INFORMATIK

Experiments with Iterative Improvement Algorithms on Completely Unimodal Hypercubes

Henrik Björklund, Viktor Petersson, and Sergei Vorobyov

MPI–I–2001–02–003 July 2001

FORSCHUNGSBERICHT ■ RESEARCH REPORT

MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

Im Stadtwald ■ 66123 Saarbrücken ■ Germany

The *Max-Planck-Institut für Informatik* in Saarbrücken is an institute of the *Max-Planck-Gesellschaft*, Germany.

ISSN: 0946 - 011X

Forschungsberichte des

Max-Planck-Instituts für Informatik

Further copies of this report are available from:

Max-Planck-Institut für Informatik
Bibliothek & Dokumentation
Im Stadtwald
66123 Saarbrücken
Germany

INFORMATIK

Experiments with Iterative Improvement Algorithms on Completely Unimodal Hypercubes

Henrik Björklund, Viktor Petersson, and Sergei Vorobyov

MPI-I-2001-02-003 July 2001

FORSCHUNGSBERICHT RESEARCH REPORT

MAX-PLANCK-INSTITUT FÜR INFORMATIK

Stuhlsatzenhausweg 85 66123 Saarbrücken Germany

Author's Address

Information Technology/Computing Science Department
Uppsala University, Box 337
751 05 Uppsala, SWEDEN

Publication Notes

Work done at the Max-Planck Institut für Informatik, Saarbrücken, Germany, July 2001.

Supported by the Swedish TFR Grant 221-2000-103 "Games for Computer-Aided Verification: Algorithms and Complexity".

Acknowledgements

We thank Harald Ganzinger and the Programming Logics Group for their hospitality. Special thanks to Leonid Khachiyan, Fritz Eisenbrand, and E. Allen Emerson for inspiring discussions.

The first author gratefully acknowledges the support from "Föreningen för datateknisk forskning i Uppsala".

Abstract

Completely unimodal (i.e., having a unique local minimum on every face) numberings of many-dimensional hypercubes are abstract versions of different optimization problems, like linear programming, decision problems for games, and abstract optimization problems. In this paper we investigate and compare the behaviors of seven iterative improvement algorithms:

1) the Greedy Single Switch Algorithm (GSSA),
2) the Random Single Switch Algorithm (RSSA),
3) the All Profitable Switches Algorithm (APSA),
4) the Random Multiple Switches Algorithm (RMSA),
5) Kalai-Ludwig's Randomized Algorithm (KLRA),
6) Weighted Random Multiple Switch Algorithm (WRMSA),
7) Weighted Greedy Multiple Switch Algorithm (WGMSA).

Our experiments were conducted on all completely unimodal four-dimensional hypercubes and on randomly generated hypercubes of dimensions up to sixteen, Hamiltonian (presumably corresponding to hard problem instances) and non-Hamiltonian.

Local-search improvement algorithms 1 and 2 have been investigated earlier, but number 3, 4, 5, 6, and 7 probably not. Algorithm 5 (first time used for completely unimodal hypercubes in this paper) is the only algorithm with the known *subexponential* expected worst-case running time. However, the algorithms 1, 3, 4, 6, 7 demonstrate superior behaviors compared to the other two investigated algorithms. This suggests that further theoretical and experimental studies of these algorithms should be carried out.

Keywords

Pseudo-Boolean functions, optimization, completely unimodal, local search, algorithms, complexity.

Contents

1 Introduction **2**

2 Completely Unimodal Pseudo-Boolean Functions **3**

3 Seven Iterative Improvement Algorithms **5**
- 3.1 Greedy Single Switch Algorithm (GSSA) 6
- 3.2 Random Single Switch Algorithm (RSSA) 6
- 3.3 All Profitable Switches Algorithm (APSA) 6
- 3.4 Random Multiple Switches Algorithm (RMSA) 7
- 3.5 Kalai-Ludwig's Randomized Algorithm (KLRA) 7
- 3.6 Weighted Random Multiple Switch Algorithm (WRMSA) 8
- 3.7 Weighted Greedy Multiple Switch Algorithm (WGMSA) 9

4 Generation of Completely Unimodal Hypercubes **11**
- 4.1 Exhaustive in Four Dimensions 11
- 4.2 Randomized in Arbitrary Many Dimensions 11
- 4.3 Randomized with Hamiltonian Path or Similar Properties 12

5 Experimental Data **13**
- 5.1 Exhaustive Tests on Four-Dimensional Hypercubes 13
- 5.2 Tests on Random Instances of Higher-Dimensional Cubes . . . 17
- 5.3 Hamiltonian Data 21

6 Conclusions **24**
- 6.1 Discussion of the Experimental Results 24
- 6.2 Directions for Future Research 24

1 Introduction

Pseudo-Boolean function optimization is a fundamental core of many problems arising in constraint solving and mathematical optimization. Pseudo-Boolean functions are real-valued functions on 0-1 variables. Some specific classes of pseudo-Boolean functions are of particular interest in the context of studying combinatorial aspects of various local-search improvement algorithms, investigating combinatorial structures of polytopes, deciding games, solving linear programs, etc. A survey of pseudo-Boolean function optimization and applications is presented in (Hansen, Jaumard & Mathon 1993).

Our interest in optimizing pseudo-Boolean functions is motivated by their relevance to solving the so-called PARITY and SIMPLE STOCHASTIC GAMES. As became apparent from our investigation of the *interior-point approach* to solving games, (Petersson & Vorobyov 2001*b*, Petersson & Vorobyov 2001*a*), different variations of local-search-type iterative improvement algorithms can be investigated in a uniform way, considering them as abstract optimization methods for well-behaved pseudo-Boolean functions. Specific pseudo-Boolean functions arising as abstract representations of such games turn out to possess an extremely favorable property (absence of isolated local minima) rendering them especially appropriate for global optimization by iterative improvement. It turns out that the games we are interested in are adequately approximated by the so-called *completely unimodal pseudo-Boolean functions* well investigated in the literature (Hammer, Simeone, Liebling & De Werra 1988, Williamson Hoke 1988). Such functions possess *unique local optima* on every face of a Boolean hypercube, a behavior shared[1] by Simple Stochastic Games in which every locally optimal stationary strategy is globally optimal. Some proper subclasses of completely unimodal functions, like *completely absorbing, or decomposable* (see below), allow for efficient randomized quadratic-time optimization. However, hypercubes obtained from games are more general, *non-saddle free*, and are not known to be optimizable in polynomial time. Consequently, we need efficient optimization algorithms for general completely unimodal hypercubes. To our knowledge, no such polynomial time algorithms currently exist.

In this paper we collect, test, and compare several known and new local-search type iterative improvement algorithms on exhaustively and randomly generated completely unimodal hypercubes, with the purpose of obtaining experimental evidence and gaining new insights for the development and analysis of new more efficient algorithms. The main conclusion of our study is that our *multiple switches algorithms*, both *greedy* and *random* (APSA,

[1] with a minor difference that can be ignored (or eliminated)

RMSA, WRMSA, and WGMSA), previously not investigated in the context of optimization on completely unimodal hypercubes, clearly demonstrate a superior behavior when compared to the known *single switches* algorithms: *random*, and *Randomized Kalai-Ludwig's* (the last one being the only proven *subexponential* algorithm for the problem). The greedy single switch algorithm (GSSA) performs best on Hamiltonian hypercubes.

2 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 $\mathcal{H}(n) \rightarrow \mathbb{R}$ 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*. 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 $\mathcal{H}(n)$ [2].

A pseudo-Boolean function on $\mathcal{H}(n)$ is called *Hamiltonian* if there exists a function-improving Hamiltonian traversal of $\mathcal{H}(n)$.

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 $1, \ldots, 2^n$.

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

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

[2] Our Random Multiple Switches Algorithm uses neighborhoods, which may be exponentially large, in general.

4. f has a unique local maximum in each 2-face.

Optimizing (minimizing) completely unimodal pseudo-Boolean functions on $\mathcal{H}(n)$ is a challenging open problem in combinatorial optimization. No polynomial-time algorithms, deterministic, randomized, quantum, or other, are currently known for general CUPBFs.

3 Seven 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.

```
Select the initial point v0 on the hypercube arbitrarily.
Set i := 0 and vi := v0.
While ( N(vi) contains an element v' with f(v') < f(vi) ) do
   Set i  := i+1;
   Set vi := v';     /* next iterate */
od.
```

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 minimizing CUPBFs. They are described in Sections 3.1 and 3.2 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 3.3 and 3.4. 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 employ and test the well-known, but probably ignored (in the framework of CUPBF-optimization), *subexponential Kalai-Ludwig's Randomized Algorithm (KLRA)*. This algorithm is the only provably subexponential algorithm for the problem[3]. Interestingly, our experiments indicate that our multiple switch algorithms APSA and RMSA behave better than Kalai-Ludwig's (KLRA).

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

3.1 Greedy Single Switch Algorithm (GSSA)

This is a local-search algorithm that at every iteration chooses the *lowest-valued neighbor* of the current vertex as the next iterate. Recall that every vertex of $\mathcal{H}(n)$ has exactly n neighbors (in the standard neighborhood). This algorithm may take *exponentially many* steps to find the global minimum (vertex numbered 1) in the worst case; see (Williamson Hoke 1988, p. 77).

3.2 Random Single Switch Algorithm (RSSA)

This is a local-search algorithm that at every iteration chooses uniformly at random one of the *lower-valued neighbors* of the current vertex as the next iterate. This algorithm may also take *exponentially many* iterations to find the global minimum. 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 $(n-1)$-dimensional face) a *last facet* if no vertex on F has a lower-valued neighbor that is not on F. A hypercube is called *decomposable* iff it has dimension 1 or has a last facet that is decomposable.

3.3 All Profitable Switches Algorithm (APSA)

Let f be an arbitrary CUPBF. Associate to each vertex v of $\mathcal{H}(n)$ the n-dimensional Boolean *successor vector* with the i-th component defined by:

$$\begin{cases} 1, & \text{if the successor of } v \text{ in the } i\text{-th direction has a smaller } f\text{-value,} \\ 0 & \text{otherwise.} \end{cases}$$

Thus, the global minimum and maximum have the all-zeros and all-ones successor vectors, respectively.

The All Profitable Switches Algorithm (APSA) at every iteration computes the successor vector 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 $v' := v$ `XOR` s.)

This algorithm may also be seen as a local-search algorithm, but the structure of a 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 maximum on the face defined by fixing all coordinates corresponding to zeros in the successor vector s. Therefore, the next iterate v' (which belongs to the same face) has a smaller f-value.

3.4 Random Multiple Switches Algorithm (RMSA)

Like APSA, the Random Multiple Switches Algorithm (RMSA) at every iteration computes the successor vector 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 $v' := v$ `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 maximum on any face defined by fixing any superset of coordinates corresponding to zeros in the successor vector s. By the same argument as above the next iterate v' (which belongs to the same face) must have a smaller f-value by definition of CUPBFs. Note that in principle, RMSA selects at random from a neighborhood, which may be *exponential* in general. So, strictly speaking, this is not a local-search improvement algorithm.

3.5 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 observed that Ludwig's algorithm without any substantial modifications performs correctly and with the same expected worst-case complexity $2^{O(\sqrt{n})}$ for minimizing 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.

The correctness and termination of the algorithm are based on the fact that every switch (bit inversion) it makes is *profitable*, i.e., improves the target CUPBF. The recursion

$$t(d+1) \leq t(d) + \frac{1}{d} \cdot \sum_{i=1}^{d} t(i),$$

with $t(1) \leq 1$, for its expected running time allows for a subexponential solution $t(n) = 2^{O(\sqrt{n})}$. This is currently the only provably subexponential algorithm for the CUPBFs optimization. It should be noted, however, that in our experiments the multiple switches algorithms, APSA and RMSA showed better behavior; see Sections 5, 6.

The fragment of Java code below gives an impression how the algorithm is implemented. Notice that the implementation bears little resemblance with the explanation given above.

```
public BitSet KalaiLudwig (Cube cube) {
    BitSet dir, pos = new BitSet(dim); // Starts in (0,...,0)
    int steps = 0, i, j, t;
    double value;
    Random rr = new Random();
    int [] ord = new int [dim = cube.getDimension()];

    for (i=0; i<dim; i++) ord[i]=i;
    for (i=dim-1; i>0; i--) { // Random permutation of directions
        t = ord[j = rr.nextInt(i)]; ord[j] = ord[i]; ord[i] = t;
    };
    while(true){
        value = cube.getValue(pos);
        dir = cube.getImproving(pos);
        if(dir.length()==0) break;  // optimum found
        for (i=0; i<dim; i++)
            if (dir.get(j = ord[i])){ // Is improving? Switch!
                if(pos.get(j)) pos.clear(j); else pos.set(j);
                steps++;
                break; // Proceed to the next iteration
            }
    }
    return pos;
}
```

3.6 Weighted Random Multiple Switch Algorithm (WRMSA)

This algorithm is a variant of the RMSA described in Section 3.4. The major difference is that:

- unlike the RMSA, which at every iteration randomly selects a nonempty subset of improving directions for a (multiple) switch to obtain the next iterate,

- the WRMSA selects directions to switch (among improving ones) based on the following randomized weighted heuristic.

  An improving direction d is included into the set of switches iff

  $$F(\zeta) \cdot i_{\max} < i_d,$$

  where $i_{\max}$ is the *maximum guaranteed* improvement provided by any single switch at the current iteration, i_d is an improvement guaranteed by a single switch in direction d, ζ is a random variable uniformly distributed on $[0,1]$, and $F : [0,1] \to [0,1]$ is a function 'transforming' a uniform distribution.

The WRMSA is in fact a generic algorithm, parameterized by a choice of the uniform distribution transformation F. Our experiments revealed that with $F = F(x) = x^2$ the algorithm demonstrates a good behavior on all randomly generated CUPBFs (see Section 5). The intuitive explanation is that the WRMSA combines (by randomization) the best features of the GSSA and the APSA. Experimental and comparison data for WRMSA and other algorithms is contained in Section 5.

3.7 Weighted Greedy Multiple Switch Algorithm (WGMSA)

This algorithm is a variant of the APSA described in Section 3.3. The main difference is that:

- unlike the APSA, which at every iteration selects *all* improving directions to make a simultaneous multiple switch to obtain the next iterate,

- the WGMSA selects among most profitable directions, some fraction of the most promising, to make a multiple switch.

  More explicitly, suppose $d_1, \ldots, d_p$ are all the improving directions ordered in the decreasing values of their guaranteed improvement profits $i_1 \geq \cdots \geq i_p$, with $\sum_{j=1}^{p} i_j = M$. To make a multiple switch, the WGMSA selects the first $q \leq p$ directions with the smallest q satisfying $\sum_{j=1}^{q} i_j \geq \rho \cdot M$ for some $0 \leq \rho \leq 1$.

Obviously, for $\rho = 1$ we get the APSA. For $\rho = 0$ the WGMSA becomes the GSSA. When ρ is randomized, the WGMSA is intuitively a randomized algorithm between the GSSA and the APSA. It shows good performance for presumably hard Hamiltonian CUPBFs. See Section 5 for experimental and comparison data between WGMSA and other algorithms.

4 Generation of Completely Unimodal Hypercubes

To test the algorithms described in the previous section we need to generate completely unimodal hypercubes. In all the generated examples the vertices are given unique integer values between 1 and 2^n (where n is the number of dimensions). Two hypercubes are considered different if one or more vertices have different values (symmetrical hypercubes are considered different). The two major problems when generating such hypercubes are:

- There are 2^n vertices in any n-dimensional hypercube, so generating such a cube takes at least exponential time (in n).
- The total number of different n-dimensional hypercubes is $2^n!$, so generating them all and checking for complete unimodality is not an option. There seems to be no straightforward way to efficiently generate only the completely unimodal hypercubes.

Nevertheless, it is possible to generate test data when looking at a reasonable amount of dimensions, and our two main approaches are briefly described below.

4.1 Exhaustive in Four Dimensions

By avoiding trivially symmetrical assignments of the highest values (16, 15, 14, 13) to vertices, one can decrease the number of possible hypercubes in four dimensions to a level where an algorithm based on exhaustive search with dead-end pruning is feasible. The dead-end pruning consists in on-the-fly checking of all thus far completed 2-dimensional faces for non-unimodality. The only problem is that two of the eight different assignments of the highest values have fewer symmetrical cases than the rest, so our average results based on the hypercubes generated by this algorithm are slightly biased. This could be avoided by further experiments, but partial results show that the bias is insignificant.

4.2 Randomized in Arbitrary Many Dimensions

Even when many symmetrical assignments are avoided, as described above, more than $2 * 10^7$ different completely unimodal 4-dimensional hypercubes are generated. This number grows very rapidly when the number of dimensions increases (for three dimensions the corresponding number is 88) so an exhaustive search for completely unimodal 5+-dimensional hypercubes is not feasible. More dimensions are however needed to test the effectiveness of the different algorithms, so the problem is shifted to generating hypercubes uniformly at random. We do this by assigning the values to random vertices in decreasing order, while doing extensive checking at each assignment to allow all completely unimodal hypercubes while at the same time avoiding backtracking. When trying to assign a value to any vertex v, the following check is done to avoid local maxima:

1. Let F be the set of all faces of dimension ≥ 2 such that v belongs to the face.
2. Select some face $f \in F$ and remove f from F.
3. If some other vertex on f has been assigned a value previously and no vertex adjacent to v on f has been assigned a value then return false. (If we proceeded we would end up with two local maxima on f).
4. Return true if F is empty, and repeat Steps 2-3 otherwise.

The actual implementation is very different (and a lot more efficient) but the idea behind it is the same. It should finally be mentioned that the highest value (2^n) always is placed in the same vertex in order to allow the algorithms to start from it as explained in Section 5.

4.3 Randomized with Hamiltonian Path or Similar Properties

The randomized generation described above is sound and efficient, but we were also interested in producing 'difficult' hypercubes. Hypercubes with relatively long simple paths (Hamiltonian paths in the extreme case) are generally considered more 'difficult', so we added the following restriction to the generator:

- If a vertex is assigned some value m then it must have a neighbor with value between m and $m + p$, where p is a parameter of the generator.

If the parameter p is 1 then the hypercube gets a Hamiltonian path, and if $p \geq 2^n$ the generator is identical to the one in the previous section. The down-side is that (for small p) generation of many-dimensional hypercubes is currently intractable .

5 Experimental Data

The programs for generating cubes and running the algorithms on them were written in Java (JDK 1.3). They were run on a Linux workstation, but since no times, only the numbers of iterations, were measured, the performance of the computer has no bearing on the results. (Generating and testing 100.000 ten-dimensional completely unimodal hypercubes takes approximately an hour).

All the algorithms were started at the vertex with the highest value. This approach was chosen because of the connection between CUPBFs and games, in which the worst strategy is easily computable.

5.1 Exhaustive Tests on Four-Dimensional Hypercubes

All the algorithms presented in Section 3 were run on all four-dimensional completely unimodal hypercubes, generated as described in Section 4.

The full computer outputs are shown below. They are summarized in Table 1. Observe that the worst case behaviors given for the randomized algorithms (RSSA, RMSA, KLRA, and WRMSA) are not the expected behaviors in the worst case, but the worst number of iterations they showed on any instance in the experiment.

```
*** Kalai-Ludwig's (dim=4)
 Iter    Instances          Fraction
 1:      498038,            0.024865774
 2:      2529329,           0.12628299
 3:      5120367,           0.25564694
 4:      5567316,           0.27796197
 5:      3727023,           0.18608081
 6:      1744069,           0.08707695
 7:      633818,            0.031644925
 8:      168083,            0.008391958
 9:      34639,             0.0017294375
 10:     5674,              2.8328845E-4
 11:     641,               3.2003507E-5
 12:     56,                2.7959381E-6
 13:     2,                 9.985493E-8
 14:     1,                 4.9927465E-8
Average: 3.9165218
Sum: 20029056
*** Single Random (dim=4)
```

```
 Iter   Instances       Fraction
 1:     333363,         0.01664397
 2:     1911516,        0.09543715
 3:     4472552,        0.22330318
 4:     5643815,        0.28178138
 5:     4363543,        0.21786064
 6:     2234683,        0.11157206
 7:     799283,         0.039906174
 8:     217008,         0.010834659
 9:     44859,          0.0022396962
 10:    7419,           3.7041187E-4
 11:    925,            4.6182904E-5
 12:    87,             4.3436894E-6
 13:    3,              1.497824E-7
Average: 4.153733
Sum: 20029056
*** Single Greedy (dim=4)
 Iter   Instances       Fraction
 1:     1470684,        0.07342753
 2:     5349166,        0.2670703
 3:     7246843,        0.3618165
 4:     4546023,        0.2269714
 5:     1296144,        0.06471319
 6:     119339,         0.0059582936
 7:     857,            4.2787837E-5
Average: 2.9605186
Sum: 20029056
*** Multiple Random (dim=4)
 Iter   Instances       Fraction
 1:     1390864,        0.06944232
 2:     4773066,        0.23830709
 3:     6487359,        0.3238974
 4:     4639893,        0.2316581
 5:     2006668,        0.100187846
 6:     584986,         0.029206868
 7:     123910,         0.0061865123
 8:     19792,          9.881645E-4
 9:     2303,           1.14982955E-4
 10:    200,            9.985493E-6
 11:    14,             6.9898454E-7
 12:    1,              4.9927465E-8
```

```
Average: 3.1729155
Sum: 20029056
*** All Profitable Switches (dim=4)
 Iter    Instances          Fraction
 1:      3641062,           0.181789
 2:      8539092,           0.42633522
 3:      6278469,           0.31346804
 4:      1473253,           0.07355579
 5:      94995,             0.0047428594
 6:      2183,              1.08991655E-4
 7:      2,                 9.985493E-8
Average: 2.2934556
Sum: 20029056
*** Weighted Random Multiple Switch (dim=4)
 Iter    Instances          Fraction
 1:      3269387,           0.1632322
 2:      8492789,           0.42402342
 3:      6484466,           0.32375294
 4:      1644243,           0.08209289
 5:      134012,            0.0066908794
 6:      4147,              2.070492E-4
 7:      12,                5.991296E-7
Average: 2.3456104
Sum: 20029056
*** Weighted Greedy Multiple Switch (dim=4)
 Iter    Instances          Fraction
 1:      2951175,           0.1473447
 2:      8801402,           0.4394317
 3:      6653571,           0.33219594
 4:      1523096,           0.07604432
 5:      98593,             0.0049224985
 6:      1219,              6.086158E-5
Average: 2.351951
Sum: 20029056
```

Algorithm	GSSA	RSSA	APSA	RMSA	KLRA	WRMSA	WGMSA
Average iterations	2.96	4.15	2.29	3.17	3.92	2.35	2.35
Worst case	7	13	7	12	14	7	6

Table 1: Tests on all unimodal four-dimensional cubes

5.2 Tests on Random Instances of Higher-Dimensional Cubes

All the algorithms described in Section 3 were run on randomly generated completely unimodal cubes of higher dimensions, generated by the random generator described in Section 4. Since generating cubes of high dimensions is costly, the numbers of cubes tested were decreased with increasing numbers of dimensions. For four-, five- and six-dimensional cubes, 1.000.000 instances of each were generated. Of seven-, eight- and nine-dimensional cubes the number was 100.000 and for ten-, eleven- and twelve-dimensional cubes 10.000. For thirteen and fourteen dimensions 1.000 cubes each were generated, and for fifteen and sixteen dimensions 200 cubes. In the experiments, all algorithms were run on the same generated set of instances for each dimension. The results are summarized in Table 2. The first figure in each entry is the average number of iterations over the set of generated instances. The second figure is the largest number of iterations performed on any instance. Note again that for the randomized algorithms, these maxima are not the expected behaviors in the worst case, but rather the worst behaviors demonstrated in the experiments. The average and worst numbers of iterations are also shown graphically in Figure 1 and Figure 2 respectively.

Dimensions	Algorithm						
	GSSA	RSSA	APSA	RMSA	KLRA	WRMSA	WGMSA
4	3.00 / 7	4.17 / 12	2.31 / 7	3.21 / 11	4.28 / 12	2.40/ 7	2.45 / 6
5	3.88 / 10	5.37 / 15	2.56 / 8	3.79 / 13	5.64 / 17	2.82 / 8	2.94 / 8
6	4.82 / 12	6.62 / 18	2.73 / 8	4.29 / 14	7.16 / 21	3.20 / 9	3.41 / 9
7	5.83 / 11	7.89 / 19	2.84 / 8	4.73 / 14	8.80 / 23	3.54 / 9	3.87 / 10
8	6.87 / 14	9.18 / 22	2.91 / 8	5.10 / 14	10.6 / 26	3.84 / 10	4.29 / 10
9	7.94 / 14	10.4 / 23	2.93 / 9	5.39 / 15	12.4 / 32	4.08 / 10	4.64 / 11
10	9.04 / 14	11.7 / 23	2.93 / 8	5.64 / 15	14.2 / 30	4.27 / 10	4.96 / 11
11	10.1 / 16	12.8 / 23	2.92 / 8	5.80 / 14	16.1 / 32	4.46 / 9	5.21 / 10
12	11.1 / 17	14.0 / 25	2.89 / 8	5.97 / 14	17.8 / 34	4.59 / 10	5.42 / 10
13	12.2 / 17	15.2 / 26	2.91 / 7	6.13 / 14	19.9 / 34	4.72 / 9	5.65 / 10
14	13.2 / 18	16.0 / 26	2.91 / 7	6.15 / 12	21.4 / 37	4.82 / 9	5.77 / 10
15	14.1 / 19	17.3 / 27	2.94 / 6	6.43 / 14	22.6 / 35	4.74 / 8	5.77 / 9
16	15.2 / 19	18.1 / 25	2.71 / 6	6.34 / 12	24.6 / 42	4.82 / 8	5.92 / 9

Table 2: Summary of results (showing Average / Maximum).

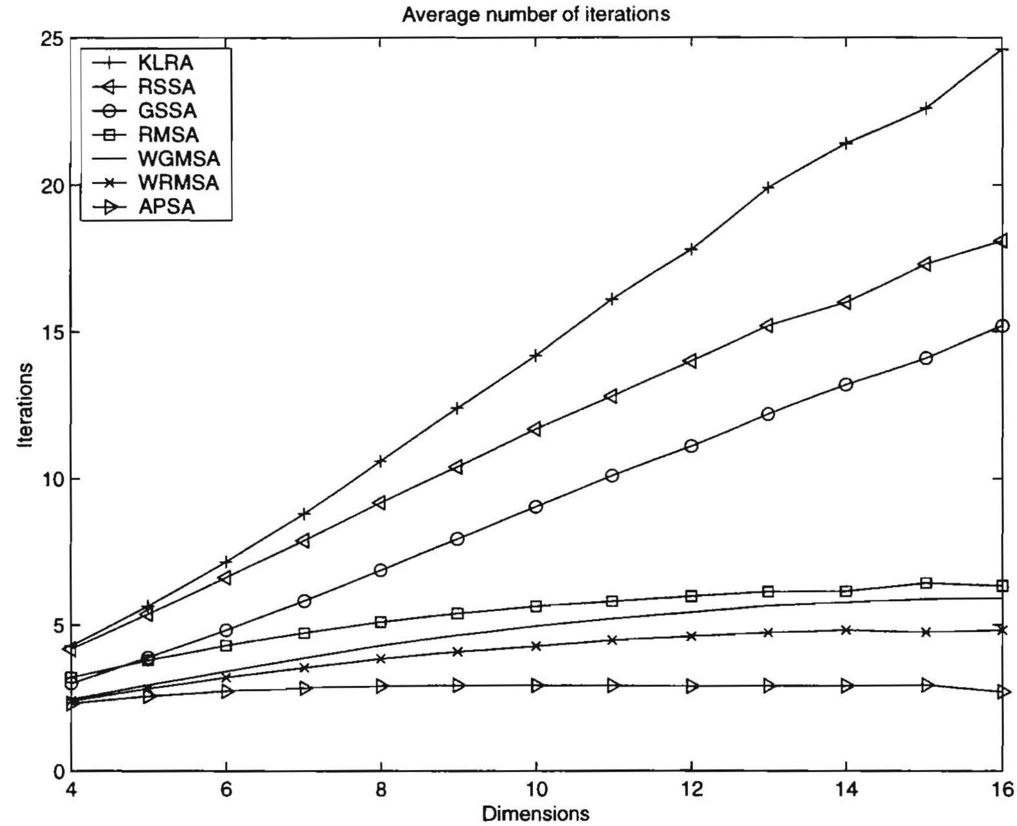

Figure 1: The graphs for the average numbers of iterations

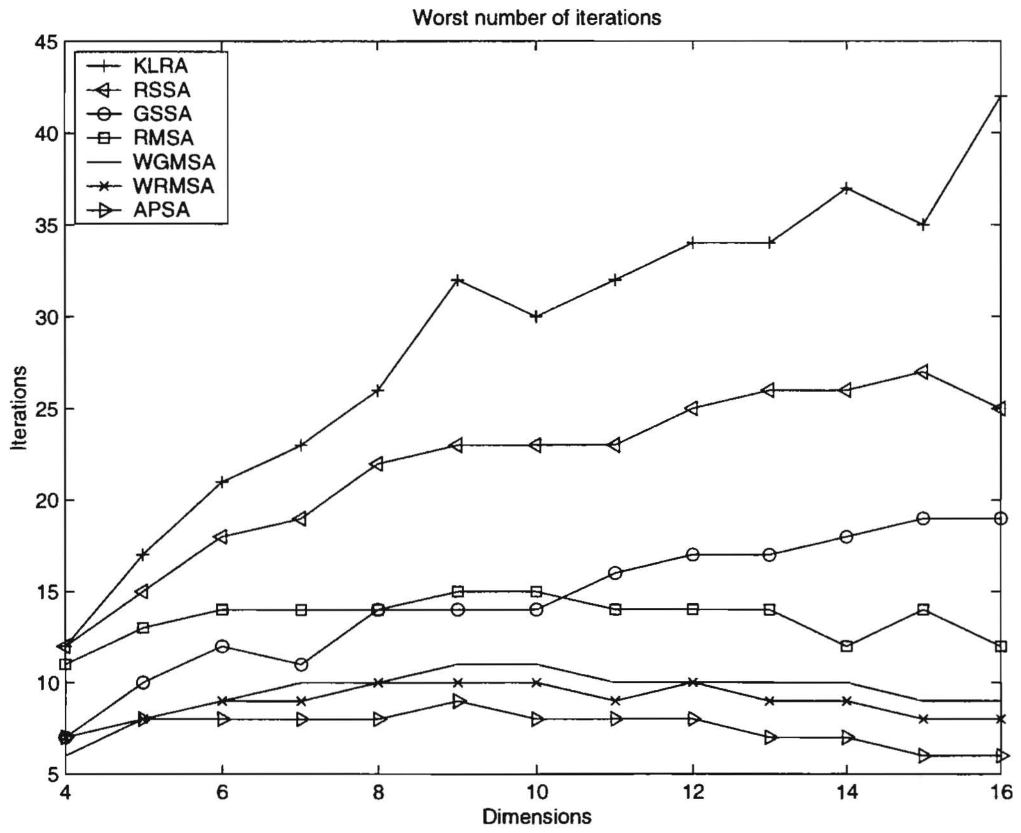

Figure 2: The graphs for the worst numbers of iterations

5.3 Hamiltonian Data

As described in Section 4 the parameterized generator can generate Hamiltonian cubes and cubes with long simple paths. Since seven is the largest number of dimensions for which the generator can produce large numbers of Hamiltonian cubes within reasonable time, all algorithms were tested on cubes of seven dimensions generated with the parameter set to 1, 2, 4, 8, 16, 32, 64, and 128. When the parameter is one, Hamiltonian cubes are generated, and for seven dimensions the parameter 128 corresponds to running the non-parameterized-generator, since seven-dimensional cubes have 128 vertices. The exponential increase of the parameter was chosen because the most dramatic changes take place at the beginning of the scale. For each choice of the parameter 10.000 cubes were generated. All algorithms were run on the same sets of cubes. Table 3 contains the results. The average numbers of iterations are also depicted in Figure 3.

The WRMSA was run with $F(x) = x^2$, and the WGMSA with $\rho = 2/3$.

It should be noted that the GSSA performs best on Hamiltonian cubes and the APSA, as expected, at the other end of the scale. What is interesting with the results is that the WGMSA and the WRMSA work reasonably well at both ends of the scale. The explanation for this is that they mimic the GSSA where it performs best, but resemble the APSA more on the types of cubes where it is superior.

Parameter	Algorithm						
	GSSA	RSSA	APSA	RMSA	KLRA	WRMSA	WGMSA
1	3.80 / 11	8.07 / 25	6.47 / 15	7.10 / 21	8.56 / 33	4.93 / 13	4.16 / 11
2	5.20 / 15	9.03 / 26	5.58 / 14	6.87 / 20	10.0 / 35	4.97 / 11	4.65 / 12
4	5.56 / 15	9.01 / 26	4.96 / 13	6.54 / 18	9.87 / 37	4.91 / 12	4.80 / 11
8	5.68 / 15	8.86 / 25	4.25 / 11	6.01 / 17	9.67 / 27	4.55 / 12	4.69 / 12
16	5.74 / 14	8.36 / 21	3.42 / 9	5.24 / 14	9.28 / 26	4.05 / 10	4.29 / 10
32	5.80 / 12	7.99 / 18	2.94 / 7	4.78 / 13	8.89 / 22	3.64 / 9	3.92 / 9
64	5.84 / 11	7.92 / 17	2.84 / 7	4.73 / 12	8.81 / 20	3.57 / 8	3.87 / 9
128	5.83 / 12	7.91 / 18	2.86 / 7	4.76 / 14	8.81 / 23	3.51 / 9	3.88 / 8

Table 3: Summary of results for tests on 7-dimensional cubes with the parameterized generator (showing Average / Maximum). With the parameter set to one, cubes with Hamiltonian paths are generated.

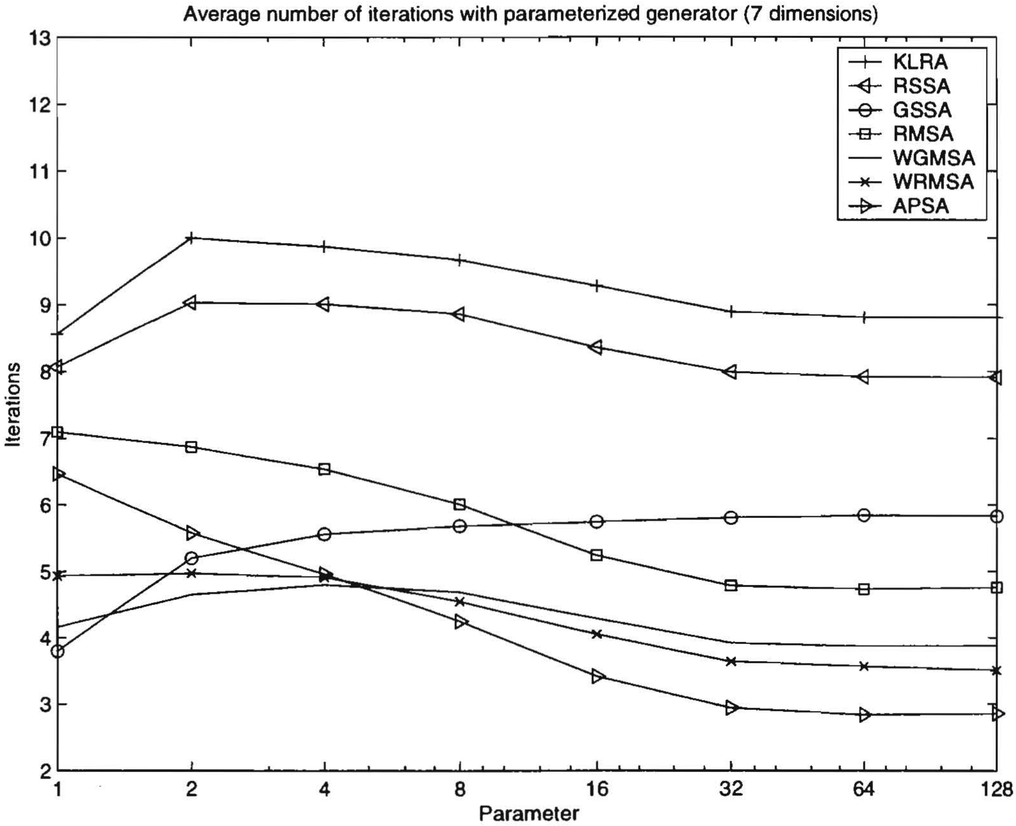

Figure 3: The graph for tests with the parameterized generator

6 Conclusions

6.1 Discussion of the Experimental Results

The experimental results presented in Section 5 are encouraging. Several extra remarks are in order, partly to put them into perspective and partly to emphasize the really good aspects of them.

1. The behavior of the multiple switch algorithms strongly indicates that most completely unimodal hypercubes are easily optimized using such algorithms.
2. Despite the fact that we tried to generate our random instances of completely unimodal hypercubes as fairly as possible, the results obtained may raise doubts that considering symmetrical hypercubes as different leads to the generation of an unproportional number of 'easy' cases. We will try to precisely account for all possible symmetries in later studies.
3. Even though the test data quite possibly is unrealistically 'easy', it seems that the multiple switch algorithms perform much better asymptotically than Kalai-Ludwig's algorithm, known to have an expected subexponential complexity. This is perhaps the most interesting, promising, and encouraging fact that was obtained from our experiments.
4. As stated earlier, the results for the randomized algorithms are not the expected numbers of iterations, but the numbers of iterations done in a single test run on each generated hypercube. The average expected number of iterations should be close to the average obtained, whereas the expected worst case behavior should be better then the experimental worst case results.

The obvious conclusion from the experimental results is that despite the questionable quality of the test data the superiority of the multiple switch algorithms compared to the well investigated single switch algorithms seems both interesting and very promising. (With the exception of the GSSA, best on Hamiltonian cubes.)

6.2 Directions for Future Research

In this paper we have considered and optimized, by using seven local-search-type algorithms, completely unimodal hypercubes with up to sixteen dimensions, and obtained promising results and conclusions. The analysis of these hypercubes and the related games is an ongoing project, and there are many interesting ways to proceed, including the following:

1. To impose some further restrictions on the hypercubes such as non-decomposable, in order to generate 'harder' random instances of completely unimodal hypercubes.
2. To generate random games (for instance Parity Games or Simple Stochastic Games) and look at the complexity of the hypercubes representing strategies in those games. Besides the fact that solving such games is an important problem in its own right, this might provide insights into the relative difficulty of distributions of completely unimodal hypercubes arising from real problems.
3. An entirely different direction would be to try to find theoretic complexity bounds on the number of iterations needed in the average case as well as the worst case for the multiple switch algorithms APSA, RMSA, WRMSA, and WGMSA. Although difficult to find, such results would be a considerable achievement in optimization and complexity theory.

Efficient algorithms for optimization of completely unimodal hypercubes would be useful for solving numerous practical problems, and this makes the quest for such algorithms interesting as well as important. We continue research in all the directions mentioned above and will present our progress in further reports.

References

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.

Hansen, P., Jaumard, B. & Mathon, V. (1993), 'Constrained nonlinear 0-1 programming (state-of-the-art survey)', *ORSA Jornal on Computing* **5**(2), 97–119.

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.

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

Petersson, V. & Vorobyov, S. (2001*b*), Parity games: interior-point approach, Technical Report 008, Uppsala University / Information Technology.

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

INFORMATIK

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Anja Becker
Stuhlsatzenhausweg 85
66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

MPI-I-2001-4-004	S.W. Choi, H. Seidel	Linear One-sided Stability of MAT for Weakly Injective Domain
MPI-I-2001-4-003	K. Daubert, W. Heidrich, J. Kautz, J. Dischler, H. Seidel	Efficient Light Transport Using Precomputed Visibility
MPI-I-2001-4-002	H.P.A. Lensch, J. Kautz, M. Goesele, H. Seidel	A Framework for the Acquisition, Processing, Transmission, and Interactive Display of High Quality 3D Models on the Web
MPI-I-2001-4-001	H.P.A. Lensch, J. Kautz, M.G. Goesele, W. Heidrich, H. Seidel	Image-Based Reconstruction of Spatially Varying Materials
MPI-I-2001-2-003	S. Vorobyov	?
MPI-I-2001-2-002	P. Maier	A Set-Theoretic Framework for Assume-Guarantee Reasoning
MPI-I-2001-2-001	U. Waldmann	Superposition and Chaining for Totally Ordered Divisible Abelian Groups
MPI-I-2001-1-002	U. Meyer	Directed Single-Source Shortest-Paths in Linear Average-Case Time
MPI-I-2001-1-001	P. Krysta	Approximating Minimum Size 1,2-Connected Networks
MPI-I-2000-4-003	S.W. Choi, H. Seidel	Hyperbolic Hausdorff Distance for Medial Axis Transform
MPI-I-2000-4-002	L.P. Kobbelt, S. Bischoff, K. Kähler, R. Schneider, M. Botsch, C. Rössl, J. Vorsatz	Geometric Modeling Based on Polygonal Meshes
MPI-I-2000-4-001	J. Kautz, W. Heidrich, K. Daubert	Bump Map Shadows for OpenGL Rendering
MPI-I-2000-2-001	F. Eisenbrand	Short Vectors of Planar Lattices Via Continued Fractions
MPI-I-2000-1-005	M. Seel, K. Mehlhorn	Infimaximal Frames A Technique for Making Lines Look Like Segments
MPI-I-2000-1-004	K. Mehlhorn, S. Schirra	Generalized and improved constructive separation bound for real algebraic expressions
MPI-I-2000-1-003	P. Fatourou	Low-Contention Depth-First Scheduling of Parallel Computations with Synchronization Variables
MPI-I-2000-1-002	R. Beier, J. Sibeyn	A Powerful Heuristic for Telephone Gossiping
MPI-I-2000-1-001	E. Althaus, O. Kohlbacher, H. Lenhof, P. Müller	A branch and cut algorithm for the optimal solution of the side-chain placement problem
MPI-I-1999-4-001	J. Haber, H. Seidel	A Framework for Evaluating the Quality of Lossy Image Compression
MPI-I-1999-3-005	T.A. Henzinger, J. Raskin, P. Schobbens	Axioms for Real-Time Logics

MPI-I-1999-3-004	J. Raskin, P. Schobbens	Proving a conjecture of Andreka on temporal logic
MPI-I-1999-3-003	T.A. Henzinger, J. Raskin, P. Schobbens	Fully Decidable Logics, Automata and Classical Theories for Defining Regular Real-Time Languages
MPI-I-1999-3-002	J. Raskin, P. Schobbens	The Logic of Event Clocks
MPI-I-1999-3-001	S. Vorobyov	New Lower Bounds for the Expressiveness and the Higher-Order Matching Problem in the Simply Typed Lambda Calculus
MPI-I-1999-2-008	A. Bockmayr, F. Eisenbrand	Cutting Planes and the Elementary Closure in Fixed Dimension
MPI-I-1999-2-007	G. Delzanno, J. Raskin	Symbolic Representation of Upward-closed Sets
MPI-I-1999-2-006	A. Nonnengart	A Deductive Model Checking Approach for Hybrid Systems
MPI-I-1999-2-005	J. Wu	Symmetries in Logic Programs
MPI-I-1999-2-004	V. Cortier, H. Ganzinger, F. Jacquemard, M. Veanes	Decidable fragments of simultaneous rigid reachability
MPI-I-1999-2-003	U. Waldmann	Cancellative Superposition Decides the Theory of Divisible Torsion-Free Abelian Groups
MPI-I-1999-2-001	W. Charatonik	Automata on DAG Representations of Finite Trees
MPI-I-1999-1-007	C. Burnikel, K. Mehlhorn, M. Seel	A simple way to recognize a correct Voronoi diagram of line segments
MPI-I-1999-1-006	M. Nissen	Integration of Graph Iterators into LEDA
MPI-I-1999-1-005	J.F. Sibeyn	Ultimate Parallel List Ranking ?
MPI-I-1999-1-004	M. Nissen, K. Weihe	How generic language extensions enable "open-world" desing in Java
MPI-I-1999-1-003	P. Sanders, S. Egner, J. Korst	Fast Concurrent Access to Parallel Disks
MPI-I-1999-1-002	N.P. Boghossian, O. Kohlbacher, H.-. Lenhof	BALL: Biochemical Algorithms Library
MPI-I-1999-1-001	A. Crauser, P. Ferragina	A Theoretical and Experimental Study on the Construction of Suffix Arrays in External Memory
MPI-I-98-2-018	F. Eisenbrand	A Note on the Membership Problem for the First Elementary Closure of a Polyhedron
MPI-I-98-2-017	M. Tzakova, P. Blackburn	Hybridizing Concept Languages
MPI-I-98-2-014	Y. Gurevich, M. Veanes	Partisan Corroboration, and Shifted Pairing
MPI-I-98-2-013	H. Ganzinger, F. Jacquemard, M. Veanes	Rigid Reachability
MPI-I-98-2-012	G. Delzanno, A. Podelski	Model Checking Infinite-state Systems in CLP
MPI-I-98-2-011	A. Degtyarev, A. Voronkov	Equality Reasoning in Sequent-Based Calculi
MPI-I-98-2-010	S. Ramangalahy	Strategies for Conformance Testing
MPI-I-98-2-009	S. Vorobyov	The Undecidability of the First-Order Theories of One Step Rewriting in Linear Canonical Systems
MPI-I-98-2-008	S. Vorobyov	AE-Equational theory of context unification is Co-RE-Hard
MPI-I-98-2-007	S. Vorobyov	The Most Nonelementary Theory (A Direct Lower Bound Proof)
MPI-I-98-2-006	P. Blackburn, M. Tzakova	Hybrid Languages and Temporal Logic
MPI-I-98-2-005	M. Veanes	The Relation Between Second-Order Unification and Simultaneous Rigid *E*-Unification
MPI-I-98-2-004	S. Vorobyov	Satisfiability of Functional+Record Subtype Constraints is NP-Hard
MPI-I-98-2-003	R.A. Schmidt	E-Unification for Subsystems of S4
MPI-I-98-2-002	F. Jacquemard, C. Meyer, C. Weidenbach	Unification in Extensions of Shallow Equational Theories
MPI-I-98-1-031	G.W. Klau, P. Mutzel	Optimal Compaction of Orthogonal Grid Drawings
MPI-I-98-1-030	H. Brönniman, L. Kettner, S. Schirra, R. Veltkamp	Applications of the Generic Programming Paradigm in the Design of CGAL