Lattice Point **E**numeration via Rational Functions, and Applications to Optimization and Statistics

By

Peter Huggins

HONORS THESIS

Submitted in partial satisfaction of the requirements for the degree of

BACHELORS IN SCIENCE

in

MATHEMATICS

at the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

_____

_____

Committee in Charge

Jesus De Loera, Rostislav Matveyev

2004

Contents

Abstract

This thesis gives an account of the author's contribution to the development of the software LattE, which implements Barvinok's polynomial time algorithm for enumerating and counting lattice points in rational polytopes in fixed dimension, via rational functions. The thesis also presents some new theoretical results and algorithms based on Barvinok's rational functions.

A more memory efficient version of Barvinok's counting algorithm is presented, along with its implementation as a new Memory Saving Mode in LattE. Then an algorithm for Integer Programming is given (with proof of correctness), called the Digging Algorithm, which extracts solutions for Integer Programming instances, out of rational functions. An implementation of the Digging Algorithm in LattE is presented, and compared to the standard Integer Programming software Cplex. Finally, a new theorem is proved that allows efficient summation of a fixed degree polynomial, over the lattice points in a finite lattice point set encoded via Barvinok's rational functions. This gives a generalized *weighted counting* algorithm, and also allows various applications to fields such as Statistics.

Chapter 1

Introduction

1.1 Preliminaries

The primary objects we will discuss are lattice points, and so naturally we should begin with a definition.

Definition 1.1. *By lattice points we mean those points in* $\mathbf{R}^d$ *which have integer coordinates.*

Thus, the set of all lattice points in $\mathbf{R}^d$ is $\mathbf{Z}^d$. We will be mainly interested in lattice points that are contained inside *rational polyhedra*, which are defined as follows:

Definition 1.2. *A rational polyhedron is a subset of* $\mathbf{R}^d$ *which is the set of all real solutions of a system of non-strict linear inequalities with integer coefficients.*

If P happens to be a bounded rational polyhedron, then we call P a *rational polytope.*

We can succinctly write a system of non-strict linear inequalities with integer coefficients as $Ax \leq b$, where $A \in \mathbf{Z}^{m \times d}$ is an integer matrix, and where $b \in \mathbf{Z}^m$. Then equivalently, a rational polyhedron is a subset $P \subset \mathbf{R}^d$ which can be written as $P = \{x \in \mathbf{R}^d : Ax \leq b\}$ for some $A \in \mathbf{Z}^{m \times d}$ and $b \in \mathbf{Z}^m$. We will often use the term *dimension* when referring to d.

It turns out that many problems in Combinatorics, Compiler Design, Representation Theory, and Number Theory—just to name a few—can be expressed as problems regarding lattice points inside rational polyhedra. See [21, 13, 11, 17, 18] for examples.

1.2 From Lattice Points to Monomials: Generating Functions

One natural way to represent lattice points is with multivariate monomials. For a given lattice point $\alpha \in \mathbf{Z}^d$, we can associate the monomial z^α in d variables $z_1, z_2, \ldots, z_d$, as follows:

$$z^\alpha := z_1^{\alpha_1} z_2^{\alpha_2} \ldots z_d^{\alpha_d} \tag{1.1}$$

Then, we can easily represent a *set* S of lattice points, by a *sum* of monomials:

$$f(S; z) = \sum_{\alpha \in S} z^\alpha \tag{1.2}$$

For the time being, we are not yet considering such $f(S; z)$ as a complex-valued function of z, because if S were infinite, it is unclear whether the Laurent series $\sum_{\alpha \in S} z^\alpha$ would even converge for some $z \in \mathbf{C}^d$. So for now, at least, we will simply regard $f(S; z)$ as a formal sum of monomials and save ourselves such troubles until later. We will often refer to $f(S; z)$ as the *generating function* for S.

1.3 A Theorem of Brion

Given a rational polyhedron $P = \{x \in \mathbf{R}^d : Ax \leq b\}$, we define the vertices of P as follows:

Definition 1.3. *A vertex $v \in P$ is a point in P which can be written as $v = A'^{-1}b'$, where A' is an $d \times d$ invertible submatrix of A, and where b' is the corresponding*

subvector of b.

Then for each such vertex $v \in P$, we define

$$I_v = \{i \in \{1, 2, \ldots, m\} : A_i \cdot v = b_i\} \tag{1.3}$$

where A_i denotes the ith row of A, and where b_i denotes the ith component of b. Then we define

$$cone(P, v) = \{x \in \mathbf{R}^n : A_i \cdot x \leq b_i, \forall i \in I_v\} \tag{1.4}$$

Such $cone(P, v)$ is called the *tangent cone* of P at vertex v. It is well known [3] that if P is a rational polyhedron containing no straight lines, then $f(P \cap \mathbf{Z}^d; z)$ does in fact have a non-empty region of convergence as a Laurent series in z, so that we may regard $f(P \cap \mathbf{Z}^d; z)$ as defining a complex-valued function of $z \in \mathbf{C}^d$. Furthermore, each $cone(P, v)$ will also be a rational polyhedron containing no straight lines, and so each $f(P \cap \mathbf{Z}^d; z)$ also has a non-empty region of convergence as a Laurent series in z.

Brion [6] was the first to discover the following beautiful result, which relates the complex-valued function $f(P \cap \mathbf{Z}^d; z)$ to the corresponding set of functions $f(cone(P, v) \cap \mathbf{Z}^d; z)$ in a very natural way:

Theorem 1.4. *[6]*

Let P be a rational polyhedron containing no straight lines, and let $V(P)$ be the set of vertices of P. Then:

$$f(P \cap \mathbf{Z}^d; z) = \sum_{v \in V(P)} f(cone(P, v) \cap \mathbf{Z}^d; z)$$

From here on, we will assume that rational polyhedra and cones contain no straight lines, unless we explicitly state otherwise.

Note that Brion's theorem reduces the problem of computing $f(P \cap \mathbf{Z}^d; z)$ for rational polyhedra P to the problem of computing $f(K \cap \mathbf{Z}^d; z)$ for which K is a rational cone.

1.4 Triangulations and Simplicial Cones

We begin with some basic terminology:

Definition 1.5. *A rational cone K is a subset of $\mathbf{R}^d$ which contains no straight lines, that can be written as $K = \{x \in \mathbf{R}^d : x = v + M\epsilon, \epsilon \in \mathbf{R}^\mu, \epsilon \geq 0\}$, for some $M \in \mathbf{Z}^{d \times \mu}$ and $v \in \mathbf{Q}^d$, where no column of M can be written as a non-negative combination of the others. Such v is called the vertex of K, and the columns of M, when regarded as vectors, are called the generators of K. If the generators of K span $\mathbf{R}^d$, then K is called full-dimensional. If the generators of K are linearly independent, then K is called simplicial.*

It is well known that a rational cone K can be expressed as a finite union of simplicial cones $K = \bigcup_{i \in I} K_i$, where each K_i has vertex v, and where the interiors of the K_i are disjoint. Additionally, this can be done so that the intersection of any pair of simplicial cones in $\{K_i\}_{i \in I}$ is again a cone, whose generators are precisely those generators that are common to the pair of cones. We call such a set simplicial cones $\{K_i\}_{i \in I}$ a *triangulation* of K.

We would like to write $f(K \cap \mathbf{Z}^d; z)$ as the sum $\sum_{i \in I} f(K_i \cap \mathbf{Z}^d; z)$, but we might "over-count" some points that are contained in more than one K_i. However, a straightforward application of the inclusion-exclusion principle shows that we can remedy this situation by subtracting all generating functions $f(K_i \cap K_j \cap \mathbf{Z}^d; z)$ for pair-wise intersections, and then adding all generating functions for 3-wise intersections, etc. Therefore, we can write $f(K \cap \mathbf{Z}^d; z)$ as a *signed* sum

$$f(K \cap \mathbf{Z}^d; z) = \sum_{i \in I'} E_i \; f(K_i \cap \mathbf{Z}^d; z) \; , \tag{1.5}$$

where each K_i is simplicial, and where each $E_i \in \{-1, 1\}$.

This further reduces the problem of computing $f(P \cap \mathbf{Z}^d; z)$ for rational polyhedra P to the problem of computing $f(K \cap \mathbf{Z}^d; z)$ for which K is a simplicial rational cone.

1.5 Toward a Formula for $f(P \cap \mathbf{Z}^d; z)$

We now seek an explicit formula for the generating function associated with a simplicial rational cone K with vertex v and generators $w_1, w_2, \ldots, w_\mu$. We begin with a familiar Laurent series,

$$\frac{1}{1 - z^\alpha} = 1 + z^\alpha + z^{2\alpha} + z^{3\alpha} + \ldots \ , \tag{1.6}$$

and note that

$$\prod_{j=1}^{\mu} \frac{1}{1 - z^{w_j}} = \prod_{j=1}^{\mu} \sum_{k=0}^{\infty} z^{k w_j} \tag{1.7}$$

is precisely equal to $\sum_{\alpha \in S} z^\alpha$, where S is the set of all non-negative integer coefficient combinations of the vectors $w_1, w_2, \ldots, w_\mu$.

Let Π be the half-open parallelepiped defined by $\Pi = \{x : x = v + W\epsilon, \epsilon \in [0, 1)^\mu\}$, where W is the matrix with columns $w_1, w_2, \ldots, w_\mu$. Then, as some geometric reasoning might suggest, the well known formula

$$f(K \cap \mathbf{Z}^d; z) = \frac{\sum_{\alpha \in \Pi \cap \mathbf{Z}^d} z^\alpha}{\prod_{j=1}^{\mu} (1 - z^{w_j})} \tag{1.8}$$

will hold anywhere inside the region of convergence of $\sum_{\alpha \in K \cap \mathbf{Z}^d} z^\alpha$, excluding the measure-zero set of poles of the rational function given in (1.8).

Let us review for a moment. Using Theorem (1.4), along with the formulae (1.5) and (1.8), if we were given $A \in \mathbf{Z}^{m \times d}$ and $b \in \mathbf{Z}^m$, then we could indeed compute the

generating function $f(P \cap \mathbf{Z}^d; z)$ associated with the polyhedron $P = \{x \in \mathbf{R}^d : Ax \leq b\}$.

The problem is, in order to write down the formula for $f(P \cap \mathbf{Z}^d; z)$ explicitly, we would have to find and list all the lattice points contained inside various parallelepipeds, in order to write down the numerators appearing in formula (1.8). But, in terms of A and b, those parallelepipeds might contain a *huge* number of lattice points. In particular, even if we considered the dimension d to be a fixed constant, there is no polynomial in the *standard input size* of A and b (see Section (1.7)) that bounds the maximum number of lattice points contained in a parallelepiped appearing in formula (1.8).

1.6 Barvinok's Algorithm

Enter Alexander Barvinok and his remarkable algorithm [2]. In 1994, Barvinok showed how to explicitly write down a formula for $f(P \cap \mathbf{Z}^d; z)$ as a sum of rational functions, in time polynomial in the input size of A and b, when the dimension d is considered to be a fixed constant. He then showed how to use this formula for $f(P \cap \mathbf{Z}^d; z)$, to count the number of lattice points contained in a rational polytope P, in polynomial time when d is fixed. This gave an extension of Lenstra's celebrated result [16] that one can determine whether P contains any lattice points in polynomial time when d is fixed. Barvinok's original version actually relied on Lenstra's result, but Dyer and Kannan [12] later showed how to modify Barvinok's algorithm so that Lenstra's algorithm was no longer needed as a subroutine.

We give an overview of Barvinok's main algorithm here. For a complete description, consult [2, 12, 4]. The basic idea is to take a simplicial rational cone K with μ generators, and use an integer vector $\omega \in \mathbf{Z}^d$ to express K as a *signed* sum of μ simplicial cones K_i (modulo the set of closed half-spaces in $\mathbf{R}^d$). Each simplicial

cone K_i is formed from K by replacing one generator with ω. By using a clever application of Minkowski's Theorem, Barvinok guarantees that a particular ω can be quickly found, so that formula (1.8) for each K_i will have an associated half-open parallelepiped Π_i containing significantly fewer lattice points than the half-open parallelepiped Π for K.

This expressing of K as a signed sum of cones K_i is called a signed decomposition of K, or simply a decomposition of K.

The same process can then be applied to obtain a signed decomposition for each K_i, and then applied again to the cones thereby obtained, and so on. In fact, Barvinok's algorithm repeatedly computes signed decompositions of cones, until all simplicial cones have parallelepipeds that contain just a single lattice point. (Such simplicial cones whose parallelepipeds contain only one lattice point are called *unimodular*.)

Furthermore, by quickly finding well-suited ω's to perform the decompositions, Barvinok's algorithm yields a signed decomposition of K into no more than $p(A, b)$ unimodular cones, where p is a polynomial in the input size of A and b, when the dimension d is a fixed constant. Furthermore, the decomposition process runs in polynomial time.

Now, if d is a fixed constant, then a polyhedron $P \subset \mathbf{R}^d$ defined by $P = \{x \in \mathbf{R}^d : Ax \leq b\}$, where $A \in \mathbf{Z}^{m \times d}$ and $b \in \mathbf{Z}^m$, has no more than $\binom{m}{d}$ vertices, each giving rise to a tangent cone having no more than $p_1(m)$ generators, where p_1 is a fixed-degree polynomial. Each of these tangent cones can be triangulated into no more than $p_2(p_1(m))$ simplicial cones, where p_2 is a fixed-degree polynomial. Note that $\binom{m}{d}$ is also a fixed-degree polynomial in m, when d is fixed, and so using Theorem (1.4), along with the formulae (1.5) and (1.8), we can write

$$f(P \cap \mathbf{Z}^d; z) = \sum_{i \in I} E_i \; f(K_i \cap \mathbf{Z}^d; z) \; , \tag{1.9}$$

where $|I|$ is bounded by a polynomial in the input size of A and b. Then applying Barvinok's signed decomposition process to each cone K_i yields the following result:

Theorem 1.6. *[2] Suppose d is fixed. Then there exists an algorithm, which given $A \in \mathbf{Z}^{m\times d}$ and $b \in \mathbf{Z}^m$, explicitly computes the generating function $f(P \cap \mathbf{Z}^d; z)$ corresponding to $P = \{x \in \mathbf{R}^d : Ax \leq b\}$, in the following form:*

$$f(P \cap \mathbf{Z}^d; z) = \sum_{i\in I} E_i \frac{z^{u_i}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})} \tag{1.10}$$

where all $E_i \in \{-1, 1\}$ and where all $u_i, v_{ij} \in \mathbf{Z}^d$, and where all v_{ij} are non-zero. Furthermore, the algorithm runs in polynomial time.

Theorem (1.6) is the main result of this chapter, and we will commonly refer to it as Barvinok's Main Algorithm. Almost everything we will discuss from this point forward will derive from it in some way. In the next chapter we discuss the software LattE, developed by our research group at UC Davis, which is the first-ever implementation of Barvinok's Main Algorithm.

But before we do, let us first take a glimpse of what we can do via Theorem (1.6). Suppose P is a rational polytope for which we have computed $f(P \cap \mathbf{Z}^d; z)$ using Barvinok's Main Algorithm. Notice that, if we were to compute the limit

$$\lim_{(z_1,z_2,\ldots,z_d)\to(1,1,\ldots,1)} f(P \cap \mathbf{Z}^d; z) = \lim_{(z_1,z_2,\ldots,z_d)\to(1,1,\ldots,1)} \sum_{\alpha\in P\cap\mathbf{Z}^d} z^\alpha \tag{1.11}$$

we would obtain $\sum_{\alpha\in P\cap\mathbf{Z}^d} 1$, which is precisely the number of lattice points inside P. Barvinok [2] showed how to compute the limit (1.11) in polynomial time, when d is fixed—thus proving that in fixed dimension, counting the number of lattice points inside rational polytopes admits a polynomial time algorithm. We will commonly refer to this algorithm for counting lattice points as Barvinok's Counting Algorithm.

1.7 Additional Notes and Miscellanea

Some Terminology Regarding Algorithms and Their Complexity

By *standard input size*, we mean the total number of binary digits needed to write down A and b in a "reasonable way." For our purposes, we will just define the *standard input size* of an $m \times n$ integer matrix W to be $cmn * (log(max\{|w_{ij}| + 2\}))$ where $W = (w_{ij})$ and where c is some universal positive constant. When we say that an algorithm is polynomial time, we mean that the number of operations required by the algorithm is bounded by a polynomial in the standard input size of its input data.

Stacks and Queues

A stack is a storage area that we may use for storing and retrieving objects. We may insert an object into the stack whenever we wish, but whenever we retrieve/remove an object from the stack, the object that we retrieve/remove will be the **most recent** object that we have added to the stack. In a sense, we "stack" inserted objects on top of each other, and always remove the object which is "on the top of the stack."

A queue is also a storage area that we may use for storing and retrieving objects. We may insert an object into the queue whenever we wish, but whenever we retrieve/remove an object from the queue, the object that we retrieve/remove will be the **least recent** object that we have added to the queue.

Chapter 2

The LattE Project: Implementing and Improving Barvinok's Algorithm

The LattE project was conceived by Jesus de Loera of UC Davis, with the original goal of implementing Barvinok's powerful algorithm described in Chapter 1. That original goal was indeed realized and thus the software LattE was born. The fruits of the labor are presented in [9, 10]. David Haws and I joined the project shortly thereafter, to help further develop and improve LattE.

2.1 A More Memory Efficient Version of Barvinok's Counting Algorithm

We first reviewed Barvinok's algorithm for counting lattice points inside a rational polytope P, and discovered that a considerable savings in memory could be accomplished through a minor modification.

2.1.1 Barvinok's Original Approach

Barvinok [2], and De Loera et. al [9], originally proposed the following approach to count the lattice points in P:

Algorithm 2.1. *Barvinok's Counting Algorithm*

1. *Compute the function $f(P \cap \mathbf{Z}^d; z)$ via Theorem (1.6)*
2. *Find a "reasonably-sized" integer vector $c \in \mathbf{Z}^d$ such that $c \cdot v_{ij} \neq 0$ for all $v_{ij} \in \mathbf{Z}^d$ appearing in the denominators in the calculated formula for $f(P \cap \mathbf{Z}^d; z)$ (By "reasonably sized," we mean that the standard input size of c is bounded by a polynomial of the standard input size of A and b, when d is considered fixed.)*
3. *Make the substitutions $z_i \to t^{c_i}$ for each $i = 1, 2, \ldots, d$, to obtain $f(P \cap \mathbf{Z}^d; z) \to g(P \cap \mathbf{Z}^d; t)$*
4. *Calculate the limit $\lim_{t \to 1} g(P \cap \mathbf{Z}^d; t)$, by first calculating a residue for each generating function appearing in the summation (1.10), (using residue techniques as described in [2, 9]), and then summing the obtained residues.*

□

Barvinok described a polynomial time method to perform step (2) of Algorithm (2.1), provided that the entire formula (1.10) for $f(P \cap \mathbf{Z}^d; z)$ was already computed and stored in memory.

2.1.2 A Modified Approach

However, if we actually knew *a priori* a reasonably-sized $c \in \mathbf{Z}^d$ that satisfied the criteria listed in step (2) of Algorithm (2.1), then instead of calculating $f(P \cap \mathbf{Z}^d; z)$ and then performing steps (2-4), we could instead use the following approach:

Algorithm 2.2. *A Different Approach for Barvinok's Counting Method*

1. *Set RESIDUE-SUM* $:= 0$
2. *Apply the algorithm in Theorem (1.6), as described in [9], for computing the function* $f(P \cap \mathbf{Z}^d; z)$*, but each time a unimodular cone* U *is produced in the cone decomposition process, such that* U *would contribute a function* $h(z) = E \frac{z^u}{\prod_{j=1}^{n} \left(1-z^{v_j}\right)}$ *to the summation (1.10), then immediately do the following:*
   (a) *Make the substitutions* $z_i \to t^{c_i}$ *(for each* $i = 1, 2, \ldots, d$*) into* h *and calculate the residue for* h *using the usual residue techniques as described in [2, 9]*
   (b) *Add the computed residue to RESIDUE-SUM, and then discard the cone* U *and the function* h *from memory*

□

2.1.3 Stacks vs. Queues; Analysis of the Cone Decomposition Process

At first glance, this modified approach does not seem to necessarily save a *great* deal of memory. But let's review for a moment on Barvinok's decomposition process. Remember that a simplicial cone K with μ generators is decomposed into μ cones. (And actually, we never have to deal with any lower dimensional cones or the inclusion-exclusion principle, thanks to "Brion's polarization trick." See [9] for further details.) Then any of the thereby obtained cones which are not yet unimodular are again decomposed, and so on.

There are two very different ways that this decomposition process could be implemented. On one hand, we could maintain a queue of simplicial cones, called NON-UNI, which contains all current simplicial cones that are not yet unimodular. Using this idea, we would implement step (2) in Algorithm (2.2) as follows:

While NON-UNI is not empty:

1. *Remove a cone K from NON-UNI*
2. *Decompose K into cones $K_1, K_2, \ldots, K_\mu$*
3. *For each obtained K_i perform steps (a) and (b) (from Algorithm (2.2) above) on K_i if K_i is unimodular. Otherwise insert K_i into the NON-UNI queue if K_i is not unimodular.*

□

Actually, if we implemented NON-UNI as a queue, then our proposed modification to Barvinok's Counting Algorithm would fail to considerably reduce the amount of memory used by the algorithm. But what if we implemented NON-UNI as a stack instead of a queue? (For a quick description of stacks and queues, see the end of Chapter 1).

We can model Barvinok's decomposition process using a tree, where nodes represent cones, and where leaf nodes correspond to unimodular cones. The children of a node K represent the cones obtained by applying the decomposition process once to the cone K. Now, if NON-UNI were a queue, then the maximum number of cones that might be stored in NON-UNI is the number of internal nodes (i.e. non-leaf nodes) in the tree. On the other hand, it is not hard to see that if NON-UNI were a stack, then NON-UNI would never contain more than $d * (L + 1)$ cones, where L is the maximum length of a path from the root of the tree down to a leaf node.

Our experiments indicate that in practice, the tree representing Barvinok's decomposition process will typically be fairly balanced, meaning that L is observed to be roughly $O(log_d\, N)$, where N is the number of nodes in the tree. On the other hand, the number of internal nodes in the tree will grow linearly with N when d is a fixed constant. So this means that implementing NON-UNI as a stack, instead of a queue, should reduce the amount of required memory from $O(N)$ down to $O(log\ N)$, which

is a tremendous improvement.

In LattE, we added a new optional *Memory Saving* mode that employs (2.2) and the stack implementation of NON-UNI as described above. We have observed that the new Memory Saving mode uses considerably less memory than traditional LattE; often the total amount of memory used is reduced by a factor of a hundred or more.

2.2 In Pursuit of a Suitable c Vector

The skeptical reader at this point may wonder how the Memory Saving version of LattE constructs the $c \in \mathbf{Z}^d$ that is demanded in step (2) of Algorithm (2.1). In practice, we just choose $c \in \{1, 2, \ldots, N\}^d$ "at random," using a pseudo-random number generator, where N is a fairly large integer whose standard input size is about the same as the standard input size of A and b. It is not hard to see that, with probability close to 1, such c will satisfy the required condition $c \cdot v_{ij} \neq 0 \; \forall v_{ij}$. If it does happen that $c \cdot v_{ij} = 0$ for some v_{ij}, then a new c is chosen and LattE restarts.

In practice, this method for finding c is very efficient and effective. However, it is not very satisfactory from a theoretical standpoint. After all, it might happen that we are unlucky, and that LattE restarts many times before a value of c is produced that satisfies the condition $c \cdot v_{ij} \neq 0$ for all v_{ij}.

2.2.1 A Deterministic Memory-Efficient Construction of c

So, in order to certify that Algorithm (2.2) can indeed be used as a memory-efficient polynomial time algorithm to count lattice points in fixed dimension, we need to show how to easily and *deterministically* construct c in a memory-efficient manner.

Notice that we can use a slightly modified version of Algorithm (2.2) to compute the maximum L_∞-norm $||v_{ij}||_\infty$ over all v_{ij}, as follows:

Algorithm 2.3. *Computing the maximum norm* $||v_{ij}||_\infty$

1. *Set MAX-NORM* $:= 0$
2. *Apply the algorithm in Theorem (1.6) for computing the function* $f(P \cap \mathbf{Z}^d; z)$*, but each time a unimodular cone* U *is produced in the cone decomposition process, such that* U *would contribute a function* $h(z) = E\ \frac{z^u}{\prod_{j=1}^{n}\left(1-z^{v_j}\right)}$ *to the summation (1.10), then immediately do the following:*
   (a) *Compare each* $||v_j||_\infty$ *to MAX-NORM. If any* $||v_j||_\infty$ *is larger than MAX-NORM, then set MAX-NORM* $:= max\{||v_j||_\infty\}$
   (b) *Discard the cone* U *from memory*

□

By the following proposition, we can define $c = (c_1, c_2, \ldots, c_d)^t$ by setting each $c_k = M^{k-1}$, where $M =$ MAX-NORM $+ 1$, and this c will satisfy $c \cdot v_{ij} \neq 0$ for all v_{ij}.

Proposition 2.4. *Suppose* $\{v_{ij}\}$ *is a collection of non-zero integer vectors in* $\mathbf{R}^d$*. Let* $M = 1 + max\{||v_{ij}||_\infty\}$*, and define* $c = (c_1, c_2, \ldots, c_d)^t$ *by setting each* $c_k = M^{k-1}$*. Then* $c \cdot v_{ij} \neq 0$ *for all* v_{ij}*.*

Proof. Suppose that we have $c \cdot v_{ij} = 0$ for some v_{ij}, where $v_{ij} = (a_1, a_2, \ldots, a_d)^t$. We have $v_{ij} \neq 0$ by definition, so let l be the largest subscript such that $a_l \neq 0$. Obviously we must have $l > 1$ because $c \cdot v_{ij} = 0$ and $c_1 \neq 0$.
Then,

$$\sum_{k=1}^{l} c_k a_k = 0 \tag{2.1}$$

so that

$$c_l a_l = -\sum_{k=1}^{l-1} c_k a_k \ . \tag{2.2}$$

Taking the absolute value of both sides, and using the fact that $a_l \in \mathbf{Z}\backslash\{0\}$, along with the triangle inequality, we have

$$|c_l| \leq |c_l a_l| = |\sum_{k=1}^{l-1} c_k a_k| \leq \sum_{k=1}^{l-1} |c_k a_k| \ , \tag{2.3}$$

which implies

$$M^{l-1} \leq \sum_{k=1}^{l-1} M^{k-1}|a_k| \leq \sum_{k=1}^{l-1} M^{k-1}(M-1) \tag{2.4}$$

where the last inequality follows from the fact that each $|a_k|$ satisfies $M \geq |a_k| + 1$ by definition of M. But this is a contradiction, because

$$\sum_{k=1}^{l-1} M^{k-1}(M-1) = \sum_{k=1}^{l-1} M^k - \sum_{k=1}^{l-1} M^{k-1} = \ (M^{l-1} - 1) \ . \tag{2.5}$$

The proposition follows. □

2.3 Concluding Remarks

Barvinok's original counting algorithm constructed a very similar c, after all the v_{ij} were calculated and stored in memory [2]. Note that Algorithm (2.3) uses essentially the same amount of memory as Algorithm (2.2). Furthermore, when d is a fixed constant, the standard input size of c is bounded by a polynomial in the standard input size of the v_{ij} (which in turn is bounded by a polynomial in the standard input size of A and b). Thus, using this c, Algorithm (2.2) will run in polynomial time when d is considered fixed.

So, by first running Algorithm (2.3) to compute c, and then running Algorithm (2.2), indeed we can efficiently implement Barvinok's method for counting lattice points, and use far less memory.

Chapter 3

Integer Programming via Barvinok's Rational Functions

3.1 Integer Programming: Overview

We begin by defining the Integer Programming problem.

Definition 3.1. *Suppose we are given $A \in \mathbf{Z}^{m\times d}$, $b \in \mathbf{Z}^m$, and $c \in \mathbf{Z}^d$. We define a rational polyhedron $P \subset \mathbf{R}^d$ by $P = \{x \in \mathbf{R}^d : Ax \leq b\}$.*
Then the Integer Programming problem for (A, b, c) asks us to determine the following:

1. *Whether $P \cap \mathbf{Z}^d$ is empty*
2. *Whether the set $\{c \cdot \alpha : \alpha \in P \cap \mathbf{Z}^d\}$ is bounded above (provided that $P \cap \mathbf{Z}^d$ is not empty)*

If it so happens that $P \cap \mathbf{Z}^d$ is not empty, and $\{c \cdot \alpha : \alpha \in P \cap \mathbf{Z}^d\}$ is indeed bounded, then the Integer Programming problem also asks us to find $x^* \in P \cap \mathbf{Z}^d$ such that

$$x^* \cdot c = max\{c \cdot \alpha : \alpha \in P \cap \mathbf{Z}^d\} \tag{3.1}$$

The art of solving instances of the Integer Programming problem is commonly called

Integer Programming. It turns out that many natural problems that arise in Industry can be restated as Integer Programming (IP) instances, and thus there has been an incredible amount of time, attention, and money devoted to developing techniques for solving them.

The commercial software Cplex is essentially the standard software used to solve IP instances in industry. As one might expect, Cplex often performs very well, even when d is large.

Intriguingly, though, there are also small, easy to state IP instances that Cplex cannot solve within reasonable time and memory constraints. Aardal et. al [1] give a collection of such examples which are low dimensional knapsack problems.

3.1.1 Hardness of Integer Programming

That Cplex sometimes fails to solve such innocent looking instances is perhaps not so surprising in light of the fact that, when d is not considered fixed, the Integer Programming problem is known to be *NP-Hard*. But moreover, even when d is considered fixed, the popular "branch-and-bound" style algorithm that Cplex implements is not guaranteed to run in polynomial time. However, Barvinok's Counting Algorithm, for instance, is guaranteed to run in polynomial time when d is fixed. Thus, when Cplex fails on small IP instances such as those given by Aardal, other approaches may fare much better.

3.2 From Rational Functions to Integer Programming

As mentioned in Chapter 1, Barvinok's Counting Algorithm gives a more generalized alternative to Lenstra's polynomial time algorithm [16] for deciding whether a rational polytope contains lattice points when d is considered fixed.

It is not hard to see that Lenstra's algorithm can be used to solve the Integer Pro-

gramming problem in polynomial time when d is fixed and P is a polytope. The gist of the approach is to add the additional constraints

$$c \cdot x \geq L, \ \ and \ \ c \cdot x \leq U \tag{3.2}$$

to our original system of inequalities $Ax \leq b$ that defines P, and then use Lenstra's algorithm to detect whether the resultant polytope contains any lattice points. By repeatedly performing this process, and systematically varying L and U, we can perform a binary search to find the optimal value M of the IP instance. Once M is known, we can add the constraints $c \cdot x \geq M$, $c \cdot x \leq M$ to our original system of inequalities, and then apply Lenstra's algorithm to the resultant polytope in order to produce an optimal solution. (Lenstra's algorithm actually outputs a lattice point from the polytope when lattice points are present).

3.2.1 The BBS Algorithm

Alternatively, we could try the same binary search idea, but using Barvinok's Counting Algorithm instead of Lenstra's algorithm. The resultant method for solving IP instances, called the *BBS Algorithm*, is presented by De Loera et. al in [8].

Intuitively, the *BBS Algorithm* might seem wasteful because it executes Barvinok's Main Algorithm (given in Theorem (1.6)) many times during the binary search process. We know that we can compute $f(P \cap \mathbf{Z}^d; z)$ by using Barvinok's Main Algorithm just once, where $P = \{x \in \mathbf{R}^d : Ax \leq b\}$. Can we then somehow use $f(P \cap \mathbf{Z}^d; z)$, in order to "read off" the solution to our IP instance, by inspection?

3.3 An Alternative to BBS: Reading Off IP Solutions from Rational Functions

If we could in fact "read off" IP solutions from $f(P \cap \mathbf{Z}^d; z)$, then we could compute $f(P \cap \mathbf{Z}^d; z)$ just once, using Barvinok's main algorithm, and then re-use it many times, to obtain IP solutions for not just one, but many different values of c. This would be particularly well suited for large families of IP instances that share the same A and b, but have different c vectors.

So, to this end, suppose $A \in \mathbf{Z}^{m \times d}$ and $b \in \mathbf{Z}^m$ are given, where d is a fixed constant. We consider the family of IP instances $\{(IP)_c\}$, defined as

$$(IP)_c : \text{ Solve the IP problem defined by}(A, b, c) \tag{3.3}$$

We restrict our attention to the case where the system of inequalities $Ax \leq b$ defines a polyhedron P $\subset \mathbf{R}^d$, such that P $\cap \mathbf{Z}^d$ is nonempty. (Because we can initially use Barvinok's Counting Algorithm, or Lenstra's algorithm, to determine whether P contains any lattice points.)

Using Barvinok's Main Algorithm, we can explicitly compute the function $f(P \cap \mathbf{Z}^d; z) = \sum_{\alpha \in P \cap \mathbf{Z}^d} z^\alpha$ in polynomial time, in the following form:

$$f(P \cap \mathbf{Z}^d; z) = \sum_{i \in I} E_i \frac{z^{u_i}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})}, \tag{3.4}$$

where all $E_i \in \{1, -1\}$, and $u_i \in \mathbf{Z}^d$ and $v_{ij} \in \mathbf{Z}^d \backslash \{0\}$.

Suppose that for a given $c \in \mathbf{Z}^d$, we have $c \cdot v_{ij} \neq 0$ for all v_{ij}.
We may then enforce that $c \cdot v_{ij} < 0$ for all v_{ij} appearing in (3.4), by employing the

identity

$$\frac{1}{1-z^{v_{ij}}} = \frac{-1}{1-z^{-v_{ij}}}, \tag{3.5}$$

in (3.4), for any v_{ij} such that $c \cdot v_{ij} > 0$. (We may have to change some of the E_i and u_i and v_{ij} by using our identity, but we will slightly 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}.)

Having enforced that $c \cdot v_{ij} < 0$ for all v_{ij} appearing in (3.4), now put

$$M = max\ \{c \cdot u_i | i \in I\} \tag{3.6}$$

and

$$T = \{i \in I | c \cdot u_i = M\} \tag{3.7}$$

Finally, put

$$\sigma = \sum_{i \in T} E_i \tag{3.8}$$

3.3.1 The "Inspection" Theorems

In the following theorems, P, c, M, σ, T are all as defined in the preceding discussion. The following result was proved by Jean B. Lasserre [15]:

Theorem 1. *If P is a polytope, and if $c \cdot v_{ij} \neq 0$ for all v_{ij} appearing in (3.4), then M is an upper bound for the optimal value of the integer program $(IP)_c$. If in addition, $\sigma \neq 0$, then M is the optimal value of the integer program $(IP)_c$.*

The author independently proved the result (1), along with the following stronger statement:

Theorem 2. *If the optimal value of $(IP)_c$ is finite, and if $c \cdot v_{ij} \neq 0$ for all v_{ij} appearing in (3.4), then M (as defined above) is an upper bound for the optimal value of the integer program $(IP)_c$. If in addition, $\sigma \neq 0$, then:*

1. *M is the optimal value of the integer program $(IP)_c$, and*
2. *Moreover, there exists an $i \in T$ such that u_i is an optimal solution for $(IP)_c$.*

While Theorem (1) is interesting from a theoretical perspective, in practice the optimal value of an IP instance is often of limited use, unless an actual *optimal solution* attaining that optimal value is also known. For example, in many IP instances, the optimal value represents the maximum amount of profit that a company can earn within production constraints, whereas the optimal solution represents how the company can adjust its production parameters in order to actually achieve that maximum profit. Thus, Theorem (2)'s most important additional feature is that when the hypotheses are satisfied, we can recover an optimal solution for $(IP)_c$, along with the optimal value.

3.3.2 An Inspection Heuristic for Solving Families of IP Instances

Algorithm 3.2. *A Heuristic for Solving a (Finite) Collection of IP Instances of the Form $(IP)_c$*

1. *Use Barvinok's algorithm to compute $f(P \cap \mathbf{Z}^d; z)$ in the form given in (3.4)*
2. *For each $(IP)_c$ we wish to solve, such that $c \cdot v_{ij} \neq 0$ for all v_{ij} in (3.4):*
   (a) *Use the identity (3.5) as necessary to enforce that $c \cdot v_{ij} < 0$ for all v_{ij} appearing in (3.4)*
   (b) *Check to see if the hypotheses of Theorem (2) are met. If not, then output "?".*
   (c) *Otherwise, find $i \in T$ that satisfies $Au_i \leq b$, and output M as the optimal value of $(IP)_c$, and output u_i as an optimal solution.*

Of course, the preceding is not really an algorithm, because we cannot guarantee that it will solve all given $(IP)_c$. (In fact, for some unlucky choice of A and b, perhaps the "heuristic" always fails to solve $(IP)_c$, regardless of c.)

3.3.3 The Case of the Vanishing σ

For a given non-zero c, we can take suitable precautions to work around the required condition that $c \cdot v_{ij} \neq 0$ holds for all v_{ij} in (3.4). (Essentially we can multiply the objective vector c by a large enough positive integer and then perturb it slightly by a suitable integer vector. Then an optimal solution for this new slightly perturbed objective vector will also be an optimal solution for the original objective vector.)

On the other hand, what if Barvinok's algorithm is doomed to produce rational functions for which the probability of $\sigma = 0$ occurring is appreciable? The LattE team has tested many examples, and while on many examples we observed that indeed $\sigma \neq 0$, we also discovered that routinely the condition failed, so that Theorems (1) and (2) could not be applied.

Hence, we need a strengthening of the Inspection Theorems, if we hope to obtain a reliable method for solving IP instances.

3.4 Inspecting Deeper: The Digging Algorithm

3.4.1 The Laurent Series Approach

The original proofs of the Inspection Theorems were somewhat dissimilar from one another, but both proofs used arguments about the asymptotic behavior of the rational functions appearing in (3.4), for particular large values of z. As it turns out, ideas regarding monomial substitutions and convergence of particular Laurent series make the proofs much easier, and also lay the groundwork for a more generalized algorithm for extracting IP solutions from rational functions. Many of the ideas re-

garding monomial substitutions and convergence of Laurent series are similar to those presented by Barvinok [2, 4] in other contexts.

We will no longer limit our consideration to generating functions for lattice points inside of rational polyhedra. Indeed, recent work by Barvinok and Woods [5] shows that generating functions can be quickly computed (in rational function form) for many other interesting types of sets of lattice points.

Instead, we will assume that we have a *pointed* lattice point set $S \subset \mathbf{Z}^d$, and an objective vector $c \in \mathbf{Z}^d$. By *pointed*, we mean that S is contained in some closed cone $K \subset \mathbf{R}^d$ that contains no straight lines.

We define a (somewhat generalized) integer programming problem Π_c by

$$\Pi_c : \quad \text{Find } \pi = \max\{c \cdot x \mid x \in S\}, \text{ and } x^* \in S \text{ satisfying } c \cdot x^* = \pi \tag{3.9}$$

As usual, we call π the optimal value of Π_c, and we call x^* an optimal solution. We will assume that $c \neq 0$, and that S is nonempty and that π actually exists and is finite. (These are mild restrictions, that basically require that Π_c actually be an interesting IP instance. At any rate, our restrictions are no stronger than those required by the Inspection Theorems.)

Define

$$f(S; z) = \sum_{\alpha \in S} z^{\alpha}. \tag{3.10}$$

We assume that the function $f(S; z)$ has been computed in Barvinok's form:

$$f(S; z) = \sum_{i \in I} E_i \frac{z^{u_i}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})} \ , \tag{3.11}$$

where I is a finite indexing set, and where all $E_i \in \{1, -1\}$ and u_i , $v_{ij} \in \mathbf{Z}^d$.

As mentioned in section (3.3.3) , we may safely assume that $c \cdot v_{ij} \neq 0$ for all v_{ij}.

Letting c_k denote the kth component of c, we make the substitutions $z_k \rightarrow y_k t^{c_k}$, for $k = 1, \ldots, d$, and then (3.11) yields a multivariate rational function in the vector variable y and scalar variable t:

$$g_S(y,t) = \sum_{i \in I} E_i \frac{y^{u_i} t^{c \cdot u_i}}{\prod_{j=1}^{n_i} (1 - y^{v_{ij}} t^{c \cdot v_{ij}})}. \tag{3.12}$$

Now, since the optimal value π of our IP instance is finite, it is clear from (3.10), and by definition of our substitutions, that the Laurent expansion

$$g_S(y,t) = \sum_{\alpha \in S} y^{\alpha} t^{c \cdot \alpha} \tag{3.13}$$

will converge absolutely, for all complex y and t such that $|t^{-1}| < 1$ and such that y lies within the region of absolute convergence for the expansion of f_S given in (3.10). Notice that since S is a pointed set, we are guaranteed some nonempty region in $\mathbf{C}^d$ where the expansion (3.10) does in fact converge absolutely [3].

We may employ the identity

$$\frac{1}{1 - y^{v_{ij}} t^{c \cdot v_{ij}}} = \frac{-y^{-v_{ij}} t^{-c \cdot v_{ij}}}{1 - y^{-v_{ij}} t^{-c \cdot v_{ij}}} \tag{3.14}$$

in (3.12), for any v_{ij} such that $c \cdot v_{ij} > 0$. We therefore enforce that all v_{ij} in (3.12) satisfy $c \cdot v_{ij} < 0$. (We may have to change some of the E_i, u_i and v_{ij} using our identity, but we will slightly 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 $0 < r < 1$ and $|t^{-1}| < r$, each of the rational functions in the summation will have a Laurent expansion of the form

$$E_i \, y^{u_i} t^{c \cdot u_i} \prod_{j=1}^{n_i} (1 + y^{v_{ij}} t^{c \cdot v_{ij}} + (y^{v_{ij}} t^{c \cdot v_{ij}})^2 + \ldots) \tag{3.15}$$

valid for those complex values of y that satisfy $|y^{v_{ij}}| < r^{-1}$ for all v_{ij}. Note that multiplication and addition of the resultant series is valid for such y and t, and we thereby obtain from (3.12) a series expansion

$$g_S(y,t) = \sum_{\alpha \in \mathbf{Z}^d, n \in \mathbf{Z}} a_{\alpha,n} y^{\alpha} t^n \tag{3.16}$$

valid for such y and t, where the coefficients $a_{\alpha,n}$ can be easily calculated.

We will now prove that the coefficients of the series (3.16) and (3.13) are equal:

Note that we only have finitely many v_{ij}. Thus for any $s > 1$, we can choose $0 < r < 1$ small enough, such that whenever the coordinates y_k of y all satisfy $\frac{1}{s} < |y_k| < s$, then we have $|y^{v_{ij}}| < r^{-1}$ for all v_{ij}.

This means that for any $s > 1$, we can choose $0 < r < 1$ small enough, so that the series (3.16) will converge for all (y,t) in the following region R:

$$R = \{(y,t) \ \mid \ t \neq 0, \ |t^{-1}| < r, \ \textit{and} \ \frac{1}{s} < |y_k| < s \ \forall k\} \tag{3.17}$$

Thus, for s sufficiently large and suitably chosen r, the region of convergence for the series (3.16) will intersect the region of convergence for the series (3.13), such that an open neighborhood is contained in the intersection. Inside this neighborhood, both series (3.13) and (3.16) are valid, but they represent the same function, namely $g_S(y,t)$, so we conclude that the corresponding coefficients of both series must be equal, as was to be shown.

3.4.2 Formal Description of the Digging Algorithm

We have thus shown that the coefficients of the series (3.13) can be algorithmically determined from the rational functions appearing in (3.11). Notice that all of the geometric series appearing in (3.15) have terms whose degree in t are strictly decreasing. Thus, when we multiply and add such series to obtain the series (3.13), we can easily

do so in a fashion that lists the terms of (3.13), in order, with respect to decreasing degree in t.

This allows an algorithm to solve Π_c. The algorithm proceeds as follows:

Algorithm 3.3. *The Digging Algorithm*

1. *Make the substitutions $z_k = y_k t^{c_k}$, for $k = 1, \ldots, d$, into (3.11), to obtain (3.12).*
2. *Use the identity (3.14) as necessary to enforce that all v_{ij} in (3.12) satisfy $c \cdot v_{ij} < 0$.*
3. *Via the expansion formulas (3.15), proceed calculating the expansion (3.13) by calculating the terms' coefficients, proceeding in decreasing order with respect to the degree of t. Continue until a degree M of t is found such that for some $\alpha \in \mathbf{Z}^d$, the coefficient of $y^\alpha t^m$ is non-zero in the expansion (3.13).*
4. *Return "$\pi = M$" as the optimal value of the integer program Π_c, and return α as an optimal solution.*

□

3.4.3 The Digging Algorithm Generalizes the Inspection Theorems

Note that when the Digging Algorithm begins step (3), the first terms that are calculated (proceeding in decreasing order with respect to degree of t) are terms of the form $y^{u_i} t^M$, where $i \in T$, where

$$M = max\ \{c \cdot u_i \mid i \in I\ \ (3.11)\} \tag{3.18}$$

and

$$T = \{i \in I \mid c \cdot u_i = M\} \tag{3.19}$$

By definition, the calculated coefficient of each such term $y^{u_i}t^M$ will either vanish or equal 1. Recalling the Inspection Theorems and the notation used there, we observe the following:

The condition "$\sigma = 0$" in the Inspection Theorems is equivalent to having all the calculated coefficients vanish for the $y^{u_i}t^M$ terms

Thus, the Digging Algorithm, and the discussion proving its correctness, give a generalization of the Inspection Theorems that is guaranteed to provide a solution even when $\sigma = 0$.

3.5 Implementing The Digging Algorithm in LattE

LattE was naturally suited for implementing the Digging Algorithm to solve integer programming instances defined over polyhedra P, since the software was already equipped to compute the necessary generating function $f(P \cap \mathbf{Z}^d; z)$. So, after inventing the Digging Algorithm, the LattE team implemented it and compared the method to both the BBS Algorithm (described earlier) and Cplex.

3.5.1 Performance Compared to BBS and Cplex

In the paper [8], De Loera et. al describe the performance of LattE's Digging Algorithm approach, compared to the BBS Algorithm approach and the software Cplex (v 6.6). In particular, the programs were run on variants of Aardal et al's "hard" low-dimensional examples [1]. As described in [8], we found that Cplex 6.6 could only solve one of the problems (Cplex ran out of memory on the rest), whereas the Digging Algorithm approach solved most of them in a matter of seconds. Somewhat surprisingly, the BBS Algorithm approach failed to solve most of the problems. (In each case, the program was killed after many hours or days of computation.)

So, it seems that in practice, the Digging Algorithm approach is a considerable alter-

native to Cplex on "hard" IP instances when d is small. Furthermore, the Digging Algorithm (unlike Cplex) has the additional benefit of being well-suited for families of IP instances that share the same matrix A and vector b. On such families, we only need to use Barvinok's Main Algorithm once to calculate $f(P \cap \mathbf{Z}^d; z)$, and then we can use the same generating function for each IP instance in the family.

3.6 Complexity of the Digging Algorithm in Fixed Dimension

At first it seemed promising that the Digging Algorithm approach might be a polynomial time algorithm in fixed dimension, since it is based upon Barvinok's Main Algorithm which enjoys such polynomial time guarantees. Unfortunately, however, the author has discovered that the Digging Algorithm might calculate an exponential number of coefficients before finding one that does not vanish. Thus, even in fixed dimension, the Digging Algorithm is not a polynomial time algorithm.

3.6.1 A Simple Example Where Digging Requires Exponential Time

Consider the family of closed convex quadrilaterals Q_N, where N is a positive integer, with vertices $(\frac{1}{2}, \frac{1}{2}), (\frac{3}{4}, \frac{1}{2}), (\frac{1}{2}, \frac{3}{4})$, and $(1, N)$. Each quadrilateral Q_N obviously contains exactly one lattice point, namely $(1, N)$. It is easy to see that the system of inequalities $Ax \leq b$ that define Q_N are given by a matrix A and vector b whose standard input size are bounded by a polynomial in $log\ N$. Thus Barvinok's Main Algorithm computes the generating function $f(Q_N \cap \mathbf{Z}^2; z)$ in time bounded by a polynomial in $log\ N$. Remember that, via Brion's theorem, $f(Q_N \cap \mathbf{Z}^2; z)$ is computed as the sum of the generating functions corresponding to the four tangent cones of Q_N. In particular, the tangent cone at vertex $(\frac{1}{2}, \frac{1}{2})$ is already unimodular, and its generating function is

$$\sum_{m,n \in \mathbf{N}} {z_1}^m {z_2}^n = \frac{z_1 z_2}{(1-z_1)(1-z_2)} \tag{3.20}$$

and thus the above rational function will appear in the formula calculated for $f(Q_N \cap \mathbf{Z}^2; z)$. So, if we take our objective vector to be $c = (-1, -1)^t$, then, upon making the corresponding substitutions $z_1 = y_1 t^{-1}$ and $z_2 = y_2 t^{-1}$, this rational function becomes

$$\sum_{m,n \in \mathbf{N}} {y_1}^m {y_2}^n t^{-m-n} = \frac{y_1 y_2 t^{-2}}{(1-y_1 t^{-1})(1-y_2 t^{-1})} \tag{3.21}$$

This rational function's Laurent expansion includes the terms $y_1 y_2 t^{-2}, y_1 {y_2}^2 t^{-3}, y_1 {y_2}^3 t^{-4}, \ldots, y_1 {y_2}^{N-1} t^{-N}$, and so the Digging Algorithm will have to calculate the coefficients (which will all vanish) for all of these $(N-1)$ terms, before finally calculating the non-vanishing coefficient of the term $y_1 {y_2}^N t^{-N+1}$. Thus, the Digging Algorithm will have to calculate coefficients for at least N terms, which is certainly exponentially many with respect to $log\ N$. Thus, for the family of quadrilaterals Q_N, the Digging Algorithm cannot run in polynomial time.

Chapter 4

Generalized Counting: Efficiently Summing Polynomials Over Lattice Point Sets in Fixed Dimension

4.1 A Motivation from Statistics

A large area of study and research in Statistics and Operations Research involves contingency tables. We borrow the following definition and example provided by Ruriko Yoshida in her soon to be published thesis:

Definition 4.1. *A* s**-table of size** $(n_1, \ldots, n_s)$ *is an array of non-negative integers* $v = (v_{i_1,\ldots,i_s})$, $1 \leq i_j \leq n_j$. *For* $0 \leq L < s$, *an* L**-marginal** *of* v *is any of the* $\binom{s}{L}$ *possible* L*-tables obtained by summing the entries over all but* L *indices.*

Example 4.2. *Consider a* 3*-table* $X = (x_{ijk})$ *of size* $(m,\ n,\ p)$, *where* m, n, *and* p *are natural numbers. Let the integral matrices* $M_1 = (a_{jk})$, $M_2 = (b_{ik})$, *and* $M_3 = (c_{ij})$ *be* 2*-marginals of* X, *where* M_1, M_2, *and* M_3 *are integral matrices of type* $n \times p$, $m \times p$, *and* $m \times n$ *respectively. Then, a* 3*-table* $X = (x_{ijk})$ *of size* $(m,\ n,\ p)$ *with given*

marginals satisfies the system of equations and inequalities:

$$\begin{aligned}
&\textstyle\sum_{i=1}^{m} x_{ijk} = a_{jk},\ (j = 1, 2, ..., n,\ k = 1, 2, ..., p),\\
&\textstyle\sum_{j=1}^{n} x_{ijk} = b_{ik},\ (i = 1, 2, ..., m,\ k = 1, 2, ..., p),\\
&\textstyle\sum_{k=1}^{p} x_{ijk} = c_{ij},\ (i = 1, 2, ..., m,\ j = 1, 2, ..., n),\\
&x_{ijk} \geq 0,\ (i = 1, 2, ..., m,\ j = 1, 2, ..., n,\ k = 1, 2, ..., p).
\end{aligned} \tag{4.1}$$

Such tables appear frequently in Statistics and Operations Research, under names such as *multi-way contingency tables*, or *tabular data*. Official Census data, and the published margins thereof, provide one of many important settings where large-scale contingency tables naturally arise.

From the definition (4.1), we see that a contingency table with particular marginals is really nothing more than a collection of integer values assigned to a set a variables, so as to obey certain linear constraints. Thus, given a complete particular set of marginals, we can easily represent the set of all contingency tables that have those marginals, as the set of lattice points inside a rational polytope. Under this identification, the coordinates of each lattice point will equal the entries in its corresponding contingency table.

It is often the goal to infer relationships among data in a given contingency table. To this end, it is natural to ask what a "typical" table with the same marginals might look like, so that we can identify any considerably atypical characteristics in our given table. Determining what a "typical" table looks like often involves summing a function over all contingency tables that have the particular margins.

For instance, given a particular contingency table, we could ask whether a particular entry's value α is unusually large. To answer such a question, we might consider the mean and variance of a random variable X, where X is defined as the entry's value α_T in a table T which is chosen at random from the set of all contingency tables that have the same marginals as our given table. Note that if we know the number of

contingency tables satisfying the particular marginals, then calculating the mean of X is computationally equivalent to just summing the particular entry's value α_T over all the contingency tables T which satisfy the marginals. Similarly, once this mean $E(X)$ is known, then calculating the variance of X is computationally equivalent to summing the function $(E(X) - \alpha_T)^2$ over all the contingency tables T which satisfy the marginals.

4.2 The General Problem

More generally, we consider the problem of efficiently summing a polynomial $q \in \mathbf{Q}[\alpha_1, \alpha_2, \ldots, \alpha_d]$ over the lattice points in a finite set $S \subset \mathbf{Z}^d$:

$$\sigma(S, q) = \sum_{\alpha \in S} q(\alpha_1, \alpha_2, \ldots, \alpha_d) \tag{4.2}$$

As usual, we will assume that d is a fixed constant. We will assume that the generating function $f(S; z)$ has been computed in Barvinok's form:

$$f(S; z) \quad = \quad \sum_{\alpha \in S} z^{\alpha} \quad = \quad \sum_{i \in I} E_i \frac{z^{u_i}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})} \ , \tag{4.3}$$

We will also assume that the degree of q is no more than D in any variable, where D is a fixed constant. This is not unreasonable in the context of statistics, for example, where often one is only interested in polynomials whose degree in any variable is at most 1 or 2.

4.3 An Algorithm for Efficiently Summing Fixed Degree Polynomials Over Lattice Point Sets, via Partial Derivatives of Barvinok's Rational Functions

In search of a solution for the general problem of evaluating $\sigma(S, q)$, note that the partial derivative of the function $f(S; z)$ with respect to z_k is

$$\frac{\partial f(S;z)}{\partial z_k} = \sum_{\alpha \in S} \frac{\partial (z^\alpha)}{\partial z_k} = \sum_{\alpha \in S} \alpha_k z^{\alpha - e_k} \tag{4.4}$$

Thus,

$$z_k \left(\frac{\partial f(S;z)}{\partial z_k} \right) = \sum_{\alpha \in S} \alpha_k z^\alpha \tag{4.5}$$

We can define a linear operator $L_k = z_k \frac{\partial}{\partial z_k}$ and succinctly express (4.5) as

$$L_k \left[f(S;z) \right] = \sum_{\alpha \in S} \alpha_k z^\alpha \tag{4.6}$$

It is not hard to show that the operators $L_1, L_2, \ldots, L_d$ are linear operators that commute, and that

$$\left({L_1}^{m_1} {L_2}^{m_2} \ldots {L_d}^{m_d} \right) \left[f(S;z) \right] = \sum_{\alpha \in S} ({\alpha_1}^{m_1} {\alpha_2}^{m_2} \ldots {\alpha_d}^{m_d}) z^\alpha \tag{4.7}$$

Thus, in the special case that the polynomial function $q(\alpha_1, \alpha_2, \ldots, \alpha_d)$ is equal to a single monomial ${\alpha_1}^{m_1} {\alpha_2}^{m_2} \ldots {\alpha_d}^{m_d}$, we have that

$$\sigma(S,q) = \sum_{\alpha\in S} {\alpha_1}^{m_1}{\alpha_2}^{m_2}\dots{\alpha_d}^{m_d} \tag{4.8}$$

$$= \lim_{(z_1,z_2,\dots,z_d)\to(1,1,\dots,1)} {L_1}^{m_1}{L_2}^{m_2}\dots{L_d}^{m_d}\,[f(S;z)] \tag{4.9}$$

We need the following lemma which states that given a function $f(z)$ in Barvinok's form, we can compute $L_k[f(z)]$ in Barvinok's form in polynomial time, when d is considered fixed.

Lemma 4.3. *Let us fix the dimension d and a positive integer M. Suppose we are given a function $f(z)$ in the following form:*

$$f(z) = \sum_{i\in I} E_i \frac{z^{u_i}}{\prod_{j=1}^{n_i}(1-z^{v_{ij}})} \ , \tag{4.10}$$

where I is a finite set, and where all $E_i \in \mathbf{Q}$, and $u_i, v_{ij} \in \mathbf{Z}^d$, $v_{ij} \neq 0$, and $n_i \leq M$. Then, for each $k = 1, 2, \dots, d$ there is a polynomial time algorithm which computes $L_k[f(z)]$ in the form:

$$L_k[f(z)] = \sum_{i\in I^*} E_i \frac{z^{u_i}}{\prod_{j=1}^{n_i}(1-z^{v_{ij}})} \ , \tag{4.11}$$

where I^* is a finite set, and where all $E_i \in \mathbf{Q}$, and $u_i, v_{ij} \in \mathbf{Z}^d$, and $v_{ij} \neq 0$. Furthermore, we have $\max_{i\in I^*} n_i \leq 2 \max_{i\in I} n_i$.

Proof. Suppose $f(z)$ is given in Barvinok's form (4.10). Let us consider any $k \in \{1, 2, \dots, d\}$. Then, because L_k is linear, we have

$$L_k[f(z)] = \sum_{i \in I} E_i L_k \left[\frac{z^{u_i}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})} \right] , \tag{4.12}$$

We will now show that for each $i \in I$, we can compute $g_i(z) = L_k \left[\frac{z^{u_i}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})} \right]$ in polynomial time in Barvinok's form:

$$g_i(z) = \sum_{i' \in I'} E_{i'} \frac{z^{u_{i'}}}{\prod_{j=1}^{n_{i'}} (1 - z^{v_{i'j}})} , \tag{4.13}$$

where I' is a finite set, and where all $E_{i'} \in \mathbf{Q}$, and $u_{i'}, v_{i'j} \in \mathbf{Z}^d$, and $v_{i'j} \neq 0$, and where $\max_{i \in I^*} n_i \leq 2 \max_{i \in I} n_i$. This will prove the lemma.

To compute $g_i(z)$ in the prescribed form, we begin with the observation that, by the quotient rule for derivatives,

$$\frac{\partial}{\partial z_k} \left(\frac{z^{u_i}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})} \right) = \frac{\left(\frac{\partial}{\partial z_k} z^{u_i} \right) \prod_{j=1}^{n_i} (1 - z^{v_{ij}}) \quad - \quad z^{u_i} \left(\frac{\partial}{\partial z_k} \prod_{j=1}^{n_i} (1 - z^{v_{ij}}) \right)}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})^2} \tag{4.14}$$

Because $n_i \leq M$, which is constant, we can completely expand the products $\prod_{j=1}^{n_i} (1 - z^{v_{ij}})$ appearing in the numerator, as a sum of no more than 2^{n_i} monomials, in polynomial time.

Then the numerator of the right-hand side of (4.14) will have the form

$$\left(\frac{\partial}{\partial z_k} z^{u_i}\right) \sum_{j=1}^{2^{n_i}} z^{w_{ij}} \quad - \quad z^{u_i} \left(\frac{\partial}{\partial z_k} \sum_{j=1}^{2^{n_i}} z^{w_{ij}}\right) \tag{4.15}$$

which we can easily compute (in polynomial time) as the following expanded sum of monomials:

$$\sum_{j=1}^{2^{n_i}} (u_{ik} - w_{ijk})\, z^{u_i + w_{ij} - e_k} \tag{4.16}$$

where u_{ik}, v_{ijk} denote the kth components of the vectors u_i and v_{ij}, respectively. Thus, in polynomial time, we can explicitly compute $g_i(z)$ as

$$g_i(z) \quad = \quad \frac{\sum_{j=1}^{2^{n_i}} (u_{ik} - w_{ijk})\, z^{u_i + w_{ij}}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})^2} \quad = \quad \sum_{j=1}^{2^{n_i}} \left((u_{ik} - w_{ijk}) \frac{z^{u_i + w_{ij}}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})^2} \right) \tag{4.17}$$

which can be easily expressed in the form prescribed in (4.13), thus completing the proof of the lemma.

□

With this lemma, we are ready to prove the following result.

Lemma 4.4. *Let us fix the dimension d and positive integers N, D. For finite $S \subset \mathbf{Z}^d$, suppose we are given the generating function $f(S; z)$ in the following form:*

$$f(S; z) = \sum_{i \in I} E_i \frac{z^{u_i}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})} \ , \tag{4.18}$$

where I is a finite set, and where all $E_i \in \mathbf{Q}$, and $u_i, v_{ij} \in \mathbf{Z}^d$, $v_{ij} \neq 0$, and $n_i \leq N$.

Suppose we are also given $m_1, m_2, \ldots, m_d \in \{0, 1, \ldots, D\}$ *that define a polynomial* $q(\alpha_1, \alpha_2, \ldots, \alpha_d) = {\alpha_1}^{m_1}{\alpha_2}^{m_2}\ldots{\alpha_d}^{m_d}$. *Then there is a polynomial time algorithm which computes* $g(z) = \sum_{\alpha \in S} q(\alpha_1, \alpha_2, \ldots, \alpha_d) z^{\alpha}$ *in the form:*

$$g(z) = \sum_{i \in I^*} E_i \frac{z^{u_i}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})} \ , \tag{4.19}$$

where I^* *is a finite set, and where all* $E_i \in \mathbf{Q}$, *and* $u_i, v_{ij} \in \mathbf{Z}^d$, *and* $v_{ij} \neq 0$. *Furthermore, we have* $\max_{i \in I^*} n_i \leq N 2^{dD}$.

Proof. Put $M = N2^{dD}$. Note M is constant because N, d, D are constant. We claim that given the generating function for $f(S; z)$ in Barvinok's form (4.3), we can apply the algorithm of Lemma 4.3 successively (each time with the same M) to compute the function $({L_1}^{m_1}{L_2}^{m_2}\ldots{L_d}^{m_d})\,[f(S; z)]$ in the form prescribed in (4.19). Indeed, note that since all $m_k \leq D$, we would successively apply the algorithm of Lemma 4.3 at most $d * D$ times. Thus, because we initially satisfy the condition that $n_i \leq N$ for all n_i, we are assured that, each time we successively apply the algorithm, we will satisfy the condition that $n_i \leq M$ for all n_i.

So, we can compute $({L_1}^{m_1}{L_2}^{m_2}\ldots{L_d}^{m_d})\,[f(S; z)]$ by successively applying the algorithm of Lemma 4.3 no more than $d * D$ times (each time with the same constant $M = N2^{dD}$). Furthermore, each time we apply the algorithm of Lemma 4.3, it runs in polynomial time.

Thus, because $d * D$ is constant, the total computation can be done in polynomial time. Furthermore, because we successively apply the algorithm of Lemma 4.3 at most $d * D$ times, we will have $\max_{i \in I^*} n_i \leq 2^{dD} \max_{i \in I} n_i$.

The proof is complete, via the identity (4.7).

□

Now, with Lemma (4.4) in hand, we are ready to prove the main result of the chapter.

Theorem 4.5. *Let us fix the dimension d and positive integers N, D. For finite $S \subset \mathbf{Z}^d$, suppose we are given the generating function $f(S; z)$ in the following form:*

$$f(S; z) = \sum_{i \in I} E_i \frac{z^{u_i}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})} \quad , \tag{4.20}$$

where I is a finite set, and where all $E_i \in \mathbf{Q}$, and $u_i, v_{ij} \in \mathbf{Z}^d$, $v_{ij} \neq 0$, and $n_i \leq N$.

Suppose we are also given $(D+1)^d$ rational numbers $b_{m_1,\ldots,m_d}$ for $m_1, m_2, \ldots, m_d \in \{0, 1, \ldots, D\}$, that define a polynomial $q(\alpha_1, \ldots, \alpha_d)$ as

$$q(\alpha_1, \ldots, \alpha_d) = \sum_{m_1,\ldots,m_d \in \{0,1,\ldots,D\}} b_{m_1,\ldots,m_d} {\alpha_1}^{m_1} \ldots {\alpha_d}^{m_d}. \tag{4.21}$$

Then there is a polynomial time algorithm which computes $g(z) = \sum_{\alpha \in S} q(\alpha_1, \ldots, \alpha_d) z^{\alpha}$ in the form:

$$g(z) = \sum_{i \in I^*} E_i \frac{z^{u_i}}{\prod_{j=1}^{n_i} (1 - z^{v_{ij}})} \quad , \tag{4.22}$$

where I^* is a finite set, and where all $E_i \in \mathbf{Q}$, and $u_i, v_{ij} \in \mathbf{Z}^d$, and $v_{ij} \neq 0$. Furthermore, we have $\max_{i \in I^*} n_i \leq N 2^{dD}$.

Proof. For any $c_1, c_2 \in \mathbf{Q}$, and any $q_1, q_2 \in \mathbf{Q}[\alpha_1, \ldots, \alpha_d]$, we have that

$$\sum_{\alpha \in S} \left(c_1 q_1(\alpha_1, \ldots, \alpha_d) + c_2 q_2(\alpha_1, \ldots, \alpha_d) \right) z^{\alpha} \tag{4.23}$$

is equal to

$$c_1 \left(\sum_{\alpha \in S} q_1(\alpha_1, \ldots, \alpha_d) z^{\alpha} \right) + c_2 \left(\sum_{\alpha \in S} q_2(\alpha_1, \ldots, \alpha_d) z^{\alpha} \right) \tag{4.24}$$

Thus, in order to compute $g(z)$ in Barvinok's form (4.22), we can simply compute

$$b_{m_1,\ldots,m_d} \sum_{\alpha \in S} {\alpha_1}^{m_1} \ldots {\alpha_d}^{m_d} z^{\alpha} \tag{4.25}$$

in Barvinok's form separately for each $(m_1, \ldots, m_d) \in \{0, 1, \ldots, D\}^d$, and then add the resultant functions. To compute each function (4.25) in Barvinok's form, we apply the algorithm given in Lemma (4.4) to compute $\sum_{\alpha \in S} {\alpha_1}^{m_1} \ldots {\alpha_d}^{m_d} z^{\alpha}$, and then multiply the resultant function by $b_{m_1,\ldots,m_d}$.

Computing $g(z)$ in this manner takes polynomial time, because there are only $(D+1)^d$ functions of the type (4.25) that we need to compute, and each can be computed in polynomial time by Lemma (4.4). (The quantity $(D+1)^d$ is constant because d, D are constant.)

Furthermore, we have $\max_{i \in I^*} n_i \leq N 2^{dD}$, because we are guaranteed as such by Lemma (4.4).

□

In [2], Barvinok showed how to compute limits of the following form in polynomial time, when d is fixed:

$$\lim_{(z_1,\ldots,z_d)\to(1,\ldots,1)} \sum_{i\in I} E_i \frac{z^{u_i}}{\prod_{j=1}^{n_i}(1-z^{v_{ij}})} \tag{4.26}$$

Thus, we immediately have the following corollary of Theorem (4.5).

Corollary 4.6. *Given the same hypotheses and input as in Theorem (4.5), there exists a polynomial time algorithm to evaluate the sum* $\sum_{\alpha\in S} q(\alpha_1,\ldots,\alpha_d)$

Proof. We can use Theorem (4.5) to compute $g(z) = \sum_{\alpha\in S} q(\alpha_1,\ldots,\alpha_d)z^{\alpha}$ in Barvinok's form in polynomial time. Then, using Barvinok's polynomial time limit calculation techniques [2], we can calculate the limit $\lim_{(z_1,\ldots,z_d)\to(1,\ldots,1)} g(z)$, which is precisely $\sum_{\alpha\in S} q(\alpha_1,\ldots,\alpha_d)$.

□

Corollary (4.6) finally provides the efficient method we were seeking, for summing polynomials over finite lattice point sets.

4.4 A Comment on Practicality

In practice, taking just one partial derivative of a sum of rational functions will increase the number of terms appearing in a formula such as (4.3) by a factor of 2^M, where $M = \max n_i$. Thus, there will be cases where a formula for $f(S;z)$ can be computed and stored in memory, whereas evaluating a reasonable degree polynomial over S, using Theorem (4.5) and Corollary (4.6), requires too much computation time and memory. We have not yet implemented the algorithm of Theorem (4.5) in LattE, so while the results of this chapter are perhaps interesting from a theoretical perspective, we do not yet know how well the presented algorithm performs in practice.

Bibliography

[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, 350-366.

[2] 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.

[3] Barvinok, A.I. *A course in convexity.* Graduate Studies in Mathematics, volume 54, American Mathematics Society.

[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.* Journal of the American Mathematical Society, 16, 2003, 957–979.

[6] Brion, M. *Points entiers dans les polyèdres convexes. Ann. Sci. École Norm. Sup.* 21, 1988, 653-663.

[7] Cox, D., Little, J., and O'Shea, D. *Using Algebraic Geometry*. Springer Verlag, Undergraduate Text, 2nd Edition, 1997.

[8] De Loera, J.A., Haws, D., Hemmecke, R., Huggins, P., and Yoshida, R. *Three kinds of integer programming algorithms based on Barvinok's rational functions*. To appear in Integer Programming and Combinatorial Optimization: 10th International IPCO Conference, 2003.

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

[10] 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/`

[11] Diaconis, P. and Gangolli, A. *Rectangular arrays with fixed margins*. Discrete probability and algorithms (Minneapolis, MN, 1993), 15–41, IMA Vol. Math. Appl., 72, Springer, New York, 1995.

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

[13] Fienberg, S.E. and Makov, U.E. and Meyer, M.M. and Steele, R.J. *Computing the exact conditional distribution for a multi-way contingency table conditional on its marginal totals*. In *Data Analysis From Statistical Foundations*, 145–166, Nova Science Publishers, 2001, A. K. Md. E. Saleh, Huntington, NY.

[14] Grötschel, M., Lovász, L., and Schrijver, A. *Geometric algorithms and combinatorial optimization*. Second edition. Algorithms and Combinatorics, 2, Springer-Verlag, Berlin, 1993.

[15] Lasserre, J.B. *Integer programming, Barvinok's counting algorithm and Gomory relaxations*. Operations Research Letters, 32, N2, 2004, 133 – 137.

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

[17] Nijehuis, A. and Wilf, H. *Representations of integers by linear forms in non-negative integers*. J. Number Theory 4, 1972, 98–106.

[18] Schmidt, J.R. and Bincer, A. *The Kostant partition function for simple Lie algebras*. In *J. Mathematical Physics* 25, 1984, 2367–2373.

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

[20] Stanley, R.P. *Combinatorics and Commutative Algebra*. Second edition. Progress in Mathematics, 41. Birkhäuser, Boston, 1996.

[21] Stanley, R.P. *Enumerative Combinatorics*. Volume I, Cambridge, 1997.