A Computational Study of Integer Programming Algorithms based on Barvinok's Rational Functions

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

[a] *University of California at Davis, One Shields Ave. Davis, CA 95616, USA*
`http://www.math.ucdavis.edu/∼latte`

Abstract

This paper discusses five algorithms to solve linear integer programming problems that use the rational function techniques introduced by A. Barvinok. We report on the first ever experimental results based on these techniques.

1 Introduction

In 1993 Barvinok gave an algorithm to *count* the lattice points in a convex rational polytope in polynomial time on the input size when the dimension of the polytope is fixed (see Barvinok (1994); Barvinok and Pommersheim (1999); Barvinok and Woods (2003) and the references within). The purpose of this article is to explain five algorithms in which Barvinok's techniques can be used to *optimize*; namely, to find the optimal value of

$$\text{maximize } \ c \cdot x \text{ subject to } Ax \leq b, \ x \geq 0, \ x \in \mathbb{Z}^d, \tag{IP}$$

where the input data are an $m \times d$ integral matrix A, an integral m-vector b, and an integral d-vector c. All five algorithms were at least partially implemented. The three most successful algorithms, the *BBS algorithm*, the *digging algorithm*, and the *single cone digging algorithm*, appear now in the second release of the computer software `LattE` (see De Loera et al. (2004a,b,c)). We solved several challenging knapsack problems and compared the performance of `LattE` with the mixed-integer programming solver CPLEX version 6.6.

For simplicity, and without loss of generality, we assume that the input data describe a nonempty full-dimensional convex polytope $P = \{x \in \mathbb{R}^d : Ax \leq b, x \geq 0\}$. Lattice points will be encoded as the exponent vectors of monomials. For example, $(2, -11)$ is represented by $z_1^2 z_2^{-11}$. It is important to note that we will often write z^a as a short notation for the multivariate monomial $z_1^{a_1} z_2^{a_2} \dots z_d^{a_d}$. The following theorem of Barvinok is the starting point of this article:

Lemma 1 (Theorem 4.4 in Barvinok and Pommersheim (1999)) *Assume d, the dimension, is fixed. Given a convex rational polytope $P = \{x : Ax \leq b, x \geq 0\}$, the multivariate generating function $f(P; z) = \sum_{a \in P \cap \mathbb{Z}^d} z^a$ can be written in polynomial time in 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 polynomial-size indexing set, and where $E_i \in \{1, -1\}$ and $u_i, v_{ij} \in \mathbb{Z}^d$ for all i and j.

Thus, the exponentially-large sum of monomials in the generating function can be written as a *short* sum of rational functions. Intuitively, to obtain $E_i \in \{1, -1\}$ and $u_i, v_{ij} \in \mathbb{Z}^d$ in (1), Barvinok's algorithm decomposes the polytope P into simple cones. The vectors u_i, v_{ij} are the rays and vertices of the pieces of the decomposition, and E_i is plus or minus one depending whether we add or subtract the cone. For a detailed description of Barvinok's algorithm proving lemma 1, examples, and our implementation of it see Barvinok and Pommersheim (1999); De Loera et al. (2004a). We call such an expression a *Barvinok rational function.* Consider for example the polygon presented in Figure 1. This is the feasible region of the integer program maximize $100x + 90y$ subject to $x + y \leq 100,\ x \leq 50,\ x, y \geq 0,\ x, y \in \mathbb{Z}^d$.

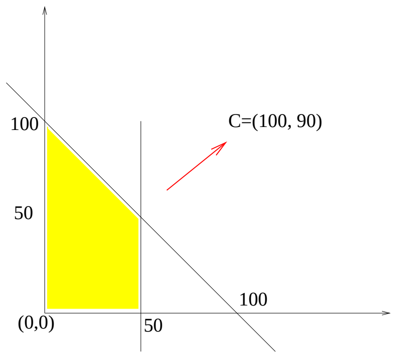

Fig. 1. A simple integer program

The associated Barvinok rational function is in this case

$$\frac{1}{(1 - z_1)(1 - z_2)} + \frac{z_1^{50}}{(1 - z_1^{-1})(1 - z_2)} + \frac{z_2^{100}}{(1 - z_1^{-1})(1 - z_2)} + \frac{z_1^{50} z_2^{50}}{(1 - z_1^{-1})(1 - z_1^{-1} z_2)}.$$

In this easy example, the cone decomposition of the polytope is trivial and the exponent vectors of denominators are the ray vectors of the four cones defining each vertex of the quadrilateral. The numerators are the just the four vertices. Of course, in general, Barvinok's decomposition is more complicated. We now describe the integer algorithms in Sections 2 and 3. Section 4 presents the computational tests we performed and some final comments.

2 Algorithm: Binary Search

We begin with the most straightforward integer programming algorithm. It is an immediate consequence of Lemma 1, no extra tools needed: Clearly using binary search, one can turn *any* feasibility or counting oracle into an algorithm that solves Problem (IP). By counting the number of lattice points in P that satisfy $c \cdot x \geq M$, we can narrow the range for the maximum value of $c \cdot x$, then we iteratively look for the largest integer M where the count is non-zero. This idea was proposed in Barvinok and Pommersheim (1999):

Algorithm: (*BBS*):

Input: $A \in \mathbb{Z}^{m \times d}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^d$.

Output: The optimal value $M = \text{maximize } \{c \cdot x : Ax \leq b, x \geq 0, x \in \mathbb{Z}^d\}$.

```
(1) Using the linear programming relaxations of problem (IP) let $M$ equal to
    $\lceil \max\{c \cdot x : Ax \leq b, x \geq 0\} \rceil$ and let $m$ equal to $\lfloor \min\{c \cdot x : Ax \leq b, x \geq 0\} \rfloor$.
    Thus $[m, M]$ is the initial range for the binary search. Let $P = \{x : Ax \leq b, x \geq 0\}$.
(2) While $M > m$ do
        $new := \lceil \frac{M+m}{2} \rceil$.
        Using Barvinok's algorithm compute $q_{new} = |P \cap \{x : cx \geq new\} \cap \mathbb{Z}^d|$.
        If $q_{new} > 0$ then set $m = new$,
        else $M = new - 1$.
    Return $M$.
```

We stress that Barvinok's original counting algorithm relied on H.W. Lenstra Jr.'s polynomial time algorithm for integer programming in a fixed number of variables (Lenstra, 1983), but shortly after Barvinok's breakthrough, Dyer and Kannan (Dyer and Kannan, 1997) showed that this dependence can be avoided by a short-vector computation using the *LLL* algorithm (Schrijver, 1986). This pure version is what is implemented in our software `LattE` (De Loera et al., 2004a,c). Therefore the *BBS* algorithm gives a new proof of the polynomiality of linear integer programming in fixed dimension. To our knowledge this is the first time such an algorithm has been fully implemented.

3 Monomial Substitution for Integer Optimization

All lattice points of a polytope P can be encoded as the Barvinok rational function $f(P;z)$ as in Equation (1). Remember that if we were to expand Equation (1) into monomials we would get $f(P;z) = \sum_{\alpha \in P \cap \mathbb{Z}^d} z^\alpha$. But this expansion is generally a very expensive computational operation, because there might be exponentially many monomials. All computations have to be done by manipulating only the rational functions that describe Equation (1).

Barvinok's encoding is not only compact, but also parametric. For example, there are, for each positive integer v, $(v+1)^2$ lattice points in the square with vertices $(0,0), (v,0), (0,v), (v,v)$. Its generating function, regardless of the value of v, is represented by the sum

$$\frac{1}{(1-z_1)(1-z_2)} + \frac{{z_1}^v}{(1-{z_1}^{-1})(1-z_2)} + \frac{{z_2}^v}{(1-{z_2}^{-1})(1-z_1)} + \frac{{z_1}^v {z_2}^v}{(1-{z_1}^{-1})(1-{z_2}^{-1})}.$$

From the Barvinok rational function $f(P;z)$ of a polytope P, the number of lattice points in P is the limit when the vector $(z_1, \ldots, z_d)$ goes to $(1,1,\ldots,1)$. We cannot simply substitute this vector because it is a singularity of the rational function expression. What we use instead is elementary complex analysis, namely residue calculations, to extract these values. See De Loera et al. (2004a) for practical details on how to carry on such calculations.

For integer programming it will be extremely important to be able to substitute general monomials $y_1^{a_1} y_2^{a_2} \ldots y_n^{a_n}$ into the variables z_i. We rely on the following result of Barvinok and Woods (Barvinok and Woods, 2003) to guarantee that the change of variables can be done efficiently using only rational functions and avoiding difficulties with singularities:

Lemma 2 (Theorem 2.6 in Barvinok and Woods (2003)) *Let us fix k, the number of binomials present in the denominator of a rational function. Given a rational function sum g of the form*

$$g(z) = \sum_{i \in I} \alpha_i \frac{z^{u_i}}{\prod_{j=1}^{k} (1 - z^{v_{ij}})},$$

where u_i, v_{ij} are integral d-dimensional vectors, α_i are rational numbers, and a monomial map $\psi : \mathbb{C}^n \longrightarrow \mathbb{C}^d$ given by the variable change $z_i \rightarrow y_1^{l_{i1}} y_2^{l_{i2}} \ldots y_n^{l_{in}}$ (with $l_{ij} \in \mathbb{Z}$) whose image does not lie entirely in the set of poles of $g(z)$, i.e. the set of roots of the denominators in $g(z)$, then there exists a polynomial time algorithm which computes the function $g(\psi(y))$ as a sum of rational functions of the same shape as $g(z)$.

For a given cost vector $c \in \mathbb{Z}^d$, if we use Lemma 2 to make the substitution $z_k = t^{c_k}$, Equation (1) yields, on one hand, a univariate rational function in t:

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

On the other hand, observe that if we make the substitution directly into the monomial sum expansion of $f(P;z)$, we have that the multivariate monomial $z_1^{a_1} z_2^{a_2} \ldots z_d^{a_d}$ becomes $t^{c \cdot a}$. We obtain the relation

$$f(P;t) = \sum_{a \in P \cap \mathbb{Z}^d} t^{c \cdot a} = k_M t^M + k_{M-1} t^{M-1} + k_{M-2} t^{M-2} + \ldots, \tag{3}$$

where M is the optimal value of our integer program and where k_s counts the number of feasible integer solutions with objective function value s. *After the monomial substitution, the IP maximum value equals the highest degree of the univariate polynomial $f(P;t)$.* If we have a way to compute the degree of this polynomial we have solved Problem (IP). For example, in the easy integer program of Figure 1, if we substitute $z_1 \to t^{100}$ and $z_2 \to t^{90}$, using the cost vector $c = (100, 90)$, and then expand the Barvinok rational function into monomials we would have t^{9500} + lower degree terms in t because 9500 is the optimal maximal value uniquely achieved at the point $(50, 50)$. There are in fact many distinct ways to find the degree of a "black-box" univariate polynomial (i.e. a polynomial which we can evaluate at rational numbers but we do not see explicitly its terms). It is worth presenting three independent methods that exemplify the main tools available:

3.1 Algorithm: Numerical Complex Integration

To find the optimum value of an integer program we can use the following elementary lemma of complex analysis (any complex analysis book has it, but we recommend Henrici (1974) as a practical reference):

Lemma 3 (Argument principle) *Let C be a simple closed curve in the complex plane that contains no roots of a polynomial $p(z)$. Then the number of roots of $p(z)$ inside the curve C, counted with multiplicity, equals*

$$\frac{1}{2\pi i}\left(\int_C \frac{p'(z)}{p(z)} dz\right).$$

By the fundamental theorem of algebra we know that the total number of roots of $p(z)$ equals the degree of $p(z)$. Thus Lemma 3 gives the degree of $p(z)$ when the curve C contains *all* the roots. In practice, we can consider a large square C centered at the origin of side $2N$. The number N needs to be large enough to contain all roots of our polynomial. Thus we need an upper bound on the absolute value of the roots. This upper bound is guaranteed by the following classical result of Cauchy (see Chapter VII of Marden (1966)):

Lemma 4 (Cauchy's bound on the absolute value of roots) *All the roots of the polynomial* $p(z) = a_n z^n + a_{n-1} z^{n-1} + \cdots + a_0$ *lie in the open disc*

$$\left\{ z \in \mathbb{C} : \ |z| < 1 + \max_{0 \le j \le n} \left| \frac{a_j}{a_n} \right| \right\}.$$

Now in our case the coefficients a_j are nothing else than the number of lattice points in a given "slice" of the polytope by a hyperplane $c \cdot x = j$. Thus, in general, $\frac{a_j}{a_n}$ is bounded by the total number of lattice points inside the polytope (a number we can compute with Barvinok's algorithm). Note that in practice, when the cost vector c is generic, all coefficients a_j are one and thus $N = 2$ suffices. Putting all these together:

Algorithm: (*Numerical Complex Integration*):

Input: $A \in \mathbb{Z}^{m \times d}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^d$.

Output: The optimal value $M = \text{maximize } \{c \cdot x : Ax \le b, x \ge 0, x \in \mathbb{Z}^d\}$.

(1) Using Barvinok's algorithm compute the rational function representation of the generating function $f(P; z)$ for $P = \{x \in \mathbb{R}^d : Ax \le b, x \ge 0\}$.
(2) Set N equal to one plus the number of lattice points in P. When the cost vector is generic $N = 2$ suffices.
(3) Using Lemma 2 perform the monomial substitution $z_i := t^{c_i}$. We obtain a univariate polynomial $f(P; t) = p(t)$.
(4) Compute $g(t) = \frac{p'(t)}{p(t)}$. It is given as a quotient of two sums of rational functions.
(5) Perform the numeric integration over the square of diagonal $-N - iN, N + iN$ numerically using, for example, Clenshaw-Curtis quadrature. Compute $M =$

$$\frac{1}{2\pi i} \left(\int_{-N}^{N} g(s - i\,N) ds + \int_{-N}^{N} i\, g(N + i\,s) ds - \int_{-N}^{N} g(s + i\,N) ds - \int_{-N}^{N} i\, g(-N + i\,s) ds \right).$$

The fact that M is an integer can be used in the precision of the calculation.

3.2 Algorithms: Digging and Single Cone Digging

In (Lasserre, 2004), Lasserre proposed an asymptotic heuristic method for solving integer programs, or at least providing an upper bound on the optimal value, based on Barvinok's rational functions: Consider again Problem (IP). From Equation (1) and the indices $i \in I$, 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 denotes the highest exponent of t appearing in the expansions of the rational functions defined for each $i \in I$ in Equation (2). The number σ is 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:

Lemma 5 (Theorem 3.1 in Lasserre (2004)) *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 Lemma 5 are met, from an easy inspection we could recover the optimal value of an integer program. If we assume that c is chosen randomly from some large cube in $\mathbb{Z}^d$, then the first condition is easy 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.

Next we present a simple improvement to Lasserre's heuristic and give another integer programming algorithm. We call it the *digging algorithm*. This algorithm *digs* for the coefficient of the next highest appearing exponent of t doing a controlled Laurent series expansion. For simplicity our explanation assumes the easy-to-achieve condition $c \cdot v_{ij} \neq 0$, for all v_{ij}:

This time we wish to find explicit optimal solutions not just the optimal value of Problem (IP). The reader will observe the modifications necessary are small. Take again Equation (1) computed via Barvinok's algorithm. Now, for the given c, instead of the substitutions $z_k = t^{c_k}$ we make the substitutions $z_k = y_k t^{c_k}$, for $k = 1,\ldots,d$. These substitutions into (1) yield 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 the following sum of monomials.

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

Both equations, (5) and (4), represent the same function $g(P;y,t)$. Thus, if we compute a Laurent series of (4) that shares a region of convergence with the series in (5), then the corresponding coefficients of both series must be equal. In particular, because P is a polytope, the series in (5) converges almost everywhere. Thus if we compute a Laurent series of (4) that has any nonempty region of convergence, then the corresponding coefficients of both series must be equal. Barvinok's algorithm provides us with the right hand side of (4). We need to obtain the coefficient of highest degree in t from the expanded Equation (5). We compute a Laurent series for it using the following procedure: 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 series of the form

$$E_i\, y^{u_i}t^{c\cdot u_i}\prod_{j=1}^{d}(1+y^{v_{ij}}t^{c\cdot v_{ij}}+(y^{v_{ij}}t^{c\cdot v_{ij}})^2+(y^{v_{ij}}t^{c\cdot v_{ij}})^3+\ldots). \tag{7}$$

Multiply out each such product of series and add the resulting series. This yields precisely the Laurent series in (5). Thus, we have now the steps of an algorithm to solve integer programs:

Algorithm: (*Digging*):

Input: $A \in \mathbb{Z}^{m\times d}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^d$.

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

(1) Using Lemmas 1 and 2 compute the rational function expression of Equation (4). Use the identity (6) as necessary to enforce that all v_{ij} in (4) satisfy $c\cdot v_{ij} < 0$.
(2) Via the expansion formula (7), multiply out the factors and add the terms, grouping together those of the same degree in t. Thus we 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), all $c\cdot v_{ij}$ are negative, so that 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 M of t is found such that for some $\alpha \in \mathbb{Z}^d$, the coefficient of $y^\alpha t^M$ is non-zero in the expansion (5).
(4) Return "M" as the optimal value of the integer program and return "α" as an optimal solution.

Note that if one needs to solve a family of integer programs where only the cost vector c is changing, then Equation (1) is computed only once applying the steps of the algorithm above for each cost vector to obtain all the optimal values.

Given the polytope $P := \{x \in \mathbb{R}^d : Ax \leq b,\ x \geq 0\}$, the *tangent cone* or *supporting cone* K_v at a vertex v of P is $K_v = v + \{u \in \mathbb{R}^d : v + \delta u \in P$ for all sufficiently small $\delta > 0\}$. A set of linear inequalities defining K_v consists of those facet inequalities of P that turn into equalities when evaluated at v. We observed in De Loera et al. (2004a) that a major practical bottleneck of the original Barvinok algorithm in Barvinok (1994) is the fact that a polytope may have too many vertices. Since originally one visits each vertex to compute a rational function at each tangent cone, the result can be costly. Therefore a natural idea for improving the digging algorithm is to compute with a single tangent cone of the polytope and revisit the digging calculation for a smaller sum of rational functions. Problem (IP) has the linear programming relaxation (LP) maximize $c \cdot x$ subject to $x \in P$. One of the vertices of P gives the optimal value for (LP) (Schrijver, 1986). Let $V(P)$ be the vertex set of P and $v \in V(P)$ be a vertex such that $c \cdot v$ is the optimal value for (LP). Then, clearly, the tangent cone K_v at v contains P. So, if we can find an integral point $x^* \in K_v$ such that $c \cdot x^* \geq c \cdot x$, $\forall x \in P \cap \mathbb{Z}^d$ and $x^* \in P$, then x^* is an optimal solution for (IP). Note that the formulas of rational functions that we used for the original digging algorithm still hold for the rational functions of the tangent cone K_v. In what follows we assume that $c \cdot \alpha \neq 0$ for all rays of K_v.

Algorithm: (*Single Cone Digging*):

Input: $A \in \mathbb{Z}^{m \times d}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^d$

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

(1) Compute a vertex v of P such that $c \cdot v =$ maximize $\{c \cdot x : Ax \leq b, x \geq 0\}$.
(2) Compute the tangent cone K_v at v and from Lemma 1 compute the Barvinok rational function (4) encoding of the lattice points inside K_v.
(3) Use the identity (6) as necessary to enforce that all v_{ij} in (4) satisfy $c \cdot v_{ij} < 0$.
(4) 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.
(5) Continue calculating the terms of the expansion (5), in decreasing order with respect to the degree of t, until a degree M of t is found such that:
   - for some $\alpha \in \mathbb{Z}^d$, the coefficient of $y^\alpha t^M$ is non-zero in the expansion (5),
   - $A\alpha \leq b$, $\alpha \geq 0$.
(6) Return "M" as the optimal value of the integer program and return "α" as an optimal solution.

Unfortunately, although the necessary rational functions can be computed in polynomial time when the dimension is assumed to be fixed, the digging algorithm may have to compute an exponential number of coefficients before finding one that does not vanish. We see this is true even for fixed dimension two in the following simple example found by P. Huggins (Huggins, 2004):

For a positive integer N define the quadrilateral Q_N, with vertices $(1/2, 1/2)$, $(3/4, 1/2)$, $(1/2, 3/4)$, $(1, N)$. It contains the single lattice point $(1, N)$. By Brion's theorem (Barvinok and Pommersheim, 1999) we know that the Barvinok rational function associated to Q_N can be written as the sum of the rational functions of the tangent cones at the vertices. In particular for the vertex $(1/2, 1/2)$ the unique rational function is $\frac{z_1 z_2}{(1-z_1)(1-z_2)}$. Now taking the cost function to be given by $c = (-1, -1)$ one must make the monomial substitutions $z_1 = y_1 t^{-1}$ and $z_2 = y_2 t^{-1}$ into the rational function above. Thus we obtain $\frac{y_1 y_2 t^{-2}}{(1-y_1 t^{-1})(1-y_2 t^{-1})}$. Its Laurent expansion contains $y_1 y_2 t^{-2}, y_1 y_2^2 t^{-3}, y_1 y_2^3 t^{-4}, \ldots, y_1 y_2^{N-1} t^{-N}$. Therefore the digging algorithm will calculate the coefficients for these $N - 1$ terms, which will vanish, before reaching the term $y_1 y_2^N t^{-N-1}$. The algorithm performs an exponential number of steps in the binary size of the input (namely $\log(N)$).

3.3 Algorithm: Hadamard products

Finally, we present the most abstract algorithm of all. Nevertheless, it has good theoretical complexity. For fixed d this algorithm runs in polynomial time (on the input size) based on the polynomiality of Barvinok's counting algorithm (Lemma 1), Lemma 2, and Corollary 1 below.

Let g_1, g_2 be Laurent power series in $z_1, z_2, \ldots, z_d$, $g_1(z) = \sum_{m \in \mathbb{Z}^d} \beta_{1m} z^m$ and $g_2(z) = \sum_{m \in \mathbb{Z}^d} \beta_{2m} z^m$. The *Hadamard product* $g = g_1 \star g_2$ is the power series $g(z) = \sum \beta_{1m}\beta_{2m} z^m$. Note that the Hadamard series depends on the particular Laurent series expansion of the functions. Barvinok and Woods (2003) used Hadamard products to carry out Boolean operations with sets of lattice points when they are encoded as rational functions. The Hadamard product is a bilinear operation on rational functions. Thus the Hadamard product of two sums of rational functions is simply the sum of Hadamard products carried out for pairs of summands as explained in Barvinok and Woods (2003). The following lemma states that this key subroutine is efficient:

Lemma 6 (Lemma 3.4 and Theorem 3.6 in Barvinok and Woods (2003)) *Let us fix k, the number of binomials in the denominators. There exists a polynomial time algorithm, which, given two functions*

$$g_1(z) = \frac{z^{p_1}}{\prod_{j=1}^{k}(1 - z^{a_{1j}})} \quad \text{and} \quad g_2(z) = \frac{z^{p_2}}{\prod_{j=1}^{k}(1 - z^{a_{2j}})}$$

with $a_{ij}, p_i \in \mathbb{Z}^d$ computes a function $h(z)$ in the form

$$h(z) = \sum_{i \in I} \beta_i \frac{z^{q_i}}{\prod_{j=1}^{s}(1 - z^{b_{ij}})}$$

with integer vectors q_i, b_{ij}, rational numbers β_i, and with $s \leq 2k$ such that $h(z)$ is a Laurent expansion of the Hadamard product $g_1(z) * g_2(z)$.

Barvinok and Wood's algorithmic proof of Lemma 6 relies on the repeated use of Barvinok's algorithm (Lemma 1) in many simpler polytopes, one for each pair of rational functions (see Lemma 3.4 in Barvinok and Woods (2003)). Note that the Hadamard product selects monomials that belong to both g_1 and g_2. The key property is that the Hadamard product $m_1 * m_2$ of two monomials m_1, m_2 is zero unless $m_1 = m_2$. That is why we recover an intersection of the monomials present in the generating functions. We use this to extract an explicit optimal value after we do the monomial substitution. This is a particular case of Lemma 8 in De Loera et al. (2004b) but we include the proof here:

Corollary 1 *Assume the dimension d is fixed. Let P be the polytope $\{x \in \mathbb{R}^d : Ax \leq b,\ x \geq 0\}$. Suppose the generating function $f(P; z) = \sum_{\alpha \in P \cap \mathbb{Z}^d} z^\alpha$ is represented as a Barvinok rational function and let c be any integer cost vector. We can extract $M := \max\{\alpha \cdot c : \alpha \in P \cap \mathbb{Z}^d\}$ in polynomial time.*

Proof: For the cost vector c perform a monomial substitution $z_i := t^{c_i}$. Such a monomial substitution can be computed in polynomial time by Lemma 2. The effect is that the polynomial $f(P; z)$ becomes, as in Equations (2) and (3), a univariate polynomial $f(P; t)$. We determine the degree of $f(P; t)$ in t using a binary search idea, but, unlike Section 2, we do not have to call explicitly Barvinok's algorithm with a new polytope at each iteration. We create the *interval polynomial* $i_{[p,q]}(t) = \sum_{i=p}^{q} t^i$ which obviously has a Barvinok rational function representation. Using Lemma 6 compute the intersection of $i_{[p,q]}(t)$ with $f(P; t)$. This yields only those monomials whose degree in the variable t lies between p and q. We will keep shrinking the interval $[p, q]$ until we find the degree, we do this following a binary search procedure to update p, q. We need a bound for the degree in t of $g(t)$ to start a binary search. An upper bound U is for example the linear programming bound. It is clear that $\log(U)$ is polynomially bounded. In no more than $\log(U)$ steps one can determine the highest degree M of $f(P; t)$. $\square$

Algorithm: (*Hadamard product*):

Input: $A \in \mathbb{Z}^{m \times d}$, $b \in \mathbb{Z}^m$, $c \in \mathbb{Z}^d$.

Output: The optimal value $M = \text{maximize } \{c \cdot x : Ax \leq b, x \geq 0, x \in \mathbb{Z}^d\}$.

(1) Using Barvinok's algorithm compute the rational function representation of the generating function $f(P; z)$ for $P = \{x \in \mathbb{R}^d : Ax \leq b, x \geq 0\}$.
(2) Following Corollary 1, do monomial substitutions and Hadamard products necessary to find the optimal value M.

4 Computational Experiments

We evaluated the five algorithms: In our experiments, the numerical complex integration algorithm and the Hadamard algorithm were not competitive: In the first case, the integration step was a major bottleneck of computation. For example, it took over seven minutes to numerically integrate rational functions of real degree 40 encoding the lattice points of a 20 by 20 square even though we simply took $N = 1$ and we used only 3 digits of accuracy. Perhaps the structure of these integrals or better integration algorithms can be exploited in the future. Finally, a dozen small examples computed with the help of `LattE` and `Maple` indicated that the Hadamard product algorithm is slower than Algorithm *BBS*.

The remaining three algorithms were compared with the default algorithm implemented in `CPLEX` MIP solver for version 6.6. The digging algorithm, the single cone digging algorithm, and the BBS algorithm, are implemented in `LattE` and available at `www.math.ucdavis.edu/∼latte`. We report our experience solving hard knapsack problems from Aardal and Lenstra (2002); Cornuéjols et al. (1997). We selected these problems because they are challenging and because the current implementation of `LattE` can only handle problems with up to 30 variables. We refer to 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 $\gcd(a_1, \dots, a_d) = 1$. For the cost vector c, we took the first d entries of the vector:
$(213, -1928, -11111, -2345, 9123, -12834, -123, 122331, 0, 0)$.

All computations were done on a 1 GHz Pentium PC running Red Hat Linux. For the Barvinok-based algorithms we set a limit of 2 hours of computation. After that limit the problem was considered unsolved. On the other hand, we allowed CPLEX 6.6 to run until it either solved the problem or it ran out of memory. Table 2 provides the optimal values and an optimal solution for each problem. As it turns out, these integer programs have very interesting geometry because there is exactly *one feasible integer solution* for each problem. With one exception, CPLEX 6.6. could not solve the given problems. It generated many nodes before running out of memory. Note that whenever the digging algorithm found the optimal value, it did so much faster than the BBS algorithm. This is interesting, because we saw the worst-case complexity for the digging algorithm is exponential even for fixed dimension, while the BBS has polynomial complexity in fixed dimension. From Table 2 and Table 3, one can see that the single cone digging algorithm is the fastest algorithm. It only failed to solve two of the instances. This algorithm is also more memory efficient than the original digging algorithm, since the number of unimodular cones for the single cone digging algorithm is much smaller. The original digging algorithm came in second place. It failed to find a solution for problems cuww2, cuww4, and cuww5. In those instances 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

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.

Problem	Value	Solution	Runtime for digging (Original)	Runtime for digging (S. Cone)	Runtime for BBS	Runtime for CPLEX 6.6
cuww1	1562142	[7334 0 0 0 0]	0.4 sec.	0.17 sec.	414 sec.	$>$ 1.5 h (OM)
cuww2	-4713321	[3 2445 0 0 0 0]	$>$ 2 h	$>$2 h	1.8 h	$>$ 0.75 h (OM)
cuww3	1034115	[4855 0 0 0 0 0]	1.4 sec.	0.24 sec.	1.7 h	$>$ 0.75 h (OM)
cuww4	-29355262	[0 0 2642 0 0 0 0]	$>$ 2 h	$>$ 2 h	$>$ 2 h	$>$ 0.75 h (OM)
cuww5	-3246082	[1 1678 1 0 0 0 0 0]	$>$ 2 h	147.63 sec.	$>$ 2 h	$>$ 0.75 h (OM)
prob1	9257735	[966 5 0 0 1 0 0 74]	51.4 sec.	18.55 sec.	$>$ 2 h	$>$ 1 h (OM)
prob2	3471390	[853 2 0 4 0 0 0 27]	24.8 sec.	6.07 sec	$>$ 2 h	$>$ 0.75 h (OM)
prob3	21291722	[708 0 2 0 0 0 1 173]	48.2 sec.	9.03 sec.	$>$ 2 h	$>$ 1.5 h (OM)
prob4	6765166	[1113 0 7 0 0 0 0 54]	34.2 sec.	9.61 sec.	$>$ 2 h	$>$ 1.5 h (OM)
prob5	12903963	[1540 1 2 0 0 0 0 103]	34.5 sec.	9.94 sec.	$>$ 2 h	$>$ 1.5 h (OM)
prob6	2645069	[1012 1 0 1 0 1 0 20 0 0]	143.2 sec.	19.21 sec.	$>$ 2 h	$>$ 2 h (OM)
prob7	22915859	[782 1 0 1 0 0 0 186 0 0]	142.3 sec.	12.84 sec.	$>$ 2 h	$>$ 1 h (OM)
prob8	3546296	[1 385 0 1 1 0 0 35 0 0]	469.9 sec.	49.21 sec.	$>$ 2 h	$>$ 2.5 h (OM)
prob9	15507976	[31 11 1 1 0 0 0 127 0 0]	0.39 h	283.34 sec.	$>$ 2 h	4.7 sec.
prob10	47946931	[0 705 0 1 1 0 0 403 0 0]	250.6 sec.	29.28 sec.	$>$ 2 h	$>$ 1 h (OM)

Table 2. Optimal values, optimal solutions, and running times for each problem. Symbol $> x$ h (OM), means CPLEX ran out of memory after x hours. The symbol $>$ 2 h means that the problem was not solved before two hours of computation.

problem	Original Digging (A)	Original Digging (B)	Single Cone Digging (A)	Single Cone Digging (B)
cuww 1	110	0	25	0
cuww 2	386	> 2,500,000	79	> 2,500,000
cuww 3	346	0	49	0
cuww 4	364	> 400,000	51	> 400,000
cuww 5	2,514	> 100,000	453	578,535
prob 1	10,618	74,150	1,665	74,150
prob 2	6,244	0	806	0
prob 3	12,972	0	2,151	0
prob 4	9,732	0	1,367	0
prob 5	8,414	1	2,336	1
prob 6	26,448	5	3,418	5
prob 7	20,192	0	2,015	0
prob 8	62,044	0	6,523	0
prob 9	162,035	3,558	45,017	3,510
prob 10	38,638	256	5,128	256

Table 3
Data for the digging algorithm. $A :=$ number of unimodular cones and $B :=$ number of digging levels

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 Barvinok and Pommersheim (1999)) 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 heuristic 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 $f(P;t)$. See Table 3. To conclude we have two comments: First, it should be mentioned these are not the only known algorithms based on Barvinok's rational functions. For example, an algorithm via Gröbner bases is presented in De Loera et al. (2004b). Finally, we would like to point out that other non-standard approaches have been tried in our test instances. For example a lattice approach to tackle feasibility version of the knapsack problems was used in Aardal and Lenstra (2002), and a test set approach was used for the cuww-instances in Cornuéjols et al. (1997).

Acknowledgments: We are grateful to K. Aardal, A. Barvinok, J.B. Lasserre, B. Sturmfels, and R. Weismantel for useful discussions. This research was supported by NSF grant DMS-0309694 and NSF VIGRE grant DMS-0135345.

References

Aardal, K. and Lenstra, A.K. *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, 350–366.

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.

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

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.

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, 1997, 92–103.

De Loera, J.A., Hemmecke, R., Tauzer, J., and Yoshida, R. *Effective lattice point counting in rational convex polytopes.* Journal of Symbolic Computation, vol. 38, 2004, 1273–1302.

De Loera, J.A, Haws, D., Hemmecke, R., Huggins, P., Sturmfels, B., and Yoshida, R. *Short rational functions for toric algebra and applications.* Journal of Symbolic Computation, Vol. 38, 2 , 2004, 959–973.

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

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

Henrici, P. *Applied and computational complex analysis.*, vol. 1, Wiley, New York, 1974.

Huggins P.M. *Lattice point enumeration via rational functions and applications to optimization and statistics.*, senior undergraduate thesis, Department of mathematics, University of California Davis, 2004.

Lasserre, J.B. *Integer programming, Barvinok's counting algorithm and Gomory relaxations.* Operations Research Letters, 32, No.2, 2004, 133–137.

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

Marden, M. *Geometry of polynomials* American Mathematical Soc., No. 3, Mathematical Surveys, Providence, Second Edition, 1966.

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