Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

# A Combinatorial Decomposition of Knapsack Cones

**Guoce Xin**
Capital Normal University
**Yingrui Zhang**
Yunnan University of Finance and Economics
**Zihao Zhang**
zihao-zhang@foxmail.com

Beijing Institute of Technology

**Research Article**

**Keywords:**

**Posted Date:** November 12th, 2026

**DOI:** https://doi.org/10.21203/rs.3.rs-11381361/v1

**License:** This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

**Additional Declarations:** No competing interests reported.

# A COMBINATORIAL DECOMPOSITION OF KNAPSACK CONES

GUOCE XIN[1] AND YINGRUI ZHANG[2] AND ZIHAO ZHANG[3,*]

Abstract. In this paper, we focus on knapsack cones, a specific type of simplicial cones that arise naturally in the context of the knapsack problem $x_1a_1+\cdots+x_na_n=a_0$. We present a novel combinatorial decomposition for these cones, named `DecDenu`, which aligns with Barvinok's unimodular cone decomposition within the broader framework of Algebraic Combinatorics. Computer experiments support us to conjecture that our `DecDenu` algorithm is polynomial when the number of variables $n$ is fixed. If true, `DecDenu` will provide the first alternative polynomial algorithm for Barvinok's unimodular cone decomposition, at least for Knapsack cones.

The `CTEuclid` algorithm is designed for MacMahon's partition analysis, and is notable for being the first algorithm to solve the counting problem for Magic squares of order 6. We have enhanced the `CTEuclid` algorithm by incorporating `DecDenu`, resulting in the `LLLCTEuclid` algorithm. This enhanced algorithm makes significant use of LLL's algorithm and stands out as an effective elimination-based approach.

*Mathematic subject classification*: Primary 52C07, Secondary 05–04, 05–08

*Keywords*: Knapsack problem; Denumerant; Lattice points; Barvinok's algorithm; MacMahon's partition analysis;

## 1. Introduction

This paper focus on the knapsack problem $x_1a_1+\cdots+x_na_n=a_0$, where $a_0,a_1,\ldots,a_n$ are positive integers with $\gcd(a_1,\ldots,a_n)=1$. The number of nonnegative integer solutions to this problem, $d(a_0;a_1,\ldots,a_n)$, is known as Sylvester's denumerant. When $n$ is part of the input and the integers are encoded in binary, deciding whether $d(a_0;a_1,\ldots,a_n)>0$ is NP-complete [1]. The complementary decision problem $d(a_0;a_1,\ldots,a_n)=0$ is therefore coNP-complete. Set $d(t;a_1,\ldots,a_n)=0$ for $t<0$. The vanishing problem is related to the Frobenius number $F(a_1,\ldots,a_n)=\max\{t\in\mathbb{Z}:d(t;a_1,\ldots,a_n)=0\}$, with $F=-1$ when some $a_i=1$; see [2] for background.

Our primary objective is to introduce a combinatorial decomposition, akin to Barvinok's unimodular cone decomposition, for a class of simplicial cones referred to as *knapsack cones* or *denumerant cones*. The knapsack (simplicial) cone is defined by

$$\mathcal{D}_\ell(a_1,a_2,\ldots,a_n)=\left\{\sum_{i=1,i\neq\ell}^{n}k_i(-a_ie_\ell+a_\ell e_i):k_i\in\mathbb{R}_{\geq 0}\right\}, \tag{1}$$

where $e_i=(0,\ldots,0,1,0,\ldots,0)^T$ is the standard $i$-th unit vector for all $i$. This cone is generated by $d=n-1$ vectors in $\mathbb{R}^n$ and is therefore not full-dimensional.

When $n$ is fixed, $d(a_0;a_1,\ldots,a_n)$ can be computed using Barvinok's polynomial algorithm[7] from the field of Computational Geometry for the lattice point counting of rational convex polytopes. The C-package `LattE` [11] is a notable implementation of

*Date*: November 9, 2020.

*Corresponding author.

Barvinok's algorithm, often lauded as "state of the art" and setting a high standard. Barvinok's rational function methods and related lattice-point algorithms have led to a range of theoretical developments and computational applications; see, for example, [4, 5, 6, 8, 9, 10, 12, 17].

Our approach falls under the domain of Algebraic Combinatorics and is in line with the `CTEuclid` algorithm developed in [16], which enables us to handle Laurent polynomial numerators uniformly. Andrews, Paule, and Riese [3] developed an earlier reduction algorithm for MacMahon's partition analysis. Xin [16] proposed combining Barvinok's geometric methods with partial fraction methods and developed the Euclid style `CTEuclid` algorithm. The present paper develops this connection for knapsack cones.

We introduce a basic building block that relies on the following established result.

**Proposition 1.** *Suppose $a_1, \ldots, a_n$ are positive integers, the $u_i$ are nonzero and independent of $\lambda$, and the factors $1 - u_i\lambda^{a_i}$ are pairwise coprime over the coefficient field. The partial fraction decomposition of the Elliott rational function $F(\lambda)$ is given by*

$$F(\lambda) = \frac{L(\lambda)}{\prod_{i=1}^{n}(1-u_i\lambda^{a_i})} = P(\lambda) + \frac{p(\lambda)}{\lambda^k} + \sum_{i=1}^{n} \frac{A_i(\lambda)}{1-u_i\lambda^{a_i}}, \tag{2}$$

*where the $u_i$'s are free of $\lambda$, $L(\lambda)$ is a Laurent polynomial, $P(\lambda), p(\lambda)$, and the $A_i(\lambda)$'s are all polynomials, $\deg p(\lambda) < k$, and $\deg A_i(\lambda) < a_i$ for all $i$.*

*Then $A_s(\lambda)$ is uniquely characterized by the following properties: i) $\deg A_s(\lambda) < a_s$; ii) $A_s(\lambda) \equiv F(\lambda)\cdot(1-u_s\lambda^{a_s}) \pmod{\langle 1-u_s\lambda^{a_s}\rangle}$ where $\langle f(\lambda)\rangle$ denotes the ideal generated by the polynomial $f(\lambda)$.*

The fundamental construction we employ is a linear operator $\mathcal{A}_{1-u_i\lambda^{a_i}} : F(\lambda) \mapsto A_i(0)$, where the $\lambda$ will be evident from the context. The `CTEuclid` algorithm utilizes recursion to compute $A_i(0)$.

For simplicity, let us elucidate this concept using the denumerant model. We first express

$$d(a_0; a_1, \ldots, a_n) = \operatorname*{CT}_{\lambda} F(\lambda; \mathbf{y})\big|_{y_i=1} \quad \text{where} \quad F(\lambda; \mathbf{y}) = \frac{\lambda^{-a_0}}{(1-\lambda^{a_1}y_1)\cdots(1-\lambda^{a_n}y_n)}.$$

Proposition 1 then comes into play, yielding

$$\operatorname*{CT}_{\lambda} F(\lambda; \mathbf{y}) = \sum_{i=1}^{n} A_i(0) = \sum_{i=1}^{n} \mathop{\mathcal{A}}_{1-y_i\lambda^{a_i}} F(\lambda; \mathbf{y}).$$

For each $i$, there exists a proper rational function $\bar{F}(\lambda, \mathbf{y})$ that vanishes at $\lambda = 0$ and is of the form

$$\bar{F}(\lambda, \mathbf{y}) = \frac{u_0\lambda^{b_0}}{(1-u_1\lambda^{b_1})\cdots(1-u_n\lambda^{b_n})}, \tag{3}$$

such that i) $\bar{F}(\lambda; \mathbf{y})\cdot(1-y_i\lambda^{a_i}) \equiv F(\lambda; \mathbf{y})\cdot(1-y_i\lambda^{a_i}) \pmod{\langle 1-y_i\lambda^{a_i}\rangle}$; ii) $0 \le b_j \le a_i/2$ for all $j \ne i$ and $b_j \equiv \pm a_j \pmod{a_i}$.

Then we have the recursion

$$\mathop{\mathcal{A}}_{1-y_i\lambda^{a_i}} F(\lambda; \mathbf{y}) = \mathop{\mathcal{A}}_{1-y_i\lambda^{a_i}} \bar{F}(\lambda; \mathbf{y}) = -\sum_{\substack{j\ne i \\ b_j>0}} \mathop{\mathcal{A}}_{1-u_j\lambda^{b_j}} \bar{F}(\lambda; \mathbf{y}). \tag{4}$$

In this way, we can give a decomposition of the form

$$\underset{1-y_s\lambda^{a_s}}{\mathcal{A}} F(\lambda;\mathbf{y}) = \sum_{i=1}^{N_s} \frac{L_i(\mathbf{y})}{(1-\mathbf{y}^{\alpha_{i,1}})\cdots(1-\mathbf{y}^{\alpha_{i,d}})}, \tag{5}$$

where $N_s$ is a positive integer and $d = n-1$ refers to the dimension of the problem. By representing $\mathrm{CT}_\lambda F(\lambda;\mathbf{y})$ as a sum of $N_1 + \cdots + N_n$ simple rational functions, we can efficiently compute its limit at $\mathbf{y} = 1$ [19]. The halving bound for this recursion does not give a polynomial bound in the binary encoding length of $a_s$, even when $n$ is fixed.

Through a meticulous examination of `LattE`, we uncover a crucial connection (see [17]). This connection reveals that:

$$\underset{1-y_s\lambda^{a_s}}{\mathcal{A}} F(\lambda;\mathbf{y}) = \sum_{\alpha\in(\mathcal{D}_s^v(a_1,a_2,\ldots,a_n)\cap\mathbb{Z}^n)} \mathbf{y}^\alpha,$$

where $v = (a_0/a_s)e_s$ and $\mathcal{D}_s^v(a_1,a_2,\ldots,a_n) = v + \mathcal{D}_s(a_1,a_2,\ldots,a_n)$.

Barvinok's algorithm provides a unimodular decomposition of $\mathcal{D}_s(a_1,a_2,\ldots,a_n)$, thereby yielding a similar but "short" formula akin to (5). In this context, "short" refers to the fact that $N_s$ is bounded by a polynomial in $\log a_s$.

Our decomposition adds a multiplier transformation to `CTEuclid`. Write $F(\lambda) = G(\lambda)/(1-u\lambda^a)$, where the denominator of $G$ is coprime to $1-u\lambda^a$. Proposition 3 gives

$$\operatorname*{CT}_\lambda \frac{1}{\underline{1-u\lambda^a}} G(\lambda) = \operatorname*{CT}_\lambda \frac{1}{\underline{1-u^{1/m}\lambda^a}} G(\lambda^m), \qquad \gcd(a,m)=1. \tag{6}$$

The multiplier $m$ determines the signed remainders $b_j = [ma_j]_a$ in the next recursion. For $d$ nonpivot exponents and $a \geq 2$, Proposition 4 gives a multiplier satisfying

$$b_j \leq \left\lfloor \frac{a}{\lfloor (a-1)^{1/d} \rfloor} \right\rfloor = O_d\big(a^{(d-1)/d}\big).$$

The existence proposition does not require the multiplier to be coprime to $a$. Thus this bound cannot in general be applied to the valid multiplier selected by `DecDenu`. Every valid multiplier still gives $b_j \leq \lfloor a/2 \rfloor$ and hence a terminating recursion.

We have developed the `DecDenu` C++ and Maple package to implement the aforementioned ideas. Please contact us or visit [Link](Password: DENU) to obtain the C++ package. The well-established LLL(Lenstra-Lenstra-Lovász) algorithm is employed to identify an effective multiplier, a method akin to that used in `LattE`. The performance of our approach rivals that of `LattE` in the decomposition of knapsack cones, leading us to conjecture that our algorithm is polynomial in the input size. The general polynomial complexity of the LLL-based multiplier policy is not proved here. Theorem 15 proves polynomial complexity for an explicit strategy when there are three pairwise coprime coefficients. Liu and Xin [13] apply the multiplier transformation (6) to obtain a fast constant term algorithm for denumerants with three variables.

The structure of this paper is as follows. In Section 2, we delineate `LattE`'s implementation of Barvinok's unimodular cone decomposition algorithm. This algorithm has set a high standard for the decomposition of knapsack cones. In Section 3, we present the key transformation and elucidate the process for determining a suitable multiplier. Additionally, we provide Algorithm `DecDenu` and compare its performance with `LattE`. Section 4 offers a concise introduction to the domain of iterated Laurent series and details

the algorithm `LLLCTEuclid`, which addresses the core problem of MacMahon's partition analysis. The central issue here is the decomposition of knapsack cones.

## 2. LattE's Implementation of Barvinok's Decomposition

This section provides a concise overview of the fundamental principles underlying Barvinok's unimodular cone decomposition and its implementation in `LattE`, which inspires our development of `DecDenu`. For a detailed treatment, we direct the reader to [11], and for a deeper exploration within the realm of algebraic combinatorics, we refer to [18].

Consider a nonsingular rational matrix $A$ of order $d$. This matrix defines a lattice $L(A) = A\mathbb{Z}^d$ and a simplicial cone $C(A) = A\mathbb{R}^d_{\geq 0}$. The LLL algorithm plays a crucial role in identifying a reduced basis for $L(A)$, which is essential for locating the smallest vector within the lattice under the infinity norm.

A simplicial cone $\mathcal{K} = C(A)$ can be normalized to $C(\bar{A})$, where $\bar{A}$ consists of primitive vectors (i.e., integral vectors with gcd 1). The index $\mathrm{ind}(\mathcal{K})$ is then determined by $|\det(\bar{A})|$, which counts the lattice points within the fundamental parallelepiped $\Pi(\mathcal{K}) = \{k_1\bar{\alpha}_1 + \cdots + k_d\bar{\alpha}_d : 0 \leq k_1, \ldots, k_d < 1\}$. The lattice point generating function $\sigma_{\mathcal{K}}(\mathbf{y}) = \sum_{\alpha\in\mathcal{K}\cap\mathbb{Z}^d} \mathbf{y}^\alpha$ is closely related to that of $\Pi(\mathcal{K})$ through Stanley's formula [14]:

$$\sigma_{\mathcal{K}}(\mathbf{y}) = \frac{\sigma_{\Pi(\mathcal{K})}(\mathbf{y})}{\prod_{k=1}^{d}(1-\mathbf{y}^{\bar{\alpha}_k})}.$$

Notably, $\sigma_{\Pi(\mathcal{K})}(\mathbf{y}) = 1$ if $\mathcal{K}$ is unimodular, i.e., $\mathrm{ind}(\mathcal{K}) = 1$.

Barvinok's theorem provides a polynomial-time algorithm for dissecting a rational cone into unimodular cones, which is instrumental in computationally addressing simplicial cones with large indices.

**Theorem 2** ([7, 11])**.** *Let $d$ be the fixed dimension. There exists a polynomial-time algorithm that, for a given rational simplicial cone $\mathcal{K} = C(A) \subset \mathbb{R}^d$, computes unimodular cones $\mathcal{K}_i = C(A_i)$, $i \in I = \{1, 2, \ldots, N\}$ and signs $\epsilon_i \in \{1, -1\}$ such that*

$$\sigma_{\mathcal{K}}(\mathbf{y}) = \sum_{i\in I} \epsilon_i \sigma_{\mathcal{K}_i}(\mathbf{y}).$$

*The sum is "short" or polynomial in size, implying that the number of summands $|I| = N$ is bounded by a polynomial in* $\log(\mathit{ind}(\mathcal{K}))$.

LattE's implementation of Barvinok's decomposition revolves around finding a lattice point $\gamma$ within a closed parallelepiped defined by $\mathcal{K}$. This is achieved by computing the smallest vector $\beta$ within the lattice $L(\bar{A}^{-1})$ and setting $\gamma = \bar{A}\beta$. If $\gamma$ meets the necessary condition, it is then utilized to decompose $\mathcal{K}$ into at most $d$ cones $\mathcal{K}_i$ with reduced indices that satisfy $\mathrm{ind}(\mathcal{K}_i) \leq (\mathrm{ind}(\mathcal{K}))^{\frac{d-1}{d}}$.

The algorithm proceeds iteratively until all cones are reduced to unimodular cones. Utilizing Brion's polarization trick facilitates this decomposition by disregarding lower-dimensional cones.

## 3. A Combinatorial Decomposition of Knapsack Cones

We first describe how to find a good multiplier, which is defined after the key transformation in Proposition 3. Then we give the pseudo code of the algorithm.

3.1. **The Key Transformation and Good Multipliers.** For the function $F(\lambda)$ as in (2), we also denote by

$$\operatorname*{CT}_{\lambda} \frac{1}{\underline{1-u_s\lambda^{a_s}}} F(\lambda)(1-u_s\lambda^{a_s}) = A_s(0) = \underset{1-u_s\lambda^{a_s}}{\mathcal{A}} F(\lambda).$$

For this notation, one can think that when taking the constant term in $\lambda$, only the single underlined factor of the denominator contributes.

Let $(k,a)=1$, and $\xi = \exp(\frac{2\pi\sqrt{-1}}{a})$ be the $a$-th root of unity. There is a basic fact that the map $x \mapsto x^k$ is an automorphism on the group $\{\xi^j : j = 0,1,\ldots,a-1\}$ of $a$th roots of unity. We establish the following proposition.

**Proposition 3.** *Let $a,k$ be positive integers with $\gcd(k,a)=1$, and let $u \neq 0$. Suppose the denominator of $F(x)$ is coprime to $1-ux^a$. In a characteristic-zero extension containing compatible fractional powers of $u$, we have*

$$\operatorname*{CT}_{x} \frac{1}{\underline{1-ux^a}} F(x) = \operatorname*{CT}_{x} \frac{1}{\underline{1-u^{1/k}x^a}} F(x^k).$$

*Proof.* By utilizing the properties of roots of unity, it is straightforward to derive that

$$\operatorname*{CT}_{x} \frac{1}{\underline{1-ux^a}} F(x) = \frac{1}{a}\sum_{i=0}^{a-1} F(u^{-1/a}\xi^i).$$

Substituting $F(x)$ with $G(x)=F(x^k)$ and $u$ with $\bar{u}=u^{1/k}$ in the above equality yields

$$\operatorname*{CT}_{x} \frac{1}{\underline{1-u^{1/k}x^a}} F(x^k) = \frac{1}{a}\sum_{i=0}^{a-1} G(\bar{u}^{-1/a}\xi^i) = \frac{1}{a}\sum_{i=0}^{a-1} F(u^{-1/a}\xi^{ki}).$$

The proposition thus follows. □

We employ the notation

$$[b]_a := \min(\ \mathrm{rem}(b,a), a - \ \mathrm{rem}(b,a))$$

to denote the absolute value of the signed remainder of $b$ when divided by $a$.

The following result parallels Minkovski's first theorem.

**Proposition 4.** *Suppose $n \geq 1$, $a_1,\ldots,a_n$ are positive integers, and $a \geq 2$ is an integer. Then there exists at least one multiplier $1 \leq k \leq a/2$ such that*

$$[ka_j]_a \leq \left\lfloor \frac{a}{\lfloor (a-1)^{\frac{1}{n}} \rfloor} \right\rfloor \qquad \textit{for all } j.$$

*Proof.* Set $\alpha = \lfloor (a-1)^{1/n} \rfloor$ and $\beta = \lceil a/\alpha \rceil$. The set $S_0 = \{0,1,\ldots,a-1\}$ is partitioned into the possibly empty sets

$$B_i = \{r \in S_0 : (i-1)\beta \leq r < i\beta\}, \qquad 1 \leq i \leq \alpha.$$

For each $k \in S_0$, consider the vector of residues $(ka_1 \bmod a, \ldots, ka_n \bmod a)$. There are $a$ such vectors and at most $\alpha^n \leq a-1$ product boxes $B_{i_1} \times \cdots \times B_{i_n}$. Two vectors, corresponding to $k_1 > k_2$, lie in the same box. Their coordinate differences have absolute value at most $\beta - 1 \leq \lfloor a/\alpha \rfloor$. Set $k = \min(k_1-k_2, a-(k_1-k_2))$. Then $1 \leq k \leq a/2$ and

$$[ka_j]_a = [(k_1-k_2)a_j]_a \leq \beta - 1 \leq \lfloor a/\alpha \rfloor, \qquad 1 \leq j \leq n.$$

□

When applying Proposition 3, $1 \le k \le a-1$ will be referred to as a *multiplier*, and is called *valid* if $(k,a)=1$. Furthermore, $k$ is termed *good* if it fulfills the conditions stipulated in Proposition 4. Note that $k$ and $a-k$ play equivalent roles, so we focus on $k$ with $1 \le k \le \frac{a}{2}$.

*Identifying a nearly good multiplier.* For $a \le 2^n$, the definition of a good multiplier in Proposition 4 must be adapted since $[ka_j]_a \le \frac{a}{2}$ holds for any $k$, and $a^{\frac{n-1}{n}} = \frac{a}{a^{\frac{1}{n}}} \ge \frac{a}{2}$.

In the case of large $a$, locating a good multiplier is challenging. This task can be rephrased as the following problem: Find a $k$ and integers $m_j$, such that $\max_j |ka_j - m_j a|$ is minimized. In the language of lattices, this is akin to seeking the smallest nonzero distance (measured using the infinite norm) between two lattices: one being the $\mathbb{Z}$ span of the single vector $(a_1, a_2, \ldots, a_n)$, and the other being $a\mathbb{Z}^n$.

We can approximate good multipliers using the LLL algorithm. This approach is akin to how `LattE` identifies the shortest vector. We construct the matrix

$$L = \begin{pmatrix} 0.01 & a_1 & \cdots & a_n \\ \mathbf{0} & & aE_n & \end{pmatrix}$$

where $E_n$ is the $n \times n$ identity matrix. The $n+1$ rows of an LLL-reduced basis have the form $(0.01k, ka_1 - m_1 a, \ldots, ka_n - m_n a)$ and provide candidate multipliers $k$. Short reduced rows are useful for obtaining small remainders, but do not guarantee the bound in Proposition 4 after invalid candidates are removed. The weight 0.01 may be replaced by another fixed positive rational number.

*Selection of valid multipliers.* Reduce each candidate modulo $a$, replace it by the smaller of $k$ and $a-k$, and discard $k=0$ and candidates with $\gcd(k,a) > 1$. Always include the fallback multiplier 1. For $a \le 13$, use the optimal choice in Proposition 14. For larger $a$, select a candidate by a fixed heuristic. For example, one may minimize $\max_{1 \le j \le n} [ka_j]_a$, breaking ties by the smaller $k$. The resulting multiplier is valid; its reduction quality remains heuristic.

**Remark 5.** *The number of valid multipliers for a given $a$ is determined by Euler's totient function $\phi(a) = a\prod_p (1-p^{-1})$, where the product ranges over all the prime factors of $a$. For $a > 2$, the restriction $1 \le k \le a/2$ leaves $\phi(a)/2$ representatives. For $a = 2$, the only representative is 1. For $a = 6$, the only representative is also 1.*

3.2. **Algorithm `DecDenu`.** In this section, we consider the function

$$F(\lambda) = \frac{\lambda^{-a_0}}{(1-y_1\lambda^{a_1})\cdots(1-y_n\lambda^{a_n})}.$$

The approach naturally extends when the numerator is replaced by a Laurent polynomial. Replacing each $y_i$ with $u_i$ yields the function presented in Proposition 1.

We first provide the pseudo code and then explain.

**Algorithm 6** (DecDenu)**.**
***Input***: *An integer $a_0 \ge 0$, positive integers $a_1, \ldots, a_n$ with $\gcd(a_1, \ldots, a_n) = 1$ and a number $s \in \{1, 2, \ldots, n\}$.*
***Output***: *A representation of $\mathcal{A}_{1-y_s\lambda^{a_s}} F(\lambda)$ as in (5).*

(1) *If $a_s = 1$ then return $F(\lambda)(1-y_s\lambda)\big|_{\lambda = y_s^{-1}}$. Otherwise, proceed recursively as follows.*

(2) *(new) Select a valid multiplier* $\mathfrak{m}$ *from the normalized LLL candidates and the fallback* 1*, using the selection rule above. Applying the key transformation (Proposition 3) yields*

$$\underset{1-y_s\lambda^{a_s}}{\mathcal{A}} F(\lambda) = \operatorname*{CT}_{\lambda} \frac{1}{\underline{1 - y_s^{\frac{1}{\mathfrak{m}}}\lambda^{a_s}}} \frac{L(\lambda^{\mathfrak{m}})}{\prod_{j=1,j\neq s}^{n}(1 - y_j\lambda^{\mathfrak{m}a_j})}.$$

(3) *Utilizing* (3)*, we can express*

$$\underset{1-y_s\lambda^{a_s}}{\mathcal{A}} F(\lambda) = \underset{1-y_s^{\frac{1}{\mathfrak{m}}}\lambda^{a_s}}{\mathcal{A}} \bar{F}(\lambda) = \operatorname*{CT}_{\lambda} \frac{1}{\underline{1 - y_s^{\frac{1}{\mathfrak{m}}}\lambda^{a_s}}} \frac{\bar{L}(\lambda)}{\prod_{j=1,j\neq s}^{n}(1 - \bar{y}_j\lambda^{b_j})},$$

*where* $b_j = [\mathfrak{m}a_j]_{a_s}$ *for all* $j \neq s$*,* $\bar{F}(\lambda)$ *is proper in* $\lambda$ *with* $\bar{F}(0) = 0$*.*

(4) *The properness of* $\bar{F}$ *and the identity* $\bar{F}(0) = 0$ *give the recurrence*

$$\underset{1-y_s\lambda^{a_s}}{\mathcal{A}} F(\lambda) = -\sum_{\substack{j\neq s \\ b_j>0}} \underset{1-\bar{y}_j\lambda^{b_j}}{\mathcal{A}} \bar{F}(\lambda). \tag{7}$$

Factors with $b_j = 0$ are independent of $\lambda$ and are retained in the coefficient field. The sign in (7) follows directly from partial fractions: since $\bar{F}$ is proper, regular at 0, and satisfies $\bar{F}(0) = 0$, the sum of the constant coefficients of its partial fraction numerators is zero. Each positive child index is at most $\lfloor a_s/2 \rfloor$. The recursion therefore terminates for every valid multiplier policy.

The aforementioned algorithm, except Step 2, appears as the core steps of the `CTEuclid` algorithm. However, it is Step 2 that enables the possibility of a polynomial-time algorithm.

**Example 7.** *Compute the following constant term:*

$$\operatorname*{CT}_{\lambda} \frac{\lambda^{-15}}{\underline{(1 - \lambda^7 y_1)}(1 - \lambda^2 y_2)(1 - \lambda^3 y_3)}.$$

*Solution.* Applying Proposition 3 with respect to $k = 2$, the constant term becomes

$$\begin{aligned}
\operatorname*{CT}_{\lambda} \frac{\lambda^{-30}}{\underline{(1 - \lambda^7 y_1^{1/2})}(1 - \lambda^4 y_2)(1 - \lambda^6 y_3)} &= \operatorname*{CT}_{\lambda} \frac{\lambda^2 y_1^3 y_2^{-1} y_3^{-1}}{\underline{(1 - \lambda^7 y_1^{1/2})}(1 - \lambda^3 y_1^{1/2} y_2^{-1})(1 - \lambda y_1^{1/2} y_3^{-1})} \\
&= \operatorname*{CT}_{\lambda} \frac{-\lambda^2 y_1^3 y_2^{-1} y_3^{-1}}{(1 - \lambda^7 y_1^{1/2})\underline{(1 - \lambda^3 y_1^{1/2} y_2^{-1})}(1 - \lambda y_1^{1/2} y_3^{-1})} \\
&\quad + \operatorname*{CT}_{\lambda} \frac{-\lambda^2 y_1^3 y_2^{-1} y_3^{-1}}{(1 - \lambda^7 y_1^{1/2})(1 - \lambda^3 y_1^{1/2} y_2^{-1})\underline{(1 - \lambda y_1^{1/2} y_3^{-1})}},
\end{aligned}$$

For the first term, we have

$$\begin{aligned}
&\operatorname*{CT}_{\lambda}\frac{-\lambda^2 y_1^3 y_2^{-1} y_3^{-1}}{(1-\lambda^7 y_1^{1/2})\underline{(1-\lambda^3 y_1^{1/2} y_2^{-1})}(1-\lambda y_1^{1/2} y_3^{-1})}\\
&\quad=\operatorname*{CT}_{\lambda}\frac{-\lambda^2 y_1^3 y_2^{-1} y_3^{-1}}{(1-\lambda y_2^2 y_1^{-1/2})\underline{(1-\lambda^3 y_1^{1/2} y_2^{-1})}(1-\lambda y_1^{1/2} y_3^{-1})}\\
&\quad=\operatorname*{CT}_{\lambda}\frac{\lambda^2 y_1^3 y_2^{-1} y_3^{-1}}{\underline{(1-\lambda y_2^2 y_1^{-1/2})}(1-\lambda^3 y_1^{1/2} y_2^{-1})\underline{(1-\lambda y_1^{1/2} y_3^{-1})}}\\
&\quad=\frac{y_1^4 y_2^{-5} y_3^{-1}}{(1-y_1^2 y_2^{-7})(1-y_1 y_2^{-2} y_3^{-1})}+\frac{y_1^2 y_2^{-1} y_3}{(1-y_3 y_2^2 y_1^{-1})(1-y_3^3 y_1^{-1} y_2^{-1})};
\end{aligned}$$

For the second term, we have

$$\operatorname*{CT}_{\lambda}\frac{-\lambda^2 y_1^3 y_2^{-1} y_3^{-1}}{(1-\lambda^7 y_1^{1/2})(1-\lambda^3 y_1^{1/2} y_2^{-1})\underline{(1-\lambda y_1^{1/2} y_3^{-1})}}=\frac{-y_3 y_1^2 y_2^{-1}}{(1-y_3^7 y_1^{-3})(1-y_3^3 y_1^{-1} y_2^{-1})}.$$

In total, we obtain three output terms. It is worth noting that we will obtain four terms if we use the `CTEuclid` algorithm (with the default value $k=1$). □

Notably, the algorithm incorporates fractional powers of the $y$ variables when Step 2 is employed, as displayed in the middle steps of Example 7. On the other hand, the three output terms contain only integer powers of the $y$ variables. This is not a coincidence. Indeed, we have the following result, which is pivotal when considering constant terms involving multiple variables:

**Proposition 8.** *The output of* `DecDenu` *only contains integer powers of the $y$ variables.*

*Proof.* We encode an Elliott rational function by a matrix by the following correspondence:

$$\frac{y^{\alpha_0}\lambda^{a_0}}{\prod_{i=1}^{n}(1-y^{\alpha_i}\lambda^{a_i})}\mapsto\begin{pmatrix}a_1 & a_2 & \cdots & a_n & a_0\\ \alpha_1 & \alpha_2 & \cdots & \alpha_n & \alpha_0\end{pmatrix}.$$

Subsequently, `DecDenu` can be applied to these matrices.

Step 2 corresponds to the transformation $\gamma_{s,\mathfrak{m}}$ as defined in Definition 9 below, and Step 3 corresponds to a sequence of operations $\epsilon_{s,l,h}$ for appropriate choices of $l$ and $h$ as per the same definition.

Column sign changes used in signed remainder reduction also preserve the integrality of these minors. Each term in the output corresponds to a matrix whose first row consists entirely of zeros, except for a single 1 in the first $n$ columns. By permuting the columns, we can assume that the sole 1 is located in the first column without loss of generality. The proposition then follows from Corollary 11 below. □

**Definition 9.** *For an integer $m\neq 0$, define $\gamma_{s,m}$ as the transformation that modifies matrix $M$ by scaling its first row by a factor of $m$, and scaling its $s$-th column by a factor of $\frac{1}{m}$. Additionally, for distinct columns $s,l$, let $\epsilon_{s,l,h}$ denote the column operation that adds $h$ times the $s$-th column to the $l$-th column of $M$.*

**Theorem 10.** *Suppose $M$ is a matrix, and $\rho$ is either $\gamma_{s,m}$ or $\epsilon_{s,l,h}$, where $m\neq 0$ and $h$ are integers and $s\neq l$ in $\epsilon_{s,l,h}$. If all $k$-th order minors of $M$ that include the first row are integers, then the corresponding $k$-th order minors in $\rho(M)$ are also integers.*

*Proof.* Consider matrix $M = (a_{ij})$, and let $M[I, J]$ be the submatrix of $M$ with row set $I = \{1, i_2, \ldots, i_k\}$ and column set $J = \{j_1, j_2, \ldots, j_k\}$.

Case $s \in J$: it is evident that for both $\gamma_{s,m}$ and $\epsilon_{s,l,h}$, the determinant $\det(\rho(M)[I, J])$ remains unchanged and equals $\det(M[I, J])$.

Case $s \notin J$: for $\rho = \gamma_{s,m}$, $\rho(M)[I, J]$ is obtained from $M[I, J]$ by scaling the first row by $m$, resulting in $\det(\rho(M)[I, J]) = m \det(M[I, J])$. For $\rho = \epsilon_{s,l,h}$, if $l \notin J$, then $\rho(M)[I, J]$ is identical to $M[I, J]$; if $l \in J$, then we have

$$\rho(M)[I, J] = \begin{pmatrix} a_{1,j_1} & \cdots & ha_{1,s} + a_{1,l} & \cdots & a_{1,j_k} \\ a_{i_2,j_1} & \cdots & ha_{i_2,s} + a_{i_2,l} & \cdots & a_{i_2,j_k} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i_k,j_1} & \cdots & ha_{i_k,s} + a_{i_k,l} & \cdots & a_{i_k,j_k} \end{pmatrix}.$$

By splitting the column $l$ in $\rho(M)[I, J]$, we obtain $\det(\rho(M)[I, J]) = h \det(M[I, J']) + \det(M[I, J])$, where $J'$ is the ordered column list obtained by replacing $l$ with $s$ in its original position in $J$. Since $\det(M[I, J'])$ and $\det(M[I, J])$ are both integers by assumption, the theorem is established. □

**Corollary 11.** *Let $M$ be an integer matrix with the gcd of its first row being 1. If $M$ undergoes a sequence of transformations $\rho_i$, each being $\gamma_{s,m}$ or $\epsilon_{s,l,h}$, to yield the matrix*

$$\begin{pmatrix} 1 & 0 \\ \alpha & A \end{pmatrix}, \tag{8}$$

*then $A$ is also an integer matrix.*

*Proof.* Given that $M$ is integral, the 2-order minors involving the first row in $M$ are integers. This property is preserved through any transformation $\rho_i$ by Theorem 10. Therefore, the 2-order minors involving the first row in the transformed matrix (8) are also integers, thus confirming the corollary. □

3.3. **Complexity Analysis.** Fix a deterministic multiplier selection policy on normalized ordered labels for `DecDenu`, and assume $d = n - 1 \geq 1$. The decomposition defines a rooted tree $T_{\mathtt{D}}$ whose root has label $(\textcircled{s}; a'_1, \ldots, a'_n)$, where $a'_s = a_s$ and $a'_i = [a_i]_{a_s}$ for $i \neq s$. A normalized node has label $(\textcircled{s}; v_1, \ldots, v_n)$ with $0 \leq v_i \leq \lfloor v_s/2 \rfloor$ for $i \neq s$ and $\gcd(v_1, \ldots, v_n) = 1$. Its index is $\mathrm{ind}(v) = v_s$, and its dimension is the number of nonzero entries minus one. For a valid multiplier $\mathfrak{m}$, set $b_s = v_s$ and $b_i = [\mathfrak{m} v_i]_{v_s}$ for $i \neq s$. The gcd condition is preserved because $\gcd(\mathfrak{m}, v_s) = 1$. Each $j \neq s$ with $b_j > 0$ gives a child of index $b_j$, which is normalized again modulo $b_j$. A node of index 1 is a leaf. The number of output terms before cancellation is $nl(T_{\mathtt{D}})$, the number of leaves.

Set $A = a_s$. Every positive child index is at most half of its parent's index, and each node has at most $d$ children. Hence

$$\mathrm{dep}(T_{\mathtt{D}}) \leq \lfloor \log_2 A \rfloor, \qquad nl(T_{\mathtt{D}}) \leq d^{\lfloor \log_2 A \rfloor} \leq A^{\log_2 d}.$$

For $d > 1$, this bound is polynomial in the numerical value of $A$, rather than in its binary encoding length. For $d = 1$, there is only one child at each internal node and the output has one term. A polynomial output bound in $\log A$ for the general LLL-based policy remains conjectural. Even such an output bound would have to be combined with

bounds on the bit lengths of intermediate exponent vectors and the cost of multiplier selection to prove polynomial running time.

For comparison, suppose a Barvinok decomposition tree $T_{\mathtt{B}}$ has at most $d$ children at each node and satisfies

$$\Delta_{j+1} \le \Delta_j^{(d-1)/d}, \qquad \Delta_j = \mathrm{ind}(\mathcal{K}_j),$$

along every path. For $d \ge 2$ and $\Delta_0 \ge 2$, this gives

$$\mathrm{dep}(T_{\mathtt{B}}) \le 1 + \left\lfloor \frac{\log(\log \Delta_0 / \log 2)}{\log(d/(d-1))} \right\rfloor.$$

Indeed, after the displayed number of steps the upper bound $\Delta_0^{((d-1)/d)^j}$ is less than 2, so the integral index must be 1. If $\Delta_0 = 1$, the tree has depth zero. Here and below, log denotes the natural logarithm and $\log_2$ denotes the logarithm to base 2.

To isolate the effect of the coprimality requirement, define a comparison tree $\overline{T_{\mathtt{D}}}$ of positive integer indices. Its root has index $A$, a node has at most $d$ children, and every child of a node of index $a \ge 2$ has index at most

$$\min\left(\left\lfloor \frac{a}{2} \right\rfloor, q_d(a)\right), \qquad q_d(a) = \left\lfloor \frac{a}{\lfloor (a-1)^{1/d} \rfloor} \right\rfloor.$$

Nodes of index 1 are leaves. This is an index comparison, not a valid cone decomposition: the multiplier supplied by Proposition 4 need not be coprime to the current index. For $d \ge 2$, set $r = (d-1)/d$ and $C_d = 2^{1+1/d}$. Since $\lfloor x \rfloor \ge x/2$ for $x \ge 1$ and $a - 1 \ge a/2$, we have

$$q_d(a) \le C_d a^r.$$

Along a path with indices $a_0 = A, a_1, \ldots,$ it follows that

$$\log a_j \le r^j \log A + \log C_d \sum_{i=0}^{j-1} r^i \le r^j \log A + d \log C_d.$$

For $A \ge 2$, choose

$$t = \left\lceil \frac{\log(\log A / \log 2)}{\log(d/(d-1))} \right\rceil.$$

If the path has not already terminated, then $a_t \le 2^{d+2}$. At most $d+2$ further halving steps are needed, so

$$\mathrm{dep}(\overline{T_{\mathtt{D}}}) \le d + 2 + \left\lceil \frac{\log(\log A / \log 2)}{\log(d/(d-1))} \right\rceil.$$

Consequently $nl(\overline{T_{\mathtt{D}}})$ is polynomial in $\log A$ for fixed $d$. This estimate does not establish the same bound for $T_{\mathtt{D}}$. Polarization can also change the starting index in a geometric decomposition; the comparison of these depth bounds alone does not establish a running-time advantage.

Bad reductions can occur for valid multipliers. For example, let the current pivot index be $2a$ and let another entry be $a$. Every multiplier coprime to $2a$ is odd, so its signed remainder at that entry is still $a$. The resulting child has index $a$, but the old pivot becomes zero modulo $a$, and the dimension decreases. More generally, dimensions in $T_{\mathtt{D}}$ never increase. If a normalized node has index $a$ and current dimension $m$ with $a < 2m$, then its $m$ positive nonpivot entries lie in the set $\{1, \ldots, \lfloor a/2 \rfloor\}$. Two entries coincide. A valid multiplier preserves this coincidence, and selecting one of them as a

child pivot makes the other zero after normalization. Thus at least one child has smaller dimension; this does not require every child to do so.

An internal node is called an LLL-node when the implementation invokes LLL to select its multiplier. The cost of these calls depends on both the lattice dimension and the bit lengths of the entries. The number of LLL-nodes is therefore an implementation statistic rather than a complete bit-complexity estimate. The remaining exact arithmetic also depends on operand bit lengths. When the implementation uses Proposition 14 for indices at most 13, LLL is needed only at larger indices. This cutoff does not imply a uniform bound on the fraction of LLL-nodes: a large-index node can have only leaf children. For the random examples in Table 1, the implementation reports that about 4% of the internal nodes are LLL-nodes. The running times in Table 3 measure the resulting computational performance.

3.4. **The optimal valid multiplier when $a_s < 14$.** In order to have a better understanding of $nl(T_{\mathtt{D}})$, we introduce the rooted tree $T^1_{\mathtt{D}}$, which is similar to $T_{\mathtt{D}}$ except that we choose $\mathfrak{m} = 1$ at each internal node. This corresponds to the `CTEuclid` algorithm [16]. Suppose the roots of $T^1_{\mathtt{D}}$ and $T_{\mathtt{D}}$ both have label $(\textcircled{s}; a_1, \ldots, a_n)$.

Denote by $f^1(\textcircled{s}; a_1, \ldots, a_n)$ the number of leaves when every multiplier is 1, and by $f^m(\textcircled{s}; a_1, \ldots, a_n)$ the number of leaves for the fixed multiplier policy used to construct $T_{\mathtt{D}}$. Define $f^{\mathrm{opt}}(\textcircled{s}; a_1, \ldots, a_n)$ to be the minimum number of leaves over all valid multiplier policies. Then

$$f^{\mathrm{opt}}(\textcircled{s}; a_1, \ldots, a_n) \le \min\big(f^1(\textcircled{s}; a_1, \ldots, a_n), f^m(\textcircled{s}; a_1, \ldots, a_n)\big).$$

For $f^1$ and $f^{\mathrm{opt}}$, permuting entries together with the pivot index and omitting zero entries do not change the value. The fixed-policy function $f^m$ is evaluated on the normalized ordered labels used by the implementation. The initial value at a normalized pivot of index 1 is 1. For $h = 1$ or $h = m$, let $\mathfrak{m}$ be the multiplier prescribed by that policy and set $b_s = a_s$, $b_i = [\mathfrak{m} a_i]_{a_s}$ for $i \neq s$. Then

$$f^h(\textcircled{s}; a_1, \ldots, a_n) = \sum_{\substack{j \neq s \\ b_j > 0}} f^h(\textcircled{j}; b_1, \ldots, b_n), \qquad a_s > 1.$$

For the optimal value, the recursion is instead

$$f^{\mathrm{opt}}(\textcircled{s}; a_1, \ldots, a_n) = \min_{\substack{1 \le \mathfrak{m} \le \lfloor a_s/2 \rfloor \\ \gcd(\mathfrak{m}, a_s) = 1}} \sum_{\substack{j \neq s \\ b_j > 0}} f^{\mathrm{opt}}(\textcircled{j}; b_1, \ldots, b_n), \qquad a_s > 1,$$

where the $b_i$ in each summand depend on $\mathfrak{m}$ as above. The minimum is well-defined because every positive child index is smaller than $a_s$.

These recursions turn a term of index $a_s$ into a sum of at most $d = n - 1$ terms of smaller indices.

For either fixed multiplier policy, the halving property gives

$$f^h(\textcircled{s}; a_1, \ldots, a_n) \le (n-1)^{\lfloor \log_2 a_s \rfloor}, \qquad h \in \{1, m\}, \quad n \ge 2.$$

For $n > 2$, this estimate does not give a polynomial bound in $\log a_s$.

**Example 12.** *Let us revisit Example 7 but focus on the number of the output terms. This will illustrate the distinction between $f^{\mathrm{opt}}$ and $f^1$. By using $\mathfrak{m}=2$, we obtain*

$$\begin{aligned} f^{\mathrm{opt}}(①;7,2,3) &\overset{\mathfrak{m}=2}{=} f^{\mathrm{opt}}(①;7,3,1)\\ &= f^{\mathrm{opt}}(②;7,3,1)+f^{\mathrm{opt}}(③;7,3,1)\\ &= f^{\mathrm{opt}}(②;1,3,1)+1=2+1=3, \end{aligned}$$

*which agrees with the solution for Example 7. While using $\mathfrak{m}=1$ gives*

$$\begin{aligned} f^1(①;7,2,3) &= f^1(②;7,2,3)+f^1(③;7,2,3)\\ &= f^1(②;1,2,1)+f^1(③;1,1,3)\\ &= 2+2=4. \end{aligned}$$

Now we give explicit formulas of $f^1(①;a_1,\dots,a_n)$ for $a_1\le 13$, and use them to find the optimal valid multiplier $\mathfrak{m}$ for $f^{\mathrm{opt}}(①;a_1,\dots,a_n)$ when $a_1\le 13$. When $a_1$ is fixed, we encode $(a_2,\dots,a_n)$ as $(1^{r_1},\dots,k^{r_k})$, where $k=\lfloor a_1/2\rfloor$, meaning that there are $r_1$ 1's, $r_2$ 2's, and so on. The recursion can be rewritten as

$$f^1(①;a_1,1^{r_1},\dots,k^{r_k})=\sum_{i=1}^{k} r_i f^1(①;i,a_1,1^{r_1},\dots,(i-1)^{r_{i-1}},(i+1)^{r_{i+1}},\dots,k^{r_k}).$$

The following formulas for $a_1<14$ are easy to compute.

$$f^1(①;2,1^{r_1})=f^1(①;3,1^{r_1})=r_1,$$

$$f^1(①;4,1^{r_1},2^{r_2})=r_1r_2+r_1,$$

$$f^1(①;5,1^{r_1},2^{r_2})=r_1r_2+r_1+r_2,$$

$$f^1(①;6,1^{r_1},2^{r_2},3^{r_3})=r_1(r_2+r_3+1)+2r_2r_3,$$

$$f^1(①;7,1^{r_1},2^{r_2},3^{r_3})=r_1(r_2+r_3+1)+(2r_3+1)r_2+r_3,$$

$$f^1(①;8,1^{r_1},2^{r_2},3^{r_3},4^{r_4})=(r_4+1)(r_2+1)(r_1+r_3)+(r_4+r_2+r_1)r_3,$$

$$f^1(①;9,1^{r_1},2^{r_2},3^{r_3},4^{r_4})=(r_1+1+r_3)(r_2r_4+r_2+r_4)+r_1+(r_4+r_2+r_1)r_3,$$

$$\begin{aligned} f^1(①;10,1^{r_1},2^{r_2},\dots,5^{r_5}) &= r_1((r_2+2)r_4+(r_2+r_3+1)(r_5+1))+3r_4(r_3+r_5)\\ &\quad+(r_4+2)(r_5+r_2)r_3+2r_5(r_4+1)r_2+r_3, \end{aligned}$$

$$\begin{aligned} f^1(①;11,1^{r_1},2^{r_2},\dots,5^{r_5}) &= r_1+(r_4r_2+r_2+r_4)(r_1+1+r_3+r_5)\\ &\quad+r_3(r_1+r_2+1+r_4+r_5)+r_5((r_1+r_4+2)(r_2+r_3+1)-1),\\ f^1(①;12,1^{r_1},2^{r_2},\dots,6^{r_6}) &= ((r_6+1)(r_2+r_3+1)+(r_2+2r_6+1)r_4)r_1\\ &\quad+((r_4+2r_6+2)r_2+r_4(3r_6+2))r_3\\ &\quad+((r_2+r_3+2)r_1+(r_4+2r_6+2)(r_2+r_3)+r_4r_2+(2r_6+3)r_4+3r_6+1)r_5,\\ f^1(①;13,1^{r_1},2^{r_2},\dots,6^{r_6}) &= r_1+r_2(r_1+1+r_3+r_5)+r_3(r_1+r_2+1+r_4+r_5)\\ &\quad+r_4(r_1+1+r_3+r_5+(r_2+r_6)(r_1+1+r_3+r_5))\\ &\quad+r_5(r_1+r_4+r_6+(r_2+r_3+1)(r_1+r_4+r_6+1))\\ &\quad+r_6(r_1+r_5+1+(r_2+r_4)(r_1+1+r_3+r_5)+r_3(r_1+r_2+1+r_4+r_5)). \end{aligned}$$

**Lemma 13.** *If $a_1\le 6$, then we have $f^{\mathrm{opt}}(①;a_1,1^{r_1},\dots,k^{r_k})=f^1(①;a_1,1^{r_1},\dots,k^{r_k})$.*

*Proof.* When $a_1 = 1$, both values are 1 by the initial condition. When $2 \le a_1 \le 4$ or $a_1 = 6$, the only valid choice with $1 \le m \le \lfloor a_1/2 \rfloor$ is 1, so $f^{\mathrm{opt}}$ equals $f^1$.

When $a_1 = 5$, the valid multipliers are 1 and 2. Utilizing 1 yields $f^1(\textcircled{1}; 5, 1^{r_1}, 2^{r_2})$, while employing 2 results in $f^1(\textcircled{1}; 5, 2^{r_1}, 1^{r_2}) = f^1(\textcircled{1}; 5, 1^{r_1}, 2^{r_2})$ according to the explicit formula. □

**Proposition 14.** *If $a_1 \le 13$, then the function $f^{\mathrm{opt}}(\textcircled{1}; a_1, 1^{r_1}, \ldots, k^{r_k})$ can be computed efficiently, with the optimal choice of $\mathfrak{m}$ determined through a few comparisons.*

*Proof.* The case $a_1 = 1$ is the initial condition. If $2 \le a_1 \le 13$, every positive child index is at most 6, regardless of the valid multiplier. By Lemma 13, we only need to make the initial choice of $\mathfrak{m}$, and there are at most 6 possible choices for $\mathfrak{m}$. Let $g_{\mathfrak{m}}$ denote the value obtained by employing the multiplier $\mathfrak{m}$. We shall select the optimal multiplier $\mathfrak{m}_0$ to be $\mathfrak{m}_0 = i$ if $g_i$ is the smallest among all valid multipliers $\mathfrak{m}$.

For some specific values of $a_1$, we can derive simple conditions. We illustrate the case $a_1 = 7$ in detail, as the other cases follow a similar pattern. In this case, the optimal multiplier $\mathfrak{m}_0$ must be chosen from the set $\{1, 2, 3\}$. We have:

$$\begin{aligned} g_1 &= f^1(\textcircled{1}; 7, 1^{r_1}, 2^{r_2}, 3^{r_3}), \\ g_2 &= f^1(\textcircled{1}; 7, 2^{r_1}, 4^{r_2}, 6^{r_3}) = f^1(\textcircled{1}; 7, 1^{r_3}, 2^{r_1}, 3^{r_2}), \\ g_3 &= f^1(\textcircled{1}; 7, 3^{r_1}, 6^{r_2}, 9^{r_3}) = f^1(\textcircled{1}; 7, 1^{r_2}, 2^{r_3}, 3^{r_1}). \end{aligned}$$

Thus, by the explicit formula of $f^1(\textcircled{1}; 7, 1^{r_1}, 2^{r_2}, 3^{r_3})$, we obtain:

$$\begin{aligned} g_1 - g_2 &= f^1(\textcircled{1}; 7, 1^{r_1}, 2^{r_2}, 3^{r_3}) - f^1(\textcircled{1}; 7, 1^{r_3}, 2^{r_1}, 3^{r_2}) = -(r_1 - r_3)r_2, \\ g_1 - g_3 &= f^1(\textcircled{1}; 7, 1^{r_1}, 2^{r_2}, 3^{r_3}) - f^1(\textcircled{1}; 7, 1^{r_2}, 2^{r_3}, 3^{r_1}) = -(r_1 - r_2)r_3. \end{aligned}$$

If $r_1 \ge r_2$ and $r_1 \ge r_3$, both differences are nonpositive, so 1 is an optimal multiplier. The conditions for $\mathfrak{m}_0$ being 2 or 3 can be similarly derived. In summary, assuming $r_i = \max(r_1, r_2, r_3)$, then if $i = 1$ then $\mathfrak{m}_0 = 1$; if $i = 2$ then $\mathfrak{m}_0 = 3$; if $i = 3$ then $\mathfrak{m}_0 = 2$.

For $a_1 = 8$, the optimal $\mathfrak{m}_0$ must be chosen from $\{1, 3\}$. We shall choose $\mathfrak{m}_0 = 3$ only when $r_1 < r_3$.

For $a_1 = 9$, the optimal $\mathfrak{m}_0$ must be chosen from $\{1, 2, 4\}$. Assume $r_i = \max(r_1, r_2, r_4)$. Then if $i = 1$, then $\mathfrak{m}_0 = 1$; if $i = 2$ then $\mathfrak{m}_0 = 4$; if $i = 4$ then $\mathfrak{m}_0 = 2$.

For $a_1 = 10$, the optimal $\mathfrak{m}_0$ must be chosen from $\{1, 3\}$. We shall choose $\mathfrak{m}_0 = 3$ only when $(r_1 + r_2 - r_3 - r_4)r_5 + 2r_1r_2 - 2r_3r_4 < 0$.

For $a_1 = 12$, the optimal $\mathfrak{m}_0$ must be chosen from $\{1, 5\}$. We shall choose $\mathfrak{m}_0 = 5$ only when $r_1 < r_5$.

For $a_1 = 11$ or 13, we cannot derive simple conclusions as in the other cases. □

Liu and Xin [13] give an algorithm for $d(a_0; a_1, a_2, a_3)$ with $O(\log a_2)$ ring operations when $a_0 \in \mathbb{Z}_{\ge 0}$, $0 < a_1 < a_2 < a_3$, and $\gcd(a_1, a_2, a_3) = 1$. The following strategy uses their modular inverse transformation and halving recursion to give an explicit `DecDenu` decomposition bound for pairwise coprime coefficients.

**Theorem 15.** *Suppose $a_1, a_2, a_3$ are pairwise coprime positive integers and $a_0 \in \mathbb{Z}_{\ge 0}$. Set*

$$F(\lambda; \mathbf{y}) = \frac{\lambda^{-a_0}}{(1 - y_1\lambda^{a_1})(1 - y_2\lambda^{a_2})(1 - y_3\lambda^{a_3})}.$$

*For each $i \in \{1,2,3\}$, an explicit valid multiplier strategy decomposes $\mathcal{A}_{1-y_i\lambda^{a_i}} F$ into at most $1 + \lfloor \log_2 a_i \rfloor$ simple rational functions. Consequently,*

$$f^{\mathrm{opt}}(①; a_1, a_2, a_3) \le 1 + \lfloor \log_2 a_i \rfloor.$$

*When $a_i = 1$, the decomposition has one term. With this strategy, $d(a_0; a_1, a_2, a_3)$ can be computed in time polynomial in the binary input length.*

*Proof.* It suffices to consider $i = 1$. If $a_1 = 1$, the first step of `DecDenu` gives one term. For this theorem, replace the LLL-based selection rule in Step 2 by the following explicit policy. Suppose $a = a_1 \ge 2$. Compute $u \in \{1, \dots, a-1\}$ with $ua_3 \equiv 1 \pmod a$, and set $m = \min(u, a-u)$. Then $\gcd(m, a) = 1$ and $[ma_3]_a = 1$. After the multiplier transformation and signed remainder reduction, the root has label

$$(①; a, b, 1), \qquad b = [ma_2]_a.$$

Since $\gcd(a, a_2) = 1$, we have $\gcd(a, b) = 1$ and $1 \le b \le \lfloor a/2 \rfloor$. At every subsequent node, choose multiplier 1.

A normalized node $(①; q, p, 1)$ with $q > 1$, $\gcd(q, p) = 1$, and $1 \le p \le q/2$ has a leaf child from the last entry 1 and another child with pivot $p$. If $p = 1$, the second child is also a leaf. If $p > 1$, its normalized label, after permuting entries, is

$$(①; p, [q]_p, 1), \qquad \gcd(p, [q]_p) = 1, \quad 1 \le [q]_p \le \lfloor p/2 \rfloor.$$

The pivot indices therefore follow the least absolute remainder Euclidean chain

$$q_0 = a, \qquad q_1 = b, \qquad q_{j+1} = [q_{j-1}]_{q_j} \quad (q_j > 1).$$

Let $\ell$ be the first index with $q_\ell = 1$. Consecutive indices remain coprime, so a remainder cannot be zero before this terminal index. For $0 \le j < \ell$, $q_{j+1} \le q_j/2$. Thus

$$1 = q_\ell \le a/2^\ell, \qquad \ell \le \lfloor \log_2 a \rfloor.$$

Each internal node adds one leaf, and the final child contributes one more. The constructed tree has exactly $\ell + 1$ leaves, proving the asserted bound and the bound for $f^{\mathrm{opt}}$.

To check binary complexity, set $B = \max(2, a_0, a_1, a_2, a_3)$. The extended Euclidean algorithm computes $u$ in polynomial time. The root multiplier is at most $B$, and all subsequent multipliers are 1. There are $O(\log B)$ nodes and a constant number of exponent-vector operations at each node. Monomial powers are stored by their exponent vectors. The first transformation makes the $\lambda$-exponents at most $B^2$ in absolute value. Remainder reductions use integer quotients of magnitude $O(B^2)$. Fractional exponents introduced at the root have a common denominator dividing $m$; subsequent nodes introduce no further multiplier denominators. After $O(\log B)$ operations, the exponent vectors have bit length $O((\log B)^2)$. Thus the decomposition is constructed in polynomial time, and its final exponents are integral by the preceding integrality result.

The sum of the three decompositions has $N = O(\log B)$ terms of the form

$$\frac{\varepsilon \mathbf{y}^\beta}{(1 - \mathbf{y}^\alpha)(1 - \mathbf{y}^\gamma)}, \qquad \varepsilon \in \{1, -1\},$$

where the denominator exponent vectors are nonzero. Choose $w = (1, T, T^2)$ with $w \cdot \alpha \ne 0$ for every denominator exponent vector. Each vector excludes at most two integer values of $T$, so some $T \in \{1, \dots, 4N+1\}$ works. It can be found in polynomial time. Substitute

$y_j = \exp(w_j t)$ for $j = 1, 2, 3$. For an individual term, set $h = w \cdot \beta$, $p = w \cdot \alpha$, and $q = w \cdot \gamma$. Its coefficient of $t^0$ is

$$\varepsilon \left( \frac{h^2 - h(p+q)}{2pq} + \frac{1}{4} + \frac{p}{12q} + \frac{q}{12p} \right).$$

These rational numbers have polynomial bit length and can be summed in polynomial time. The complete sum is the generating polynomial

$$\sum_{\substack{x_1,x_2,x_3 \in \mathbb{Z}_{\geq 0} \\ a_1x_1+a_2x_2+a_3x_3=a_0}} y_1^{x_1} y_2^{x_2} y_3^{x_3}.$$

It is regular at $\mathbf{y} = 1$, so the sum of the displayed constant coefficients is $d(a_0; a_1, a_2, a_3)$. □

3.5. **Computer Experiments.** This section presents the outcomes of our computational experiments aimed at decomposing knapsack cones. We utilized two C++ packages: `LattE 1.7.5` and our own `DecDenu`. The results are multivariate rational generating functions corresponding to unimodular cones. We report the computation time and the number of terms $nl(T_{\mathtt{D}})$ and $nl(T_{\mathtt{B}})$ in the output.

In Table 1, we list two random knapsack cones for each $9 \leq n \leq 15$. Notably, there is only one instance (when $n = 10$ and boldfaced) for which $nl(T_{\mathtt{D}}) > nl(T_{\mathtt{B}})$.

| $n$ | Knapsack cone $\mathcal{D}_1(a_1, \ldots a_n)$ | | | | | | | | | | | | | | | $nl(T_{\mathtt{D}})$ | $nl(T_B)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | **1285** | 2549 | 2209 | 2402 | 2018 | 2789 | 1181 | 2369 | 121 | | | | | | | 10403 | 11341 |
| 9 | **1565** | 2594 | 2882 | 2988 | 2876 | 544 | 1621 | 740 | 2372 | | | | | | | 11015 | 12808 |
| 10 | **422** | 1980 | 2478 | 1360 | 2179 | 1992 | 2857 | 1326 | 78 | 2421 | | | | | | **7955** | 7701 |
| 10 | **2937** | 600 | 2895 | 538 | 584 | 2175 | 1636 | 2942 | 1905 | 509 | | | | | | 42673 | 50123 |
| 11 | **681** | 640 | 1082 | 2115 | 2937 | 965 | 2690 | 1572 | 701 | 596 | 224 | | | | | 21838 | 25557 |
| 11 | **1576** | 2362 | 226 | 2059 | 2078 | 2694 | 1824 | 1320 | 1908 | 2968 | 1547 | | | | | 29358 | 40456 |
| 12 | **1439** | 799 | 2358 | 241 | 743 | 2370 | 2188 | 1713 | 1114 | 783 | 922 | 1124 | | | | 77245 | 89577 |
| 12 | **2934** | 1928 | 2894 | 1687 | 2542 | 2633 | 662 | 2545 | 1184 | 1250 | 2357 | 1539 | | | | 104834 | 155285 |
| 13 | **951** | 1249 | 1796 | 2396 | 1838 | 728 | 930 | 1266 | 196 | 2353 | 701 | 1906 | 1301 | | | 55751 | 71119 |
| 13 | **2097** | 1020 | 2525 | 628 | 1080 | 581 | 2709 | 1322 | 149 | 1125 | 2309 | 1210 | 1878 | | | 297169 | 501261 |
| 14 | **1300** | 1340 | 2934 | 1188 | 1696 | 1716 | 67 | 167 | 2390 | 950 | 1218 | 1201 | 2757 | 2584 | | 242632 | 314738 |
| 14 | **2800** | 631 | 608 | 2136 | 2925 | 163 | 628 | 1387 | 1337 | 2370 | 2226 | 2562 | 1550 | 739 | | 744882 | 999965 |
| 15 | **758** | 148 | 1880 | 43 | 281 | 2169 | 528 | 243 | 1589 | 1187 | 1145 | 290 | 268 | 2643 | 317 | 216216 | 428843 |
| 15 | **2841** | 438 | 775 | 2129 | 2919 | 1284 | 1374 | 613 | 2917 | 674 | 1740 | 843 | 1834 | 1314 | 924 | 1241253 | 1393263 |

TABLE 1. Knapsack cones with dimensions ranging from 8 to 14.

For a comparative analysis, we list several hard knapsack problems from Table 1 of the September 2002 preprint of [1] in Table 2. The first column provides the instance names, with the data for instances cuww1-3 omitted due to their negligible computation times. Instances cuww4-5, and prob1-10 are specifically crafted to possess the structure $a_i = Mp_i + Nr_i$ for certain integers $M$ and $N$, and for short integer vectors $p = (p_1, \ldots, p_n)$ and $r = (r_1, \ldots, r_n)$. The far-right column specifies the Frobenius number $F(a_1, \ldots, a_n)$. It is a notoriously difficult task for many algorithms, including the standard branch and bound approach, to confirm that $d(F; a_1, \ldots, a_n) = 0$. Instances prob11-20 represent random examples.

| instance | a | | | | | | | | | | Frobenius # |
|---|---|---|---|---|---|---|---|---|---|---|---|
| cuww4 | 13211 | 13212 | 39638 | 52844 | 66060 | 79268 | 92482 | | | | 104723595 |
| cuww5 | 13429 | 26850 | 26855 | 40280 | 40281 | 53711 | 53714 | 67141 | | | 45094583 |
| prob1 | 25067 | 49300 | 49717 | 62124 | 87608 | 88025 | 113673 | 119169 | | | 33367335 |
| prob2 | 11948 | 23330 | 30635 | 44197 | 92754 | 123389 | 136951 | 140745 | | | 14215206 |
| prob3 | 39559 | 61679 | 79625 | 99658 | 133404 | 137071 | 159757 | 173977 | | | 58424799 |
| prob4 | 48709 | 55893 | 62177 | 65919 | 86271 | 87692 | 102881 | 109765 | | | 60575665 |
| prob5 | 28637 | 48198 | 80330 | 91980 | 102221 | 135518 | 165564 | 176049 | | | 62442884 |
| prob6 | 20601 | 40429 | 40429 | 45415 | 53725 | 61919 | 64470 | 69340 | 78539 | 95043 | 22382774 |
| prob7 | 18902 | 26720 | 34538 | 34868 | 49201 | 49531 | 65167 | 66800 | 84069 | 137179 | 27267751 |
| prob8 | 17035 | 45529 | 48317 | 48506 | 86120 | 100178 | 112464 | 115819 | 125128 | 129688 | 21733990 |
| prob10 | 45276 | 70778 | 86911 | 92634 | 97839 | 125941 | 134269 | 141033 | 147279 | 153525 | 106925261 |
| prob11 | 11615 | 27638 | 32124 | 48384 | 53542 | 56230 | 73104 | 73884 | 112951 | 130204 | 577134 |
| prob12 | 14770 | 32480 | 75923 | 86053 | 85747 | 91772 | 101240 | 115403 | 137390 | 147371 | 944183 |
| prob13 | 15167 | 28569 | 36170 | 55419 | 70945 | 74926 | 95821 | 109046 | 121581 | 137695 | 765260 |
| prob14 | 11828 | 14253 | 46209 | 52042 | 55987 | 72649 | 119704 | 129334 | 135589 | 138360 | 680230 |
| prob15 | 13128 | 37469 | 39391 | 41928 | 53433 | 59283 | 81669 | 95339 | 110593 | 131989 | 663281 |
| prob16 | 35113 | 36869 | 46647 | 53560 | 81518 | 85287 | 102780 | 115459 | 146791 | 147097 | 1109710 |
| prob17 | 14054 | 22184 | 29952 | 64696 | 92752 | 97364 | 118723 | 119355 | 122370 | 140050 | 752109 |
| prob18 | 20303 | 26239 | 33733 | 47223 | 55486 | 93776 | 119372 | 136158 | 136989 | 148851 | 783879 |
| prob19 | 20212 | 30662 | 31420 | 49259 | 49701 | 62688 | 74254 | 77244 | 139477 | 142101 | 677347 |
| prob20 | 32663 | 41286 | 44549 | 45674 | 95772 | 111887 | 117611 | 117763 | 141840 | 149740 | 1037608 |

TABLE 2. Hard knapsack problems from [1].

By employing `DecDenu` and `LattE`, we have re-evaluated the values of $d(F+i; a_1, \ldots, a_n)$ for $i$ ranging from 0 to 100 for the instances listed in Table 2. The reported times correspond solely to the computation of the unimodular cones. It is worth noting that the transformation from unimodular cones to $d(F + i; a_1, \ldots, a_n)$ is a critical step, but the two software packages utilize distinct algorithms, as detailed in [19] and [10], respectively. For example, the prob8 instance is processed by `LattE` to transform 62044 unimodular cones in 6.0 seconds, whereas `DecDenu` transforms 139188 unimodular cones in 1.2 seconds. We observe that the computation time for both packages is nearly insensitive to the value of $i$. The lattice basis reduction method of [1] addresses integer feasibility and the computation of Frobenius numbers.

In the initial 12 hard instances, `DecDenu` generates a larger number of cones than `LattE`, yet its computation time is notably less. For the subsequent 10 random instances, `DecDenu` yields fewer cones than `LattE`, but its computation time is approximately one-twentieth that of `LattE`.

We conclude this section with the following remarks.

The multiplier chosen by `DecDenu` may not always be optimal. For instance, let $\alpha = (13429, 26850, 26855, 40280, 40281, 53711, 53714, 67141)$ be the vector in problem cuww5. The LLL policy selects multiplier 2447 at the root with pivot 26850, giving the normalized label

$$(\textcircled{1}; 26850, 3637, 12235, 1190, 1257, 67, 7408, 1123)$$

and 2226 leaves. If the root multiplier is instead forced to be 1, while the same policy is retained at all descendants, the normalized root label is

$$(\textcircled{1}; 26850, 13421, 5, 13420, 13419, 11, 14, 13409),$$

| | LattE | | DecDenu | | LattE/DecDenu | |
|---|---|---|---|---|---|---|
| | # of unimodular cone | time(s) | # of unimodular cone | time(s) | Number ratio | Time ratio |
| cuww4 | 364 | 0.01 | 1036 | 0.012 | 0.351 | 0.8333333 |
| cuww5 | 2514 | 0.16 | 5548 | 0.041 | 0.453 | 3.902 |
| prob1 | 10618 | 0.85 | 24786 | 0.178 | 0.428 | 4.775 |
| prob2 | 6244 | 0.51 | 11072 | 0.077 | 0.564 | 6.623 |
| prob3 | 12972 | 1.01 | 11490 | 0.08 | 1.129 | 12.625 |
| prob4 | 9732 | 0.72 | 15438 | 0.112 | 0.630 | 6.429 |
| prob5 | 8414 | 0.73 | 29595 | 0.221 | 0.284 | 3.303 |
| prob6 | 26448 | 2.82 | 52916 | 0.409 | 0.500 | 6.895 |
| prob7 | 20192 | 2.54 | 43552 | 0.35 | 0.464 | 7.257 |
| prob8 | 62044 | 7.56 | 139188 | 1.09 | 0.446 | 6.936 |
| prob9 | 36354 | 4.07 | 69808 | 0.547 | 0.521 | 7.441 |
| prob10 | 38638 | 3.88 | 53766 | 0.42 | 0.719 | 9.238 |
| prob11 | 5600284 | 701 | 4455683 | 36.7 | 1.257 | 19.101 |
| prob12 | 10778801 | 1417 | 6961202 | 57.9 | 1.548 | 24.473 |
| prob13 | 8400814 | 1076 | 6085420 | 52.4 | 1.380 | 20.534 |
| prob14 | 9068188 | 1221 | 7026995 | 59.3 | 1.290 | 20.590 |
| prob15 | 7647072 | 944 | 5183979 | 43.7 | 1.475 | 21.602 |
| prob16 | 6242020 | 818 | 4921562 | 40.9 | 1.268 | 20.000 |
| prob17 | 9000574 | 1237 | 6519150 | 54.2 | 1.381 | 22.823 |
| prob18 | 10097931 | 1309 | 6450759 | 53.4 | 1.565 | 24.513 |
| prob19 | 8506304 | 1074 | 6041508 | 49.8 | 1.408 | 21.566 |
| prob20 | 9568498 | 1248 | 6527133 | 54.6 | 1.466 | 22.857 |

TABLE 3. Data for `LattE` and `DecDenu`.

and the computation gives 910 leaves. These are two different root-selection policies, illustrating that the LLL candidate need not minimize the leaf count.

We can seamlessly transition to `LattE` at any node of $T_{\mathtt{D}}$, but we are unable to switch to `DecDenu` at nodes in $T_{\mathtt{B}}$.

(1) When encountering a bad vertex in $T_{\mathtt{D}}$, we have the option to switch to `LattE`.
(2) In all of the finite cases where $a_1 \leq 13$ and $d < 10$, the proportion of cases where $nl(T_{\mathtt{D}}) > nl(T_{\mathtt{B}})$ is less than $1/10$. It is feasible to log these instances and employ `LattE` when such cases arise.

## 4. LLLCTEuclid: A Novel Algorithm for Constant Term Extraction

We commence by elucidating the domain of the iterated Laurent series and the foundational elements essential for this section. Subsequently, we delineate the `LLLCTEuclid` algorithm, an innovative application of our `DecDenu` algorithm for the extraction of constant terms from Elliott rational functions. The `LLLCTEuclid` algorithm is grounded in the principles of `CTEuclid` but leverages the LLL's algorithm extensively. This integration positions it as a formidable tool for constant term extraction, making it a compelling choice among elimination-based algorithms.

### 4.1. The field of iterated Laurent series.

Iterated Laurent series provides a powerful framework for handling multivariate rational functions, particularly in the context

of constant term extraction. For any field $K$, the field $K((x))$ of Laurent series is constructed as $\{\sum_{k=N}^{\infty} a_k x^k : N \in \mathbb{Z}, a_k \in K\}$. The field of iterated Laurent series can be recursively defined; its use for constant term extraction is developed in [15].

In this sub-section, we provide a succinct overview of the iterated Laurent series. Our working field is $K = \mathbb{Q}((x_m))((x_{m-1}))\cdots((x_1))$, which encompasses the field of rational functions as a subfield. This field is specified by the sequence of variables $(x_1, \ldots, x_m)$, and the order of the variables is crucial in the series expansion of a rational function.

*Unique Series Expansion in $K$.* Working within $K$, every element possesses a unique series expansion. Consequently, the constant term operators $\mathrm{CT}_{x_i}$ are well-defined and commute with each other, as evidenced by the relations $\mathrm{CT}_{x_i}\,\mathrm{CT}_{x_j} = \mathrm{CT}_{x_j}\,\mathrm{CT}_{x_i}$. These commute relations are indispensable. Importantly, there is no convergence issue; the constant term of an iterated Laurent series remains an iterated Laurent series.

We are particularly interested in Elliott-rational functions, which are rational functions with binomial factors only in their denominator. Elliott-rational functions are named after Elliott's reduction identity, which demonstrates that the constant term of an Elliott rational function is still an Elliott rational function. The expansion of an Elliott rational function in $K$ is straightforward yet pivotal. Every monomial $M \neq 1$ can be compared with 1 in $K$ by identifying the "smallest" variable $x_j$ appearing in $M$, i.e., $\deg_{x_i} M = 0$ for all $i < j$. If $\deg_{x_j} M > 0$, then $M$ is considered "small", denoted $M < 1$; otherwise, $M$ is "large", denoted $M > 1$. For the unique series expansion of Elliott-rational functions in $K$, it suffices to consider the following two types of series expansions:

$$\frac{1}{1-M} = \begin{cases} \sum_{k\geq 0} M^k, & \text{if } M < 1; \\ \frac{1}{-M(1-1/M)} = \sum_{k\geq 0} -\frac{1}{M^{k+1}}, & \text{if } M > 1. \end{cases}$$

When expanded as a series in $K$, an Elliott-rational function $E$ is typically expressed in its proper form:

$$E = \frac{\text{a Laurent polynomial}}{\prod_{j=1}^{n}(1-\text{Monomial}_j)} = \frac{L}{\prod_{j=1}^{n}(1-M_j)} = L\prod_{j=1}^{n}\Big(\sum_{k\geq 0}(M_j)^k\Big), \text{ (proper form)} \tag{9}$$

where $L$ is a Laurent polynomial and $M_j < 1$ for all $j$. It is important to note that the proper form of $E$ is not unique; for instance, $1/(1-x) = (1+x)/(1-x^2)$ are both proper forms.

Now, we elucidate the *core problem* within MacMahon's partition analysis. Let $\Lambda = \{\lambda_1, \ldots, \lambda_r\} = \{x_{i_1}, \ldots, x_{i_r}\}$ be a subset of variables, and we employ the concise notation $\Lambda^{\mathbf{b}} = \lambda_1^{b_1}\cdots\lambda_r^{b_r}$. We can then express

$$E = \frac{L(\Lambda)}{\prod_{j=1}^{n}(1-\Lambda^{\mathbf{c}_j}M_j)}, \quad \text{(proper form in } K) \tag{10}$$

where $M_j$ are independent of $\Lambda$ for all $j$. The *type* of $E$ (with respect to $\Lambda$) is defined to be the matrix $A = (c_1, \ldots, c_n)$. It is typically assumed that $A$ is of rank $r$, and $d = n - r$ is referred to as the dimension of $\mathrm{CT}_\Lambda E$. If $L(\Lambda) = \Lambda^{-\mathbf{b}}$, then

$$\operatorname*{CT}_{\Lambda} E = \sum_{\alpha\in P\cap\mathbb{N}^n} M_1^{\alpha_1}\cdots M_n^{\alpha_n},$$

where $P$ is the polyhedron defined by $P = \{\alpha \in \mathbb{R}^n : A\alpha = \mathbf{b}\}$.

**Core Problem.** Given an Elliott rational function $E$ as described, the objective is to represent the constant term of $E$ in $K$ as follows:

$$\operatorname*{CT}_{\lambda_1,\dots,\lambda_r} E = \text{ a finite sum of simple rational functions free of the } \lambda\text{-variables.}$$

### 4.2. The `LLLCTEuclid` Algorithm.

**Algorithm 16** (LLLCTEuclid)**.**
***Input:*** *An Elliott rational function $E$ as in* (10) *and a working field $K$.*
***Output:*** $\mathrm{CT}_\Lambda E$ *as a finite sum of simple rational functions.*

1.1 *Initialize*

$$O_0' = \frac{L(\Lambda)}{(1 - z_1 M_1(x)\Lambda^{c_1})(1 - z_2 M_2(x)\Lambda^{c_2})\cdots(1 - z_n M_n(x)\Lambda^{c_n})}, \tag{11}$$

*and work in $K' = K((z_n))\cdots((z_1))$, the field specified by $(z_1, \dots, z_n, x_1, \dots, x_m)$.*

1.2 *(Optimization) Set $s = \operatorname{rank}(A)$. Apply Proposition 18 to obtain $O_0$ with the same constant term, a Laurent polynomial numerator with integral exponents, and a reduced type matrix of full row rank $s$. If the filtered numerator is zero, return zero. If $s = 0$, return the $\Lambda$-constant coefficient of the numerator divided by the $\Lambda$-independent denominator, followed by the specialization in Step 3. Otherwise, retain only the active variables $\lambda_1, \dots, \lambda_s$ and reset $r = s$ and $\Lambda = (\lambda_1, \dots, \lambda_r)$.*

2.1 *Compute $O_{i+1}$ from $O_i$ for $i = 0, 1, \dots, r-1$ as follows:*

2.2 *For each summand $O_{ij}$, select a variable $\lambda$ that has not yet been eliminated in that summand. Write its nonzero denominator exponents as $a_j$, and set $g = \gcd(|a_j| : a_j \neq 0)$. If there are no such exponents, replace the summand by the $\lambda$-constant coefficient of its numerator divided by its unchanged denominator. Otherwise, delete numerator monomials whose $\lambda$-exponent is not in $g\mathbb{Z}$, then divide every remaining $\lambda$-exponent, including those in the denominator, by $g$. The constant term is preserved: denominator expansions shift exponents only by multiples of $g$, and division by $g$ preserves whether an exponent is zero. Dividing by positive $g$ also preserves the sign of the first nonzero exponent in the prescribed variable order, so the small or large expansion choice of each binomial is unchanged. Factors independent of $\lambda$ stay in the coefficient field. Rewrite the $\lambda$-dependent factors with positive exponents, absorbing the monomial factors and signs into the numerator. Set $F = F^+ + F^-$, where $F^+$ contains the numerator monomials with positive $\lambda$-exponents and $F^-$ those with nonpositive exponents, with the same denominator. Then $F^-$ is proper in $\lambda$, while $F^+(0) = 0$. The partial fraction formulas of* [16] *give*

$$\operatorname*{CT}_\lambda F = \sum_{\substack{j:\, a_j>0 \\ u_j\lambda^{a_j}<1}} \mathcal{A}_{1-u_j\lambda^{a_j}} F^- - \sum_{\substack{j:\, a_j>0 \\ u_j\lambda^{a_j}>1}} \mathcal{A}_{1-u_j\lambda^{a_j}} F^+,$$

*where the inequalities refer to the prescribed monomial order in $K'$.*

2.3 *(Optimization) Compute each $\mathcal{A}_{1-u_i\lambda^{a_i}} F^\pm(\lambda)$ using Algorithm 6 (`DecDenu`) in Section 3.*

3 *After eliminating the $r$ active variables, set $d = n - r$. We obtain a finite sum*

$$O_r = \operatorname*{CT}_\Lambda O_0 = \sum_{i\in I} \frac{L_i(x; z)}{\prod_{j=1}^{d_i}(1 - M_{ij} z^{B_{ij}})}, \qquad 0 \le d_i \le d.$$

*Here $I$ is the finite output index set; when $d_i = 0$, the denominator is an empty product. Provided that the total expression has a well-defined specialization at $z = (1, \dots, 1)$, eliminate the slack variables using generalized Todd polynomials as in* [19].

The novel algorithm, `LLLCTEuclid`, is constructed within the framework of `CTEuclid`, with modifications to Steps 1.2, 2.3, and 3. The computation of $\mathcal{A}_{1-u_i\lambda^{\alpha_i}} F^{\pm}(\lambda)$ in Step 2.3 is the focal point of this study, which we have addressed in Section 3. Step 3 employs the log-exponential trick for generalized Todd polynomials, as detailed in our recent work [19].

We now elaborate on Step 1, which introduces a new simplification facilitated by the LLL algorithm. For simplicity, we consider the case where $L(\Lambda) = \Lambda^{-\mathbf{b}}$ to illustrate the concept. The approach extends naturally for Laurent polynomial $L$. The inclusion of slack variables $z_j$ in Step 1 ensures that the denominator factors of $O_{ij}$ remain coprime throughout our elimination process. Consequently, Proposition 1 can be applied consistently when computing $\mathrm{CT}_\lambda O_{ij}$ in Step 2.

Using direct series expansion, it is evident that

$$\operatorname*{CT}_{\Lambda} E = \left( \operatorname*{CT}_{\Lambda} O_0' \right)\Big|_{z_i=1, i=1,2,\dots,n}.$$

The type of $O_0'$ is $A = (c_1, \dots, c_n)$, and the constant term corresponds to a weighted count of lattice points within the polytope $P = \{\alpha \in \mathbb{R}^n : A\alpha = \mathbf{b}\}$. Elementary row operations do not alter the solution set, thus $P = \{\alpha \in \mathbb{R}^n : WA\alpha = W\mathbf{b}\}$ for any nonsingular matrix $W$. We aim for $WA$ to have small integral entries. Additionally, we seek to simplify equations such as $k\alpha_1 - 2k\alpha_n = b$ to $\alpha_1 - 2\alpha_n = \frac{b}{k}$, ensuring that $P$ contains no lattice points when $\frac{b}{k}$ is not an integer. This corresponds to dividing a row of $WA$ by their greatest common divisor.

To utilize the aforementioned idea, we define the action of a nonsingular matrix $W = (w_{ij})_{r\times r}$ by

$$WF(\Lambda) := W \circ F(\Lambda) = F(\gamma_1, \dots, \gamma_r), \qquad \gamma_i = \prod_{j=1}^{r} \lambda_j^{w_{ji}}, \quad 1 \le i \le r.$$

We allow rational entries in $W$. If $q$ clears their denominators, the transformed Laurent polynomials are interpreted using powers of $\lambda_i^{1/q}$. For a monomial, $W \circ \Lambda^\alpha = \Lambda^{W\alpha}$.

**Lemma 17.** *Suppose $W$ is a nonsingular matrix and $F(\Lambda)$ is a Laurent polynomial in $\Lambda$. Then we have*

$$\operatorname*{CT}_{\Lambda} F(\Lambda) = \operatorname*{CT}_{\Lambda}(W \circ F(\Lambda)).$$

*Proof.* By linearity, we can assume $F(\Lambda) = \Lambda^\alpha$. Then $W \circ F(\Lambda) = \Lambda^{W\alpha}$. Since $\alpha = 0 \Leftrightarrow W\alpha = 0$, the lemma follows. □

The same argument applies coefficientwise to series in the slack variables whose coefficients are finite Laurent polynomials in the original variables. In $K'$, each $z_j$ precedes the original variables, so every factor of $O_0'$ is expanded geometrically in $z_j$. Each coefficient of $\mathbf{z}^\nu$, $\nu \in \mathbb{Z}^n_{\ge 0}$, is a finite Laurent polynomial. The lemma therefore applies to $O_0'$ coefficientwise, also when rational powers are introduced by $W$. This justification uses the slack-variable expansion; an arbitrary monomial substitution need not preserve the expansion of an arbitrary iterated Laurent series.

**Proposition 18.** *Let $O_0'$ be as in* (11)*, with a Laurent polynomial numerator with integral exponents and type matrix $A \in \mathbb{Z}^{r\times n}$ of rank $s$. If $s > 0$, its constant term can be represented by a rational function $O_0$ such that: i) $O_0$ depends only on $\lambda_1, \dots, \lambda_s$; ii) its numerator is an integral-exponent Laurent polynomial; iii) its type matrix $A^L \in \mathbb{Z}^{s\times n}$ has rank $s$ and $s$-th determinant factor* 1*; iv) the rows of $A^L$ form an LLL-reduced basis. If $s = 0$, the constant term is obtained directly from the numerator.*

*Proof.* If $s = 0$, then $A = 0$ and the denominator is independent of $\Lambda$. Thus $\mathrm{CT}_\Lambda O_0' = (\mathrm{CT}_\Lambda L)/\prod_{j=1}^n (1 - z_j M_j)$. Assume $s > 0$. Choose unimodular matrices $U \in \mathbb{Z}^{r\times r}$ and $V \in \mathbb{Z}^{n\times n}$ such that

$$UAV = S = \begin{pmatrix} \mathrm{diag}(d_1, \dots, d_s) & 0 \\ 0 & 0 \end{pmatrix}$$

is the Smith normal form. Set

$$W_1 = \mathrm{diag}(d_1^{-1}, \dots, d_s^{-1}, E_{r-s})U, \qquad A' = W_1 A = \begin{pmatrix} E_s & 0 \\ 0 & 0 \end{pmatrix} V^{-1}.$$

The matrix $W_1$ is nonsingular, so Lemma 17, applied coefficientwise in the slack variables, preserves the constant term. The denominator after this transformation has integral exponents in the active variables and is independent of $\lambda_{s+1}, \dots, \lambda_r$.

Write the transformed numerator as the finite sum

$$W_1 \circ L(\Lambda) = \sum_{\alpha\in\mathcal{S}} \ell_\alpha \Lambda^\alpha, \qquad \mathcal{S} \subset \mathbb{Q}^r,$$

and set $\ell_\alpha = 0$ for $\alpha \notin \mathcal{S}$. A numerator monomial with a nonzero inactive exponent cannot contribute to the constant term. Neither can one whose active exponent vector lies outside $\mathbb{Z}^s$, since every denominator expansion shifts that vector by an element of $\mathbb{Z}^s$. Consequently we may replace the numerator by

$$L'(\lambda_1, \dots, \lambda_s) = \sum_{\substack{\beta\in\mathbb{Z}^s \\ (\beta,0)\in\mathcal{S}}} \ell_{(\beta,0)} \lambda_1^{\beta_1} \cdots \lambda_s^{\beta_s}.$$

Let $A''$ be the first $s$ rows of $A'$. Then $A''V = (E_s, 0)$, so its $s$-th determinant factor is 1. Choose a unimodular $W_2 \in \mathbb{Z}^{s\times s}$ such that $A^L = W_2 A''$ has LLL-reduced rows. With $\mathbf{c}_j'$ the $j$-th column of $A''$, set

$$O_0 = W_2 \circ \frac{L'(\lambda_1, \dots, \lambda_s)}{\prod_{j=1}^n (1 - z_j M_j \lambda_1^{c_{1j}'} \cdots \lambda_s^{c_{sj}'})}.$$

A second application of Lemma 17 preserves the constant term. Unimodularity preserves integral exponents and the determinant factors. Thus $O_0$ has all the asserted properties. □

**Example 19.** *Compute $\mathrm{CT}_\Lambda F$, where $F = F(\Lambda)$ has the following proper form:*

$$F = \frac{\frac{\lambda_1^{11}\lambda_4^{33}}{\lambda_2^{33}\lambda_3^{39}} + \frac{\lambda_3^{32}}{\lambda_1^4\lambda_2^{112}\lambda_4^7} + \frac{\lambda_3^{111}}{\lambda_1^{15}\lambda_2^{358}\lambda_4^{45}}}{\left(1 - \frac{\lambda_1^{19}\lambda_2^{165}\lambda_4^{57}z_1}{\lambda_3^{99}}\right)\left(1 - \frac{\lambda_3^{36}z_2}{\lambda_1^8\lambda_2^{30}\lambda_4^{24}}\right)\left(1 - \frac{\lambda_3^5 z_3}{\lambda_1\lambda_2^7\lambda_4^3}\right)\left(1 - \frac{\lambda_3^{50}z_4}{\lambda_1^{10}\lambda_2^{72}\lambda_4^{30}}\right)\left(1 - \frac{\lambda_2^{28}z_5}{\lambda_3^4}\right)}.$$

*Solution.* Using `CTEuclid` produces 23 terms, while using `LLLCTEuclid` only produces 9 terms. We only illustrate the first step of the `LLLCTEuclid`.

The type of $F$ is represented by the matrix

$$A = \begin{pmatrix} 19 & -8 & -1 & -10 & 0 \\ 165 & -30 & -7 & -72 & 28 \\ -99 & 36 & 5 & 50 & -4 \\ 57 & -24 & -3 & -30 & 0 \end{pmatrix}.$$

The Smith Normal Form of matrix $A$ is given by

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} = UAV = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -4 & 1 & 1 & 0 \\ -3 & 0 & 1 & 0 \\ -3 & 0 & 0 & 1 \end{pmatrix} A \begin{pmatrix} -2 & 0 & -1 & -1 & -1 \\ 0 & 0 & 0 & 1 & 0 \\ 251 & 10 & 121 & 3 & 1 \\ -29 & -1 & -14 & -3 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Let $W_1$ be defined as

$$W_1 = \operatorname{diag}\left(1, \frac{1}{2}, \frac{1}{4}, 1\right) U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & \frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{3}{4} & 0 & \frac{1}{4} & 0 \\ -3 & 0 & 0 & 1 \end{pmatrix}$$

and then the transformed matrix $A'$ is given by

$$A' = W_1 A = \begin{pmatrix} 19 & -8 & -1 & -10 & 0 \\ -5 & 19 & 1 & 9 & 12 \\ -39 & 15 & 2 & 20 & -1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Set $A''$ as the first 3 rows of $A'$ and apply the LLL algorithm on the rows of $A''$, we obtain

$$A^L = W_2 A'' = \begin{pmatrix} -1 & -1 & 0 & 0 & -1 \\ 0 & 0 & -1 & -1 & -1 \\ 2 & -1 & 0 & -1 & 0 \end{pmatrix},$$

where $W_2 = \begin{pmatrix} 2 & 0 & 1 \\ -222 & -9 & -107 \\ 25 & 1 & 12 \end{pmatrix}$. Hence, we derive the following transformation:

$$\begin{aligned} \operatorname*{CT}_{\Lambda} F = \operatorname*{CT}_{\Lambda} W_2 \circ (W_1 \circ F) &= \operatorname*{CT}_{\Lambda} \frac{\lambda_1^4 \lambda_2^6 \lambda_3 + \frac{\lambda_1^3 \lambda_4^5}{\lambda_2} + \frac{\lambda_1^9}{\lambda_2^{3/2} \lambda_3^{1/2}}}{\left(1 - \frac{\lambda_3^2 z_1}{\lambda_1}\right)\left(1 - \frac{z_2}{\lambda_1 \lambda_3}\right)\left(1 - \frac{z_3}{\lambda_2}\right)\left(1 - \frac{z_4}{\lambda_2 \lambda_3}\right)\left(1 - \frac{z_5}{\lambda_1 \lambda_2}\right)} \\ &= \operatorname*{CT}_{\Lambda} \frac{\lambda_1^4 \lambda_2^6 \lambda_3}{\left(1 - \frac{\lambda_3^2 z_1}{\lambda_1}\right)\left(1 - \frac{z_2}{\lambda_1 \lambda_3}\right)\left(1 - \frac{z_3}{\lambda_2}\right)\left(1 - \frac{z_4}{\lambda_2 \lambda_3}\right)\left(1 - \frac{z_5}{\lambda_1 \lambda_2}\right)}. \end{aligned}$$

□

**Declaration:** The authors declared that they had no conflicts of interest with respect to their authorship or the publication of this article.

**Data availability:** All computational data reported in this study are included in this article. The original hard knapsack benchmark instances are available in the cited source [1].

**Funding:** This work is partially supported by the National Natural Science Foundation of China [12571355].

## References

[1] K. Aardal and A. K. Lenstra, *Hard equality constrained integer knapsacks*, in *Integer Programming and Combinatorial Optimization*, W. J. Cook and A. S. Schulz, eds., Lecture Notes in Computer Science, vol. 2337, Springer, Berlin, 2002, pp. 350–366. https://doi.org/10.1007/3-540-47867-1_25. A full preprint with the benchmark data is available as Utrecht University Preprint 1256 (September 12, 2002), https://www.math.uu.nl/publications/preprints/1256.pdf.

[2] J. L. Ramírez Alfonsín, *The Diophantine Frobenius Problem*, Oxford University Press, Oxford, 2005. https://doi.org/10.1093/acprof:oso/9780198568209.001.0001.

[3] G. E. Andrews, P. Paule, and A. Riese, *MacMahon's Partition Analysis VI: A New Reduction Algorithm*, Annals of Combinatorics, 5 (2001), pp. 251–270. https://doi.org/10.1007/s00026-001-8011-y.

[4] E. Bajo, R. Davis, J. A. De Loera, A. Garber, S. Garzón Mora, K. Jochemko, and J. Yu, *Weighted Ehrhart theory: Extending Stanley's nonnegativity theorem*, Advances in Mathematics, 444 (2024), Article 109627. https://doi.org/10.1016/j.aim.2024.109627.

[5] V. Baldoni, N. Berline, J. A. De Loera, M. Köppe, and M. Vergne, *Computation of the highest coefficients of weighted Ehrhart quasi-polynomials of rational polyhedra*, Foundations of Computational Mathematics, 12 (2012), no. 4, pp. 435–469. https://doi.org/10.1007/s10208-011-9106-4.

[6] A. Barvinok and K. Woods, *Short rational generating functions for lattice point problems*, Journal of the American Mathematical Society, 16 (2003), no. 4, pp. 957–979. https://doi.org/10.1090/S0894-0347-03-00428-4.

[7] A. I. Barvinok, *A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed*, Mathematics of Operations Research, 19 (1994), no. 4, pp. 769–779. https://doi.org/10.1287/moor.19.4.769.

[8] F. Breuer and Z. Zafeirakopoulos, *Polyhedral Omega: a new algorithm for solving linear Diophantine systems*, Annals of Combinatorics, 21 (2017), no. 2, pp. 211–280. https://doi.org/10.1007/s00026-017-0349-x.

[9] J. A. De Loera, D. Haws, R. Hemmecke, P. Huggins, and R. Yoshida, *A computational study of integer programming algorithms based on Barvinok's rational functions*, Discrete Optimization, 2 (2005), no. 2, pp. 135–144. https://doi.org/10.1016/j.disopt.2005.04.001.

[10] J. A. De Loera, D. C. Haws, and M. Köppe, *Ehrhart polynomials of matroid polytopes and polymatroids*, Discrete & Computational Geometry, 42 (2009), no. 4, pp. 670–702. https://doi.org/10.1007/s00454-008-9080-z.

[11] J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida, *Effective lattice point counting in rational convex polytopes*, Journal of Symbolic Computation, 38 (2004), no. 4, pp. 1273–1302. https://doi.org/10.1016/j.jsc.2003.04.003.

[12] P. Dongre, B. Drabkin, J. Lim, E. Partida, E. Roy, D. Ruff, A. Seceleanu, and T. Tang, *Computing rational powers of monomial ideals*, Journal of Symbolic Computation, 116 (2023), pp. 39–57. https://doi.org/10.1016/j.jsc.2022.08.018.

[13] F. Liu and G. Xin, *A fast algorithm for denumerants with three variables*, Journal of Symbolic Computation, 129 (2025), article 102414. https://doi.org/10.1016/j.jsc.2024.102414.

[14] R. P. Stanley, *Enumerative Combinatorics, Volume 1*, 2nd ed., Cambridge Studies in Advanced Mathematics, vol. 49, Cambridge University Press, Cambridge, 2011. https://doi.org/10.1017/CBO9781139058520.

[15] G. Xin, *A fast algorithm for MacMahon's partition analysis*, The Electronic Journal of Combinatorics, 11 (2004), no. 1, Research Paper R58. https://doi.org/10.37236/1811.
[16] G. Xin, *A Euclid style algorithm for MacMahon's partition analysis*, Journal of Combinatorial Theory, Series A, 131 (2015), pp. 32–60. https://doi.org/10.1016/j.jcta.2014.11.006.
[17] G. Xin and X. Xu, *A polynomial time algorithm for calculating Fourier-Dedekind sums*, Experimental Mathematics, 34 (2025), no. 3, pp. 342–349. https://doi.org/10.1080/10586458.2024.2364264.
[18] G. Xin and C. Zhang, *An algebraic combinatorial approach to Sylvester's denumerant*, The Ramanujan Journal, 66 (2025), article 64. https://doi.org/10.1007/s11139-024-01022-1.
[19] G. Xin, Y. Zhang, and Z. Zhang, *Fast evaluation of generalized Todd polynomials: Applications to MacMahon's partition analysis and integer programming*, Journal of Symbolic Computation, 130 (2025), article 102420. https://doi.org/10.1016/j.jsc.2025.102420.

[1]School of Mathematical Sciences, Capital Normal University, Beijing 100048, PR China

Yunnan University of Finance and Economics, School of Statistics and Mathematics, 650221 Kunming, China [2]

[3]School of Mathematics and Statistics, Beijing Institute of Technology, Beijing 102400, P.R. China

*Email address*: [1]guoce_xin@163.com & [2]zyrzuhe@126.com & [3]zihao-zhang@foxmail.com