Three Kinds of Integer Programming Algorithms based on Barvinok's Rational Functions *

J.A. De Loera, D. Haws, R. Hemmecke, P. Huggins, R. Yoshida

November 13, 2003

1 Introduction

This paper presents three algebraic-analytic algorithms for solving integer and mixed integer programming problems. We report both theoretical and experimental results. We use the generating function techniques of A. Barvinok [3] and the recent advances by Barvinok and Woods [5].

In 1993 A. Barvinok gave an algorithm that *counts* lattice points in convex rational polyhedra in polynomial time when the dimension of the polytope is fixed (see [3, 4] and the references within). Originally, Barvinok's counting algorithm relied on H. Lenstra's polynomial time algorithm for integer programming in a fixed number of variables [13], but shortly after Barvinok's breakthrough, Dyer and Kannan [10] showed that this step can be replaced by a short-vector computation using the LLL algorithm. Therefore, using binary search, one can turn Barvinok's counting oracle into an algorithm that solves integer programming problems with a fixed number of variables in polynomial time (i.e. by counting the number of lattice points in P that satisfy $c \cdot x \geq \alpha$, we can narrow the range for the maximum value of $c \cdot x$, then we iteratively look for the largest α where the count is non-zero). This idea was proposed by Barvinok in [4]. We call this IP algorithm the *BBS algorithm*.

More recently, J.B. Lasserre outlined a very easy asymptotic heuristic method for solving integer programs [12], or at least providing an upper bound on the optimal value, which is also based on Barvinok's rational functions (it comes without complexity guarantees). Unfortunately, Lasserre's criteria, needed to find an optimum value, often fail in practice. Within this topic we make three contributions:

1. In Section 2 we present yet another way to use Barvinok's rational functions to solve integer programs, the (A, b, c)-*test set algorithm*. This time

*Research supported by NSF Grant DMS-0309694 and by NSF VIGRE Grant No. DMS-0135345.

the rational functions encode a *test set*. A test set for a family of integer programs is a finite collection of integral vectors with the property that every feasible non-optimal solution of any integer program in the family can be improved by adding a vector in the test set. There has been considerable activity in the area of test sets and augmentation methods (e.g. Graver and Gröbner bases, integral basis method, etc.) [2, 15]. Here we promote a new way of looking at test sets, as explained in [8, 11]: First, encode the whole test set in a "short" sum of rational functions. Second, instead of improving one step at a time, many times, the optimum can be obtained via a single Hadamard product of two rational functions. We obtain the following complexity results:

Theorem 1 *Let $A \in \mathbb{Z}^{m\times d}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^d$, and assume that number of variables d is fixed. Given the mixed-integer programming problem*

maximize $c{\cdot}x$ *subject to* $x \in \{x | Ax \leq b, x \geq 0,\ x_i \in \mathbb{Z}$ *for* $i \in J \subset \{1, \ldots, d\}\}$,

(A) *We can use rational functions to encode the set of vectors (the (A, b, c)-test set):*

$\{u - v | u$ *is a* $c-$ *optimal solution,* v *feasible solution,* $u, v \in \mathbb{Z}^d\}$,

and then solve the MIP problem in time polynomial in the size of the input.

(B) *More strongly, the (A, b, c) test set can be replaced by smaller test sets, such as Graver bases or reduced Gröbner bases.*

2. We improve Lasserre's heuristic and give a third deterministic IP algorithm based on Barvinok's rational function algorithms, the *digging algorithm*. In this case the algorithm can have an exponential number of steps even for fixed dimension, but performs well in practice. See Section 3 for details.

3. We implemented the *BBS algorithm* and the *digging algorithm* in the second release of the computer software `LattE` (see [7, 8, 9]). We solved several challenging knapsack problems and compared the performance of `LattE` with the mixed-integer programming solver CPLEX version 6.6. See Section 4 for details.

2 The (A, b, c) Test Set Algorithm

In all our discussions below, the input data are an $m \times d$ integral matrix A and an integral m-vector b. For simplicity we assume it describes a polytope $P = \{x \in \mathbb{R}^d | Ax \leq b, x \geq 0\}$. Our point of view is that lattice points will be encoded as the exponent vectors of monomials. For example, $(2, -11)$ is represented by $z_1^2 z_2^{-11}$. The set of all lattice points in a polyhedron P will be represented by the multivariate generating function $f(P; z) = \sum_{a \in P \cap \mathbb{Z}^d} z_1^{a_1} z_2^{a_2} \ldots z_d^{a_d}$ with one

monomial in the sum per lattice point inside P. The crucial fact from Barvinok's theory is that the exponentially large sum $f(P;z)$ can be written as a short sum of rational functions of the form

$$f(P;z) = \sum_{i \in I} E_i \frac{z^{u_i}}{\prod_{j=1}^{d} (1 - z^{v_{ij}})}, \tag{1}$$

where I is a finite indexing set, and where $E_i \in \{1, -1\}$ and $u_i, v_{ij} \in \mathbb{Z}^d$ for all i and j. More recently Barvinok and Woods (2003) further showed how to carry out Boolean operations with sets of lattice points when they are encoded as rational functions:

Lemma 2 (Theorem 3.6 in [5]) *Let S_1, S_2 be finite subsets of $\mathbb{Z}^d$, for d fixed. Let $f(S_1;x)$ and $f(S_2;x)$ be their generating functions, given as short rational functions with at most k binomials in each denominator. Then there exists a polynomial time algorithm, which, given $f(S_i,x)$, computes*

$$f(S_1 \cap S_2; x) \quad = \quad \sum_{i \in I} \gamma_i \cdot \frac{x^{u_i}}{(1 - x^{v_{i1}}) \ldots (1 - x^{v_{is}})}$$

with $s \leq 2k$, where the γ_i are rational numbers and u_i, v_{ij} are nonzero integral vectors.

We will use this *Intersection Lemma* to extract special monomials present in the expansion of a generating function. The essential step in the intersection algorithm is the use of the *Hadamard product* [5, Definition 3.2] and a special monomial substitution. The Hadamard product is a bilinear operation on rational functions (we denote it by $*$). The computation is carried out for pairs of summands as in [5]. Note that the Hadamard product $m_1 * m_2$ of two monomials m_1, m_2 is zero unless $m_1 = m_2$. Another key subroutine introduced by Barvinok and Woods is the following *Projection Theorem.*

Lemma 3 (Theorem 1.7 in [5]) *Assume the dimension d is a fixed constant. Consider a rational polytope $P \subset \mathbb{R}^d$ and a linear map $T : \mathbb{Z}^d \to \mathbb{Z}^k$. There is a polynomial time algorithm which computes a short representation of the generating function $f\big(T(P \cap \mathbb{Z}^d); z\big)$.*

To extract an explicit optimal solution we need the following lemma.

Lemma 4 (Lemma 7 in [8]) *Assume the dimension d is fixed. Let $S \subset \mathbb{Z}_+^d$ be finite. Suppose the polynomial $f(S;z) = \sum_{\beta \in S} z^\beta$ is represented as a short rational function and let c be a cost vector. We can extract the (unique) lexicographic largest leading monomial of $f(S;z)$ with respect to c, in polynomial time.*

Proof of Theorem 1: For lack of space, we only show the proof of part (A). For the details of part (B) see [8]. We first explain how to solve integer programs (all variables are integral). This part of the proof is essentially the proof of Lemma 3.1 given in [11] for the case $Ax = b,\ x \geq 0$, instead of $Ax \leq b$, but we emphasize the fact that b is fixed here. We will see how the techniques can be extended to mixed integer programs later.

Using Barvinok's algorithm in [4], compute the following generating function in $2d$ variables:

$$f(x,y) \quad = \quad \sum \{\, x^u y^v \ :\ Au \leq b,\ Av \leq b,\ u,v \geq 0,\ \text{and } c\cdot u - c\cdot v \geq 1 \}.$$

This is possible because we are clearly dealing with the lattice points of a rational polyhedron. The monomial expansion of $f(x,y)$ has a clear order on the variables: $x^u y^v$ where $c \cdot u > c \cdot v$. Hence v is not an optimal solution. In fact, optimal solutions will never appear as exponents in the y variables.

Let now $g(y)$ be the projection of $f(x,y)$ onto the y-variables variables. Thus $g(y)$ is encoding all non-optimal feasible integral vectors (because the exponent vectors of the x's are better feasible solutions, by construction), and it can be computed from $f(x,y)$ in polynomial time by Lemma 3. Let $f(x,y)$ and $g(x)$ be as above and compute the Hadamard product

$$H(x,y) \quad := \quad f(x,y) * \left[\left(\prod_{i=1}^{d} \frac{1}{1-x_i} - g(x) \right) \left(\prod_{i=1}^{d} \frac{1}{1-y_i} \right) \right].$$

This is the sum over all monomials $x^u y^v$ where u is an optimal solution. The reader should note that the vectors $u - v$ form a test set (an enormous one), since they can be used to improve from any feasible non-optimal solution v. This set is what we called the (A,b,c)-test set. It should be noted that one may replace $H(x,y)$ by a similar encoding of other test sets, like the Graver test set or a Gröbner basis (see [8] for details).

We now use $H(x,y)$ as one would use a traditional test set for finding an optimal solution: Find a feasible solution a inside the polyhedron $P = \{x | Ax \leq b,\ x \geq 0\}$ using Lemma 4 and Barvinok's Equation (1) for the polyhedron P. Improve or augment to an optimal solution by computing the Hadamard product

$$H(x,y) * \left(y^a \prod_{i=1}^{d} \frac{1}{1-x_i} \right).$$

The result is the set of monomials of the form $x^u y^a$ where u is an optimal solution. One monomial of the set, say the lexicographic largest, can be obtained by applying Lemma 4. This concludes the proof of the case when all variables are integral.

Now we look at the mixed integer programming case, where only x_i with $i \in J \subset \{1,\ldots,d\}$ are required to be integral. Without loss of generality, we may assume that $J = \{r,\ldots,d\}$ for some r, $1 \leq r \leq d$. Thus, splitting A into $(B|C)$,

we may write the polyhedron P as $\{(x,x')|Bx+Cx' \leq b,\ x,x' \geq 0\}$ where the variables corresponding to B are not demanded to be integral. Consider a vertex optimal solution $\bar{x}$ to the mixed integer problem. The first key observation is that its fractional part can be written as $\bar{x}_J = \hat{B}^{-1}(b - C\bar{x}')$ where $b - C\bar{x}'$ is an integer vector. Here $\hat{B}^{-1}$ denotes the inverse of a submatrix of B. This follows from the theory of linear programming, when we solve the mixed integer program for fixed $x' = \bar{x}'$.

The denominators appearing are then contributed by $\hat{B}^{-1}$. Then every appearing denominator is a factor of M, the least common multiple of all determinants of a square submatrix of A. It is clear M can be computed in polynomial time in the size of the input. This complexity bound holds, since the number of such square submatrices is bounded by a polynomial in m, the number of rows of A, of degree d, the number of columns of A. Moreover, each of these determinants can be computed in time polynomial in the size of the input, and therefore, M itself can be computed in time polynomial in the size of the input in fixed dimension d. Thanks to this information, we know that if we dilate the original polyhedron P by M, the optimal solutions of the mixed integer program become, in the dilation MP, optimal integral solutions of the problem

$$\text{maximize } c \cdot x \text{ subject to } \quad x \in MP,\ x \in \mathbb{Z}^d$$

with the additional condition that the coordinates with index in J are multiples of M. Ignoring this condition involving multiples of M for a moment, we see that, as we did before, we can obtain an encoding of *all* optimal improvements as a generating function $H(x,y)$. To extract those vectors whose coordinates indexed by J are multiples of M, we only need to intersect (Hadamard product again) our generating function $H(x,y)$ with the generating function $\left(\prod_{i \notin J} \frac{1}{1-x_i}\right)\left(\prod_{i \in J} \frac{1}{1-x_i^M}\right)$. Then only those vectors whose coordinates indexed by J are multiples of M remain. This completes the proof of the theorem. □

3 The Digging Algorithm

In what follows we present a strengthening of Lasserre's heuristic and discuss how to use Barvinok's short rational functions to solve integer programs using digging. We consider the integer programming problem maximize$\{c \cdot x|Ax \leq b, x \geq 0, x \in \mathbb{Z}^d\}$, where $c \in \mathbb{Z}^d$ is arbitrary, and $A \in \mathbb{Z}^{m \times d}$, $b \in \mathbb{Z}^m$ are fixed. We assume that the input system of inequalities $Ax \leq b, x \geq 0$ defines a bounded polytope $P \subset \mathbb{R}^d$, such that $P \cap \mathbb{Z}^d$ is nonempty. As before, all integer points are encoded as a short rational function $f(P;z)$ in Equation (1) for P, where the rational function is given in Barvinok's form. Remember that if we were to expand Equation (1) into monomials (generally a very bad idea!) we would get $f(P;z) = \sum_{\alpha \in P \cap \mathbb{Z}^d} z^\alpha$, where z^α denotes ${z_1}^{\alpha_1}{z_2}^{\alpha_2} \ldots {z_d}^{\alpha_d}$. For a given $c \in \mathbb{Z}^d$, we make the substitution $z_i = t^{c_i}$, Equation (1) yields a univariate rational function in t:

$$f(P;t) = \sum_{i \in I} E_i \frac{t^{c \cdot u_i}}{\prod_{j=1}^{d} (1 - t^{c \cdot v_{ij}})}. \tag{2}$$

The key observation is that if we make that substitution directly into the monomial expansion of $f(P;z)$, we have that $z^{\alpha} \rightarrow t^{c \cdot \alpha}$. Moreover we would obtain the relation

$$f(P;t) = \sum_{\alpha \in P \cap \mathbb{Z}^d} t^{c \cdot \alpha} = kt^M + (\text{lower degree terms}), \tag{3}$$

where M is the optimal value of our integer program and where k counts the number of optimal integer solutions. Unfortunately, in practice, M and the number of lattice points in P may be huge and we need to avoid the monomial expansion step altogether. All computations have to be done by manipulating short rational functions.

Lasserre [12] suggested the following approach: For $i \in I$, define sets η_i by $\eta_i = \{j \in \{1, ..., d\} | c \cdot v_{ij} > 0\}$, and define vectors w_i by $w_i = u_i - \sum_{j \in \eta_i} v_{ij}$. Let n_i denote the cardinality of η_i. Now define $M = \max\{c \cdot w_i | i \in I\}$, $S = \{i \in I | c \cdot w_i = M\}$ and set $\sigma = \sum_{i \in S} E_i (-1)^{n_i}$. Note that M simply denotes the highest exponent of t appearing in the expansions of the rational functions defined for each $i \in I$ in (2). The number σ is in fact the sum of the coefficients of t^M in these expressions, that is, σ is the coefficient of t^M in $f(P;t)$. Now with these definitions and notation we can state the following result proved by Lasserre [12].

Theorem 5 (Theorem 3.1 in [12]) *If $c \cdot v_{ij} \neq 0$ for all $i \in I, j \in \{1, \ldots, d\}$, and if $\sigma \neq 0$, then M is the optimal value π of the integer program* maximize$\{c \cdot x | Ax \leq b, x \geq 0, x \in \mathbb{Z}^d\}$.

When the hypotheses of Theorem 5 are met, from an easy inspection, we could recover the optimal value of an integer program. If we assume that c is random, the first condition is not difficult to obtain. Unfortunately, our computational experiments (see Section 4) indicate that the condition $\sigma \neq 0$ is satisfied only occasionally. Thus an improvement on the approach that Lasserre proposed is needed to make the heuristic terminate in all instances. Here we explain the details of an algorithm that *digs* for the coefficient of the next highest appearing exponent of t. For simplicity our explanation assumes the easy-to-achieve condition $c \cdot v_{ij} \neq 0$.

As before, take Equation (1) computed via Barvinok's algorithm. Now, for the given $c \in \mathbb{Z}^d$, we make the substitutions $z_k = y_k t^{c_k}$, for $k = 1, \ldots, d$. Then substitution into (1) yields, for the right-hand side of Equation (1), a sum of multivariate rational functions in the vector variable y and scalar variable t:

$$g(P;y,t) = \sum_{i \in I} E_i \frac{y^{u_i} t^{c \cdot u_i}}{\prod_{j=1}^{d} (1 - y^{v_{ij}} t^{c \cdot v_{ij}})}. \tag{4}$$

On the other hand, the substitution on the left-side of Equation (1) gives a sum of monomials, also shown grouped in terms of t.

$$g(P; y, t) = \sum_{\alpha \in P} y^{\alpha} t^{c \cdot \alpha} = \sum_{\alpha \in \mathbb{Z}^d, n \in \mathbb{Z}} a_{\alpha,n} y^{\alpha} t^{n}. \tag{5}$$

Both equations, (5) and (4), represent the same function $g(P; y, t)$, so the corresponding coefficients of the Laurent series expansion of both expressions must be equal. What Barvinok's algorithm provides us is the right-hand side of Equation (4) and we need to manipulate it to obtain the coefficient of highest degree in t from Equation (5) (because P is a polytope, there will be a highest degree monomial, from which we recover the optimal value). The process is the following. Apply the identity

$$\frac{1}{1 - y^{v_{ij}} t^{c \cdot v_{ij}}} = \frac{-y^{-v_{ij}} t^{-c \cdot v_{ij}}}{1 - y^{-v_{ij}} t^{-c \cdot v_{ij}}} \tag{6}$$

to Equation (4), so that any v_{ij} such that $c \cdot v_{ij} > 0$ can be changed in "sign" to be sure that, for all v_{ij} in (4), $c \cdot v_{ij} < 0$ is satisfied (we may have to change some of the E_i, u_i and v_{ij} using our identity, but we abuse notation and still refer to the new signs as E_i and the new numerator vectors as u_i and the new denominator vectors as v_{ij}). Then, for each of the rational functions in the sum of Equation (4) compute a Laurent expansion of the form

$$E_i \, y^{u_i} t^{c \cdot u_i} \prod_{j=1}^{d} (1 + y^{v_{ij}} t^{c \cdot v_{ij}} + (y^{v_{ij}} t^{c \cdot v_{ij}})^2 + \ldots). \tag{7}$$

Add terms of the same degree in t from the different resultant series. We obtain the coefficients $a_{\alpha,n}$ appearing in the terms of the series (5). Thus, we have an algorithm to solve integer programs:

Algorithm: (*Digging Algorithm*):

Input: A, b, c.

Output: optimal value and optimal solution of maximize$\{c \cdot x | Ax \leq b, x \geq 0, x \in \mathbb{Z}^d\}$.

1. Use the identity (6) as necessary to enforce that all v_{ij} in (4) satisfy $c \cdot v_{ij} < 0$.

2. Via the expansion formulas (7), find (5) by calculating the terms' coefficients. Proceed in decreasing order with respect to the degree of t. This can be done because, for each series appearing in the expansion formulas (7), the terms of the series are given in decreasing order with respect to the degree of t.

3. Continue calculating the terms of the expansion (5), in decreasing order with respect to the degree of t, until a degree n of t is found such that for some $\alpha \in \mathbb{Z}^d$, the coefficient of $y^\alpha t^n$ is non-zero in the expansion (5).

4. Return "n" as the optimal value of the integer program and return α as an optimal solution.

We close this section mentioning that we can recover not only the optimal value, but also an explicit optimal solution. There is also a variation of the digging algorithm where instead of using Barvinok's rational function for the whole polytope, one uses the Barvinok rational function of the tangent cone at a chosen vertex (typically the vertex is LP relaxation optimal solution). This provides in practice speed ups in some of the cases (this is not reported here).

4 Computational Experiments

In this section we report our experience solving hard knapsack problems from [1, 6]. See Table 1 for the data used here. Their form is maximize $c \cdot x$ subject to $ax = b, x \geq 0, x \in \mathbb{Z}^d$, where $b \in \mathbb{Z}$ and where $a \in \mathbb{Z}^d$ with $\gcd(a_1, \ldots, a_d) = 1$. For the cost vector c, we choose the first d components of the vector $(213, -1928, -11111, -2345, 9123, -12834, -123, 122331, 0, 0)$. We compared the digging algorithm and the BBS algorithm, both implemented in `LattE` (available at `www.math.ucdavis.edu/~latte`), with `CPLEX` version 6.6. The computations were done on a 1 GHz Pentium PC running Red Hat Linux. Table 2 provides the optimal values and an optimal solution for each problem. As it turns out, there is exactly one optimal solution for each problem.

Problem	a										b
cuww1	12223	12224	36674	61119	85569						89643482
cuww2	12228	36679	36682	48908	61139	73365					89716839
cuww3	12137	24269	36405	36407	48545	60683					58925135
cuww4	13211	13212	39638	52844	66060	79268	92482				104723596
cuww5	13429	26850	26855	40280	40281	53711	53714	67141			45094584
prob1	25067	49300	49717	62124	87608	88025	113673	119169			33367336
prob2	11948	23330	30635	44197	92754	123389	136951	140745			14215207
prob3	39559	61679	79625	99658	133404	137071	159757	173977			58424800
prob4	48709	55893	62177	65919	86271	87692	102881	109765			60575666
prob5	28637	48198	80330	91980	102221	135518	165564	176049			62442885
prob6	20601	40429	40429	45415	53725	61919	64470	69340	78539	95043	22382775
prob7	18902	26720	34538	34868	49201	49531	65167	66800	84069	137179	27267752
prob8	17035	45529	48317	48506	86120	100178	112464	115819	125128	129688	21733991
prob9	3719	20289	29067	60517	64354	65633	76969	102024	106036	119930	13385100
prob10	45276	70778	86911	92634	97839	125941	134269	141033	147279	153525	106925262

Table 1: knapsack problems.

With one exception, CPLEX 6.6. could not solve the given problems. Note that whenever the digging algorithm found the optimal value, it did so much faster than the BBS algorithm. This is interesting, as the worst-case complexity for the digging algorithm is exponential even for fixed dimension, while BBS has polynomial complexity in fixed dimension. The digging algorithm fails to find a solution for problems prob2, prob3, and prob5. What happens is that

Problem	Value	Solution	Digging	BBS	CPLEX 6.6
cuww1	1562142	[7334 0 0 0 0]	0.4 sec.	414 sec.	> 1.5h
cuww2	-4713321	[3 2445 0 0 0 0]	> 3.5h	6,600 sec.	> 0.75h
cuww3	1034115	[4855 0 0 0 0 0]	1.4 sec.	6,126 sec.	> 0.75h
cuww4	-29355262	[0 0 2642 0 0 0 0]	> 1.5h	38,511 sec.	> 0.75h
cuww5	-3246082	[1 1678 1 0 0 0 0 0]	> 1.5h	> 80h	> 0.75h
prob1	9257735	[966 5 0 0 1 0 0 74]	51.4 sec.	> 3h	> 1h
prob2	3471390	[853 2 0 4 0 0 0 27]	24.8 sec.	> 10h	> 0.75h
prob3	21291722	[708 0 2 0 0 0 1 173]	48.2 sec.	> 12h	> 1.5h
prob4	6765166	[1113 0 7 0 0 0 0 54]	34.2 sec.	> 5h	> 1.5h
prob5	12903963	[1540 1 2 0 0 0 0 103]	34.5 sec.	> 5h	> 1.5h
prob6	2645069	[1012 1 0 1 0 1 0 20 0 0]	143.2 sec.	> 4h	> 2h
prob7	22915859	[782 1 0 1 0 0 0 186 0 0]	142.3 sec.	> 4h	> 1h
prob8	3546296	[1 385 0 1 1 0 0 35 0 0]	469.9 sec.	> 3.5h	> 2.5h
prob9	15507976	[31 11 1 1 0 0 0 127 0 0]	1,408.2 sec.	> 11h	4.7 sec.
prob10	47946931	[0 705 0 1 1 0 0 403 0 0]	250.6 sec.	> 11h	> 1h

Table 2: Optimal values, optimal solutions, and running times for each problem.

the expansion step becomes costly when more coefficients have to be computed. In these three examples, we computed coefficients for more than 2,500,000, 400,000, and 100,000 powers of t; all turning out to be 0. The Digging algorithm is slower than CPLEX in problem prob9 because during the execution of Barvinok's unimodular cone decomposition (see pages 15 and 16 of [4]) more than 160,000 cones are generated, leading to an enormous rational function for $f(P;t)$. Moreover, for prob9 more than 3,500 coefficients turned out to be 0, before a non-zero leading coefficient was detected. Finally, in problems cuww1, cuww3, prob2, prob3, prob4, prob6, and prob8, no digging was necessary at all, that is, Lasserre's condition did not fail here. For all other problems, Lasserre's condition did fail and digging steps were necessary to find the first non-vanishing coefficient in the expansion of $f(P;t)$.

References

[1] Aardal, K., Lenstra, A.K., and Lenstra, H.W. Jr. *Hard equality constrained integer knapsacks,* preliminary version in W.J. Cook and A.S. Schulz (eds.), Integer Programming and Combinatorial Optimization: 9th International IPCO Conference, Lecture Notes in Computer Science vol. 2337, Springer-Verlag, 2002, pp 350–366.

[2] Aardal, K., Weismantel, R., and Wolsey, L. *Non-Standard Approaches to Integer Programming* Discrete Applied Mathematics 123, 2002, 5–74

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

[4] Barvinok, A.I. and Pommersheim, J. *An algorithmic theory of lattice points in polyhedra*, in: *New Perspectives in Algebraic Combinatorics* (Berkeley, CA, 1996-1997), 91–147, Math. Sci. Res. Inst. Publ. 38, Cambridge Univ. Press, Cambridge, 1999.

[5] Barvinok, A.I. and Woods, K. *Short rational generating functions for lattice point problems*, available at arXiv.math.CO.0211146. J. Amer. Math. Soc. 16 (2003), 957-979.

[6] Cornuéjols, G., Urbaniak, R., Weismantel, R., and Wolsey, L.A. *Decomposition of integer programs and of generating sets.* R. E. Burkard, G. J. Woeginger, eds., *Algorithms*–ESA 97. Lecture Notes in Computer Science 1284, Springer-Verlag, 92–103, 1997.

[7] De Loera, J.A, Hemmecke, R., Tauzer, J., and Yoshida, R. *Effective lattice point counting in rational convex polytopes.* To appear in Journal of Symbolic Computation.

[8] De Loera, J.A, Haws, D., Hemmecke, R., Huggins, P., Sturmfels, B., and Yoshida, R. *Short rational functions for toric algebra and applications*, available at math arXiv math.CO/0307350, 2003.

[9] De Loera, J.A., Haws, D., Hemmecke, R., Huggins, P., Tauzer, J., Yoshida, R. *A User's Guide for* `LattE` *v1.1*, 2003, software package `LattE` is available at `http://www.math.ucdavis.edu/~latte/`

[10] Dyer, M. and Kannan, R. *On Barvinok's algorithm for counting lattice points in fixed dimension*, Math of Operations Research 22 (1997) 545–549.

[11] Hosten, S. and Sturmfels, B. *Computing the integer programming gap*, available at math arXiv math.OC/0301266, 2003.

[12] Lasserre, J.B. *Integer programming, Barvinok's counting algorithm and Gomory relaxations*, to appear in Operations Research Letters.

[13] Lenstra, H.W. *Integer Programming with a fixed number of variables* Mathematics of Operations Research, 8, 1983, 538–548.

[14] Schrijver, A. *Theory of Linear and Integer Programming.* Wiley-Interscience, 1986.

[15] Thomas, R. *Algebraic methods in integer programming*, Encyclopedia of Optimization (eds: C. Floudas and P. Pardalos), Kluwer Academic Publishers, Dordrecht, 2001.