0/1 vertex and facet enumeration with BDDs*

Markus Behle† Friedrich Eisenbrand‡

Abstract

In polyhedral studies of 0/1 polytopes two prominent problems exist. One is the vertex enumeration problem: Given a system of inequalities, enumerate its feasible 0/1 points. Another one is the convex hull problem: Given a set of 0/1 points in dimension d, enumerate the facets of the corresponding polytope. We present two new approaches for both problems. The novelty of our algorithms is the incorporation of binary decision diagrams (BDDs), a datastructure which has become very popular and effective in hardware verification and computational logic.
Our computational results show the strength of our methods. We introduce our new tool `azove` which is currently the fastest software for counting and enumerating 0/1 points in a polytope.

1 Introduction

In combinatorial optimization an important part in understanding and designing algorithms for a certain problem is the investigation of the polyhedral structure of the associated polytope. For many problems in this field the underlying polytope is a 0/1 polytope, i.e. all vertices are 0/1 points.

One frequently arising problem is the *0/1 vertex enumeration* problem:

> Given a set of inequalities $Ax \leq b$, $A \in \mathbb{Z}^{m \times d}$, $b \in \mathbb{Z}^m$, compute a list of all 0/1 points satisfying the system.

In other words, if P denotes the polyhedron $P = \left\{x \in \mathbb{R}^d \mid Ax \leq b,\ 0 \leq x \leq 1\right\}$ then one is interested in the vertices of the *integer hull* P_I of P which generate the convex hull of all integer points of P.

The other problem which we consider here is the *0/1 facet enumeration* problem:

> Given a set $S \subseteq \{0,1\}^d$ of 0/1 points, enumerate all facets of the convex hull $conv(S)$.

A successful approach to difficult optimization problems with integer programming often requires some understanding of the facets of the integer hull of the solution space. A software package which computes the inequality representation $A'x \leq b'$ of the integer hull P_I of P, given an inequality representation $Ax \leq b$ of P can here become very useful. Such an inequality representation is currently computed in a two-step approach. In a first step, one solves the 0/1 vertex enumeration problem, and then in a second step the 0/1 facet enumeration problem for the previously generated 0/1 points is solved.

In this paper we present tools to solve the 0/1 vertex and facet enumeration problems which are based on binary decision diagrams. Our tool `azove` [9] which solves the vertex enumeration problem outperforms the currently best codes for this task by several orders of magnitude. Second we report on a gift-wrapping approach[15] to solve the facet enumeration problem. Here we use BDDs to rotate a facet-defining inequality along a ridge to find a new facet. Ridges are computed with existing codes. We can recommend our approach for polytopes whose facets contain few vertices.

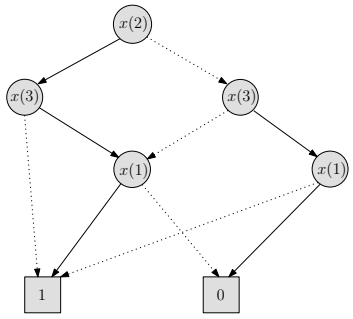
Related work Bussieck and Lübbecke [14] presented a backtracking algorithm for solving the 0/1 vertex enumeration problem. Their algorithm intersects the given polytope with two opposite facets of the unit hypercube. If this intersection is not empty it is divided into two branches which are then treated recursively. They proved that the vertex set of a 0/1 polytope given in inequality description is *strongly $\mathcal{P}$-enumerable*. The tool `zerone` [25] is based on this idea.

To describe related work on the convex hull problem, we first review some definitions. The *faces* of a convex polytope P are $\emptyset$, P and the intersection of a supporting hyperplane of P and P itself. The *dimension* d of P is the dimension of its affine hull. Faces of dimension 0, $d-2$, and $d-1$ are called *vertices*, *ridges*, and *facets* respectively. A polytope P is called *simpli-*

*This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See `www.avacs.org` for more information.

†Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, `behle@mpi-inf.mpg.de`

‡Universität Paderborn, Warburger Str. 100, 33098 Paderborn, Germany, `eisen@math.uni-paderborn.de`



(a) BDD

$x(1)$	$x(2)$	$x(3)$
0	0	1
0	1	0
1	0	0
1	1	0
1	1	1

(b) Truth table

Figure 1: A simple BDD represented as a directed graph. Edges with parity 0 are dashed. The table shows the represented 0/1 points of the set T.

cial, if every facet contains exactly d vertices. P is called *simple*, if every vertex is the intersection of exactly d facets. The input for the facet enumeration problem is called *nondegenerate* if there are no $d+1$ points which lie on a common hyperplane, and *degenerate* otherwise. For further definitions on polytopes see [32].

Many algorithms have been developed for the (general) convex hull problem, see [28] for an overview. *Incremental methods* like Kallay and Seidel's *beneath and beyond* and the *double description method* by Motzkin et. al. successively compute a description for $P_i := \{p_1, \ldots, p_i\}$ from the description for P_{i-1} and the point p_i. Bremner [11] showed that incremental convex hull algorithms are not output sensitive. Note that the so-called Fourier-Motzkin elimination is dual to the double description method and also an incremental method.

Algorithms that construct the face lattice of a polytope are called *graph traversal methods*. Among them is Chand and Kapur's *gift wrapping* [15] which has been improved later by many others (see e.g. [26] and [30]), and in the dual sense *pivoting algorithms* like Avis and Fukuda's *reverse search* [4]. Another approach is the *primal-dual method* by Bremner, Fukuda and Marzetta [12].

At present no polynomial runtime algorithm for the convex hull problem for general (degenerate) polytopes is known. A comparison of the main convex hull algorithms is given in [3].

BDDs Our approaches are are based on *Binary Decision Diagrams* (BDDs for short), a datastructure which represents a set of 0/1 points in a compact way. We provide a short definition of BDDs as they are used in this paper.

A BDD for a set of variables $x(1), \cdots, x(d)$ is a directed acyclic graph $G = (V, A)$, see figure 1. It has one node with in-degree zero, called the root and two nodes with out-degree zero, called leaf 0 resp. leaf 1. There is a labeling function $\ell\colon V \setminus \{\text{leaf } 0, \text{leaf } 1\} \to \{x(1), \ldots, x(d)\}$ and a parity function $\text{par}\colon A \to \{0,1\}$. All nodes labelled with $x(i)$ lie on the same level, which means, we have an *ordered* BDD (OBDD). In this paper all BDDs are ordered. If each path from the root to one of the leafs contains exactly d edges the BDD is called *complete*. A path $e_1, \ldots, e_d$ from the root to one of the leafs represents a variable assignment, where the label $x(i)$ of the starting node of e_j is assigned to the value $\text{par}(e_j)$. All paths from the root to leaf 1 represent the set $T \subseteq \{0,1\}^d$ of true-assignments, whereas the paths from the root to leaf 0 represent the set $F \subseteq \{0,1\}^d$ of false-assignments. We always have $F \dot\cup T = \{0,1\}^d$. In case a BDD is not complete there are long edges that cross a level with nodes labeled $x(k)$. In that case the assignment for $x(k)$ is free. Long edges are the key for very compact representations of a set of boolean vectors.

BDDs were first proposed by Lee in 1959 [23]. Bryant [13] presented efficient algorithms for the synthesis of BDDs. After that, BDDs became very popular in the area of hardware verification and computational logics, see e.g. [31].

2 0/1 vertex enumeration

Given a set of inequalities $Ax \le b$, $A \in \mathbb{Z}^{m\times d}$, $b \in \mathbb{Z}^m$ we want to enumerate all 0/1 points which satisfy this set of inequalities. This is done as follows. We we build the BDD of each of the m linear inequalities $a_i^T x \le b_i$ (see section 2.1). Then we "combine" the inequalities pairwise by conjuncting the corresponding pairs of BDDs (see section 2.2). After m conjunctions we have the final BDD representing the 0/1 solutions. The enumeration of all paths from the root to the leaf 1 gives then all 0/1 points.

2.1 Building a BDD for a linear constraint Consider the function $f\colon \{0,1\}^d \to \mathbb{Z}$ with $f(x) := a^T x - b$, $a \in \mathbb{Z}^d$, $b \in \mathbb{Z}$. Algorithm 1 shows how to build the BDD for the threshold function $f(x) \leq 0$. In fact it is a dynamic programming approach (see e.g. [27]) which is very similar to the dynamic programming approach to solve the knapsack problem.

Algorithm 1 Build BDD for $f(x) \leq 0$

BUILDBDD(f)
(1) **if** $\max(f) \leq 0$ **return** leaf 1
(2) **if** $\min(f) > 0$ **return** leaf 0
(3) **if** $f \in$ table **return** table[f]
(4) x(i) = NEXTVARIABLE(f)
(5) BDD low = BUILDBDD($f|_{x(i)=0}$)
(6) BDD high = BUILDBDD($f|_{x(i)=1}$)
(7) BDD result = $x_i \cdot$ high $+\bar{x}_i \cdot$ low
(8) table[f] = result
(9) **return** result

Define $a^+ := \sum_{a_i>0} a_i$ and $a^- := \sum_{a_i<0} a_i$. We set up a table of size $d \times (a^+ - a^-)$ in which we save results (step 8) and look up already computed BDDs in constant time (step 3). To start building the BDD we call BuildBDD($a^T x - b$). First we check for trivial cases (steps 1 and 2). Note that it is sufficient to compute the global minimum and maximum for f once. All other values can be computed in constant time by the recursive calls. After the selection of a variable $x(i)$ according to the variable order in the BDD (step 4) we build the children of the actual node with restriction of the variable $x(i)$ to 0 resp. 1 (steps 5 - 6). In step 7 a new node will be inserted on top of both children.

Let $\|a\|_\infty$ be the maximum absolute value of all components of the vector a. Then $d\|a\|_\infty \geq a^+ - a^-$ holds. Thus the runtime and space complexity for building a BDD for a linear constraint are both pseudo-polynomial which is stated in the following lemma.

LEMMA 2.1. *The runtime and the space complexity for building a BDD for a linear constraint $a^T x \leq b$ are both* $O(d^2\|a\|_\infty)$.

Note that algorithm 1 can easily be adapted to build the BDD for a linear equation, which means it can also solve the subset sum problem.

2.2 Conjunction of BDDs Let f and g be two linear constraints and be $G_f = (V_f, A_f)$ resp. $G_g = (V_g, A_g)$ the corresponding graph respresentations of the BDDs. The algorithm for the synthesis of two BDDs via a binary operator like AND (see e.g. algorithm 3.3.5 in [31]) is a straight forward recursive approach. The synthesis is possible in time and space $O(|V_f||V_g|)$. In practice, the typical performance is closer to the size of the resulting BDD which is smaller than $|V_f||V_g|$.

LEMMA 2.2. *The runtime and the space complexity for the conjunction of m BDDs defined by the system of linear inequalities $Ax \leq b$ are both* $O((d^2\|A\|_\infty)^m)$.

Let ν be the number of 0/1 vertices of the polytope, i.e. the cardinality of the output set. In a BDD these vertices correspond to all paths from the root to the leaf 1 which can be enumerated in $O(\nu d)$.

The total runtime of our algorithm for the enumeration of all 0/1 points that are contained in the relexation given by the system of linear inequalities $Ax \leq b$ is $O((d^2\|A\|_\infty)^m + \nu d)$. In practice, the typical performance is much better as we will describe in the next section.

2.3 Computational results We developed `azove` which is *a*nother *z*ero *o*ne *v*ertex *e*numeration tool. It can be downloaded from [9]. The presented computational results were achieved with version 1.0. For managing the BDDs we used the CUDD 2.4.1 library [29].

We compare our tool with `zerone` 1.81 [25], which we patched to run with CPLEX 9.0 [21] as linear solver. Table 1 shows the comparisson of the runtimes in seconds. The time spent building the BDDs and just counting the 0/1 vertices is given in brackets. The tests were run on a Linux system with kernel 2.6.13 and gcc 3.4.4 on a Pentium 4 CPU with 2.6 GHz and 1.5 GB memory. Our benchmark set contains two instance classes OA and TC which we took from a collection of 0/1 polytopes that has been compiled in connection with [32] and can be found on the polymake [19] homepage. The convex hull of these polytopes served as input. The other problems have been taken from the MIPLIB [10]. They are relaxations of 0/1 polytopes. Although not necessary the bounds on the variables $0 \leq x_i \leq 1$ are given explicitly and thus included in the number of inequalities. For the instances p0040 and stein45 most of the time is spent for the output of the vertices. Obviously our tool `azove` outperforms `zerone` even on inequality systems that describe 0/1 polytopes and not only relaxations. We also tried `vint` from the porta 1.4.0 package [16] which enumerates all integral points in a polytope. In nearly all cases it reported that it could not handle that many inequalities. In case it succeeded the runtime was not comparable, possibly because it is not specialized in 0/1 vertices.

2.4 Vertex counting Counting the vertices of a 0/1 polytope is equivalent to counting the number of paths from the root to leaf 1 in a graph representation $G = (V, A)$ of a BDD. This can be done in $O(|V|)$ (see

Name	Dim	Inequalities		0/1 Vertices	zerone	azove	(counting)
OA:8-25	8	524	(ch)	25	0.06	0.02	(0.01)
OA:9-33	9	1870	(ch)	33	0.48	0.11	(0.10)
OA:10-44	10	9708	(ch)	44	11.09	1.06	(0.94)
TC:8-38	8	1675	(ch)	38	0.38	0.06	(0.05)
TC:9-48	9	6875	(ch)	48	3.45	0.46	(0.40)
TC:10-83	10	41591	(ch)	83	89.74	4.51	(4.00)
TC:11-106	11	250279	(ch)	106	5713.19	54.53	(50.98)
bm23	27	74	(rel)	2168	15.48	3.91	(3.72)
p0033	33	81	(rel)	10746	11.41	0.14	(0.01)
p0040	40	103	(rel)	519216	166.84	8.39	(0.01)
stein15	15	66	(rel)	2809	0.41	0.02	(0.01)
stein27	27	172	(rel)	367525	110.47	4.14	(0.22)
stein45	45	421	(rel)	244049633	166115.17	4386.08	(232.95)

Table 1: Comparisson of the 0/1 vertex enumeration tools `zerone` and `azove`

e.g. [31]) in the following way. W.l.o.g. be the BDD complete. Label each node $v \in V$ with a number $c(v)$ in a bottom-up fashion. Set $c(\text{leaf } 0) = 0$ and $c(\text{leaf } 1) = 2^d$. For a node v with its two successors v_0 and v_1 set $c(v) = (c(v_0) + c(v_1))/2$ since a path starting from v can choose any of the two edges. $c(\text{root})$ finally states the number of 0/1 vertices of the polytope. In table 1 the time given in brackets reflects the time that we need to build the BDDs and just count the number of 0/1 vertices with `azove`. Counting the number of 0/1 vertices is often desired to decide in advance whether an enumeration makes sense.

We also tried another tool capable of counting integral points in polytopes which is `latte` 1.2 [24]. It implements Barvinok's algorithm [7] which is polynomial in fixed dimension. The runtimes are considerably higher. A comparison however is not fair since `latte` is not specialized for the 0/1 case.

3 Facet enumeration

In this section we consider the following problem: Given a set S of 0/1 points in dimension d, find an inequality description $P = \left\{x \in \mathbb{R}^d \mid Ax \leq b,\ A \in \mathbb{Z}^{m \times d},\ b \in \mathbb{Z}^m\right\}$ of its convex hull, which means $P = conv(S)$. We assume here that $P = conv(S)$ is full-dimensional. Then each inequality of the system corresponds to a facet of P.

Algorithm 2 incorporates the BDD structure in a gift-wrapping approach. We build and then traverse a graph $G_P = (F, R)$. The nodes $f \in F$ represent the facets of P and the edges $r \in R$ represent the ridges which are the facets of a facet. Note that a ridge is the intersection of two facets, i.e. $r = (f, f')$. A ridge r is called *open* if only one of its incident facets $r = (f, \cdot)$ is known.

We start in step 1 with an empty BDD. For every $p \in S$ we build a BDD which represents p by its path. Then we build the synthesis of the BDDs with the OR operator (see e.g. [31]). In this case the time and space complexity is naturally bounded by $O(|S|d)$.

Algorithm 2 Finding the convex hull with BDDs

```
ConvexHullBDD(S)
(1) BDD = buildBDD(S)
(2) f_1 = findFirstFacet(S)
(3) F = {f_1}
(4) R = findRidges(f_1)
(5) while (∃ open ridge r = (f, ·) ∈ R) {
(6)     f' = findNewFacet(f, r, BDD)
(7)     F = F ∪ {f'}
(8)     R = R ∪ findRidges(f') }
(9) return F
```

In step 2 we have to find the first facet f_1 of P to start with. Facets are represented by their normalvector and their right hand side as $a^T x \leq b$. Be S_c the translation of the set S in such a way that $\mathbf{0} \in \text{interior}(conv(S_c))$. S_c can be computed using the center of S. Any vertex of the polar set S_c^Δ of S_c can be used as the normalvector of the first facet (see e.g. [32]). Such a vertex can be computed in polynomial time in $|S|$ and d. Note that it is sufficient to know the normalvector a since we can compute the right hand side b and all 0/1 vertices that lie on the facet by optimizing over the BDD according to the linear objective function a. This reduces to a shortest path problem on the BDD, see also [8]. The value b is given as optimal value and the shortest paths correspond to the 0/1 points that are tight at the facet.

Be S_r the set of 0/1 points that lie on a ridge r. Since a BDD implicitly represents 0/1 points in a lexicographical order, S_r is lexicographically sorted. Then the ridge r is uniquely defined by the first $d-1$

Name	Dim	Vertices	Facets	glrs	gchbdd	cddr+	gchbddCDD	chbdd
MJ:32-33	32	33	33	**0.001**	0.290	0.230	0.290	0.290
BIR5:16-120	16	120	25	1836.340	5576.010	**2.200**	27.550	1501.050
CUT6:15-32	15	32	368	5.670	17.290	**1.320**	28.880	4.100
HC:7-64	7	64	78	0.300	0.250	0.190	0.580	**0.070**
HC:8-128	8	128	144	4.990	3.720	0.970	2.850	**0.910**
OA:8-25	8	25	524	**0.120**	0.540	0.410	1.890	0.180
OA:9-33	9	33	1870	**0.850**	2.930	2.370	10.220	0.940
OA:10-44	10	44	9708	7.670	21.320	26.680	67.810	**6.640**
TC:7-30	7	30	432	0.110	0.310	0.350	0.950	**0.090**
TC:8-38	8	38	1675	0.540	1.710	2.140	5.220	**0.530**
TC:9-48	9	48	6875	4.870	9.980	17.180	31.100	**3.070**
TC:10-83	10	83	41591	105.570	129.030	629.120	422.460	**39.980**
TC:11-106	11	106	250279	979.970	1185.470	24532.110	6852.290	**374.440**
stein9	9	172	31	25.890	24.180	**1.570**	5.860	5.990

Table 2: Comparison on instances from the literature

$0/1$ points that are affinely independent. We use this fact to store ridges as bit-strings of size $(d-1)\cdot d$.

Given a facet f we need to know all of its ridges in the steps 4 and 8. We optimize according to the normalvector of f and get the set S_f. If f is simplicial, i.e. $|S_f| = d$, all d ridges can be enumerated directly. Otherwise we compute all normalvectors a_r for all ridges of f with a sub-algorithm (e.g. lexicographical reverse search or double description method) in dimension $d-1$. For each normalvector a_r we calculate S_r via optimization over the BDD. Solving a system of linear equations then gives us the $d-1$ affinely independent points that are the smallest regarding lexicographical order in time $O(|S_r|^2 d)$.

We keep the open ridges in an additional hash-set to answer the query in step 5. The number of open ridges is bounded by the total number of facets of P. Be r a ridge which has been found by findRidges in step 8. If r is not contained in the hash-set we add it. Otherwise we delete r from the hash-set as we know both facets that are incident to it.

Algorithm 3 Gift-wrapping with BDDs

findNewFacet(f, r, BDD)
(1) define $a' \in \mathbb{Z}^d$, $b' \in \mathbb{Z}$, $p \in \{0,1\}^d$ and $p' \in \{0,1\}^d$
(2) p = $0/1$ point contained in f but not contained in r
(3) a' = - normalvector of f
(4) (p', b') = shortestPath(a', BDD)
(5) **while** $({a'}^T p \neq b')\{$
(6) $\quad p = p'$
(7) $\quad a'$ = computeNormalvector(r, p)
(8) $\quad (p', b')$ = shortestPath(a', BDD) $\}$
(9) **return** $({a'}^T x \leq b')$

The sub-routine findNewFacet in step 6 is explained in detail in our algorithm 3. We use the fact that all $d-1$ points contained in the ridge r together with a $0/1$ point $p \in S$ which is not contained in r define a hyperplane. We start with the given facet f and rotate it around the ridge r until a new facet f' is found. A new point p is found by optimizing over the BDD via shortest path computation in $O(|S|d)$. We have to rotate at most $|S|$ times before we find a new facet, but in practice just a few rotations are needed. Computing a normalvector of the hyperplane defined by r and p is done by solving a system of linear equations. It is possible in $O(d^2)$ since r is fixed and we can do a precomputation once which costs $O(d^3)$. So the overall runtime of algorithm 3 is $O(d^3 + |S|(|S|d + d^2))$.

If the polytope is simplicial, all steps of our algorithm 2 can be performed in time polynomial in d, $|S|$ and $|F|$, and thus our algorithm 2 is output sensitive.

3.1 Computational results In the following section we want to compare implementations for the convex hull problem which compute exact results with arbitrary precision. We used the same hardware setup as in section 2.3. The instance MJ32-33 taken from the polymake website (see section 2.3) serves as a test instance. It is numerically difficult since the coefficients of the normalvectors of its facets are extremely large. In our testbed it is the only simplicial instance.

Neither `traf` from the porta 1.4.0 package [16] which implements the Fourier Motzkin elimination nor the quickhull algorithm [5] implementation in `qhull` 2003.1 [6] compute with arbitrary precision. Therefore we do not take these programs into account. The runtimes of the primal-dual method implemented in `pd` 1.7 [12] are extremely high and thus not comparable. In higher dimensions the beneath and beyond imple-

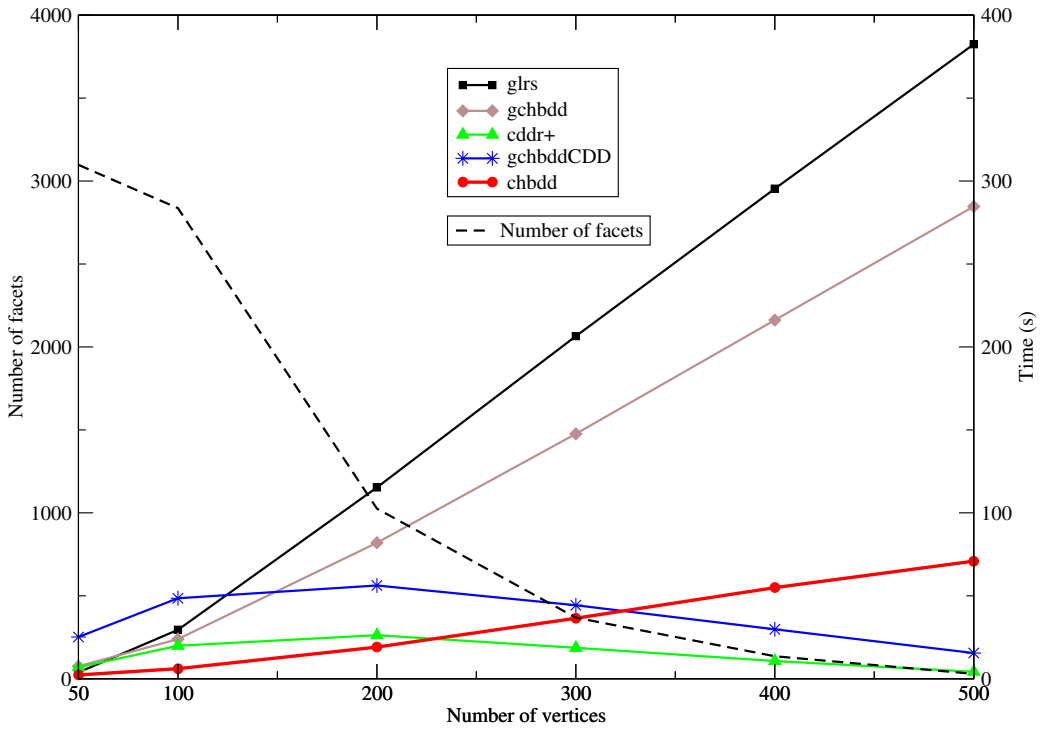


Figure 2: Comparison on randomly generated instances in dimension 9

mentation in `polymake` 2.2 [19] requires a lot of memory and the runtimes are very high. Therefore we restrict the comparison to `glrs` 4.2 [2] which implements the reverse search algorithm [1] and `cddr+` 0.77 [17] which is an implementation of the double description method [18]. Both programs were compiled with the GMP 4.1.4 [20] for arbitrary precision support.

In our implementation `chbdd` we work with the datatype `long` until we catch an integer overflow in which case we switch to the GMP. This results in a speed advantage. To be able to compare with `glrs` and `cddr+` we force the usage of the GMP in our implementation `gchbdd`. In `chbdd` and `gchbdd` we use `lrs` as a subroutine for finding the ridges whereas we use `cdd` in `gchbddCDD`. Our algorithm decomposes the problems in convex hull problems in one dimension lower. For these subproblems the input is lexicographically sorted. This fact might help incremental methods, see e.g. [22].

The input of our test instances is a set of $0/1$ points. All runtimes are given in seconds. The fastest runtime is printed in bold. The instances presented in table 2 are from the same source as those in section 2.3. For the HC instances we can speed up `glrs` with our hybrid approach. For the higher dimensional TC problems the performance of `cdd` can be improved with our algorithm. The structures of the instances in table 2 do not reveal much information about the behaviour of all algorithms in general. Therefore we generated random instances S. We generate one $0/1$ point $s \in \{0,1\}^d$ by a sequence of d coin flips. If $s \notin S$ and S does not have the desired cardinality we add it in such a way that S is lexicographically sorted.

The figures 2, 3 and 4 show the comparisons on our randomly generated instances in dimensions 9, 10 and 11. We generated 3 instances per dimension. The figures show the median of the number of facets and the median of the runtimes.

The runtime of `lrs` increases with the number of vertices of the input whereas `cdd`'s runtime mainly depends on the number of facets of the output for larger instances. The reverse search and the double description method behave complementary to each other.

If the number of vertices exceeds 100 we can improve `glrs` with our hybrid approach `gchbdd` since it is easy to find all ridges in dimension $d-1$. Only for the instances with up to 300 vertices in dimension 11 we could improve `cddr+` with our hybrid approach `gchbddCDD`. We guess that this is related to the number of ridges that arise. For a small number of vertices our approach `chbdd` is usually the fastest. In every dimension there exists a threshold value for the number of vertices from which on our approach is inferior to `cddr+`. In dimension 9, 10 resp. 11 the crucial number of vertices lies between 200-300, 300-400 resp. 500-600.

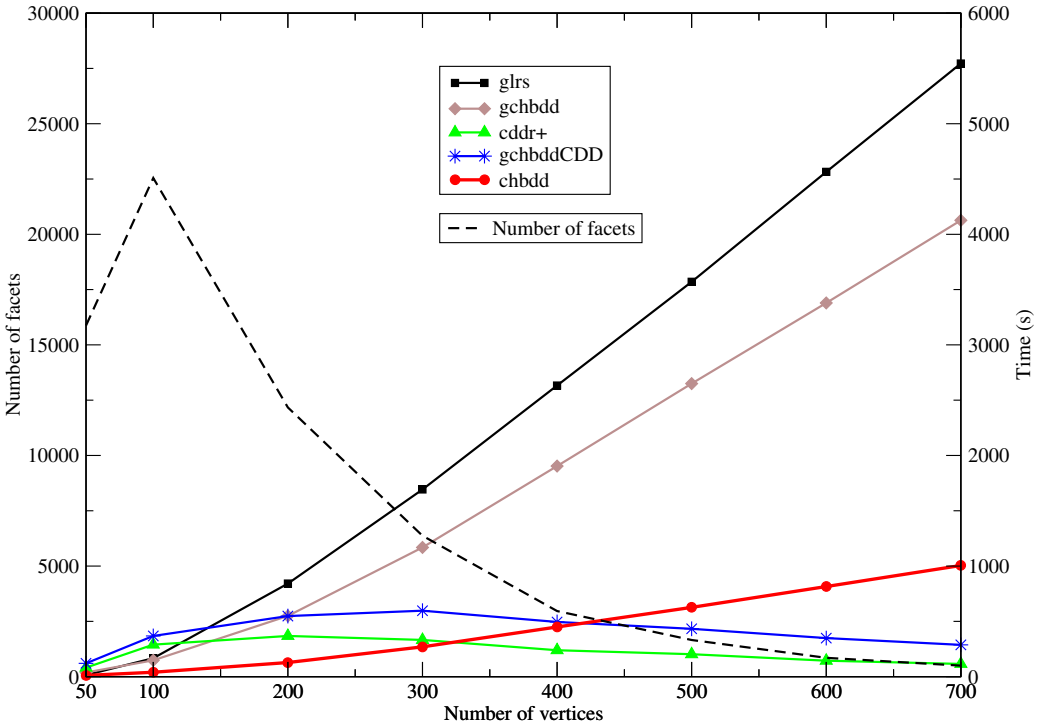


Figure 3: Comparison on randomly generated instances in dimension 10

Generally speaking our approach can cope better with polytopes whose facets contain few vertices.

An interesting point of further research could be the correlation between the number n of vertices of random 0/1 polytopes in dimension d and the expected number of facets, which we can observe in the figures 2, 3 and 4.

References

[1] D. Avis. lrs: A revised implementation of the reverse search vertex enumeration algorithm. In G. Kalai and G. M. Ziegler, editors, *Polytopes – Combinatorics and Computation*, pages 177 – 198. Birkhäuser, 2000.

[2] D. Avis. *lrslib 4.2 Homepage*, 2005. http://cgm.cs.mcgill.ca/~avis/C/lrs.html.

[3] D. Avis, D. Bremner, and R. Seidel. How good are convex hull algorithms? *Comput. Geom.: Theory and Appl.*, 7(5–6):265 – 301, 1997.

[4] D. Avis and K. Fukuda. A pivoting algorithm for convex hull and vertex enumeration of arrangements and polyhedra. *Discrete Comput. Geom.*, 8(3):295 – 313, 1992.

[5] C. B. Barber, D. P. Dobkin, and H. T. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469 – 483, 1996.

[6] C. B. Barber and H. T. Huhdanpaa. *Qhull 2003.1 Homepage*, 2003. http://www.qhull.org.

[7] A. Barvinok. Polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Math of Operations Research*, 19:769–779, 1994.

[8] B. Becker, M. Behle, F. Eisenbrand, and R. Wimmer. BDDs in a branch and cut framework. In S. Nikoletseas, editor, *Experimental and Efficient Algorithms, Proceedings of the 4th International Workshop on Efficient and Experimental Algorithms (WEA '05)*, volume 3503 of *Lecture Notes in Computer Science*, pages 452–463. Springer, 2005.

[9] M. Behle. *Another Zero One Vertex Enumeration tool 1.0 Homepage*, 2006. http://www.mpi-inf.mpg.de/~behle/azove.html.

[10] R. E. Bixby, E. A. Boyd, and R. R. Indovina. MIPLIB: A test set of mixed integer programming problems. *SIAM News*, 25(2), 1992.

[11] D. Bremner. Incremental convex hull algorithms are not output sensitive. In *Proceedings of the 7th International Symposium on Algorithms and Computation*, volume 1178 of *Lecture Notes in Computer Science*, pages 26 – 35. Springer, 1996.

[12] D. Bremner, K. Fukuda, and A. Marzetta. Primal-dual methods for vertex and facet enumeration. *Discrete Comput. Geom.*, 20(3):333 – 357, 1998.

[13] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35:677–691, 1986.

[14] M. R. Bussieck and M. E. Lübbecke. The vertex set of a 0/1-polytope is strongly P-enumerable. *Computational*

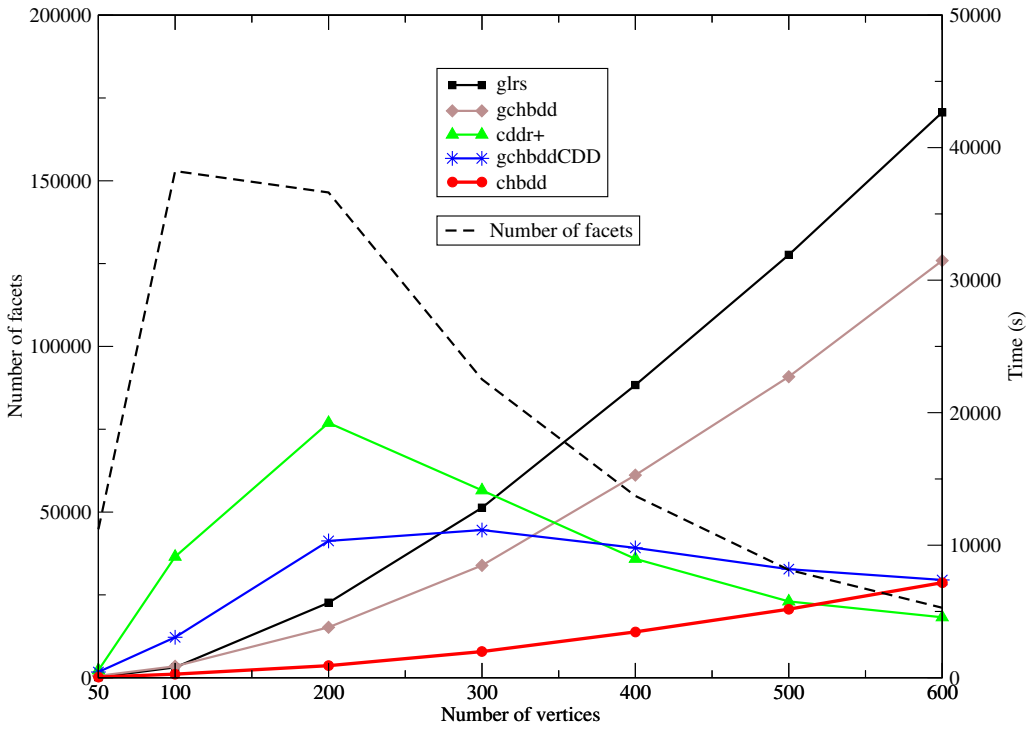


Figure 4: Comparison on randomly generated instances in dimension 11

Geometry, 11(2):103 – 109, 1998.

[15] D. R. Chand and S. S. Kapur. An algorithm for convex polytopes. *J. Assoc. Comput. Mach.*, 17(78):78 – 86, 1970.

[16] T. Christof and A. Löbel. *POlyhedron Representation Transformation Algorithm 1.4.0 Homepage*, 1997. http://www.zib.de/Optimization/Software/Porta/.

[17] K. Fukuda. *cdd 0.77 and cdd+ 0.94b Homepage*, 2003. http://www.ifor.math.ethz.ch/~fukuda/cdd_home.

[18] K. Fukuda and A. Prodon. Double description method revisited. In *Selected papers from the 8th Franco-Japanese and 4th Franco-Chinese Conference on Combinatorics and Computer Science*, volume 1120 of *Lecture Notes in Computer Science*, pages 91 – 111. Springer, 1995.

[19] E. Gawrilow and M. Joswig. Polymake: A framework for analyzing convex polytopes. In G. Kalai and G. M. Ziegler, editors, *Polytopes – Combinatorics and Computation*, pages 43 – 74. Birkhäuser, 2000.

[20] T. Granlund. *GNU Multiple Precision Arithmetic Library 4.1.4 Homepage*, 2004. http://swox.com/gmp.

[21] ILOG. *CPLEX 9.0 Homepage*, December 2003. http://www.ilog.com.

[22] V. Kaibel and R. Mechtel. Revlex-initial 0/1-polytopes. Preprint. To appear in: J. Comb. Theory, Ser. A, 2005.

[23] C. Y. Lee. Representation of switching circuits by binary-decision programs. *The Bell Systems Technical Journal*, 38:985 – 999, 1959.

[24] J. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida. Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation*, 38(4):1273 – 1302, 2004.

[25] M. Lübbecke. *Zerone 1.81 Homepage*, 1999. http://www.math.tu-bs.de/mo/research/zerone.html.

[26] G. Rote. Degenerate convex hulls in high dimensions without extra storage. In *Proceedings of the 8th annual Symposium on Computational Geometry*, pages 26 – 32, 1992.

[27] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley, 1986.

[28] R. Seidel. Convex hull computations. In J. Goodman and J. O'Rouke, editors, *Handbook of Discrete and Computational Geometry*, chapter 19. CRC Press, 1997.

[29] F. Somenzi. *CU Decision Diagram Package Release 2.4.1 Homepage*. Department of Electrical and Computer Engineering, University of Colorado at Boulder, May 2005. http://vlsi.colorado.edu/~fabio/CUDD.

[30] G. F. Swart. Finding the convex hull facet by facet. *J. Algorithms*, 6:17 – 48, 1985.

[31] I. Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, PA, 2000.

[32] G. M. Ziegler. *Lectures on Polytopes*. Springer, 1995.