})$ positive iff the largest value of p on the cycle C is even. A usual method (Puri 1995, Jurdziński 1998) consists in defining w_p to be $(-|V|)^p$ for every p, because in this case one edge with cost w_p dominates the sum of the costs of all edges with cost less than w_p.

However, there are smaller values of (w_i) for which the reduction works. If we define n_i to be the number of vertices of priority i, then the validity condition is fulfilled if the following holds:

$$\begin{aligned} w_i &\geqslant 1 - \textstyle\sum_{0<2j+1<i} n_j w_j \quad \text{if } i \text{ is even,} \\ w_i &\leqslant -1 - \textstyle\sum_{0\leqslant 2j<i} n_j w_j \quad \text{if } i \text{ is odd,} \end{aligned} \tag{1}$$

because then the absolute value of each w_i is larger than the sum of the absolute value of all w_j of the opposite sign with $j < i$ multiplied by the number of vertices of priority j. If we define (w_i) from the formulas above using equalities, we can reduce the definition into[1]

$$w_0 = 1, \qquad w_1 = -n_0 - 1 \qquad \text{and} \qquad w_i = w_{i-2} - n_{i-1}w_{i-1} \quad \text{for all } i \geqslant 2, \tag{2}$$

and the resulting sequence (w_i), resembling Fibonacci's, satisfies our condition. We call this cost assignment method the "*smallest*"[2] It turns out to be crucial for the ability to solve large games with 20K vertices, while in the Puri-Jurdziński method the weights grow too fast to be practical. The aim of defining the weights by (1), (2) is to keep them as small as possible, which is crucial for the GKK algorithm performance both theoretically and practically, as we additionally became convinced by experimenting.

As a consequence, $f(c, \mathcal{V})$ is never zero, and it is positive iff the highest value of $p(u)$ along the cycle is even. This proves that a cyclic play is winning for a given player in $\tilde{\mathcal{N}}$ iff it is also winning for him in $\mathcal{N}$.

We leave the reader to convince herself (see Beffara & Vorobyov (2001)) that winning strategies in $\tilde{\mathcal{N}}$ are actually winning strategies in $\mathcal{N}$.

[1]We could have started with $w_0 = 0$, but we wish to avoid the 0-valued cycles.

[2]Since it assigns the smallest possible values if one only consider the number of vertices of different color in and ignore the the structure of the graph. Better (smaller) results could be obtained by studying elementary cycles in the graph, which will be implemented later.

3 The GKK Algorithm

The algorithm we use to solve PGs is an adaptation of the algorithm due to Gurvich et al. (1988) used for solving MPGs. We will need some

Definitions, Notation. Given an MPG $(V_\mathbf{1}, V_\mathbf{2}, E, c)$, the *extremal* value of a vertex v in V is defined as

$$ext(c, v) = \begin{cases} \max\{c(v,u) \mid (v,u) \in E\} & \text{for } v \in V_\mathbf{1}, \\ \min\{c(v,u) \mid (v,u) \in E\} & \text{for } v \in V_\mathbf{2}. \end{cases}$$

Extremal edges are thus edges whose value is extremal for their origin vertex, and we define the *extremal successors* of a given vertex v as

$$V_{ext}(c, v) = \{u \mid (v,u) \in E, c(v,u) = ext(c,v)\}.$$

Finally, two cost functions c and c' are said to be *equivalent* if there exists a *potential function* ε from V to $\mathbb{R}$ such that for all edge (u, v),

$$c'(u,v) = c(u,v) + \varepsilon(u) - \varepsilon(v).$$

In the latter case we write $c' = c_\varepsilon$. The equivalence implies that, for every finite path $(v_0, v_1, \ldots, v_p)$ the cost of the path for c' is equal to the cost of c plus a term $\varepsilon(v_0) - \varepsilon(v_n)$, which is clearly bounded. Therefore, the mean payoff of every play $(v_i)_{i\in\mathbb{N}}$, when it exists, is the same for c and c'. The transformation of c into c_ε is called a *potential transformation.* This notion is central for the algorithm (see below).

3.1 The Method of Potential Transformations

The GKK algorithm constructs an optimal stationary strategy for both players by reducing (iteratively applying cleverly selected potential transformations) the initial game $(V_\mathbf{1}, V_\mathbf{2}, E, c)$ into a so-called *canonical form* $(V_\mathbf{1}, V_\mathbf{2}, E, c')$ with some appropriate cost function c' *equivalent* to c.

Definition 3.1 (Canonical Form) *An MPG $(V_\mathbf{1}, V_\mathbf{2}, E, c)$ is said to be in* canonical form *if there exists a function p from V to $\mathbb{R}$ such that for all v in V:*

1. *$ext(c, v) = p(v)$,*
2. *$p(u) = p(v)$ for all u in $V_{ext}(c, v)$,*
3. *if $v \in V_\mathbf{1}$ then $p(u) \leqslant p(v)$ for all u in $V(v)$,*
4. *if $v \in V_\mathbf{2}$ then $p(u) \geqslant p(v)$ for all u in $V(v)$.* □

Consider a play $(v_i)_{i\in\mathbb{N}}$ in such a game, and assume that player **1** always greedily chooses extremal edges. Then at each step i the following conditions hold:

1. if $v_i \in V_\mathbf{1}$ then $c(v_i, v_{i+1}) = p(v_i)$,
2. if $v_i \in V_\mathbf{2}$ then $c(v_i, v_{i+1}) \geqslant p(v_i)$.

By definition of the canonical form, the fact that player **1** chooses extremal edges guarantees that the value of p does not decrease along the path, therefore each cost along the path is at least $p(v_0)$. As a consequence, the payoff of this game is at least $p(v_0)$. By the same argument, if player **2** always greedily chooses extremal edges, then the payoff of the game is at most p. Therefore, in games in canonical form, any greedy strategy consisting in choosing extremal edges is optimal.

3.2 Original Version of the GKK

The algorithm was first published by Gurvich et al. (1988) and later reformulated and somewhat generalized by Pisaruk (1999) for a wider class of games. We first present the original algorithm and afterwards see how it adapts to reductions of parity games.

Throughout the algorithm's description, we will use the following notation. Given a subset X of V, we write $X_{\mathbf{1}} = X \cap V_{\mathbf{1}}$ and similarly $X_{\mathbf{2}} = X \cap V_{\mathbf{2}}$. At any time, the sets K, S, and T are defined as:

$$\begin{array}{ll} K(c) = \{v \mid ext(c,v) = p\} & \text{– } \textit{critical} \text{ vertices,} \\ S(c) = \{v \mid ext(c,v) < p\} & \text{– } \textit{insufficient} \text{ vertices,} \\ T(c) = \{v \mid ext(c,v) > p\} & \text{– } \textit{redundant} \text{ vertices,} \end{array}$$

where p is a given threshold value between $m = \min_{v\in V} ext(c,v)$ and $M = \max_{v\in V} ext(c,v)$. By *shifting* we call a modification of the cost function c by an amount δ with respect to a subset L of V, i.e., substituting c' for c where, for each edge (u,v) the cost $c'(u,v)$ is defined as:

$$c'(u,v) = \begin{cases} c(u,v) + \delta & \text{if } u \in L \text{ and } v \notin L, \\ c(u,v) - \delta & \text{if } u \notin L \text{ and } v \in L, \\ c(u,v) & \text{otherwise.} \end{cases}$$

Thus c' is equivalent to c because it corresponds a potential transformation $\delta \cdot \chi_L$, where χ_L denotes the indicator function of L in V.

In every iteration the set L above is defined as a set of vertices starting from which player **2** has a stationary strategy forcing player **1** into S, when both players use only extremal edges.

The algorithm keeps doing the above cost shifts with $\delta > 0$ while it is possible. When it stops, either 1) L is empty, or 2) $\bar{L}$ is empty, or 3) $(L, \bar{L})$ is a *regular* partition of the game. The latter means that player **1** cannot escape from L and player **2** cannot escape from $\bar{L}$, when their opponents follow extremal edges. Moreover, the player **2** guaranteed payoff in L is at most p and the player **1** guaranteed payoff in $\bar{L}$ is at least p. In cases 1) and 2), the new threshold value p is selected in the middle of $[m, p]$ or $[p, M]$, respectively, and the algorithm iterates. In case 3), one obtains two smaller games to which the algorithm applies recursively. The full details and correctness proofs are contained in the paper (Gurvich et al. 1988).

3.3 Reduced Version of the GKK

The GKK algorithm thus provides for computing the optimal payoff for each starting position in a mean-payoff game. We need less — to determine the winner for each starting position in a parity game, which only requires (after the reduction above) to determine the sign of the optimal payoff. Therefore, several simplifications may be made to the GKK algorithm.

Since we only need to find the sign of the optimal mean payoff starting from every position, using 0 as the threshold value p, *exactly one* execution of the algorithm is needed to get the desired result.

The simplified algorithm can be formulated the following way, provided that $(V_{\mathbf{1}}, V_{\mathbf{2}}, E, c)$ is an MPG reduced from a PG. It produces a potential function a such that any greedy strategy always choosing extremal edges is optimal.

for all $v \in V$ **do** $a(v) \leftarrow 0$
while $S(c_a) \neq \emptyset$ and $T(c_a) \neq \emptyset$ **do**
    $L \leftarrow S(c_a)$
    repeat
        $X \leftarrow \{\, v \mid v \in K(c_a)_{\mathbf{1}} \setminus L, V_{ext}(c_a, v) \subseteq L \,\}$
        $Y \leftarrow \{\, v \mid v \in K(c_a)_{\mathbf{2}} \setminus L, V_{ext}(c_a, v) \cap L \neq \emptyset \,\}$
        $L \leftarrow L \cup X \cup Y$
    until $X = Y = \emptyset$
    $E_1 \leftarrow \{\, (u, v) \mid (u, v) \in E, u \in L_{\mathbf{1}}, v \in \bar{L} \,\}$
    $E_2 \leftarrow \{\, (u, v) \mid (u, v) \in E, u \in \bar{L}_{\mathbf{2}}, v \in L \,\}$
    $\hat{S} \leftarrow \{\, v \mid v \in S(c_a)_{\mathbf{2}}, V_{ext}(c_a, v) \subseteq \bar{L} \,\}$
    $\hat{T} \leftarrow \{\, v \mid v \in T(c_a)_{\mathbf{1}}, V_{ext}(c_a, v) \subseteq L \,\}$
    if $E_1 = E_2 = \emptyset$ and $\hat{S} = \hat{T} = \emptyset$ **then return** a
    $\delta \leftarrow \min\Big(\big\{|c_a(e)| \mid e \in E_1 \cup E_2\big\} \cup \big\{|\, ext(c_a, v)| \mid v \in \hat{S} \cup \hat{T}\big\}\Big)$
    for all $v \in L$ **do** $a(v) \leftarrow a(v) + \delta$
end while
return a

Let us discuss the correctness of this algorithm. Two exit conditions represent different situations:

1. If $S(c_a)$ is empty then the extremal value of each vertex is non-negative, therefore all edges leaving vertices that belong to **2** have a non-negative cost. Consequently, if **1** always chooses extremal edges, every game has a positive payoff. In this case **1** is always the winner. If $T(c_a)$ is empty, then the situation is symmetric and **2** is always the winner.
2. If at some point the sets E_1, E_2, $\hat{S}$ and $\hat{T}$ are all empty, then starting in L, **2** can guarantee that the game never exits L, and that starting in $\bar{L}$ **1** can force the game to stay in $\bar{L}$. For the same reason as in the previous cases, if the game stays in L, then **2** is the winner, and if it stays in $\bar{L}$, then **1** is the winner. Therefore, this provides optimal strategies for both players.

So whenever the algorithm terminates, the obtained potential a provides the expected result. To demonstrate that the algorithm always terminates (Pisaruk 1999) introduced the function

Φ defined as

$$\Phi(a) = \sum_{v \in V} \min\Big(\|a\|_\infty - a(v),\ \big(|V|-1\big)\|c\|_\infty\Big),$$

where $\|c\|_\infty$ denotes the maximum absolute value of the function c. He demonstrated that the value of this function Φ is monotonically increasing, as the algorithms proceeds, and is bounded. This provides for a pseudopolynomial worst case bound on the running time. A pseudopolynomial worst-case bound on the number of iteration of the simplified algorithm for PGs can be obtained by considering the function Φ above and taking into account the fact that for PGs $\|c\|_\infty = w_{p_{\max}}$, where $p_{\max}$ is the maximal priority of a game.

However, extensive experiments on PGs performed in this work witnesses much better, low-degree polynomial-time behavior on the average. We hope to explain this behavior and get better upper bounds for the algorithm in the particular case of parity games.

3.4 Randomization of the Algorithm

Randomization is a well-known tool improving the worst-case behavior of many algorithms (Motwani & Raghavan 1995) by allowing them to 'foil' bad inputs. Classical example is randomized QuickSort. Also the simplex method is exponential on Klee-Minty cubes, but its naive randomization becomes quadratic polynomial on these cubes. There are several known versions of randomized subexponential simplex algorithm.

Other known algorithms for the μ-calculus model checking or parity games (Browne et al. 1997, Seidl 1996, Jurdzinski 2000, Vöge & Jurdzinski 2000) do not allow randomization, and for all of them[3] the examples forcing their worst-case exponential behavior.

The advantage of our adaptation is that it naturally allows for randomization by applying (random) potential transformations to the games before running the algorithm. This allows for studying the influence of the actual edge costs on several MPGs equivalent to the same parity game. By choosing the initial potential transformation randomly it is also possible to randomize the whole algorithm, which may allow it to foil bad input instances and avoid worst-case behavior (similar to randomized Quick Sort, making initial random permutations). Our experiments show that this randomization improves the behavior of the algorithm; see (Beffara & Vorobyov 2001) and Section 7 below, where we show how randomized GKKbeats hard known instances of games, forcing deterministic GKK make exponentially many iterations.

One of the random potential transformations we apply is defined as follows. Let $\mathcal{N}$ be an MPG obtained as the reduction of a PG with priority function p. Let (w_i) be the sequence of costs used in the reduction (as explained in Section 2.1). Given an integer k_0, we apply the transformation associated with the potential function a defined as:

$$\text{for all } v \in V, \qquad a(v) = \begin{cases} 0 & \text{if } p(v) < k_0 \\ -w_{p(v)} & \text{if } p(v) \geqslant k_0 \end{cases}$$

Clearly, this transformation may only decrease the norm of the cost function, i.e., does not influence the pseudopolynomial worst-case behavior of the algorithm. Our experiments

[3] Except maybe the last one, on which we do not have available data.

actually show that when k_0 is in the middle of the set of priorities, the average number of required iterations is slightly less, by approximately 20% (Beffara & Vorobyov 2001). However, the main purpose of such transformations (we are also using other ones) is to help the algorithm foil bad cases, if they happen.

4 Implementation

Fully documented C++ implementation of the GKK algorithm (adapted for PARITY GAMES) in the CWEB system of D. Knuth's "literate programming" occupies 18 pages and is presented in full in (Beffara & Vorobyov 2001). The procedural language was chosen (in favor of functional languages such as SML), because the algorithm is strongly imperative making it is much easier to represent and manipulate graphs. CWEB allowed for writing clear, reliable, well-documented programs in C++, easy to modify and extend, which was especially valuable during experimentation. The Standard Template Library STL (`http://www.sgi.com/tech/stl/`) was used extensively. Our primary goal was to measure the essential characteristics of the algorithm, like the average and maximal number of iterations of the algorithm (which never exceeded the number of vertices) in a series of experiments (and not, say, the CPU time). Therefore, some auxiliary graph algorithms are not implemented optimally. For example, instead of the linear time finite reachability game algorithm to compute the set L, we implemented a straightforward quadratic one. Also we have not yet implemented the unlimited precision integer arithmetic, which will be needed to experiment with unlimited numbers of colors. We were able, however to deal with games on up to 10.000 vertices with 7 colors and 20.000 with up to 6 colors. In an ongoing work we lift those limitations and hope to obtain more substantial experimental data.

The implementation intentionally contains a lot of redundancies for the algorithm's self-control, because we needed to be completely aware of results. For example, there are means to randomly transform the same game into several equivalent versions (unknown to the algorithm) and comparing results.

5 Experiments on Big Random Parity Games

We tested the algorithm on a considerable number of large problem instances (games)[4], we provide a form of random generation of games with different parameters (maximum outdegrees and number of colors). In this generation, we select uniformly at random:

1. the owner of each vertex (player making a move),
2. its outdegree (between one and a specified upper bound, or equal to a specific value),
3. its successors,
4. its priority/color (between zero and a specified upper bound).

Other natural methods, and distributions, could be easily incorporated, and will be investigated in further work. In this section we only generate games that are reductions of PGs. This means that all vertices have a given priority and that all edges leaving a given edge receive the *same* initial cost, depending on this priority. Different methods are possible for choosing these costs, and we implemented two of them. The first one due to (Puri 1995, Jurdziński 1998) uses too large costs and appeared to be impractical even for small games. Instead, we implemented the "smallest" weights method, as described in Section 2.1.

5.1 Number of Iterations

Varying color numbers and outdegrees. In (Beffara & Vorobyov 2001) we presented numerous surface plots made from the average number of iterations in randomly generated games of size 1000, with varying outdegrees (2, 3, 4, ..., 64) and numbers of priorities (2, 3, 4, 5, 6, 7), analyzing 100 randomly generated games in each case. It appears that the highest number of iterations occurs for small (expected values of) outdegrees, namely 4 or 5 (average around 2 – 3) depending on the number of priorities. The same happens to be true for games of larger sizes. Figures 1, 2, 3 represent analogous results, but for games of size 20000 with 6 colors and outdegrees 3, 4, 5, giving the range, average, and maximum number of iterations, and the distribution diagram. Similar experiments and diagrams are now available for games of size up to 20.000 and with 6-7 priorities[5]. All of them witness similar behavior of the algorithm.

Small outdegrees, for any sizes and number of colors represent more difficult cases for the GKK algorithm.

5.2 Random Potential Transformations

As discussed in Section 3.4, the randomized potential transformations do not increase the norm of the initial cost function and do not influence the worst-case behavior of the algorithm, but may help it to avoid/foil 'bad' inputs. Considering the ratio between the number of iterations of GKK required with such transformations and without them shows (Beffara & Vorobyov 2001) that in our experiments the behavior is rather significantly improved when choosing values of k_0 around the middle of the priorities interval.

[4]In our experiments, we manipulate about 10^7 games of size of order 10^4.

[5]See our web site `http://www.csd.uu.se/ vorobyov/eb-gkk/`

More importantly, randomization allows to beat, with a few iterations examples of games requiring exponentially many iterations of the deterministic GKK.

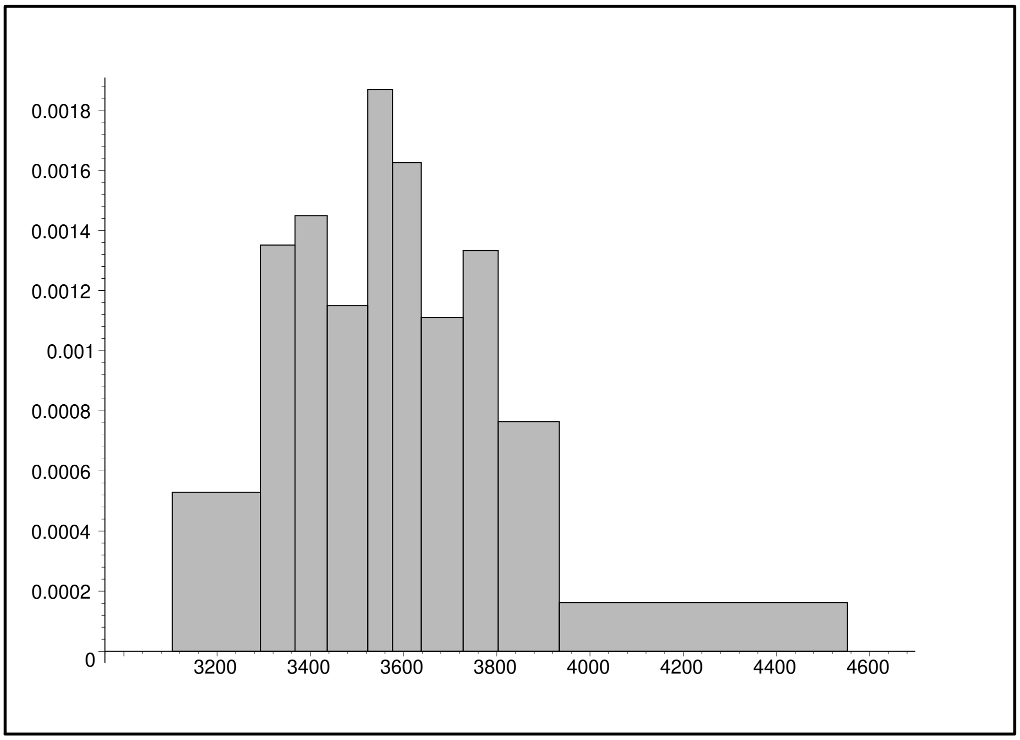

Figure 1: 20000 vertices, 6 colors, outdegree 3, 106 games, iteration range: 3104..4552, average number of iterations: 3605, standard deviation: 263.

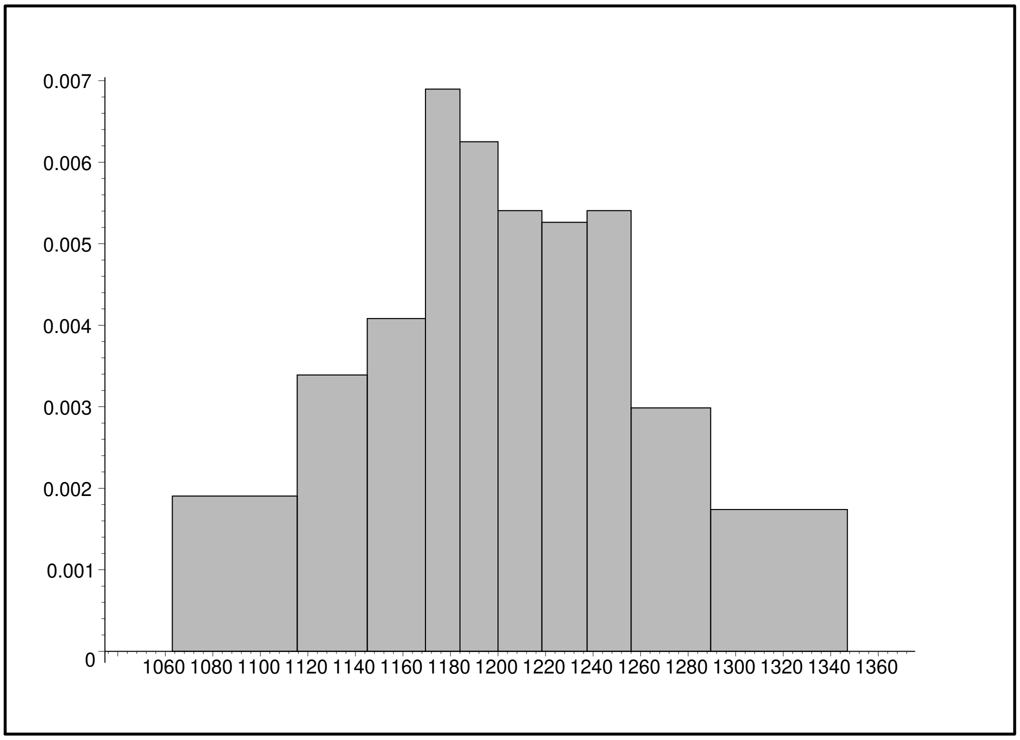

Figure 2: 20000 vertices, 6 colors, outdegree 4, 100 games, iteration range: 1063..1347, average number of iterations: 1203, standard deviation: 64.

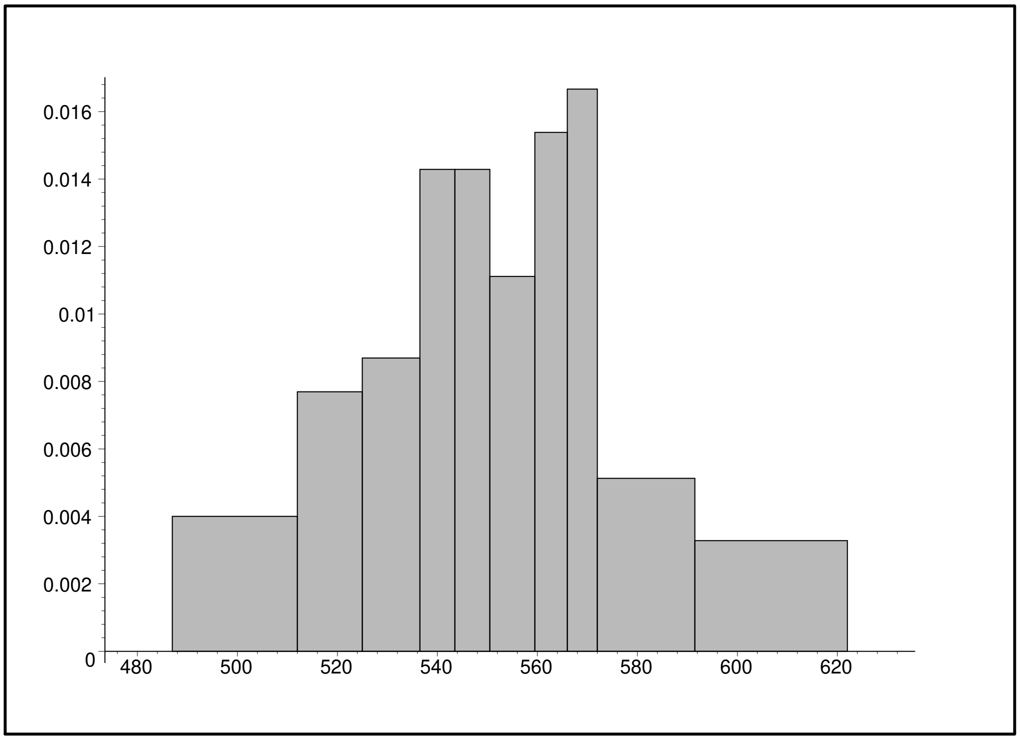

Figure 3: 20000 vertices, 6 colors, outdegree 5, 100 games, iteration range: 487..662, average number of iterations: 551, standard deviation: 29.

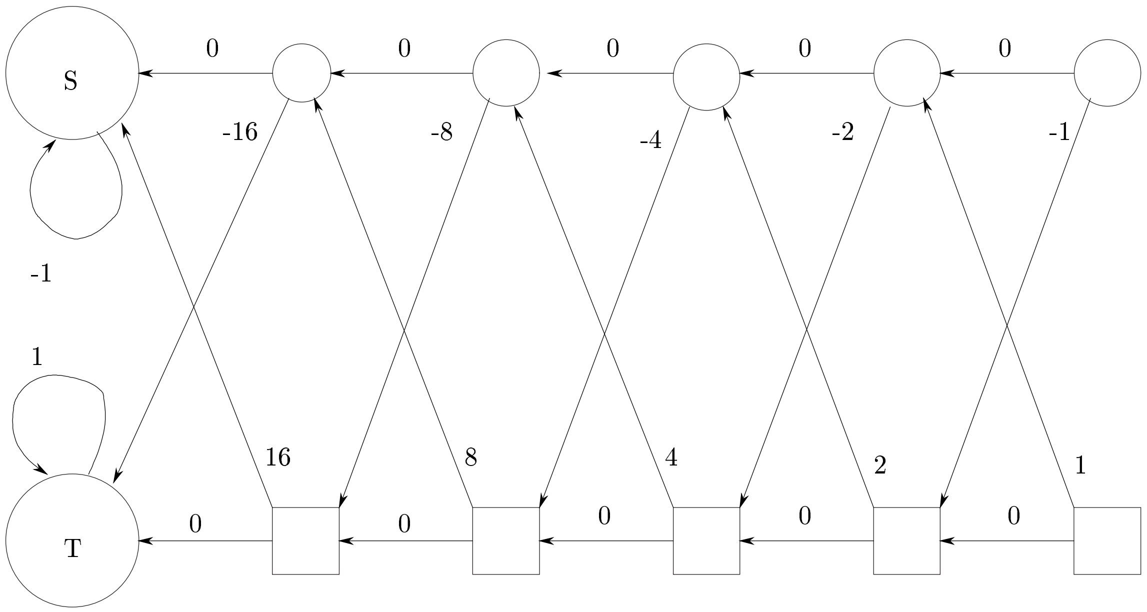

Figure 4: Lebedev-Gurvich's example, case $n = 6$. Forces deterministic GKK to make $2^{n-1} = 2^5$ iterations. Straightforwardly generalizes to any n.

6 Lebedev-Gurvich's 'Hard' Game Instances

Vladimir Gurvich has kindly presented us (Gurvich 2001) an example (possibly the same as mentioned by Gurvich et al. (1988, p. 90)) of the exponential behavior of the GKK algorithm. This example is shown in Figure 4 (for the case $n = 6$, and can be straightforwardly generalized for any n).

The reader is invited to verify and convince herself that the deterministic GKK algorithm, as described above, indeed behaves *exponentially*, making precisely 2^{n-1} iterations. We checked our implementation on this example (the whole family, for $n = 2, \ldots, 20$), and indeed it makes 2^{n-1} iterations, as predicted.

The remedy, allowing to avoid this exponentially bad behavior of the GKK algorithm lies in a simple randomization, as we describe below.

7 Random Initial Potential Transformations

A *potential* is an arbitrary function assigning an integer value $p(v)$ to every vertex v of the game graph. Given a potential function $p(.)$, a *potential transformation* of the game graph consists in changing the weight $c(uv)$ of each edge uv with the new one defined by

$$c'(uv) := c(uv) + p(u) - p(v). \tag{3}$$

Obviously, every potential transformation preserves the optimal values and strategies for both players.

The following idea of randomizing GKK turns out very efficient in practice.

1. Suppose the weights assigned to edges of the game graph are integers in the interval $I = [-W_1, \ldots, W_2]$.

2. Generate a random potential by letting, for each vertex v of the graph, the potential value $p(v)$ as a randomly and uniformly chosen integer from the interval I.

3. Perform the corresponding potential transformation on the game graph, and afterwards run GKK on the resulting graph.

We implemented this simple scheme and it turned out to be very efficient in practice as we describe below.

Different modifications of the above scheme are possible, including:

1. non-uniform selection of potentials, depending on the frequency of different weights,

2. selection of potentials that do not change the maximum weight assigned – this is crucial for non-deteriorating the pseudopolynomial worst case bound for GKK proved by (Pisaruk 1999).

Those will be investigated in future work.

8 Limiting the Number of Iterations

Another very simple yet productive idea, which happens to be work very robustly in practice, consists in giving up long iterations of GKK, and restarting the whole computation by performing an independent new random potential transformation to the initial game. It seems, although we do not currently have theoretical explanations, that with the high probability a random initial potential transformation is '*good*', i.e., leads to a short convergence of GKK. We present an extensive experimental data to support this conjecture.

The limit, after each randomized GKK gives up is expressed in terms of the maximal number of iterations during which non of the sets S, T is changed. We generously set this limit to 2^13 for Lebedev-Gurvich's example of size $n = 50$ and 2^14 for size $n = 60$. Our experiments show that setting this limit just to several hundreds would be enough; see below.

This idea is known and performs very well in other applications, like in Schöning's $O((4/3)^n)$ multistart Monte Carlo local search for 3SAT. Recall that this algorithm generates random truth assignments and performs linearly short local searches from each one, by flipping a bit corresponding to a randomly chosen literal in an unsatisfied clause. If it fails in a short run, it gives up and tries a new initial random assignment.

Although we do not yet know theoretically 'optimal' bounds on the limit of iterations, for Lebedev-Gurvich's examples of sizes $n = 50, ..., 60$ this limit is 300-400 hundreds iterations. See discussion of experimental data below and compare with $2^{49} = 562949953421312, \ldots,$ $2^{59} = 576460752303423488$ iterations of the non-randomized GKK.

9 Experiments with Randomized GKK on Lebedev-Gurvich's Games

We conducted extensive experiments with randomized GKK and describe here several representative cases. Namely, we experimented with Lebedev-Gurvich's examples of sizes $n = 50$ and $n = 60$.

1. For $n = 50$, in two series of 1000 trials each, the randomized GKK algorithm was set to give up after $2^{13} = 8K$ iterations (for comparison, the deterministic one should do $2^{49} = 512\text{Tera} \approx 5.63 \cdot 10^{14}$ iterations), In those two series 764 and 749 trials, respectively, were successful (others gave up).
2. For $n = 60$, in one series of 1000 trials, the randomized GKK algorithm was set to give up after $2^{14} = 16K$ iterations (for comparison, the non-randomized one should do $2^{59} = 2^{19}\text{Tera} \approx 5.76 \cdot 10^{17}$ iterations). In this series 407 trials were successful.

We combine relevant information about those experiments in Table 1 and Figures 6, 5 placed in Appendix B.

Size n of the example	50	50	60
Number of Trials	1000	1000	1000
Successful trials	764	749	407
Average number of iterations	2048	2039	3525
Maximum number of iterations	19250	18916	33023
Standard deviation	3100	3166	5182
Non-randomized iterations 2^{n-1}	562949953421312	562949953421312	576460752303423488
Percentage of successful trials with the number of iterations in the range:			
[0..100]	0%	0%	0%
[100..200]	22.64%	25.90%	9.09%
[200..300]	9.69%	8.68%	14.00%
[300..400]	7.07%	6.94%	6.14%
[400..500]	4.58%	5.07%	5.16%
[500..600]	2.75%	2.00%	2.70%
[600..700]	5.10%	4.80%	4.42%
[700..800]	2.88%	2.27%	4.18%
[800..900]	1.70%	1.47%	2.21%
[900..1000]	1.44%	0.93%	1.47%
[1000..10000]	38.61%	38.05%	40.54%
[10000..1000000]	3.53%	3.87%	10.07%
[1000000..5000000]	0%	0%	0%

Table 1: Summary of results: Randomized GKK on Hard Lebedev-Gurvich Instances of sizes 50 and 60

10 Conclusions

Many key theoretical advances on average-case analyses of the celebrated algorithms and problems such as simplex method, bin packing, Hamiltonian tour local search algorithms, were stimulated by experimental research. Theoretical results of Karmarkar (1984) about his polynomial interior-point algorithm for linear programming did not attract much attention in the practical optimization community until he produced experimental results suggesting that his algorithm was practical. We conducted a similar case study for the randomized version of Gurvich-Karzanov-Khachiyan's algorithm adapted for PARITY GAMES in this paper.

1. GKK is an efficient, robust, easily randomizable algorithm, readily adaptable to solving the PARITY GAMES or the MU-CALCULUS MODEL CHECKING problems.

2. Extensive experiments on large randomly generated instances of games, and also on provably difficult instances of games clearly illustrate that the algorithm converges rapidly, presumably after linearly many iterations, on the average.

3. We investigated the productive multistart idea, when GKK gives up after a small number of iterations, and restarts with the new random potential transformation. It seems that the proportion of 'good' initial potential transformations, leading to fast convergence of GKK on 'hard' Lebedev-Gurvich game instances is very high.

4. We invite everybody to submit their favorite 'hard' game instances for cracking by our randomized GKK and join us in proving that the randomized GKK is a polynomial Monte Carlo algorithm for the PARITY GAMES or the MU-CALCULUS MODEL CHECKING problems.

References

Beffara, E. & Vorobyov, S. (2001), Adapting Gurvich-Karzanov-Khachiyan's algorithm for parity games: Implementation and experimentation, Technical Report 2001-020, Department of Information Technology, Uppsala University. `http://www.it.uu.se/research/reports/`.

Björklund, H., Petersson, V. & Vorobyov, S. (2001), Experiments with iterative improvement algorithms on completely unimodal hypercubes, Technical Report 2001-017, Information Technology/Uppsala University. `http://www.its.uu.se/research/reports/`.

Browne, A., Clarke, E. M., Jha, S., Long, D. E. & Marrero, W. (1997), 'An improved algorithm for the evaluation of fixpoint expressions', *Theor. Comput. Sci.* **178**, 237–255. Preliminary version in CAV'94, LNCS'818.

Ehrenfeucht, A. & Mycielski, J. (1979), 'Positional strategies for mean payoff games', *International Joourn. of Game Theory* **8**, 109–113.

Emerson, E. A. (1990), Temporal and modal logic, *in* J. van Leeuwen, ed., 'Handbook of Computer Science', Vol. B, Elsevier, pp. 996–1072.

Emerson, E. A. (1997), Model checking and the Mu-calculus, *in* N. Immerman & P. G. Kolaitis, eds, 'DIMACS Series in Discrete Mathematics', Vol. 31, pp. 185–214.

Emerson, E. A., Jutla, C. & Sistla, A. P. (1993), On model-checking for fragments of μ-calculus, *in* 'Computer Aided Verification, Proc. 5th Int. Conference', Vol. 697, Lect. Notes Comput. Sci., pp. 385–396.

Gurvich, V. (2001), Private communication. October 2001.

Gurvich, V. A., Karzanov, A. V. & Khachiyan, L. G. (1988), '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.

Jurdziński, M. (1998), 'Deciding the winner in parity games is in UP∩co-UP', *Information Processing Letters* **68**, 119–124.

Jurdzinski, M. (2000), Small progress measures for solving parity games, *in* '17th STACS', Vol. 1770 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 290–301.

Karmarkar, N. K. (1984), 'A new polynomial-time algorithm for linear programming', *Combinatorica* **4**, 373–395.

Knuth, D. (1984), 'Literate programming', *The Computer Journal* **27**, 97–11.

Knuth, D. (1992), *Literate Programming*, Vol. 27 of *CSLI Lecture Notes*, CSLI, Center for the Study of Language and Information.

Motwani, R. & Raghavan, P. (1995), *Randomized Algorithms*, Cambridge Univ. Press.

Moulin, H. (1976), Prolongement des jeux à deux joueurs de somme nulle, *in* 'Bulletin de la Société Mathématique de France', SMF, mémoire 45.

Papadimitriou, C. (2001), Algorithms, games, and the internet, *in* 'ACM Annual Symposium on Theory of Computing', ACM, pp. 749–753.

Petersson, V. & Vorobyov, S. (2001*a*), Interior-point approach to parity games, *in* 'IEEE Annual Symposium on Logic in Computer Science (LICS'2001)', Boston, Massachusetts. Short communication.

Petersson, V. & Vorobyov, S. (2001*b*), 'A randomized subexponential algorithm for parity games', *Nordic Journal of Computing* . To appear. Preliminary version: Technical Report #1, 2001, Department of Information Technology, Uppsala Unitersity. Also: Proceedings of the 12th Nordic Workshop on Programming Theory.

Pisaruk, N. (1999), 'Mean cost cyclical games', *Mathematics of Operations Research* **24**(4), 817–828.

Puri, A. (1995), Theory of hybrid systems and discrete events systems, PhD thesis, EECS Univ. Berkeley.

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

Vöge, J. & Jurdzinski, M. (2000), A discrete strategy improvement algorithm for solving parity games, *in* 'CAV'00: Computer-Aided Verification', Vol. 1855 of *Lect. Notes Comput. Sci.*, Springer-Verlag, pp. 202–215.

Zwick, U. & Paterson, M. (1996), 'The complexity of mean payoff games on graphs', *Theor. Comput. Sci.* **158**, 343–359.

A APPENDIX: MAPLE SESSION for Randomized GKK on Lebedev-Gurvich Games

This is the Maple session analyzing data obtained from running Randomized Gurvich-Karzanov-Khachiyan algorithm on Lebedev-Gurvich's example, for n=50 and n=60. This example forces the nonrandomized GKKA to make 2ˆ49 and 2ˆ59 iterations respectively. Randomized GKK in most trials converges in a few hundreds iterations only, as show histograms in pages 4-6. To avoid repetitions we erase the user input, which is repeated by Maple in structured form after parsing.

$$M := 0$$

$$m := 0$$

$$p0 := [0..100,\ 100..200,\ 200..300,\ 300..400,\ 400..500,\ 500..600,\ 600..700,\ 700..800,\ 800..900,\ 900.1000]$$

$$p2 := [0..1000,\ 1000..10000,\ 10000..1000000]$$

$$p3 := [0..100,\ 100..200,\ 200..300,\ 300..400,\ 400..500,\ 500..600,\ 600..700,\ 700..800,\ 800..900,\ 900..1000,\ 1000..10000,\ 10000..1000000,\ 1000000..5000000]$$

```
RD := proc(file)
  local data, i, j, L, c, d;
  global MX, mx, M, m;
    MX := NULL;
    mx := NULL;
    m := 0;
    M := 0;
    data := readdata(file, integer, 2);
    for i to nops(data) do
      L := data[i];
      MX := MX, L[1];
      mx := mx, L[2];
      if M < L[1] then M := L[1] fi;
      if m < L[2] then m := L[2] fi
    od;
    NULL
  end
```

$an01 := \mathbf{proc}(\mathit{file},\, p,\, desc,\, n,\, dsc)$
$\quad\mathbf{local}\, h,\, histdata,\, data,\, hh,\, h0,\, h1,\, h2,\, h3;$
$\quad\mathbf{global}\, MX,\, mx,\, M,\, m;$
$\qquad \mathrm{RD}(\mathit{file});$
$\qquad \mathrm{with}(stats);$
$\qquad \mathrm{with}(describe);$
$\qquad \mathrm{with}(statplots);$
$\qquad \mathrm{with}(plots);$
$\qquad h0 := transform_{tallyinto}([MX],\, p);$
$\qquad h := transform_{statsort}(h0);$
$\qquad h1 := transform_{scaleweight_{100.0\times 1/describe_{count}(h)}}(h);$
$\qquad h2 := transform_{apply_{x\to 1/100\times x}}(h1);$
$\qquad h3 := transform_{split_2}(h2);$
$\qquad$ printf("********** NEW SET OF EXPERIMENTAL DA\
TA *******\n%s. File: %s. Success (of 1000): %d. Averag\
e: %d. Max: %d. Stand Dev: %d Non-randomized iteration\
s: %d (for comparison).\n* Tallied in intervals %a \n* In pe\
rcent: %a\n* Scaled: %a\n* First 50%% %a\n* Remainin\
g 50%% %a\n\ n*** %s", $desc$, file, count([MX]),
ceil(mean([MX])), M, ceil(standarddeviation([MX])), $2^{(n-1)}$, h,
$h1$, $h2$, $h3_1$, $h3_2$, dsc);
$\qquad hh := \mathrm{histogram}(h3_1);$
$\qquad \mathrm{display}(hh)$
$\quad\mathbf{end}$

```
> an01('ge50x1000a', p3, '50x1000 with abort', 50, 'First 50% histogram.
  X-scale: x100 iterations');

 ********** NEW SET OF EXPERIMENTAL DATA *******

 50x1000 with abort. File: ge50x1000a. Success (of 1000): 764. Average:
 2084. Max: 19250. Stand Dev: 3100 Non-randomized iterations:
 562949953421312 (for comparison).

 * Tallied in intervals [Weight(0 .. 100,0), Weight(100 .. 200,173),
 Weight(200 .. 300,74), Weight(300 .. 400,54), Weight(400 .. 500,35),
 Weight(500 .. 600,21), Weight(600 .. 700,39), Weight(700 .. 800,22),
 Weight(800 .. 900,13), Weight(900 .. 1000,11), Weight(1000 ..
 10000,295), Weight(10000 .. 1000000,27), Weight(1000000 .. 5000000,0)]


 * In percent: [Weight(0 .. 100,0), Weight(100 .. 200,22.64397907),
 Weight(200 .. 300,9.685863878), Weight(300 .. 400,7.068062830),
 Weight(400 .. 500,4.581151834), Weight(500 .. 600,2.748691100),
 Weight(600 .. 700,5.104712044), Weight(700 .. 800,2.879581153),
```

```
Weight(800 .. 900,1.701570681), Weight(900 .. 1000,1.439790576),
Weight(1000 .. 10000,38.61256546), Weight(10000 ..
1000000,3.534031415), Weight(1000000 .. 5000000,0)]

* Scaled: [Weight(0 .. 1,0), Weight(1 .. 2,22.64397907), Weight(2 ..
3,9.685863878), Weight(3 .. 4,7.068062830), Weight(4 ..
5,4.581151834), Weight(5 .. 6,2.748691100), Weight(6 ..
7,5.104712044), Weight(7 .. 8,2.879581153), Weight(8 ..
9,1.701570681), Weight(9 .. 10,1.439790576), Weight(10 ..
100,38.61256546), Weight(100 .. 10000,3.534031415), Weight(10000 ..
50000,0)]

* First 50% [Weight(0 .. 1,0), Weight(1 .. 2,22.64397907), Weight(2 ..
3,9.685863878), Weight(3 .. 4,7.068062830), Weight(4 ..
5,4.581151834), Weight(5 .. 6,2.748691100), Weight(6 .. 7,3.27225129)]

* Remaining 50% [Weight(6 .. 7,1.83246075), Weight(7 ..
8,2.879581153), Weight(8 .. 9,1.701570681), Weight(9 ..
10,1.439790576), Weight(10 .. 100,38.61256546), Weight(100 ..
10000,3.53403138)]


*** First 50% histogram. X-scale: x100 iterations
```

See Figure 6

```
> an01(‘ge50x1000b‘, p3, ‘50x1000 with abort‘, 50, ‘First 50% histogram.
  X-scale: x100 iterations‘);

********** NEW SET OF EXPERIMENTAL DATA *******

50x1000 with abort. File: ge50x1000b. Success (of 1000): 749. Average:
2039. Max: 18916. Stand Dev: 3166 Non-randomized iterations:
562949953421312 (for comparison).

* Tallied in intervals [Weight(0 .. 100,0), Weight(100 .. 200,194),
Weight(200 .. 300,65), Weight(300 .. 400,52), Weight(400 .. 500,38),
Weight(500 .. 600,15), Weight(600 .. 700,36), Weight(700 .. 800,17),
Weight(800 .. 900,11), Weight(900 .. 1000,7), Weight(1000 ..
10000,285), Weight(10000 .. 1000000,29), Weight(1000000 .. 5000000,0)]


* In percent: [Weight(0 .. 100,0), Weight(100 .. 200,25.90120161),
Weight(200 .. 300,8.678237653), Weight(300 .. 400,6.942590122),
Weight(400 .. 500,5.073431243), Weight(500 .. 600,2.002670228),
Weight(600 .. 700,4.806408546), Weight(700 .. 800,2.269692925),
Weight(800 .. 900,1.468624834), Weight(900 .. 1000,.9345794395),
Weight(1000 .. 10000,38.05073432), Weight(10000 ..
1000000,3.871829107), Weight(1000000 .. 5000000,0)]

* Scaled: [Weight(0 .. 1,0), Weight(1 .. 2,25.90120161), Weight(2 ..
3,8.678237653), Weight(3 .. 4,6.942590122), Weight(4 ..
5,5.073431243), Weight(5 .. 6,2.002670228), Weight(6 ..
7,4.806408546), Weight(7 .. 8,2.269692925), Weight(8 ..
9,1.468624834), Weight(9 .. 10,.9345794395), Weight(10 ..
100,38.05073432), Weight(100 .. 10000,3.871829107), Weight(10000 ..
50000,0)]
```

```
* First 50% [Weight(0 .. 1,0), Weight(1 .. 2,25.90120161), Weight(2 ..
3,8.678237653), Weight(3 .. 4,6.942590122), Weight(4 ..
5,5.073431243), Weight(5 .. 6,2.002670228), Weight(6 .. 7,1.40186915)]

* Remaining 50% [Weight(6 .. 7,3.40453940), Weight(7 ..
8,2.269692925), Weight(8 .. 9,1.468624834), Weight(9 ..
10,.9345794395), Weight(10 .. 100,38.05073432), Weight(100 ..
10000,3.87182908)]


*** First 50% histogram. X-scale: x100 iterations
```

See Figure 6

```
> an01(‘ge60x1000b‘, p3, ‘60x1000 with abort‘, 60, ‘First 50% histogram.
  X-scale: x100 iterations‘);

********** NEW SET OF EXPERIMENTAL DATA *******

60x1000 with abort. File: ge60x1000b. Success (of 1000): 407. Average:
3525. Max: 33023. Stand Dev: 5182 Non-randomized iterations:
576460752303423488 (for comparison).

* Tallied in intervals [Weight(0 .. 100,0), Weight(100 .. 200,37),
Weight(200 .. 300,57), Weight(300 .. 400,25), Weight(400 .. 500,21),
Weight(500 .. 600,11), Weight(600 .. 700,18), Weight(700 .. 800,17),
Weight(800 .. 900,9), Weight(900 .. 1000,6), Weight(1000 ..
10000,165), Weight(10000 .. 1000000,41), Weight(1000000 .. 5000000,0)]


* In percent: [Weight(0 .. 100,0), Weight(100 .. 200,9.090909091),
Weight(200 .. 300,14.00491400), Weight(300 .. 400,6.142506143),
Weight(400 .. 500,5.159705160), Weight(500 .. 600,2.702702703),
Weight(600 .. 700,4.422604423), Weight(700 .. 800,4.176904177),
Weight(800 .. 900,2.211302211), Weight(900 .. 1000,1.474201474),
Weight(1000 .. 10000,40.54054054), Weight(10000 ..
1000000,10.07371007), Weight(1000000 .. 5000000,0)]

* Scaled: [Weight(0 .. 1,0), Weight(1 .. 2,9.090909091), Weight(2 ..
3,14.00491400), Weight(3 .. 4,6.142506143), Weight(4 ..
5,5.159705160), Weight(5 .. 6,2.702702703), Weight(6 ..
7,4.422604423), Weight(7 .. 8,4.176904177), Weight(8 ..
9,2.211302211), Weight(9 .. 10,1.474201474), Weight(10 ..
100,40.54054054), Weight(100 .. 10000,10.07371007), Weight(10000 ..
50000,0)]

* First 50% [Weight(0 .. 1,0), Weight(1 .. 2,9.090909091), Weight(2 ..
3,14.00491400), Weight(3 .. 4,6.142506143), Weight(4 ..
5,5.159705160), Weight(5 .. 6,2.702702703), Weight(6 ..
7,4.422604423), Weight(7 .. 8,4.176904177), Weight(8 ..
9,2.211302211), Weight(9 .. 10,1.474201474), Weight(10 ..
100,.61425062)]
```

```
* Remaining 50% [Weight(10 .. 100,39.92628992), Weight(100 ..
10000,10.07371007)]
```

```
*** First 50% histogram. X-scale: x100 iterations
```

See Figure 5

B Histograms For Randomized GKK on Lebedev-Gurvich's Hard Game Instances

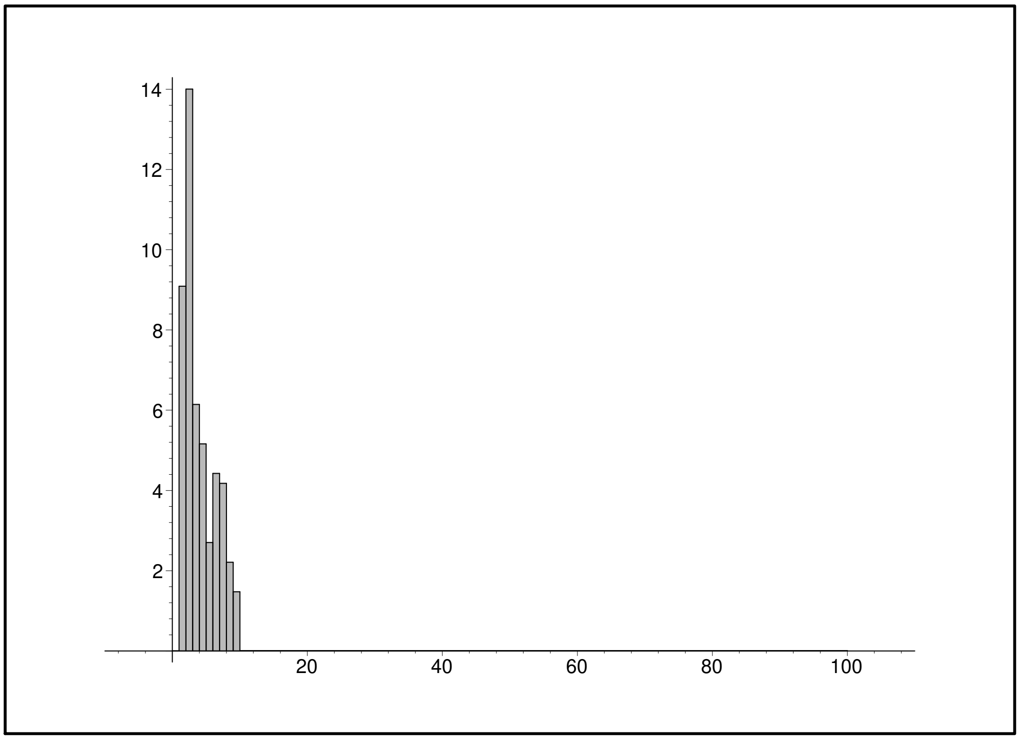

Figure 5: Histogram of Randomized GKK runs on 1000 trials with Lebedev-Gurvich's example of size $n = 60$. Shows distribution of 50% fastest trials. Horizontal axis measures in hundreds of iterations, vertical axis gives percentages. Note that almost all of them make less than 1000 iterations, and 50% of these 50% make less than 350 iterations.

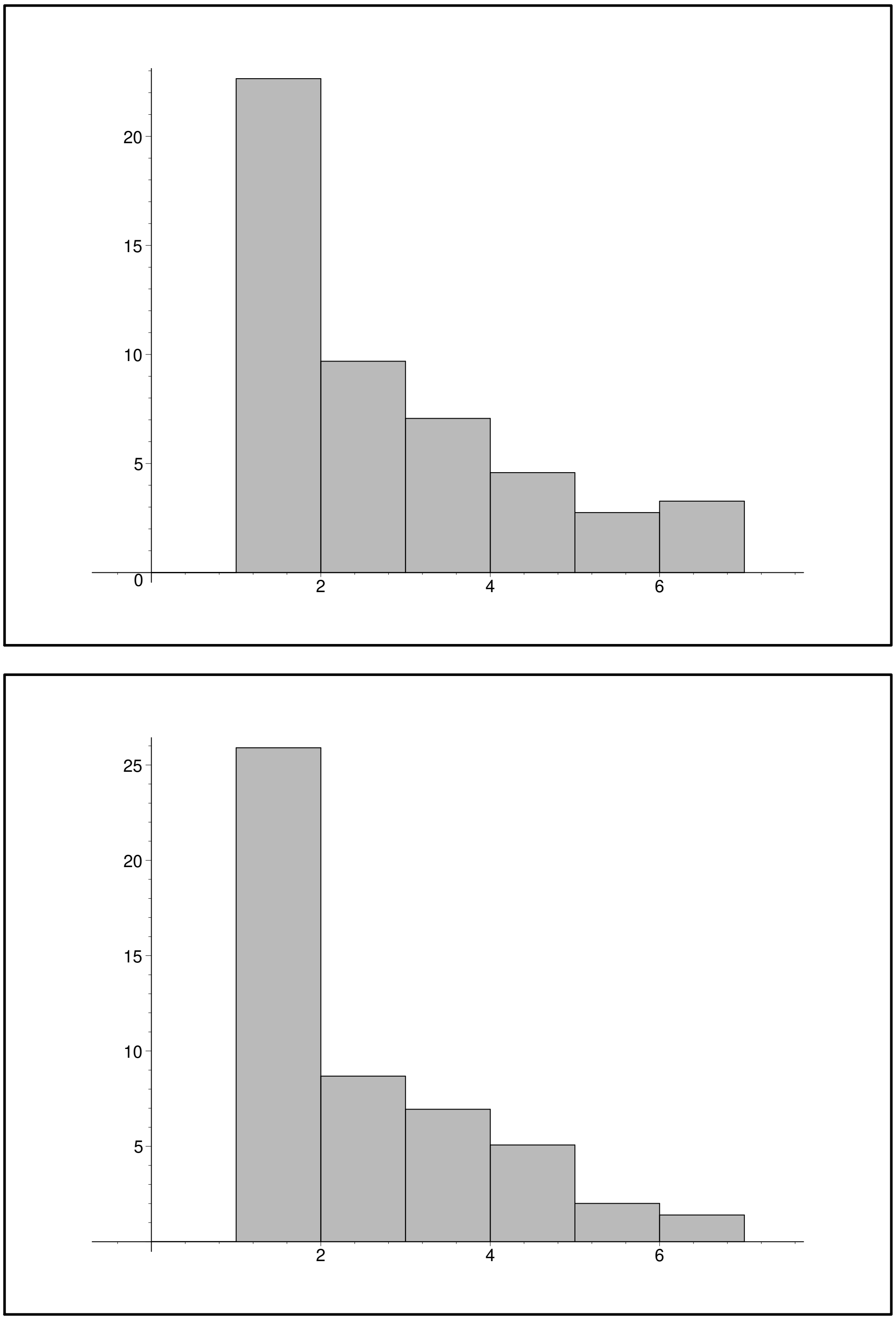

Figure 6: Histograms of Randomized GKK runs on two sets of 1000 trials with Lebedev-Gurvich's example of size $n = 50$. Shows distribution of 50% fastest trials. Horizontal axis measures in hundreds of iterations, vertical axis gives percentages. Note that 50% of these 50% make less than 300 iterations.

Recent technical reports from the Department of Information Technology

2001-001	Viktor Petersson and Sergei Vorobyov: *A Randomized Subexponential Algorithm for Parity Games*
2001-002	Johan Steensland: *Dynamic Structured Grid Hierarchy Partitioners Using Inverse Space-Filling Curves*
2001-003	Karl Hörnell and Per Lötstedt: *Adaptive Iteration to Steady State of Flow Problems*
2001-004	Bernhard Müller and H.C. Yee: *High Order Numerical Simulation of Sound Generated by the Kirchhoff Vortex*
2001-005	Yordanos G. Beyene: *Firewalls in Linux: Principles and Implementation*
2001-006	Erik K. Larsson and Torsten Söderström: *Identification of Continuous-Time AR Processes by Using Limiting Properties of Sampled Systems*
2001-007	Sven Montan: *Validation of Cycle-Accurate CPU Simulators against Real Hardware*
2001-008	Viktor Petersson and Sergei Vorobyov: *Parity Games: Interior-Point Approach*
2001-009	Johan Bengtsson: *Reducing Memory Usage in Symbolic State-Space Exploration for Timed Systems*
2001-010	Thiemo Voigt and Per Gunningberg: *Dealing with Memory-Intensive Web Requests*
2001-011	Torsten Söderström, Umberto Soverini and Kaushik Mahata: *Perspectives on errors-in-variables estimation for dynamic systems*
2001-012	Friedhelm Stappert, Andreas Ermedahl and Jakob Engblom: *Efficient Longest Executable Path Search for Programs with Complex Flows and Pipeline Effects*
2001-013	Eva Olsson, Lena Kecklund, Michael Ingre and Anders Jansson: *Lokförarens informationsmiljö och ATC. Ett användarperspektiv*
2001-014	Paul Pettersson and Sergio Yovine: *Workshop on Real-Time Tools (Proceedings)*
2001-015	Magnus Svärd: *On coordinate transformations for summation-by-parts operators*
2001-016	Robert Stjernström: *User-Centred Design of a Train Driver Display*
2001-017	Henrik Björklund, Viktor Petersson and Sergei Vorobyov: *Experiments with Iterative Improvement Algorithms on Completely Unimodal Hypercubes*
2001-018	Emad Abd-Elrady: *An adaptive grid point RPEM algorithm for harmonic signal modeling*
2001-019	Wendy Kress and Jonas Nilsson: *Boundary conditions and estimates for the linearized Navier-Stokes equations on staggered grids*
2001-020	Emmanuel Beffara and Sergei Vorobyov: *Adapting Gurvich-Karzanov-Khachiyan's Algorithm for Parity Games: Implementation and Experimentation*
2001-021	Inger Boivie: *Usability and Design Decisions in Software Development*
2001-022	Pierre Flener, Alan Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, Justin Pearson and Toby Walsh: *Symmetry in Matrix Models*
2001-023	Pierre Flener, Alan Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel and Toby Walsh: *Matrix Modelling*
2001-024	Larisa Beilina, Klas Samuelsson and Krister Åhlander: *A hybrid method for the wave equation*

UPPSALA UNIVERSITY

October 2001
ISSN 1404-3203
http://www.it.uu.se/