Complexity of Model Checking by Iterative Improvement: the Pseudo-Boolean Framework ⋆ ⋆⋆

Henrik Björklund, Sven Sandberg, and Sergei Vorobyov

Information Technology Department, Box 337, Uppsala University,
751 05 Uppsala, Sweden

{henrikbj,svens}@it.uu.se, vorobyov@csd.uu.se

Abstract. We present several new algorithms as well as new lower and upper bounds for optimizing functions underlying infinite games pertinent to computer-aided verification.

1 Introduction

Infinite two-person adversary full information games are a well established framework for modeling interaction between a system and its environment. A correct system can be naturally interpreted as a player possessing a winning strategy against any strategy of the malicious environment. Respectively, a verification process can be considered as a proof that a system does possess such a strategy. If the system loses, then a winning strategy for the environment encompasses possible system improvements. During the last decades a substantial progress has been achieved both on fitting diverse approaches to computer-aided verification into the game-theoretic paradigm and, simultaneously, on developing efficient algorithms for solving games, i.e., determining the winner and its strategy [12, 8, 15, 19, 28, 5]. A naturally appealing approach to solving games [18, 10, 28] consists in taking an initial strategy of the system and gradually 'improving' it, since a non-optimal strategy is outperformed by some counterstrategy of the environment. This allows for improving either the strategy, or the system, or both. In this paper we address the complexity of such an approach in an abstract model based on pseudo-Boolean functions possessing essential properties of games.

Game theory suggests, for numerous types of games, a nice characterization of the optimal solutions for behaviors of rational players, the so-called *Nash equilibria*. The 'only' remaining problem left widely open is: '*What is the exact computational complexity of finding Nash equilibria (optimal strategies) in two-person games with finitely many strategies? Could such games be solved in polynomial time?*' This is a fundamental problem, the solution to which determines whether or not, and to what extent game theory will be applicable to solving and optimizing real large-scale games emerging from practical verification of complex reactive systems. The problem is known to belong to the complexity class NP∩coNP (therefore, most probably not NP-complete, luckily!) but is *not known to be polynomial time solvable*. The best currently known algorithms are *exponential* (sometimes *subexponential*). Clearly, superpolynomial algorithms will not allow for practical solutions of the real-life verification games, no matter how fast the progress in hardware continues. Consequently, this problem is one of the most practical and (together with the NP versus P) most fundamental problems in the foundations of computing, complexity theory, and automata theory [24].

In this paper we address the efficiency of iterative improvement algorithms for solving games in the following abstract setting. Consider a game where one of the players has n binary choices, e.g., selects whether to move left or right[1] in n places, and every combination of choices is given a

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

⋆⋆ Extended abstract in PSI'03, LNCS 2890, © Springer-Verlag

[1] The restriction to two successors is no loss of generality, since a game with non-binary choices may be reduced to binary. However, complexity gets worse, and this issue is dealt with in Section 9.

cost, reflecting how good this strategy (combination of choices) is against a perfect adversary. This results in an n-dimensional valued Boolean hypercube, or a *pseudo-Boolean function*. Consequently, everything boils down to optimizing a pseudo-Boolean function. How fast can we optimize such a function? Despite its simplicity, the question appears extremely difficult for the classes of functions with special properties pertinent to games. Not surprisingly, there are not so many strong and general results of this kind in the literature. The best bound in Tovey's survey [27] is $O(2^{n/2})$; Mansour and Singh [22] suggest an $O(2^{0.78n})$ algorithm (for Markov decision processes), and we independently obtain an improved bound $O(2^{0.46n})$ for a similar algorithm in a more general setting [4]. Ludwig [21] suggested a subexponential $O(2^{\sqrt{n}})$ algorithm for binary simple stochastic games, which becomes exponential for games of unbounded (non-binary) outdegree. We suggested several [5, 6] subexponential algorithms for games of arbitrary outdegrees.

The development of our theory is primarily motivated by the so-called *parity games*, which are infinite games played on finite directed bipartite leafless graphs, with vertices colored by integers. Two players alternate moving a pebble along edges. The goal of Player 0 is to ensure that the biggest color visited by the pebble infinitely often is even, whereas Player 1 tries to make it odd. The complexity of determining the winner in parity games, equivalent to the Rabin chain tree automata non-emptiness, as well as to the μ-calculus[2] model checking [13, 12], is a fundamental open problem in complexity theory [24]. The problem belongs to NP$\cap$coNP, but its PTIME-membership status remains widely open. *Discounted mean payoff* [31] and *simple stochastic games* [9, 10], relevant for verification of probabilistic systems, are two other classes to which our theory applies.

We exploit the fundamental fact, apparently unnoticed before, that functions arising from such games possess extremely favorable structural properties, close to the so-called *completely unimodal functions* [16, 30, 29, 27]. These functions are well known in the field of pseudo-Boolean optimization [7, 17], an established area of combinatorial optimization, in which local search [1, 27] is one of the dominating methods. The complexity of local search for different classes of functions were carefully investigated [27, 16, 30]. However, all previously known algorithms for completely unimodal functions were *exponential* [27, 7]. A fruitful idea came from linear programming. In the early 90's Kalai [20] and Matoušek, Sharir, Welzl [26, 23] came up with strongly subexponential randomized algorithms for linear programming. Later Ludwig [21] adapted it to binary simple stochastic games, and we to binary parity games [25]. We also figured out that Kalai-Sharir-Welzl's randomized schemes fit perfectly well for completely unimodal optimization [2, 4, 3] and parity games [25]. Later we succeeded to generalize and adapt those subexponential schemes to create a discrete subexponential algorithm for general (non-binary) parity games [5], and extend it to combinatorial structures (which we call *completely local-global*) more directly reflecting games [6].

In this paper we present new upper bounds on the number of iterations of various iterative improvement algorithms for optimization of *completely unimodal pseudo-boolean functions* (CUPBFs). Such functions possess several remarkable properties: 1) unique minimum and maximum on every subcube, 2) every vertex has a unique neighborhood improvement signature. The problem of optimizing CUPBFs has been studied before [16, 30], but only few and weak bounds are known. Only a very restrictive subclass of decomposable CUPBFs is known to allow for polynomial time randomized optimization. The best known upper bounds are exponential, but it is conjectured that any CUPBF can be optimized in polynomial time [30].

We investigated and compared, both theoretically and practically[3], five algorithms for CUPBF optimization: the Greedy Single Switch Algorithm (GSSA), the Random Single Switch Algorithm (RSSA), the All Profitable Switches Algorithm (APSA), the Random Multiple Switches Algorithm (RMSA), and Kalai-Ludwig's Randomized Algorithm (KLRA). The GSSA and the RSSA have been studied by others in the context of CUPBF optimization. We first proposed the use of the APSA and RMSA in this context in [2]. It turns out that complete unimodality allows, knowing all improving neighbors of a vertex, to improve by simultaneously jumping towards all or some of them. (Actually

[2] One of the most expressive temporal logics of programs [12].

[3] Our results on practical evaluation and comparison of the algorithms can be found in [2, 4, 3]

any 0-1 linear combination of improving directions gives an improvement; this allows for non-local search algorithms that behave surprisingly well in practice.) The only nontrivial lower bound that has been shown is an exponential one for the GSSA. Local search iterative improvement algorithms are appealing, easy to implement, and efficient in practice. But no nontrivial upper bounds are known for those algorithms, except for subclasses of all CUPBFs. We settle several such nontrivial bounds in this paper. The fifth algorithm, KLRA, was first proposed in [21] for solving simple stochastic games, with a subexponential upper bound on the expected running time. We show that it can be modified to solve the CUPBF optimization problem, still in expected subexponential time.

Outline of the paper. In Section 2 we recall the definitions and main results concerning completely unimodal functions. Section 3 describes five iterative improvement algorithms for completely unimodal functions. Section 4 is devoted to the random multiple switches algorithm, whereas Section 5 covers single greedy single random, and all profitable switches algorithms. Section 6 adds random sampling to the random multiple switches algorithm, while Section 7 does the same to all other algorithms. Section 8 describes the Kalai-Ludwig-style algorithm. Section 9 generalizes the previous results to completely local-global functions and presents the Sharir-Welzl-style algorithm for these functions.

2 Completely Unimodal Pseudo-Boolean Functions

Parity and simple stochastic games can be solved by maximizing appropriately defined functions on neighborhood graphs representing sets of strategies. The prototypical case of such neighborhood graphs is the Boolean hypercube, and the essential structure of games pertinent to optimization by iterative improvement is captured, in the first approximation, by completely unimodal functions on Boolean cubes [16, 29, 30, 27]. Much of the theory we present can be understood and developed in terms of completely unimodal functions, and until Section 9 we concentrate on this case.

In general, parity and simple stochastic games require less restrictive neighborhood structures and functions. In [6] we succeeded to characterize them as a class we called *completely local-global* (CLG) functions, defined on Cartesian products of arbitrary finite sets, rather than on Boolean hypercubes. These functions, considered in Section 9, were crucial in our development [5, 6] of subexponential algorithms for parity games with arbitrary outdegree. This is because reducing such games to binary ones quadratically increases the number of vertices. As a consequence, a straightforward reduction renders subexponential algorithms exponential [5].

Let $H(n)$ denote an *n-dimensional Boolean hypercube* $\{0,1\}^n$, for $n \in \mathbb{N}^+$. A *pseudo-Boolean function* is an injective[4] mapping $H(n) \to \mathbb{N}$ associating a natural number to every n-dimensional Boolean vector. For $0 \leq k \leq n$, a *k-dimensional face* of $H(n)$, or a *k-face*, is a collection of Boolean vectors obtained by fixing $n-k$ arbitrary coordinates and letting the k remaining coordinates take all possible Boolean values. Faces of dimension 0 are called *vertices*, faces of dimension 1 are called *edges*. Faces of dimension $n-1$ are called *facets*. Two vertices that share an edge are called *neighbors*. Each vertex v in $H(n)$ has exactly n neighbors, forming the *standard neighborhood* of v on $H(n)$.

Complete Unimodality. A pseudo-Boolean function f is called *completely unimodal* (CUPBF for short) if one of the following four equivalent conditions holds [16, 30]:

1. f has a unique local minimum on each face,
2. f has a unique local maximum on each face,
3. f has a unique local minimum on each 2-face,
4. f has a unique local maximum on each 2-face.

[4] The standard injectiveness restriction is lifted in Section 9.

Improving directions. Let $f : H(n) \to \mathbb{R}$ be a CUPBF and let v be a vertex on $H(n)$. Number the dimensions of $H(n)$ from 0 to $n-1$. For each $i \in \{0, 1, ..., n-1\}$, let v_i be the neighbor of v that is reached by moving in coordinate i from v. Let p_i be 1 if $f(v) < f(v_i)$, otherwise 0. Then we call $\overline{p} = [p_0, p_1, ..., p_{n-1}]$ the *vector of improving directions* (VID) of v under f.

In the sequel we will usually abuse terminology by identifying a function and a (valued) hypercube it is defined upon. By 'optimizing' we mean '*maximizing*'.

3 Five Iterative Improvement Algorithms

A *standard local-search improvement algorithm* starts in an arbitrary point v_0 of the hypercube $H(n)$ and iteratively improves by selecting a next iterate with a better value from a *polynomial* neighborhood $N(v_i)$ of the current iterate.

Specific instances of the standard algorithm are obtained when one fixes:

1. the neighborhood structure on $H(n)$,
2. the disciplines of selecting the initial point and the next iterate.

Two major local-search improvement algorithms, the *Greedy Single Switch Algorithm (GSSA)* and the *Random Single Switch Algorithm (RSSA)* have previously been investigated and used for optimizing CUPBFs [27]. We also investigate the *All Profitable Switches Algorithm (APSA)* and the *Random Multiple Switches Algorithm (RMSA)*. Strictly speaking, neither APSA, nor RMSA is a local-search algorithm. The first one operates with neighborhood structures which vary depending on the CUPBF being optimized. The second chooses the next iterate from a non-polynomially bounded (in general) neighborhood of the current iterate. Finally, we show how the *subexponential Kalai-Ludwig's Randomized Algorithm (KLRA)* for solving binary simple stochastic games can be modified to optimize CUPBFs, and show that it is subexponential for this problem as well, with expected worst case behavior is $2^{O(\sqrt{n})}$. This algorithm has so far been ignored in the field of CUPBF optimization.

Greedy Single Switch Algorithm (GSSA) This is a local-search algorithm that at every iteration chooses the *highest-valued neighbor* of the current vertex as the next iterate. Recall that every vertex of $H(n)$ has exactly n neighbors (in the standard neighborhood). Unfortunately, this algorithm may take *exponentially many* steps to find the maximum of a CUPBF [29].

Random Single Switch Algorithm (RSSA) This is a local-search algorithm that at every iteration chooses uniformly at random one of the *higher-valued neighbors* of the current vertex as the next iterate. This algorithm may also take *exponentially many* iterations to find the global maximum. Although its expected running time for general CUPBFs is unknown, Williamson Hoke [30] has shown, using a proof technique due to Kelly, that the RSSA has expected quadratic running time on any *decomposable* hypercube. Call a facet F an *absorbing facet* if no vertex on F has a higher-valued neighbor that is not on F. A hypercube is called *decomposable* iff it has dimension 1 or has an absorbing facet that is decomposable. This result, together with the fact that in a CUPBF there is a short improving path from any vertex to the maximum (i.e., Hirsch conjecture holds), form grounds for the polynomial time optimization conjecture for CUPBFs [30] (currently open).

All Profitable Switches Algorithm (APSA) The All Profitable Switches Algorithm (APSA) at every iteration computes the VID s of a current iterate v and inverts the bits of v in positions where s has ones to get the new iterate v' (i.e., $v' := v$ `XOR` s.)

This algorithm may also be seen as a local-search algorithm, but the structure of the neighborhood is not fixed a priori (as for GSSA and RSSA), but rather changes for each CUPBF.

APSA is a stepwise improvement algorithm for CUPBFs because the current iterate v is the unique global minimum on the face defined by fixing all coordinates corresponding to zeros in the VID s. Therefore, the next iterate v' (which belongs to the same face) has a better function value.

Random Multiple Switches Algorithm (RMSA) Like APSA, the Random Multiple Switches Algorithm (RMSA) at every iteration computes the VID s of a current iterate v. However, to get the next iterate v' RMSA inverts bits in v corresponding to a nonempty subset s' of the nonzero bits in s, chosen uniformly at random (i.e., $v' := v$ `XOR` s'.)

RMSA is a stepwise improvement algorithm for CUPBFs because the current point v is the unique global minimum on the face defined by fixing the coordinates in v corresponding to zeros in s'. Thus the next iterate v', belonging to this face, must have a better function value. Note that RMSA selects at random from a neighborhood that may be *exponentially big* in the dimension. So, strictly speaking, this is not a polynomial local-search improvement algorithm.

Kalai-Ludwig's Randomized Algorithm (KLRA) In a major breakthrough Kalai [20] suggested the first *subexponential* randomized simplex algorithm for linear programming. Based on Kalai's ideas, Ludwig [21] suggested the first *subexponential* randomized algorithm for simple stochastic games with binary choices. We show that Ludwig's algorithm without any substantial modifications performs correctly and with the same expected worst-case complexity $2^{O(\sqrt{n})}$ for optimizing CUPBFs. Kalai-Ludwig's algorithm may informally be described as follows.

1. Start at any vertex v of $H(n)$.
2. Choose at random a coordinate $i \in \{1, \ldots, n\}$ (not chosen previously).
3. Apply the algorithm recursively to find the best point v' with the same i-th coordinate as v_i.
4. If v' is not optimal (has a better neighbor), invert the i-th component in v', set $v := v'$ and repeat.

4 The Random Multiple Switches Algorithm

We start this section by stating a couple of lemmas, that will prove useful later on.

Lemma 1. *For $0 < k < n/2$ one has*

$$\sum_{i=0}^{k-1} \binom{n}{i} < \frac{k}{n-2k} \cdot \binom{n}{k}.$$

Proof. See [11, p. 122]. □

Lemma 2. *For n, k big enough one has*

$$\binom{n}{k} \approx \sqrt{\frac{n}{2\pi k(n-k)}} \cdot \frac{n^n}{(n-k)^{n-k} \cdot k^k}$$

Proof. Apply Stirling approximations for factorials in $n!/(k!(n-k)!)$. □

The only two algorithms that were previously studied for the CUPBF optimization problem are: 1) the GSSA, 2) the RSSA. The GSSA makes an exponential number of steps on CUPBFs constructed in [29]. If extremely unlucky, the RSSA can make an exponential number of steps on CUPBFs generated by Klee-Minty cubes, but nevertheless its expected runtime on such cubes is $O(n^2)$, quadratic in the number of dimensions. The same expected quadratic upper bound holds for

the RSSA running on the class of the so-called *decomposable* CUPBFs (of which Klee-Minty's form a proper subclass) [30, p. 77-78]. Besides these results, there are no other known: 1) nontrivial lower bounds for the problem, 2) better upper bounds for any specific algorithms. Nevertheless, [30, p. 78] conjectures that the RSSA is (expected) polynomial on *all CUPBFs*.

It is worth mentioning, however, that for any CUPBF with optimum v^* and any initial vertex v_0 there is always an improving path from v_0 to v^* of length $hd(v_0, v^*)$, the Hamming distance between v_0 and v^*. Therefore, the 'Hirsch conjecture' about short paths to the optimum holds for CUPBFs (thus the potential existence of 'clever' polynomial time algorithms is not excluded).

Simultaneously, nothing except this 'trivial linear' $\Omega(n)$ lower bound is currently known for the CUPBF optimization problem. For the broader classes of all pseudo-Boolean and all unimin functions[5] there are the $\Omega(2^v/\sqrt{n})$ and the $\Omega(2^v/n^{1.5})$ lower bounds, respectively, for any deterministic algorithms, and the $\Omega(2^{n/2} \cdot n)$ lower bound for any randomized algorithms [27, Thm. 14, p. 66, Coroll. 21, p. 71, Coroll. 19, p. 69].

In view of the above results our new, presented in this section, $O(2^{0.775n}) = O(1.71^n)$ upper bound for our new RMSA (Randomized Multiple Switch Algorithm)[6] should be considered an improvement. The key step consists in exploiting complete unimodality and the next simple lemma, giving a lower bound on the per-step improvement of the target function value by the RMSA, exponential in the number of improving directions in the current vertex.

Lemma 3. *The expected function value improvement of the RMSA step from a vertex v with $i > 0$ improving directions is at least 2^{i-1}.*

Proof. Since each improving direction is chosen with probability $= 1/2$, the algorithm will jump, with equal probability, to any of the 2^i vertices in the i-dimensional subcube defined by the improving directions. Since each of these vertices must have better function values than in v, the expected number of steps of improvement must be at least $2^i/2 = 2^{i-1}$. □

This makes the RMSA attractive: we can guarantee that the expected 'value jump' in each step is relatively high, provided, there are many improving directions. If the average number of improving directions per vertex visited during a run of the RMSA were at least k, the algorithm would terminate in at most $O(2^{n-k})$ steps. In particular, $n/2$ average improving directions would give an $O(2^{n/2})$ upper bound. Can we guarantee any nontrivial lower bound on the number of improving directions in any run of the RMSA? Fortunately, the fundamental property that in every completely unimodal cube every possible bit vector of improving directions is present exactly once [30, p. 75-76] allows us to do this. The proof of the following theorem assumes the worst (and seemingly unrealizable) case that the algorithm is always unlucky, selecting a vertex with the *fewest* possible number of improving directions, and the cube generated by these direction is numbered by the *smallest* possible successors of the current value. This forces the worst case and settles an upper bound on the number of RMSA iterations in the worst case.

Theorem 4. *The expected number of iterations made by the RMSA on an n-dimensional CUPBF is less than $2^{0.775n} = 1.71^n$, for sufficiently large n.*

Proof. By Lemma 3, jumping from a vertex with i improving directions, the RMSA is expected to improve the target function value by at least 2^{i-1}. The algorithm can never, during a whole run on a CUPBF, improve by more than 2^n in total. The smallest expected improvement per iteration is achieved when the algorithm only encounters the vertices with the fewest possible number of improving directions, and the subcube generated by improving directions is numbered by the least possible values exceeding the current value. This gives a lower bound on the expected total improvement during a run[7]. If the RMSA visits all vertices with $\leq k$ improving directions, then the

[5] Possessing a unique minimum

[6] Which is not local-search type. Maybe this is the reason it was not considered in general pseudo-Boolean optimization; it is only correct for CUPBFs.

[7] We do not believe this worst case scenario may actually take place for many iterations for any CUPBF.

total number of its steps, the least possible expected total target value improvement and the least possible average (per step) improvement are given, respectively, by:

$$\sum_{i=1}^{k}\binom{n}{i}, \quad \sum_{i=1}^{k}\binom{n}{i}2^{i-1}, \quad \frac{\sum_{i=1}^{k}\binom{n}{i}2^{i-1}}{\sum_{i=1}^{k}\binom{n}{i}}.$$

Therefore, we start by proving that there are constants $c \in (0, 1/2)$ and $d \in (0, 1)$ such that

$$\sum_{i=1}^{\lfloor cn \rfloor}\binom{n}{i} \leq 2^{dn} \tag{1}$$

$$\frac{2^n}{\left(\frac{\frac{1}{2}\sum_{i=1}^{\lfloor cn \rfloor}\binom{n}{i}2^i}{\sum_{i=1}^{\lfloor cn \rfloor}\binom{n}{i}}\right)} \leq 2^{dn} \tag{2}$$

The intuition here is that (1) limits the number of iterations and the denominator in (2) is a lower bound on the expected average improvement per iteration. Thus if both properties hold, the algorithm cannot make 2^{dn} or more steps, since it would mean that the total improvement in the whole run would be bigger than 2^n, which is impossible.

We start by showing how to satisfy (1) and to find the dependency of d on c, $d = d(c)$. By Lemma 1,

$$\sum_{i=1}^{\lfloor cn \rfloor}\binom{n}{i} \leq \sum_{i=1}^{\lfloor cn \rfloor - 1}\binom{n}{i} + \binom{n}{\lfloor cn \rfloor} < \left(\frac{\lfloor cn \rfloor}{n - 2 \cdot \lfloor cn \rfloor} + 1\right)\binom{n}{\lfloor cn \rfloor} \approx$$

(using Stirling approximation from Lemma 2)

$$\approx \left(\frac{\lfloor cn \rfloor}{n - 2\lfloor cn \rfloor} + 1\right) \cdot \sqrt{\frac{n}{2\pi(n - \lfloor cn \rfloor)\lfloor cn \rfloor}} \cdot \frac{n^n}{(n - \lfloor cn \rfloor)^{n - \lfloor cn \rfloor}\lfloor cn \rfloor^{\lfloor cn \rfloor}} =$$

$$= p(n) \cdot \frac{n^n}{(n - \lfloor cn \rfloor)^{n - \lfloor cn \rfloor}\lfloor cn \rfloor^{\lfloor cn \rfloor}} \leq p(n) \cdot \left(\frac{1}{(1-c)^{1-c}c^c}\right)^n,$$

where $p(n)$ is a polynomial. The last step is valid since by removing the floor function in the denominator, both $(n - \lfloor cn \rfloor)$ and $\lfloor cn \rfloor$ are brought closer to $n/2$ (remember that c should be strictly smaller than $1/2$, and the 'middle' binomial coefficient is the biggest).

Let R stand for the expression

$$\frac{1}{(1-c)^{1-c}c^c}.$$

Immediate computation for $c = 1/2$ gives $R = 2$, as well as $R < 2$ for all $c \in (0, \frac{1}{2})$. Thus there exist $R < 2$ and $d = \log(R) < 1$ such that

$$\sum_{i=1}^{\lfloor cn \rfloor}\binom{n}{i} < R^n = 2^{dn},$$

and thus (1) can be satisfied. Actually, we are interested in selecting c as small as possible, since it determines the total number of steps $\sum_{i=1}^{cn}\binom{n}{i}$, but we must also select c big enough to satisfy (2).

We now proceed to showing how (2) can be satisfied. Using (1) it is enough to prove that $c < 1/2$ can be chosen such that

$$2 \cdot 2^n \leq \sum_{i=1}^{\lfloor cn \rfloor}\binom{n}{i}2^i.$$

The sum on the right-hand side is bigger than its last term:

$$\sum_{i=1}^{\lfloor cn \rfloor} \binom{n}{i} 2^i > \binom{n}{\lfloor cn \rfloor} 2^{\lfloor cn \rfloor}.$$

Unless cn is an integer the following holds:

$$\frac{\binom{n}{\lceil cn \rceil}}{\binom{n}{\lfloor cn \rfloor}} = \frac{n - \lfloor cn \rfloor}{\lfloor cn \rfloor + 1} < \frac{n}{cn} = \frac{1}{c}.$$

If the restriction $c > b$, for $0 < b < 1/2$ is imposed, then

$$\binom{n}{\lfloor cn \rfloor} 2^{\lfloor cn \rfloor} \geq b \cdot \binom{n}{\lceil cn \rceil} 2^{\lfloor cn \rfloor}.$$

Applying Lemma 2 to the right side gives:

$$b \cdot \binom{n}{\lceil cn \rceil} 2^{\lfloor cn \rfloor} \geq p(n) \left(\frac{1}{(1-c)^{1-c} c^c} \right)^n 2^{\lfloor cn \rfloor}$$

where p is a polynomial[8]. Here we recognize our expression R. If c is chosen such that $\log R + c > 1$ we get:

$$p(n) \cdot R^n \cdot 2^{\lfloor cn \rfloor} = p(n) \cdot 2^{n \log R} \cdot 2^{\lfloor cn \rfloor} \geq$$

$$\geq p(n) \cdot 2^{n \log R + cn - 1} = \frac{p(n)}{2} \cdot 2^{n(\log R + c)} > 2^n$$

for sufficiently large n. In fact, $\log R + c > 1$ holds whenever $0.228 \leq c < 1/2$.

This shows that there are constants c and d such that both the above properties (1), (2) are satisfied. If we select c as small as possible (approximately 0.228), numerical calculation gives that we can select any $d > 0.775$. Thus, for sufficiently large n, the expected number of iterations made by the RMSA on an n-dimensional CUPBF is less than $2^{0.775n} \approx 1.71^n$. This concludes the proof. □

After we obtained the bound from Theorem 4, we were pointed out that a similar bound for the same algorithm, but applied to Markov decision processes, was proved earlier in [22]. Later it became clear that our result is stronger, after we succeeded to reduce simple stochastic games to CLG-functions and CLG-functions to CU-functions [6]. Moreover, we substantially improve the bound from Theorem 4 below $2^{n/2}$ in Section 6.

5 Single Greedy, Single Random, and All Profitable Switches Algorithms

We start with a simple lower bound on the per-step improvement for all three algorithms. This bound is weaker than for the RMSA. Consequently, the upper bounds we can settle for these algorithms are weaker. Nevertheless, the practical behavior of these algorithms shown in experiments [2, 4, 3] make them very attractive.

Proposition 5. *When the APS, the GSS, or the RSS algorithms make a switch in a vertex with i improving directions the value of the target function increases at least by i (by expected value at least $i/2$ for the RSSA).*

[8] When we remove the ceiling function we get cn instead of $\lceil cn \rceil$. Since the middle binomial coefficients are the biggest, we get something smaller on the right-hand side. If cn is an integer, it works since $cn = \lfloor cn \rfloor = \lceil cn \rceil$.

Proof. For the GSSA it is straightforward, it selects the best out of i better and different values of neighbors. For the APSA, there should exist a value-improving path of length i from the current vertex to the opposite corner of the subcube defined by the improving directions. Each step of the path improves at least by one. The RSSA selects uniformly at random among the i different neighbors. Therefore the function value improvement is expected to be at least $i/2$. □

Theorem 6. *For any $0 < \varepsilon < 1$ the All Profitable Switches Algorithm, the Greedy Single Switch Algorithm, and the Randomized Single Switch Algorithm make fewer than $2^{n-(1-\varepsilon)\log(n)}$ iterations on any n-dimensional CUPBF, for sufficiently large n.*

Proof. Let $f(n) = (1-\varepsilon)\log(n)$ and assume that on an n-dimensional hypercube H the algorithm (APSA, RSSA, or GSSA) makes $2^{n-f(n)}$ iterations. Let k be the largest integer satisfying

$$\sum_{i=1}^{k} \binom{n}{i} \leq 2^{n-f(n)}. \tag{3}$$

(Obviously, $k < \lceil n/2 \rceil$ since otherwise the sum is not less than 2^{n-1}.)

Let A be the average number of vertices 'skipped' in one iteration, when the algorithm is run on H. Then

$$A \geq \frac{\sum_{i=1}^{k} \binom{n}{i} \frac{i}{2}}{\sum_{i=1}^{k} \binom{n}{i}}.$$

(Here and in (3) we assume the worst case: an adversary presents a cube H which forces the algorithm to make switches with the fewest possible numbers of improving directions, and each i-switch improves by the least possible expected value $i/2$. Note that $i/2$ is used to make the proof valid for the RSSA. For the APSA and the GSSA we could use i and get a stronger lower bound on A, but for the purpose of this proof it makes no difference.)

Since k must be smaller than $n/2$ we get

$$A \geq \frac{\sum_{i=1}^{k} \binom{n}{i} \frac{i}{2}}{\sum_{i=1}^{k} \binom{n}{i}} \geq \frac{\binom{n}{1}\frac{k}{4} + \binom{n}{2}\frac{k}{4} + \ldots \binom{n}{k}\frac{k}{4}}{\binom{n}{1} + \binom{n}{2} + \ldots \binom{n}{k}} \geq \frac{k}{4}.$$

The algorithm cannot make more than $\frac{2^n}{A}$ steps (the value domain $\{1, \ldots, 2^n\}$ is exhausted), so we want to prove that

$$\frac{2^n}{A} < \frac{2^n}{2^{f(n)}},$$

which holds when $A > 2^{f(n)}$. Since $A \geq k/4$, let us prove that $k/4 > 2^{f(n)}$ for some k satisfying (3) (otherwise taking $k = n/2$ would do the job). Note that by the choice of k,

$$2^{n-f(n)} \leq \sum_{i=1}^{k+1} \binom{n}{i} \leq n^{k+2}.$$

By taking logarithms of both sides we get

$$n - f(n) \leq (k+2)\log(n) \Leftrightarrow \frac{n-f(n)}{\log(n)} - 2 \leq k \Leftrightarrow \frac{k}{4} \geq \frac{n-f(n)}{4\log(n)} - \frac{1}{2},$$

and k may be selected smaller than $n/2$. Now it suffices to show

$$\frac{n-f(n)}{4\log(n)} - \frac{1}{2} > 2^{f(n)},$$

which holds because asymptotically we have

$$\frac{n-f(n)}{4\log(n)} - \frac{1}{2} > n^{1-\epsilon} = 2^{f(n)}.$$

This concludes the proof. □

Remark 7. 1) Note that it is stronger than claiming 'fewer than 2^{n-c} (for a constant $c > 0$)', but 2) weaker than 'fewer than 2^{cn} (for a constant $0 < c < 1$)'. 3) This upper bound is better than the $5/24 \cdot 2^n - 45$ lower bound for any local improvement algorithm on a uniminmax function [27, Thm 23, p. 72].

6 Adding Random Sampling to the RMSA

The RMSA can be considerably improved by adding random sampling. If we start the RMSA from a 'good' vertex, with a value close to the optimum, the RMSA guarantees a reasonably short run before finding it, as is shown in Section 4. The trick is to select a good initial vertex by making an optimal number of random probes picking the one with the best value, and to minimize the overall running time. We call the modified algorithm the RMSA-RS and parameterize it by the number of randomly sampled vertices. For this modified algorithm, a better upper bound can be shown, when we choose the parameter optimally:

Theorem 8. *The RMSA-RS can be parameterized in such a way that its expected running time on an n-dimensional CUPBF is $O(2^{0.46n}) = O(1.376^n)$.*

Proof. Suppose that we first take uniformly at random 2^{xn} samples of vertices on the CUPBF, for some $x \in (0, 1)$. The expected difference between the best value in any of the sampled vertices and the optimal value is at most $2^{n(1-x)}$. If we select $x = 1/2$, the best expected probed value will be at most $2^{n/2}$ away from the optimum, and *any* subsequent iterative improvement will take at most $2^{n/2}$ iterations. But we hope to select x smaller than $1/2$, since the RMSA converges faster. Formally, we want to find constants $c \in (0, 1/2)$ and $d \in (0, 1/2)$ such that

$$\sum_{i=1}^{\lfloor cn \rfloor} \binom{n}{i} \leq 2^{dn} \tag{4}$$

$$\frac{2^{n(1-x)}}{\left(\frac{\frac{1}{2} \sum_{i=1}^{\lfloor cn \rfloor} \binom{n}{i} 2^i}{\sum_{i=1}^{\lfloor cn \rfloor} \binom{n}{i}} \right)} \leq 2^{dn} \tag{5}$$

Using the same arguments as in Section 4, it can be shown that (4) can be satisfied by selecting any $0 < c \leq 0.11$. As in Section 4, to satisfy (5) it is enough to provide

$$2 \cdot 2^{n(1-x)} \leq \sum_{i=1}^{cn} \binom{n}{i} 2^i.$$

(Here we are less strict about floors and ceilings than in Section 4; they can be treated rigorously in the same way as before.)

Note again (using the Stirling approximation) that

$$\sum_{i=1}^{cn} \binom{n}{i} 2^i > \binom{n}{cn} 2^{cn} \approx p(n) \left(\frac{1}{(1-c)^{1-c} c^c} \right)^n 2^{cn} = p(n) \cdot R^n \cdot 2^{cn},$$

where p is a polynomial. If c is chosen such that $\log R + c > (1 - x)$ we get

$$p(n) \cdot R^n \cdot 2^{cn} = p(n) \cdot 2^{n \log R} \cdot 2^{cn} = p(n) \cdot 2^{n(\log R + c)} \geq 2 \cdot 2^{n(1-x)}$$

for sufficiently large n. Now the optimal value x is chosen such that there is a $c \leq 0.11$ satisfying $\log R + c \geq 1 - x$ and $d = \log R$ being as close to x as possible. Numerical calculations give that for $x = 0.46$ one can choose $c = 0.095$ and $d = 0.45$.

Therefore, after sampling $2^{0.46n}$ vertices the algorithm is expected to do less than $2^{0.45n}$ iterations. Thus the RMSA-RS with random sampling of $2^{0.46n}$ vertices has an expected running time $O(2^{0.46n})$ on an n-dimensional CUPBFs. □

As described, the RMSA-RS always makes $2^{0.46n}$ random samplings, before starting any optimizations, so its expected best case is $\Omega(2^{0.46n})$. Although other single or multiple switch algorithms we consider have worse known upper bounds, they show much better practical behavior.

7 Adding Random Sampling to the APSA, the GSSA, and the RSSA

As we saw in the previous section, a better bound can be proved for the RMSA when random sampling is added. In this section we show that random sampling also allows for better bounds for the APSA, the GSSA, and the RSSA. The bounds are not as strong, however, as the one for the RMSA with random sampling.

Theorem 9. *For any $0 < \varepsilon < 1/2$ the All Profitable Switches, the Greedy Single Switch, and the Randomized Single Switch Algorithms with initial random sampling of $2^{\frac{n}{2}-(\frac{1}{2}-\varepsilon)\log(n)}$ vertices make less than $2^{\frac{n}{2}-(\frac{1}{2}-\varepsilon)\log(n)}$ iterations on any n-dimensional CUPBF, for sufficiently large n.*

Proof. Let $f(n) = (\frac{1}{2} - \varepsilon)\log(n)$. After making $2^{\frac{n}{2}-f(n)}$ initial random samplings, we are expected to have found a starting vertex with a value p satisfying $2^n - p \leq 2^{\frac{n}{2}+f(n)}$. Now assume that starting from such a vertex the algorithm (APSA, RSSA, or GSSA) makes $2^{\frac{n}{2}-f(n)}$ iterations. Let k be the largest integer satisfying

$$\sum_{i=1}^{k} \binom{n}{i} \leq 2^{\frac{n}{2}-f(n)}. \tag{6}$$

(Obviously, $k < \lceil n/2 \rceil$ since otherwise the sum is not less than 2^{n-1}.)

Let A be the average number of vertices 'skipped' in one iteration. Then

$$A \geq \frac{\sum_{i=1}^{k} \binom{n}{i}\frac{i}{2}}{\sum_{i=1}^{k} \binom{n}{i}}.$$

(Here and in (6) we assume the same worst case as in the previous sections: the algorithm encounters only the vertices with the fewest number of improving directions, and each switch improves by the least possible expected value $i/2$.)

Since k must be smaller than $n/2$, we get

$$A \geq \frac{\sum_{i=1}^{k} \binom{n}{i}\frac{i}{2}}{\sum_{i=1}^{k} \binom{n}{i}} \geq \frac{\binom{n}{1}\frac{k}{4} + \binom{n}{2}\frac{k}{4} + \ldots\binom{n}{k}\frac{k}{4}}{\binom{n}{1} + \binom{n}{2} + \ldots\binom{n}{k}} \geq \frac{k}{4}.$$

The algorithm cannot make more than $\dfrac{2^{\frac{n}{2}+f(n)}}{A}$ steps (the value domain is exhausted), so we want to prove that

$$\frac{2^{\frac{n}{2}+f(n)}}{A} < \frac{2^{\frac{n}{2}}}{2^{f(n)}}.$$

Since $A \geq k/4$, it is enough to prove that $k/4 > 2^{2f(n)}$ for some k satisfying (6) (otherwise taking $k = n/2$ would do the job). Note that by the choice of k,

$$2^{\frac{n}{2}-f(n)} \leq \sum_{i=1}^{k+1} \binom{n}{i} \leq n^{k+2}.$$

By taking logarithms of both sides we get

$$\frac{n}{2} - f(n) \leq (k+2)\log(n) \Leftrightarrow \frac{n - 2f(n)}{2\log(n)} - 2 \leq k \Leftrightarrow \frac{k}{4} \geq \frac{n-2f(n)}{8\log(n)} - \frac{1}{2},$$

and k may be selected smaller than $n/2$. Now it suffices to show

$$\frac{2^{\frac{n}{2}+f(n)}}{\left(\frac{n-2f(n)}{8\log(n)}-\frac{1}{2}\right)} < \frac{2^{\frac{n}{2}}}{2^{f(n)}}, \quad \text{or} \quad \frac{1}{\left(\frac{n-2f(n)}{8\log(n)}-\frac{1}{2}\right)} < \frac{1}{2^{2f(n)}},$$

which holds, because asymptotically we have

$$2^{2f(n)} = n^{2(\frac{1}{2}-\varepsilon)} = n^{1-2\varepsilon} < \frac{n-2f(n)}{8\log(n)} - \frac{1}{2}.$$

This concludes the proof. □

Corollary 10. *With the initial random sampling of $2^{\frac{n}{2}-(\frac{1}{2}-\varepsilon)\log n}$ vertices, for any $\varepsilon \in (0, 1/2)$, the APSA, the GSSA, and the RSSA have running times (expected in the case of RSSA) that are $O(2^{\frac{n}{2}-(\frac{1}{2}-\varepsilon)\log n})$.*

8 Kalai-Ludwig's Algorithm for CUPBFs

We were the first to observe [2] that the Kalai-Ludwig's Randomized Algorithm (KLRA) initially designed for linear programming [20] and later adapted for simple stochastic games [21], works perfectly for CUPBF optimization, also providing *subexponential* expected running time. Modified for CUPBF optimization the KLRA is shown below.

Algorithm 1: Kalai-Ludwig's Algorithm for CUPBFs

```
KLRA(CUPBF H, initial vertex v_0)
(1)   if dim(H) = 0
(2)       return v_0
(3)   else
(4)       choose a random facet F of H containing v_0
(5)       v* ← KLRA(F, v_0)
(6)       if neighbor u of v* on H \ F is better than v* then return KLRA(H \ F, u)
(7)                                                    else return v*
```

It turns out that the algorithm is correct and terminating:

Theorem 11. *For every CUPBF KLRA terminates and returns the global maximum.*

Proof. The algorithm terminates because recursive calls are made on strictly smaller hypercubes. The proof of correctness is by induction on the dimension d of H. If $d = 0$ there is only one vertex and it is optimal. Suppose the algorithm returns the correct vertex for all cubes of dimension less than d. Then the first recursive call returns the optimum v^* on F. If this is the optimum on H then the algorithm correctly returns v^*. Otherwise, the optimum is on $H \setminus F$, and by inductive assumption the call on line 6 returns it. □

The following adjusts the theorem and proof in [21] to the case of CUPBFs.

Theorem 12. *The expected running time of KLRA on a CUPBF is $2^{O(\sqrt{n})}$.*

Proof. Everything in the algorithm, except the recursive calls, takes polynomial time. The key issue is the number of recursive calls in line 6, referred to as *switches* (since they change a facet of the cube). Following Ludwig, let $f(d)$ denote the expected number of switches on a d-dimensional

CUPBF, and $h(F)$ denote the maximum value on the face F. When the algorithm chooses a facet of H, it has d different facets to chose from. Order these facets so that

$$h(F_1) \geq h(F_2) \geq \ldots \geq h(F_n).$$

Suppose the algorithm randomly chooses facet F_r in line 4. By solving a subproblem of size $d-1$ it finds the maximum vertex on this facet. Since the algorithm can only switch if the neighbor of the maximum vertex has a bigger value, it can never switch to any vertex in the facets $F_{r+1}, F_{r+2}, ..., F_n$. This means that $n-r$ dimensions will remain fixed for the rest of the computation. and narrows down the search to an r-dimensional face of H. Therefore, after solving one recursive problem, making no more than $f(d-1)+1$ switches, the algorithm solves the remaining problem using no more than $f(r-1)$ switches. This gives the following recurrence:

$$f(d) \leq f(d-1) + \frac{1}{d}\sum_{i=1}^{d-1} f(i) + 1$$

Ludwig solves this recurrence and shows that $f(d) \leq e^{2\sqrt{d-1}}$ Thus the algorithm has an expected subexponential running time. □

Our experiments on randomly generated CUPBFs [2, 4, 3] indicate that KLRA performs better than its theoretical subexponential upper bound. Surprisingly, the other four algorithms considerably outperform KLRA, although no subexponential bounds are currently known for them. It is reasonable to believe that for some of the algorithms there may be subexponential or better upper bounds. In this work we prove the first nontrivial upper bounds for these algorithms. The bounds we show are still exponential, but not expected to be tight. Rather, they are to be viewed as a first step towards settling the precise complexity of these algorithms on CUPBFs.

9 Completely Local-Global (CLG) Functions

So far we restricted our attention to binary games. Although every non-binary game can be reduced to a binary one, the resulting number of vertices is proportional to the size of the initial graph. This may give a quadratic blow-up in the number of vertices (e.g., for graphs of linear outdegree), and the $2^{O(\sqrt{n})}$ bound becomes exponential, since n is quadratic in the initial number of vertices. In this section we show how to apply more sophisticated algorithms directly on non-binary structures, maintaining the subexponential bounds. We start with a non-binary generalization of hypercubes.

Definition 13 (Hyperstructure). *For each $j \in \{1, \ldots, d\}$ let $\mathcal{P}_j = \{e_{j,1}, \ldots, e_{j,\delta_j}\}$ be a finite set. Call $\mathcal{P} = \prod_{j=1}^{d} \mathcal{P}_j$ a d-dimensional* hyperstructure*, or* structure *for short.* □

A *substructure* $\mathcal{P}' \subseteq \mathcal{P}$ is a product $\mathcal{P}' = \prod_{j=1}^{d} \mathcal{P}'_j$, where $\mathcal{P}'_j \subseteq \mathcal{P}_j$ for all j. If each $\mathcal{P}'_j$ has only two elements, then identify $\mathcal{P}'$ with H. Call $\mathcal{P}'$ a *facet* of $\mathcal{P}$ if there is a j such that $\mathcal{P}'_k = \mathcal{P}_k$ for all $k \neq j$, and $\mathcal{P}'_j$ has only one element. Say that $x, y \in \mathcal{P}$, are *neighbors* iff they differ in only one coordinate. Thus each $x \in \mathcal{P}$ has exactly $\sum_{j=1}^{d}(\delta_j - 1)$ neighbors. The neighbor relation defines a graph with elements of the hyperstructure as nodes, allowing us to talk about paths and distances in the hyperstructure. A structure $\mathcal{P} = \prod_{j=1}^{d} \mathcal{P}_j$ has d dimensions and $n = \sum_{j=1}^{d} \delta_j$ facets.

Throughout this section, let $\mathcal{D}$ be some partially ordered set. We now consider functions defined on $\mathcal{P}$ with values in $\mathcal{D}$. Functions with partially ordered codomains are better suited for games [6].

A *local maximum* of a function $f : \mathcal{P} \to \mathcal{D}$ is a vertex in $\mathcal{P}$ with value bigger than or equal to all its neighbors. A *global maximum* is a vertex with a function value bigger than or equal to the values of all other vertices. In particular, any global maximum is comparable with all other vertices. Local and global minima are defined symmetrically.

Definition 14 (CLG-function on a hyperstructure). *Let $f : \mathcal{P} \to \mathcal{D}$ be a function such that all neighbors have comparable values. Say that f is a* CLG-function *if the following properties hold on every substructure of $\mathcal{P}$:*

1. any local maximum of f is also global;
2. any local minimum of f is also global;
3. any two local maxima are connected by a path of local maxima;
4. any two local minima are connected by a path of local minima.

By a CLG-structure *we mean a CLG-function together with the underlying hyperstructure.*

We note in passing that CLG-functions can be defined on hypercubes as well (with the same four properties), and CUPBFs can be defined: 1) with a partial set $\mathcal{D}$ as co-domain, and 2) on hyperstructures. The relaxation of the co-domain from $\mathbb{R}$ to $\mathcal{D}$ is an advantage (the important properties rely only on the order of neighbors), important for the applications to games. Many properties are also carried on from hypercubes to hyperstructures, usually with a modified formulation. Multiple switches algorithms APSA, RMSA generalize to hyperstructures as well.

As shown in [6], Algorithm 2 optimizes CLG-functions on hyperstructures in expected time $2^{O(\sqrt{d \log(n/\sqrt{d})} + \log n)}$, where $n = \sum_{j=0}^{d} \delta_j$. For games, where the maximal outdegree is d and $n = O(d^2)$, the bound collapses to $2^{O(\sqrt{d \log d})}$. The algorithm is adapted from the linear programming algorithm by Matoušek, Sharir and Welzl [26, 23]. If $\mathcal{P}$ happens to be binary, the algorithm coincides with Kalai-Ludwig's algorithm; see Section 8.

Algorithm 2: MSW-Style Optimization Algorithm

```
OPTIMIZE(CLG-structure P, initial vertex v_0)
(1)   if P = v_0
(2)       return v_0
(3)   else
(4)       choose a random facet F of P, not containing v_0
(5)       v* ← OPTIMIZE(P \ F, v_0)
(6)       if neighbor u of v* on F is better than v* then return OPTIMIZE(F, u)
(7)                                                    else return v*
```

The importance of CLG-functions stems from the fact that the strategy measures from [21, 28, 5] are indeed CLG-functions, as shown in [6]. Moreover, CLG-functions on hypercubes can be transformed to CUPBFs by introducing an artificial order on unordered neighbors [6]. Also, CLG-functions on hyperstructures can be transformed analogously to CUPBF-like functions on hyperstructures. Finally, we showed in [6] that any CLG-function reduces to an *LP-type problem*, an abstract framework for linear programming [26] and problems in computational geometry [23, 14]. These reductions provide a strong argument for CLG-functions as a link between well-studied areas that look very different on the surface.

CLG-functions simultaneously allow for subexponential and multiple switch algorithms. Although the latter currently have worse known upper bounds, good practical behavior of such algorithms, confirmed by experiments, make them very attractive and competitive [2, 4, 3]. We hope that a thorough investigation of random walks on the favorable CU- or CLG-structures will allow for improved bounds for multiple switching algorithms, both on the average and in the worst case.

References

1. E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley & Sons, 1997.

2. 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/`.
3. 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/`.
4. H. Björklund, S. Sandberg, and S. Vorobyov. Optimization on completely unimodal hypercubes. Technical Report 2002-018, 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, 2003. `http://www.it.uu.se/research/reports/`.
7. E. Boros and P. L. Hammer. Pseudo-boolean optimization. Technical Report RRR 48-2001, RUTCOR Rutger Center for Operations Research, 2001.
8. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.
9. A. Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.
10. A. Condon. On algorithms for simple stochastic games. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 13:51–71, 1993.
11. Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT electrical engineering and computer science series. MIT Press, Cambridge, MA, 6th printing edition, 1992.
12. 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.
13. 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.
14. B. Gärtner. A subexponential algorithm for abstract optimization problems. *SIAM Journal on Computing*, 24:1018–1035, 1995.
15. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata Logics and Infinite Games. A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer-Verlag, 2003.
16. P. L. Hammer, B. Simeone, Th. M. Liebling, and D. De Werra. From linear separability to unimodality: a hierarchy of pseudo-boolean functions. *SIAM J. Disc. Math.*, 1(2):174–184, 1988.
17. P. Hansen, B. Jaumard, and V. Mathon. Constrained nonlinear 0-1 programming (state-of-the-art survey). *ORSA Journal on Computing*, 5(2):97–119, 1993.
18. A. J. Hoffman and R. M. Karp. On nonterminating stochastic games. *Management Science*, 12(5):359–370, 1966.
19. M. Jurdziński. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *17th STACS*, volume 1770 of *Lect. Notes Comput. Sci.*, pages 290–301. Springer-Verlag, 2000.
20. G. Kalai. A subexponential randomized simplex algorithm. In *24th ACM STOC*, pages 475–482, 1992.
21. W. Ludwig. A subexponential randomized algorithm for the simple stochastic game problem. *Information and Computation*, 117:151–155, 1995.
22. Y. Mansour and S. Singh. On the complexity of policy iteration. In *Uncertainty in Artificial Intelligence'99*, 1999.
23. J. Matoušek, M. Sharir, and M. Welzl. A subexponential bound for linear programming. In *8th ACM Symp. on Computational Geometry*, pages 1–8, 1992.
24. C. Papadimitriou. Algorithms, games, and the internet. In *ACM Annual Symposium on Theory of Computing*, pages 749–753. ACM, July 2001.
25. V. Petersson and S. Vorobyov. A randomized subexponential algorithm for parity games. *Nordic Journal of Computing*, 8:324–345, 2001.
26. M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *9th Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 577 of *Lecture Notes in Computer Science*, pages 569–579, Berlin, 1992. Springer-Verlag.
27. 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.

28. J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In E. A. Emerson and A. P. Sistla, editors, *CAV'00: Computer-Aided Verification*, volume 1855 of *Lect. Notes Comput. Sci.*, pages 202–215. Springer-Verlag, 2000.
29. D. Wiedemann. Unimodal set-functions. *Congressus Numerantium*, 50:165–169, 1985.
30. K. Williamson Hoke. Completely unimodal numberings of a simple polytope. *Discrete Applied Mathematics*, 20:69–81, 1988.
31. U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158:343–359, 1996.