Optimization on Completely Unimodal Hypercubes

Henrik Björklund, Sven Sandberg & Sergei Vorobyov

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

Technical report 2002-018
May 2002
ISSN 1404-3203

Abstract

We investigate and compare, both theoretically and practically, several old and new algorithms for the completely unimodal pseudo-boolean function optimization problem. We settle the first nontrivial upper bounds for two well-known local search algorithms, the Random and the Greedy Single Switch algorithms. We also present the new Random Multiple and All Profitable Switches algorithms. These are not local search-type algorithms, and have not been previously considered for this problem. We justify the algorithms and show nontrivial upper bounds. In particular we prove a $O(2^{0.775n})$ bound for the Random Multiple Switches Algorithm, and also use random sampling to improve the bounds for all above algorithms. In particular we give a $O(2^{0.46n}) = O(1.376^n)$ upper bound for the Random Multiple Switches Algorithm with Random Sampling.

We also show how Kalai-Ludwig's algorithm for Simple Stochastic Games (Ludwig 1995) can be modified to solve the problem at hand, and that the modified algorithm preserves the subexponential, $2^{O(\sqrt{n})}$, running time.

We introduce and justify a new method for random generation of 'presumably hard' completely unimodal pseudo-boolean functions. We also present experimental results indicating that all the above algorithms perform well in practice. The experiments, surprisingly, show that in all generated example cases, the subexponential Kalai-Ludwig's algorithm is outperformed by all the other algorithms.

Contents

1 Introduction

In this paper we settle 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 (Hammer, Simeone, Liebling & De Werra 1988, Williamson Hoke 1988), 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 (Williamson Hoke 1988). The importance of the problem stems in part from its connection to game theory. Finding the winner in the so called *Simple Stochastic Games* is polynomial time reducible to optimizing CUPBFs. Many other games, like *Discounted Payoff Games*, *Mean Payoff Games*, and *Parity Games*, can in turn be reduced to Simple Stochastic Games. Since the PARITY GAMES problem is known to be polynomial time equivalent to that of μ-CALCULUS MODEL CHECKING, the existence of a polynomial time algorithm for optimizing CUPBFs would imply the existence of a polynomial time algorithm for μ-CALCULUS MODEL CHECKING. Studying the problem in the abstract setting of pseudo-boolean optimization simplifies and unifies the study and comparison of different algorithms.

1.1 Theoretical Investigations of the Algorithms

We investigate and compare, both theoretically and practically, 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 (Björklund, Petersson & Vorobyov 2001). 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 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 lower bound that has been shown is an exponential one for the GSSA. Local search 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 (Ludwig 1995) 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.

Our experiments on randomly generated CUPBFs 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 those algorithms. 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.

1.2 Cube Generation and Experimental Evaluation of the Algorithms

Generating nontrivial random CUPBFs, which would be difficult for any of the above-mentioned algorithms, is a challenging open problem. (Recall that it is a long-standing conjecture that any CUPBF is solvable by the RSSA in expected polynomial time!) Most 'regular' ways of generating CUPBFs produce 'easy-to-crack' problems. We invented a method that may possibly produce some 'difficult' examples, if there are any (cf., the conjecture above). In any case, until now we were unable to prove that our cubes are easy for our algorithms. The extensive experimental studies show, however, that all the cubes we generated so far are *easily* breakable by all the algorithms in seemingly low-degree polynomial time. It is also surprising that the theoretically 'best' (with the strongest known upper bound) Kalai-Ludwig's algorithm demonstrates the worst practical behavior among all the algorithms.

The idea of our generation method is as follows. We create an $n+1$-dimensional CUPBF recursively, out of two n-dimensional CUPBFs by connecting their vertices using directed edges. This is nontrivial, since complete unimodality should be preserved. There are two simple methods for doing this: 1) throw all new edges in one direction (creates an absorbing facet) and 2) take two identical copies throwing connecting edges randomly. It is not too hard to prove that these methods are actually equivalent, generating the same set of cubes but with the coordinates in reverse order. While guaranteeing complete unimodality, these methods create provably 'easy' to solve CUPBFs. Our method is more involved. It first computes classes of vertices that should simultaneously throw an outgoing or receive an incoming connecting edge from/to the opposite facet (this exploits the property that every possible binary vector of improving directions should be present exactly once), and then by using randomization, backtracking, and acyclicity checking proceeds to orienting classes of connecting edges. This process is quite expensive, not speaking about the necessity of keeping the full n-dimensional cube in memory. Until now we only dealt with CUPBFs of dimensions up to 27 (and continue upwards). Except finding worst-case times for our algorithms on the generated cubes, we are also checking different characteristics of the cubes. For example, once for almost all vertices of the cube at least one half of improving directions are 'correct', diminishing the Hamming distance to the optimum, then most of our algorithms are provably polynomial. It turns out that the 'above 1/2-correctness' property does not hold in our generation, which could be considered as an evidence of their 'difficulty'. Despite this, all the algorithms we investigated behave excellently on the instances we generated so far. Currently our research is concentrated on theoretically explaining these phenomena and improving upper bounds for the algorithms.

2 Preliminaries

2.1 Completely Unimodal Pseudo-Boolean Functions

Let $\mathcal{H}(n)$ denote an *n-dimensional Boolean hypercube* $\{0,1\}^n$, for $n \in \mathbb{N}^+$.

A *pseudo-Boolean function* is a mapping $\mathcal{H}(n) \to \mathbb{R}$ associating a real number to every n-dimensional Boolean vector. For $0 \leq k < n$, a *k-dimensional face* of $\mathcal{H}(n)$, 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 $\mathcal{H}(n)$ has exactly n neighbors, forming the *standard neighborhood* of v on $\mathcal{H}(n)$.

Proviso. All pseudo-Boolean functions in this paper will be considered *injective*. This simplifies the analysis and is adequate for our purposes. Consequently, without loss of generality we always assume throughout this paper that all pseudo-Boolean functions on $\mathcal{H}(n)$ take different integer values in the range $1, \ldots, 2^n$. Also, in this paper we will always assume that we are interested in *maximizing* the function value.

Complete Unimodality. A pseudo-Boolean function f is called *completely unimodal* (CUPBF for short) if one of the four equivalent conditions holds (Hammer et al. 1988, Williamson Hoke 1988):

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.

Improving directions. Let $f : \{0,1\}^n \to \mathbb{R}$ be a CUPBF and let v be a vertex on the n-dimensional boolean hypercube H. Number the dimensions of H 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 dimension 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. □

2.2 Five Iterative Improvement Algorithms

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

(1) $v_0 \leftarrow$ arbitrary initial point on the hypercube
(2) $i \leftarrow 0$
(3) **while** $N(v_i)$ contains an element v' with $f(v') > f(v_i)$
(4) $\quad i \leftarrow i + 1$
(5) $\quad v_i \leftarrow v'$

Specific instances of the SLSIA are obtained when one fixes:

1. the neighborhood structure on $\mathcal{H}(n)$,

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. They are described in Sections 2.2.1 and 2.2.2 respectively.

We introduce, describe, and justify two new, previously uninvestigated algorithms, the *All Profitable Switches Algorithm (APSA)* and the *Random Multiple Switches Algorithm (RMSA)* in Sections 2.2.3 and 2.2.4. None of them are **local-search algorithms**. 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 Simple Stochastic Games can be modified to optimize CUPBFs, and show that it is subexponential for this problem as well[1]. This well-known algorithm has so far been ignored in the field of CUPBF optimization.

2.2.1 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

[1] Its expected worst case behavior is $2^{O(\sqrt{n})}$.

vertex of $\mathcal{H}(n)$ has exactly n neighbors (in the standard neighborhood). This algorithm may take *exponentially many* steps to find the global maximum in the worst case; see (Williamson Hoke 1988, p. 77).

2.2.2 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 (1988) has shown, using a proof-technique due to Kelly, that the RSSA has an 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.

2.2.3 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'. (Think of it as $v' := v \texttt{ XOR } s$.)

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.

The fact that APSA is a stepwise improvement algorithm (for CUPBFs) follows from the fact that the current point 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.

2.2.4 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 new iterate v' RMSA uniformly at random selects a non-empty set of nonzero components s' of s and inverts the bits of v in the nonzero positions of s'. (Think of it as $v' := v \texttt{ XOR } s'$, where s' contains randomly selected non-zero bits of s.)

The fact that RMSA is a stepwise improvement algorithm (for CUPBFs) follows from the fact that the current point v is the unique global minimum on any face defined by fixing any superset of coordinates corresponding to zeros in the VID s. By the same argument as above the next iterate v' (which belongs to the same face) must have a better function value by definition of CUPBFs. Note that in principle, RMSA selects at random from a neighborhood, which may be *exponential*. So, strictly speaking, this is not a local-search improvement algorithm.

This is the algorithm in pseudo-code:

RMSA(H, v_0)
(1) $v \leftarrow v_0$
(2) $vid \leftarrow$ the vector of improving directions from v
(3) **while** vid is not the all-zero vector
(4) $vid' \leftarrow$ non-zero subvector of vid, chosen uniformly at random
(5) $v \leftarrow$ the vertex reached by moving in all directions indicated in vid' from v
(6) $vid \leftarrow$ the vector of improving directions from v
(7) **return** v

2.2.5 Kalai-Ludwig's Randomized Algorithm (KLRA)

In a major breakthrough Kalai (1992) suggested the first *subexponential* randomized simplex algorithm for linear programming. Based on Kalai's ideas, Ludwig (1995) suggested the first *subexponential* randomized algorithm for simple stochastic games. 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 $\mathcal{H}(n)$.
2. Choose at random a coordinate $i \in [1..n]$ (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.

2.3 Useful Lemmas

Lemma 2.1 *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 (Cormen, Leiserson & Rivest 1992, p. 122). $\square$

Lemma 2.2 *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 $n!/(k!(n-k)!)$. $\square$

3 The Random Multiple Switches Algorithm

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 generated by Klee-Minty's polytopes. If extremely unlucky, the RSSA may also make an exponential number of steps, 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) (Williamson Hoke 1988, 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, Williamson Hoke (1988, 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 pseudoboolean and all unimin functions[2] 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 (Tovey 1997, 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)[3] 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 3.1 *The expected target function value improvement of the RMSA step from a vertex v with $i > 0$ improving directions is at least 2^{i-1}.*

[2]Possessing a unique minimum

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

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 $2^i/2 = 2^{i-1}$. $\square$

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 non-trivial lower bound on the number of improving directions in any run of the RMSA? Fortunately, the complete unimodality property (every possible vector of improving directions is present exactly once (Williamson Hoke 1988, 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 3.2 *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 3.1, 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[4]. If the RMSA visits all vertices with $\leq k$ improving directions, then the 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:

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

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

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

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

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

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 2.1,

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

(using Stirling approximation from Lemma 2.2)

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

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

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 2.2 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[5]. Here we recognize our expression R. If c is chosen such that $\log R + c > 1$ we get:

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

[5]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 $cn = \lfloor cn \rfloor = \lceil cn \rceil$.

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

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.

$\square$

4 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.

Proposition 4.1 *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).*

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 4.2 *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.*

Remark. 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 (Tovey 1997, Thm 23, p. 72). □

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. $\square$

5 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 3. 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 5.1 *The RMSA-RS can be parameterized in 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 3, it can be shown that (4) can be satisfied by selecting any $0 < c \leq 0.11$. As in Section 3, 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 3; 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. $\square$

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.

6 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 6.1 *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[6] 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

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

[6]Is expected to make, in the case of the RSSA.

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 6.2 *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 $O(2^{\frac{n}{2}-(\frac{1}{2}-\varepsilon)\log n})$.*

7 Kalai-Ludwig's Algorithm for CUPBFs

We were the first to observe (Björklund et al. 2001) that the Kalai-Ludwig's Randomized Algorithm (KLRA) initially designed for LINEAR PROGRAMMING (Kalai 1992) and later adapted for SIMPLE STOCHASTIC GAMES (Ludwig 1995), works perfectly for the CUPBF optimization, also providing *subexponential* expected running time. Modified to the CUPBF optimization the KLRA is as follows:

KLRA(CUPBF H, initial vertex v_0)
(1) **if** $dim(H) = 0$
(2)   **return** the only vertex of H
(3) $s \leftarrow$ randomly chosen dimension of H
(4) $F \leftarrow$ the facet of H obtained by fixing dimension s to 0 or 1 so that $v_0 \in F$
(5) $v \leftarrow$ KLRA(F, v_0)
(6) $\overline{F} \leftarrow H - F$
(7) $u \leftarrow$ the neighbor of v in $\overline{F}$
(8) **if** $value(u) < value(v)$
(9)   **return** v
(10) **else**
(11)   **return** KLRA($\overline{F}$, u)

It turns out that the algorithm is correct and terminating:

Theorem 7.1 *The above algorithm, when run on a hypercube representing a CUPBF, terminates and returns the vertex of the cube with the maximum value.*

Proof. Recursive calls are only made with hypercubes of smaller dimension. Therefore the algorithm always terminates.

We prove that it finds the maximal-valued vertex by induction on the number of dimensions d. If $d = 0$ there is only vertex, and it must have the maximal value. Suppose that the algorithm returns the correct vertex for all cubes of smaller dimensions than d. Then the recursive call on line 6 will return the maximal-valued vertex v on the chosen facet F. If v has a bigger value than its neighbor on the other facet $\overline{F}$, it is returned. Since all its neighbors have smaller values, and the pseudo-boolean function that defines the cube is known to be completely unimodal, it must also be the global maximum.

If v has a smaller value than its neighbor on $\overline{F}$, the global maximum must be located on $\overline{F}$, since v is, by inductive assumption, the maximum on F. In this case we call the algorithm on $\overline{F}$, and it will, again by inductive assumption, return the maximum on $\overline{F}$, which is also the global maximum. □

The following is a modification of the theorem and proof by Ludwig (1995).

Theorem 7.2 *The expected running time of the above algorithm is subexponential in the number of dimensions of a CUPBF.*

Proof. Everything in the algorithm, except the recursive calls, takes polynomial time. The key issue is the number of recursive calls in line 11, 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

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

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. □

8 CUPBF Generation

In (Björklund et al. 2001) we used random CUPBF generation based on the following idea. Take a cube and assign the maximum value 2^n to one of its vertices. The preceding biggest value $2^n - 1$ should be a random neighbor of the maximum (to avoid local maxima). In general, if the k biggest values are already assigned to vertices V', the biggest remaining value $2^n - k$ should be assigned to a (random) neighbor v of some vertex in V'. Moreover, in every subcube containing v, v is either the first vertex assigned a value, or has a neighbor from V'. This algorithm was able to generate random CUPBFs of dimension up to 16. In this paper we describe a different, more powerful method, allowing us to go up to 27 dimensions with currently available memory.

Let $f : \{0,1\}^n \to \mathbb{R}$ be a CUPBF. Associate to each vertex v of the cube $\{0,1\}^n$ the vector of improving directions (VID), as described in Section 2, and call the result the *VID-function* $\hat{f} : \{0,1\}^n \to \{0,1\}^n$ associated to f.

Our CUPBF generation is based on the following remarkable theorem.

Theorem 8.1 (Williamson Hoke (1988)) *For any CUPBF f its associated VID-function $\hat{f}$ is* bijective*.* □

We use it in the 'opposite' way (see Theorem 8.2 below) to generate CUPBFs. Take a bijective function $\hat{f} : \{0,1\}^n \to \{0,1\}^n$. In general, there may be no CUPBF f for which $\hat{f}$ is the associated VID-function (there are examples even in 3 dimensions). It turns out that acyclicity is needed. Let the binary 'improvement' relation I on $\{0,1\}^n$ generated by $\hat{f}$ be defined as follows: $I(u,v)$ iff u, v differ only in their i-th coordinate (they are neighbors) and the i-th bit of $\hat{f}(u)$ is 1. We have the following inverse of Theorem 8.1:

Theorem 8.2 *If $\hat{f}$ is bijective and the improvement relation I generated by $\hat{f}$ is acyclic then $\hat{f}$ is the associated VID-function for some CUPBF f* [7]*.* □

Note that it follows that if u and v differ only in the i-th coordinate, then the i-th bit of $\hat{f}(u)$ is 0 iff the i-th bit of $\hat{f}(v)$ is 1. In particular, exactly one of $I(u,v)$ and $I(v,u)$ holds for all neighbors u and v.

This allows for a recursive generation. We take two $(n-1)$-dimensional VID-functions $\hat{f}_0$, $\hat{f}_1$, each being injective and having acyclic improvement relation. From these we create an n-dimensional VID-function $\hat{f}$ as follows: for every $x \in \{0,1\}^{n-1}$ define $\hat{f}(0x) = b_0(x)\hat{f}_0(x)$, $\hat{f}(1x) = b_1(x)\hat{f}_1(x)$ for

[7] The latter can be obtained by topologically sorting I.

some bits $b_{0,1}(x)$ that keep $\hat{f}$ satisfying the premises of Theorem 8.2. Selecting the bits $b_{0,1}(x)$ corresponds to connecting the respective vertices of $(n-1)$-dimensional cubes by directed edges. Call these new edges *connecting edges*.

The comment just after the theorem implies that $b_0(x) = 1 - b_1(x)$, reducing the set of feasible solutions. In the following subsections we describe other methods for reducing this set even further.

8.1 Classes of Dependent Edges

We can group connecting edges together in classes so that all edges in the same class must be given the same direction.

Define the relation R on the connecting edges by: $R((0x, 1x), (0y, 1y))$ iff $\hat{f}_0(x) = \hat{f}_1(y) \vee \hat{f}_1(x) = \hat{f}_0(y)$. To provide for bijectivity of the $\hat{f}$ constructed, we need to guarantee that $\hat{f}_0(x) = \hat{f}_1(y) \Rightarrow b_0(x) \neq b_1(y)$. Or, equivalently, $\hat{f}_0(x) = \hat{f}_1(y) \Rightarrow b_i(x) = b_i(y)$. Therefore, in every equivalence class C of R^+ (the transitive closure of R, which in this case can be proved to be an equivalence relation) we have to assign the same bit $b_0(u)$ to all vertices $0u$ such that $(0u, 1u) \in C$ and assign the opposite bit to $1u$. This reduces the number of feasible solutions from being exponential in the number of vertices to being exponential in the number of equivalence classes under R^+.

We can reduce the number of solutions further. Take $u, v \in \{0,1\}^{n-1}$ so that $I(0u, 0v)$ and $I(1v, 1u)$ (i.e., the edges between u and v are oppositely directed in $\hat{f}_0$ and $\hat{f}_1$). Then we need to guarantee that $b_i(u) = b_i(v)$. If we did not, the two-dimensional face consisting of $0u, 0v, 1u, 1v$ would not be completely unimodal: either it would have two local minima and two local maxima, or it would be cyclic. Thus, if there are two equivalence classes $[(0u, 1u)]$ and $[(0v, 1v)]$ of R^+, such that $I(0u, 0v)$ and $I(1v, 1u)$ (for some choice of the representatives), then the two classes can be merged. This merging process is continued until no pair of edges belonging to different classes satisfies the above condition. The classes in the resulting partitioning will be referred to as *classes of dependent edges* or simply *classes* for brevity.

In our experiments, combining edges into classes using R^+ has never yielded more than 100 classes. The second stage of merging the classes reduces this worst observed case to 31. This is important, because the worst case running time of the overall algorithm is exponential in the number of classes, although this worst case has never been observed in practice. This behavior was observed after generating over twenty 26-dimensional and many more lower-dimensional cubes.

8.2 Orienting Classes

We now present an algorithm that searches through assignments of directions to classes and finds one that yields a VID-function. Moreover, this assignment is non-absorbing if such an assignment exists (recall that an assignment is absorbing if all connecting edges point in the same direction). We will only assign directions to classes, implicitly meaning that all edges in the class are directed that way.

The recursive backtracking algorithm FIND-ASSIGNMENT takes two parameters: an assignment A of a direction to each class and a set S of classes. Call an assignment that is non-absorbing, does not yield cycles, and may differ from A only in the direction of classes contained in S *a valid (A,S)-assignment*. FIND-ASSIGNMENT searches for a valid (A,S)-assignment, i.e., it is only allowed to change directions of the classes in S; all other classes must remain directed the same way as in A. It returns SUCCESS if it finds a valid (A,S)-assignment. If no valid (A,S)-assignment exists, it returns FAILURE.

8.2.1 Invoking Find-Assignment

FIND-ASSIGNMENT is initially invoked the following way. Let c be a class, taken uniformly at random. Take A to be the orientation where all classes are directed the same (random) way, except c, directed the opposite way. Assign S the set of all classes except c. If the first call to FIND-ASSIGNMENT(A, S) returns FAILURE, it means that there is no non-absorbing acyclic assignment where c has the given direction. But there may still be one where c goes the opposite way. So we invert the directions of all classes and try again. If the second call also returns FAILURE, it means that all acyclic assignments are absorbing, so we pick one such randomly. The top-level function that generates a VID-function by calling FIND-ASSIGNMENT is listed in Algorithm 1.

Algorithm 1: Generate-Cube
Input: An integer n.
Output: An n-dimensional CUPBF.
Generate-Cube(n)
(1) **if** $n = 1$
(2)   **return** Either of the two possible one-dimensional CUPBFs
(3) $\hat{f}_0 \leftarrow$ Generate-Cube($n-1$)
(4) $\hat{f}_1 \leftarrow$ Generate-Cube($n-1$)
(5) Generate-Classes($\hat{f}_0$, $\hat{f}_1$)
(6) $A \leftarrow$ An assignment of the same (random) direction to all classes
(7) $S \leftarrow$ The set of all classes
(8) $c \leftarrow$ A class taken uniformly at random
(9) A.flip(c)
(10) $S \leftarrow S \setminus \{c\}$
(11) **if** Find-Assignment(A, S) = failure
(12)   Flip all classes in A
(13)   **if** Find-Assignment(A, S) = failure
(14)   **return** $\hat{f}_0$ and $\hat{f}_1$ glued together with an absorbing assignment
(15) **return** $\hat{f}_0$ and $\hat{f}_1$ glued together with the assignment produced by the previous successful call to Find-Assignment

The function Generate-Classes called in line 5 of Algorithm 1 is not listed. It can be efficiently implemented using standard algorithms for finding strongly connected components.

8.2.2 Find-Assignment

Now we proceed to describing how Find-Assignment is implemented. Call the two directions in which classes can be oriented *left* and *right*. Assume without loss of generality that the single class initially oriented oppositely to the others is oriented left. If A is acyclic and non-absorbing, then A itself is a valid (A,S)-assignment, so we save A and return success.

Assume A is absorbing. It will be guaranteed that S contains only right-going edges and that recursive calls are 'consistent' with A and S, meaning that no elements are added to S and the direction of a class is only changed if it is a member of S. (See the proof of Theorem 8.3 below.) So all edges in the absorbing assignment must be left-going since there is initially a left-going edge not in S. It follows that S is empty and there is no valid (A,S)-assignment, so we return FAILURE.

It remains to describe what FIND-ASSIGNMENT does if A is non-absorbing but there is a cycle. There may be many cycles, but pick one arbitrarily. We must break this cycle by flipping a class containing a connecting edge on the cycle. If all classes containing connecting edges are in the complement of S, it follows that there is no valid (A,S)-assignment, so we return FAILURE. Otherwise, let c be a class in S containing a connecting edge on the cycle. There are two kinds of valid (A,S)-assignments: those where c is flipped with respect to A and those where it is intact. We try both possibilities if we have to. Let $S' = S \setminus \{c\}$ and $A' = A$ with c flipped. Then call FIND-ASSIGNMENT recursively with (A', S') as parameters. If it fails, call FIND-ASSIGNMENT again with (A, S') as parameters. If one of these calls returns SUCCESS then we return SUCCESS too, otherwise we return FAILURE.

The function FIND-ASSIGNMENT is listed in Algorithm 2.

Algorithm 2: Find-Assignment
Input: An assignment A of a direction to each class and a set S of classes.
Output: If there exists a valid (A,S)-assignment, return success and save the assignment in a global variable. Otherwise, return failure.

Find-Assignment(A, S)
(1) **if** A is absorbing
(2)   **return** failure
(3) Check if there is a cycle when orientation A is used
(4) **if** there are no cycles
(5)   Save A
(6)   **return** success
(7) **if** no class in S contains a connecting edge on the cycle
(8)   **return** failure
(9) $c \leftarrow$ A class in S containing a connecting edge on the cycle
(10) $S' \leftarrow S \setminus \{c\}$
(11) $A' \leftarrow A$ with c flipped
(12) **if** Find-Assignment(A', S') $=$ success
(13)   **return** success
(14) **return** Find-Assignment(A, S')

The correctness of Find-Assignment can be proved by induction on $|S|$, the number of elements of S.

Theorem 8.3 *If there exists a valid (A,S)-assignment then Algorithm 2, when invoked from Algorithm 1, finds one and returns* success. *If not, it returns* failure.

Proof. First note that all edges in S are right-going since this property holds the first time when Find-Assignment is called and is preserved in the recursive calls. All recursive calls preserve the orientation in A of the classes in S. No elements are ever added to S, only removed.

The proof now proceeds by induction on $|S|$, the number of elements of S. Assume $|S| = k$ and the algorithm is correct whenever $|S| < k$.

If A is absorbing, then S must be empty, so no valid (A,S)-assignment exists and it is correct to return failure on line 2. Otherwise, if there are no cycles, then A is itself a valid (A,S)-assignment so the algorithm correctly returns SUCCESS on line 6. Otherwise, if no class in S contains a connecting edge on the cycle, then any assignment that coincides with A on all edges not in S must be cyclic. So the algorithm is correct when it returns FAILURE on line 8.

Now assume that A is non-absorbing, there is a cycle, and there is a class in S that contains a connecting edge on the cycle. The recursive calls are correct by the induction hypothesis, and they try all possibilities to find a valid (A,S)-assignment (namely, with c flipped and with c intact). So the algorithm is correct. □

8.3 Further Improvements

In our implementation, we also employ several heuristics in order to improve randomness and speed. The algorithm described above generates coordinates in order. Since the i-th bits of $\hat{f}_0$ and $\hat{f}_1$ were generated at the same depth of recursion, one might suspect some kind of dependency between them. To avoid this, we randomly permute the order of coordinates of each generated cube. This does not affect complete unimodality since the definition does not depend on the order of coordinates.

We use depth-first search to look for cycles. Let $L \subseteq \{0,1\}^n$ be the set of vertices in the right cube that belong to right-going classes, and let $R \subseteq \{0,1\}^n$ be the set of vertices in the left cube that belong to left-going classes. Recall that the 'standard' implementation of a cycle tester using depth-first search loops through all vertices, taking each as a starting point for the DFS-VISIT function. We have an advantage over the general cycle-testing algorithm when there are no assumptions on the graph, since we know that both $\hat{f}_0$ and $\hat{f}_1$ are acyclic. So any cycle must traverse at least one connecting edge from $\hat{f}_0$ to $\hat{f}_1$ and equally many the opposite direction. In particular it must pass through some vertex in L and some vertex in R. So we calculate which of L and R is smallest, and start only from vertices in this smallest set.

Another improvement is based on the following property (Hammer et al. 1988, Theorem 3.6).

Theorem 8.4 *If f is a CUPBF then for any vertex v there is a path in the graph induced by the improvement relation I from v to the maximum of f. Similarly, there is a path from the minimum of f to v.* □

Let $0v$ be the vertex such that $\hat{f}_0(v) = 00\ldots0$, that is, v is the global optimum of $\hat{f}_0(v)$. Assume $b_0(v) = 1$, i.e., the connecting edge goes from

$0v$ to $1v$. Let $1u$ be a vertex reachable from $1v$. If $b_0(u) = 0$ then it would give a cycle: $0u \rightsquigarrow$ (by the theorem) $0v \rightsquigarrow 1v \rightsquigarrow 1u \rightsquigarrow 0u$. So any class containing an edge incident to a vertex $1u$ reachable from $1v$ must be oriented the same way as the class containing the edge $(0v, 1v)$. Note that this only holds under the assumption that $b_0(v) = 1$, so it does not give a method to merge more classes. To employ this property we could instead pre-calculate the set of classes reachable from the vertex next to the optimum and augment FIND-ASSIGNMENT so that it takes this into account. The same reasoning can be applied to both the maxima and the minima in both of the $(n-1)$-dimensional sub-cubes. It can also be applied to optima in lower-dimensional sub-cubes. Our current implementation does not use this heuristic.

Although the suggested scheme was designed to create possibly hard instances of CUPBFs without giving advantages to any algorithms, it nevertheless revealed considerable differences in their worst-case running times. A generally good behavior of the algorithms on the generated cubes may be explained in several ways, in particular, by the polynomial time conjecture for CUPBFs.

9 Experimental Results

In (Björklund et al. 2001) we investigated the iterative improvement algorithms presented in Section 2 experimentally. We were only able to experiment with CUPBFs of up to 16 dimensions, but the results were promising. In particular, the fact that all other algorithms clearly outperformed the provably subexponential KLRA was encouraging. As is clear from previous sections, the KLRA is still the algorithm with the best known upper bound. In this section we present new experimental data. Thanks to new algorithms for generating test instances (described in Section 8), the algorithms could be tested on CUPBFs of up to 27 dimensions. This new data reinforces the impression that on most CUPBFs, the KLRA is actually the worst of the investigated algorithms, and the hope that for some of the algorithms much stronger upper bounds than those settled in this paper might be provable. Some caution is, however, appropriate when interpreting the experimental results, since the GSSA, which is exponential in the worst case, is among the algorithms exhibiting a very nice behavior.

9.1 Worst Exhibited Behaviors

All the algorithms were run on a number CUPBFs of dimensions 3 to 27, generated randomly as described in Section 8. On each cube, the algorithms were run starting from every vertex. The randomized algorithms were run four times from each vertex on the cubes of dimensions 3 through 20, twice from each vertex for dimensions 21 through 24 and 27, and once from each vertex for dimensions 25 and 26. The maximal number of iterations for each algorithm on each cube, as well as the average over the different starting points on all cubes were gathered.

In our experiments, the deterministic algorithms outperform the randomized ones. This is to be expected, since the main motivation for randomization lies in evening out the running time over the set of all inputs. It is, however, somewhat surprising that the GSSA is the algorithm with the best exhibited behavior (together with the APSA). This to some extent contradicts the results presented in (Björklund et al. 2001), which indicated that both the multiple switching algorithms (the APSA and the RMSA) performed better than all the single switching algorithms. This could be explained by different methods of random CUPBF generation.

For exact data on the average and worst exhibited behaviors of the algorithms, see Figure 1 and Figure 2, respectively. Figure 3 and Figure 4 show the same results for the three fastest algorithms separately. They show the average and worst numbers of iterations over all vertices of all generated CUPBFs. The general picture is that the APSA and the GSSA showed the best performance, followed by the RMSA. The RSSA performed significantly worse, while the KLRA has a worst exhibited behavior that is very much worse than that of all the other algorithms.

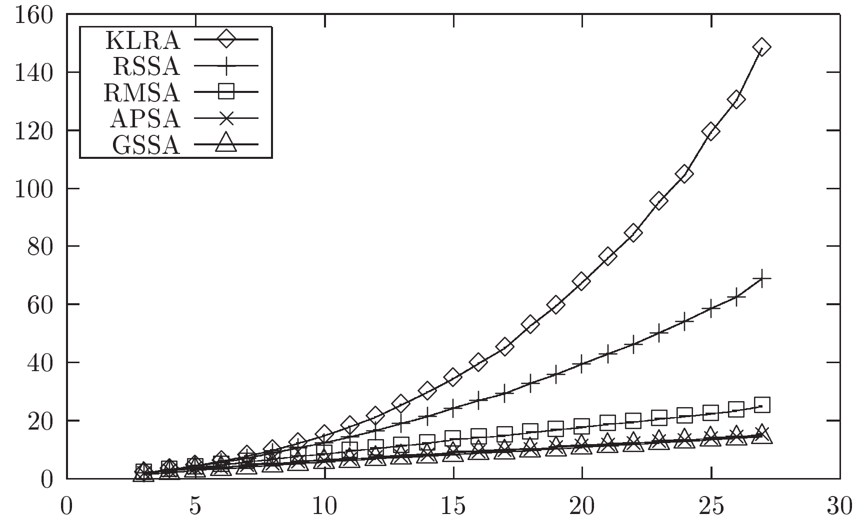

Figure 1: Average numbers of iterations for five algorithms when starting from each corner of cubes of dimensions between 3 and 27. For dimensions 3 through 24, 50 to 60 cubes of each dimension were generated. For dimensions 25 and 26 the numbers of cubes are 19 and 18, respectively, and for dimension 27 it is 3.

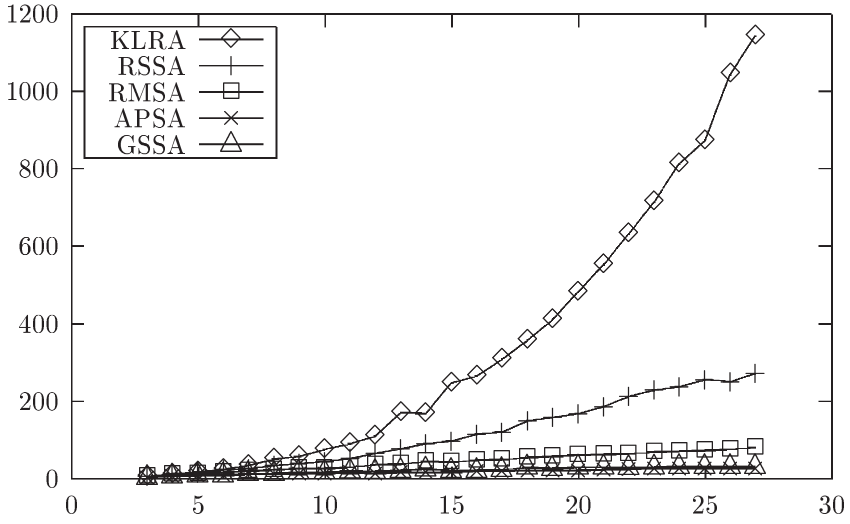

Figure 2: Worst numbers of iterations for each algorithm in the same experiments as in Figure 1.

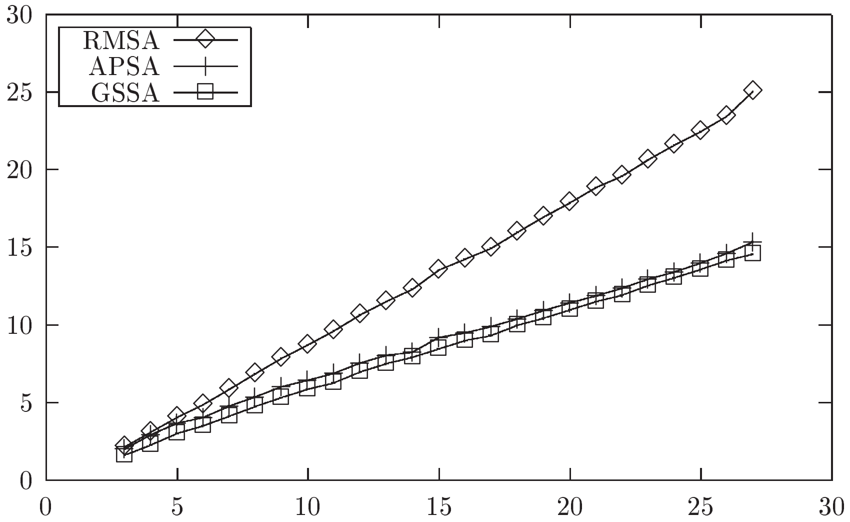

Figure 3: Average numbers of iterations for the three fastest algorithms in the same experiments as in Figure 1.

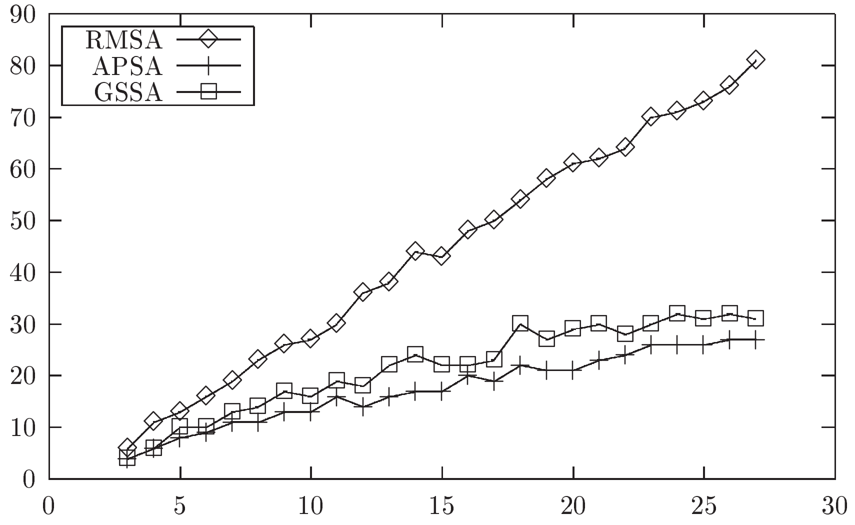

Figure 4: Worst numbers of iterations for the three fastest algorithms in the same experiments as in Figure 1.

10 Conclusions

1. We presented improved upper bounds for a number of old and new optimization algorithms for the CUPBF optimization. In particular, we were able to give better than previously known upper bounds for the Greedy and Random Single Switch local search algorithms. We also suggested and provided upper bounds for Multiple Switches Algorithms (All Profitable and Randomized), which work only for the CUPBF optimization. It turns out that the Randomized Multiple Switches Algorithms gives the best known upper bound $O(2^{0.775n}) = O(1.71^n)$ among all switching algorithms.

2. By combining Randomized Multiple Switching with random sampling we improved the upper bound for the RMS Algorithm to $O(2^{0.46n}) = O(1.376^n)$.

3. We have also shown that Kalai-Ludwig's randomized algorithm can be adapted for the CUPBF optimizations and has expected subexponential running time.

4. To test and compare the practical behavior of different algorithms we suggested a new method to generate (presumably difficult) random CUPBFs instances.

5. Our experiments show that all algorithms easily optimize the generated CUPBFs, although there are considerable differences in their running times.

6. Although the Kalai-Ludwig's algorithm has the best currently known upper bound, its practical behavior turns out to be the worst in our experiments. We stress that the CUPBF generation was not intended to create data easy for some and difficult for the other algorithms.

7. Further research will be concentrated on improving the upper bounds for different switching algorithms. We feel there is a big potential source of improvement due to the fact that intuitively these algorithms provide for a much bigger average stepwise improvement compared to our worst case considerations, which apparently do not realize themselves for any CUPBFs.

References

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/`.

Cormen, T. H., Leiserson, C. E. & Rivest, R. L. (1992), *Introduction to Algorithms*, The MIT electrical engineering and computer science series, 6th printing edn, MIT Press, Cambridge, MA.

Hammer, P. L., Simeone, B., Liebling, T. M. & De Werra, D. (1988), 'From linear separability to unimodality: a hierarchy of pseudo-boolean functions', *SIAM J. Disc. Math.* **1**(2), 174–184.

Kalai, G. (1992), A subexponential randomized simplex algorithm, *in* '24th ACM STOC', pp. 475–482.

Ludwig, W. (1995), 'A subexponential randomized algorithm for the simple stochastic game problem', *Information and Computation* **117**, 151–155.

Tovey, C. A. (1997), Local improvement on discrete structures, *in* E. Aarts & L. J. K., eds, 'Local Search in Combinatorial Optimization', John Wiley & Sons, pp. 57–89.

Williamson Hoke, K. (1988), 'Completely unimodal numberings of a simple polytope', *Discrete Applied Mathematics* **20**, 69–81.

Recent technical reports from the Department of Information Technology

2001-029	Henrik Lundgren, David Lundberg, Johan Nielsen, Erik Nordström and Christian Tschudin: *A Large-scale Testbed for Reproducible Ad hoc Protocol Evaluations*
2001-030	Jakob Engblom, Andreas Ermedahl and Friedhelm Stappert: *Validating a Worst-Case Execution Time Analysis Method for an Embedded Processor*
2002-001	Sônia M. Gomes and Bertil Gustafsson: *Combining Wavelets with Finite Differences: Consistency Analysis*
2002-002	S. Seipel and M. Lindkvist: *Methods and application of interactive 3D computer graphics in anthropology*
2002-003	Ken Mattsson, Magnus Svärd and Jan Nordström: *Stable and accurate artificial dissipation*
2002-004	Mats Löfberg, Jan Gulliksen and Torsten Norlander: *Användare i systemutvecklingsprocessen - ett fenomenologiskt perspektiv*
2002-005	Kajsa Ljungberg, Sverker Holmgren and Örjan Carlborg: *Efficient Kernel Algorithms for QTL Mapping Problems*
2002-006	Anders Berglund: *How do students understand network protocols? A phenomenographic study*
2002-007	F. Edelvik, G. Ledfelt, P. Lötstedt and D. J. Riley: *An Unconditionally Stable Subcell Model for Arbitrarily Oriented Thin Wires in the FETD Method*
2002-008	Henrik Björklund and Sergei Vorobyov: *Two Adversary Lower Bounds for Parity Games*
2002-009	Andy Bavier, Thiemo Voigt, Mike Wawrzoniak, Larry Peterson and Per Gunningberg: *SILK: Scout Paths in the Linux Kernel*
2002-010	Jonas Boustedt: *Automated Analysis of Dynamic Web Services*
2002-011	Sheayun Lee, Andreas Ermedahl, Sang Lyul Min and Naehyuck Chang: *Statistical Derivation of an Accurate Energy Consumption Model for Embedded Processors*
2002-012	Lars Ferm and Per Lötstedt: *Accurate and Stable Grid Interfaces for Finite Volume Methods*
2002-013	Jakob Engblom: *Effects of Branch Predictors on Execution Time*
2002-014	Samuel Sundberg and Lina von Sydow: *Analysis of a semi-Toeplitz preconditioner for a convection-diffusion problem*
2002-015	Andreas Johnsson: *Differentiated Security in Wireless Networks*
2002-016	Fredrik Edelvik: *A New Technique for Accurate and Stable Modeling of Arbitrarily Oriented Thin Wires in the FDTD Method*
2002-017	Eva Olsson: *Safer Navigation at Sea Through Augmented Reality? An introductory study*
2002-018	Henrik Björklund, Sven Sandberg and Sergei Vorobyov: *Optimization on Completely Unimodal Hypercubes*

UPPSALA
UNIVERSITET

May 2002
ISSN 1404-3203
http://www.it.uu.se/