2021

Loss of Precision in Implementations of the Toom-Cook Algorithm

Marcus Elia
University of Vermont

Loss of Precision in Implementations of the Toom-Cook Algorithm

A Dissertation Presented

by

Marcus Elia

to

The Faculty of the Graduate College

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
Specializing in Mathematical Sciences

May, 2021

Defense Date: March 25th, 2021
Dissertation Examination Committee:

Christelle Vincent, Ph.D., Advisor
Joseph Near, Ph.D., Chairperson
Greg Warrington, Ph.D.
Chris Danforth, Ph.D.
Cynthia J. Forehand, Ph.D., Dean of Graduate College

Abstract

Historically, polynomial multiplication has required a quadratic number of operations. Several algorithms in the past century have improved upon this. In this work, we focus on the Toom-Cook algorithm. Devised by Toom in 1963, it is a family of algorithms parameterized by an integer, n. The algorithm multiplies two polynomials by recursively dividing them into smaller polynomials, multiplying many small polynomials, and interpolating to obtain the product. While it is no longer the asymptotically fastest method of multiplying, there is a range of intermediate degrees (typically less than 1000) where it performs the best.

Some applications, like quantum-resistant cryptosystems, require the use of polynomials whose coefficients belong to the ring of integers modulo a power of 2. A problem arises with using the Toom-Cook algorithm to multiply these polynomials because the interpolation step of the algorithm requires division by even numbers. This results in a loss of 2-adic precision. If too many bits of precision are lost, the product will be incorrect.

Interpolating a polynomial from some of its values is generally easy, and different works have solved the interpolation step of the Toom-Cook algorithm with different equations. In order to track the loss of precision, it is necessary to establish and prove the general form of the solution to the system of equations. We present three sets of interpolation formulas: the *matrix*, *natural*, and *efficient* formulas. For any integer $n > 2$, we seek to find a general expression for each of the three sets of formulas, and to prove the respective loss of precision. First, for the efficient interpolation, we prove the general set of formulas. Then, for the natural interpolation, we conjecture a general set of formulas that depends on two combinatorial identities. We prove the first identity and some cases of the second identity. Finally, we prove the loss of precision of the matrix interpolation formulas.

Acknowledgements

I am thankful for the guidance I have received from my adviser, Christelle Vincent.

I am thankful to my committee for their careful review and feedback.

Over the years, I had several helpful conversations with Richard Foote and Greg Warrington.

This work was partially funded by the NSF (grant DMS-1802323).

Table of Contents

Chapter 1

Introduction

1.1 An Overview of Multiplication Algorithms

Multiplying two polynomials of degree d in the simplest way, using distributivity, requires $O(d^2)$ operations. We say *schoolbook multiplication* to refer to the process of multiplying polynomials by distributing. This requires $O(d^2)$ multiplications between coefficients, and a few additions. This was improved upon by Karatsuba [16] in 1963. The Karatsuba algorithm recursively splits the polynomials in half, multiplying smaller polynomials, and then combining them back together. This has a complexity of $O(d^{\log_2(3)})$. The Karatsuba algorithm can be viewed as a special case of the Toom-Cook algorithm, discovered by Toom [28] and formalized by Cook [8]. The Toom-Cook algorithm is a family of algorithms, parameterized by any integer $n \geq 2$. It works by recursively dividing the polynomials into n parts, multiplying the smaller polynomials, and combining them back together. It achieves a com-

plexity of $O(d^{\log_n(2n-1)})$. We will explain in more detail how the Karatsuba and Toom-Cook algorithms work in Chapter 2. In 1971, Schonhage and Strassen introduced a new multiplication algorithm based on the fast Fourier transform (FFT) that achieves complexity $O(d\log(d)\log(\log(d)))$ [26]. This algorithm is now known as the Schonhage-Strassen algorithm. This method was refined by Furer [12], and finally improved to $O(d\log(d))$ by Harvey and van der Hoeven [14].

We are primarily concerned with performing multiplication over $(\mathbb{Z}/2^m\mathbb{Z})[x]$. Some public key cryptosystems require efficient multiplication in such rings. Notable examples include NIST Post-Quantum Cryptography candidate algorithms NTRU [31] and Saber [10]. Multiplication algorithms that are based on the FFT (those due to Schonhage and Strassen, Furer, and Harvey and van der Hoeven) require a ring in which 2 is a unit, so they are not directly applicable to this situation. This leaves Toom-Cook as a leading candidate for multiplying polynomials for those applications. Recent work has shown that multiplication based on the FFT can, with some extra work, be applied to these situations and potentially result in faster multiplication [7].

1.2 Previous Work on the Toom-Cook Algorithm

After the Toom-Cook algorithm was discovered in the 1960s, further work has been done to improve it and apply it to different situations. Knuth gives a thorough explanation of complexity and implementation aspects [17, pp. 294-318]. Bodrato

applies the Toom-Cook algorithm to multivariate polynomials in [2], and Bodrato and Zanoni extend the algorithm to perform multiplication when the operands have different degrees [3]. Mera, Karmakar, and Verbauwhede optimize the evaluation and interpolation stages of Toom-Cook by pre-computing and saving results [21]. Kronenburg provides analysis of a multithreaded implementation [18]. Zanoni shows that the Toom-8 variant of Toom-Cook can efficiently multiply large integers [30]. Dutta, Bhattacharjee, and Chattopadhyay give a quantum circuit to implement Toom-Cook multiplication [11], and Maji and Mullins introduce a way that Toom-Cook can speed up a specific type of neural network [20].

1.3 Structure of this Thesis

In Chapter 2, we formally describe the Toom-Cook algorithm, the loss of precision, and introduce three sets of interpolation formulas. These are called the efficient, natural, and matrix formulas. In Chapter 3, we prove the general closed form for the efficient formulas. In Chapter 4, we conjecture the general closed form for the natural formulas, and show that the conjecture depends on two combinatorial identities. We prove the first identity, and present some progress toward the second identity. In Chapter 5, we apply known results about Vandermonde matrices to prove the loss of precision of the matrix formulas. In Chapter 6, we discuss methods for tracking the loss of precision in Toom-Cook interpolation formulas. In Chapter 7, we present graphs that show which Toom-Cook decompositions are more efficient.

Chapter 2

The Toom-Cook Algorithm

2.1 Precise Description of the Toom-Cook Algorithm

We present a brief description of the Toom-Cook algorithm. Suppose we want to multiply two polynomials $f(x)$ and $g(x)$ of degree d_1 and d_2, respectively. Then for $d \geq d_1, d_2$ and n dividing d, we write

$$f(x) = f_0(x) + f_1(x)x^{d/n} + f_2(x)x^{2d/n} + \cdots + f_{n-1}(x)x^{(n-1)d/n}$$

and

$$g(x) = g_0(x) + g_1(x)x^{d/n} + g_2(x)x^{2d/n} + \cdots + g_{n-1}(x)x^{(n-1)d/n}$$

for some degree d/n polynomials f_i, g_i, $i = 0, \ldots, n-1$. Letting $X = x^{d/n}$, we define $F(X)$ and $G(X)$ as

$$F(X) = f_0(x) + f_1(x)X + f_2(x)X^2 + \cdots + f_{n-1}(x)X^{n-1}$$

and

$$G(X) = g_0(x) + g_1(x)X + g_2(x)X^2 + \cdots + g_{n-1}(x)X^{n-1}.$$

We define $r(X)$ to be the product of $F(X)$ and $G(X)$, which is a degree $2n-2$ polynomial in the variable X whose coefficients are degree $2d/n$ polynomials in the variable x. The strategy of the Toom-Cook algorithm to obtain $r(X)$ is to evaluate r at $2n-1$ different points, and solve the resulting system of linear equations to find the coefficients $r_0(x), r_1(x), \ldots, r_{2n-2}(x)$. Since $r(X) = F(X)G(X)$, evaluating r at a point requires evaluating F and G at the same point and multiplying the results. We choose to evaluate at the points $0, \pm 1, \pm 2, \ldots, \pm(n-2), n-1, \infty$, where evaluation at infinity means returning the leading coefficient of the polynomial. There are other sets of evaluation points, but this is the simplest one. It is also used for implementations of the NTRU cryptosystem [9]. So we perform $2n-1$ multiplications of polynomials

of degree d/n to find

$$\begin{aligned} r(0) &= F(0)G(0) \\ r(1) &= F(1)G(1) \\ r(-1) &= F(-1)G(-1) \\ &\vdots \\ r(n-1) &= F(n-1)G(n-1) \\ r(\infty) &= F(\infty)G(\infty). \end{aligned}$$

Given that these values have been computed, it is now a matter of finding the coefficients of $r(X)$. This will be discussed later in this chapter. Once we have all of the coefficients of $r(X)$, we can obtain the product $f(x)g(x)$ by evaluating $r(X)$ at $X = x^{d/n}$.

2.2 Loss of Precision

Working over $(\mathbb{Z}/2^m\mathbb{Z})[x]$, most of the Toom-Cook algorithm is performed just like it would be over $\mathbb{Z}[x]$. However, the interpolation formulas have divisions. Dividing by an odd number is acceptable, because odd numbers are invertible and the division can be replaced with a multiplication by the inverse. Even numbers in $\mathbb{Z}/2^m\mathbb{Z}$, on the other hand, do not have multiplicative inverses. One way to avoid this problem is to perform all computations in $\mathbb{Z}$, and then reduce the final answer modulo 2^m. This is often not applicable, since many situations require the hardware to consistently

use a fixed-size integer type (16-bit integers, for example). We use M to denote the number of bits being used to represent integers. When all arithmetic is done using M-bit integers, the operations are performed modulo 2^M. Moving forward, we assume all computations are done in $\mathbb{Z}/2^M\mathbb{Z}$ and the final product needs to have correct coefficients in $\mathbb{Z}/2^m\mathbb{Z}$, for some $m \leq M$. In practice, M is likely to be 16 or 32, as determined by the hardware.

Even though division by 2 is not defined the in ring $\mathbb{Z}/2^M\mathbb{Z}$, the divisions by 2 in the Toom-Cook algorithm will still result in a well-defined element of $\mathbb{Z}/2^M\mathbb{Z}$. To help see why, we define $a \,\%\, 2^M$.

Definition 2.2.1. For any integer a, let $a \,\%\, 2^M$ be the unique integer between 0 and $2^M - 1$ such that $a \,\%\, 2^M \equiv a \bmod 2^M$.

We can still perform divisions by smaller powers of 2 because reducing a to $a \,\%\, 2^M$ does not affect this divisibility.

Lemma 2.2.2. *For any integer a, and for any positive integer $k < M$, if $2^k | a$, then $2^k | a \,\%\, 2^M$.*

Proof. By definition, there exists some integer j_1 such that $a \,\%\, 2^M = a - 2^M j_1$. Since $2^k | a$, there exists some integer j_2 such that $a = 2^k j_2$. So

$$a \,\%\, 2^M = a - 2^M j_1 = 2^k j_2 - 2^m j_1 = 2^k (j_2 - 2^{M-k} j_1).$$

□

So division by 2 still "works" in the sense that the quotient is an integer, but it is not guaranteed to be the answer we want. As an example, suppose we want to compute

$$\frac{101 + 139}{16} \,\%\, 64.$$

The result is $(240/16)\%64 = 15$. But if we perform the operations in a different order, and instead compute

$$\frac{(101 + 139) \,\%\, 64}{16},$$

the result is $(240 \,\%\, 64)/16 = 48/16 = 3$. This is because dividing by powers of 2 loses bits of precision. Consider the previous example in terms of binary numbers:

$$240 = 11110000_2$$

and

$$48 = 00110000_2.$$

These numbers agree on the least significant 6 bits (since they are congruent modulo 2^6), but they disagree on any bits after that. Dividing by 16 has the effect of right shifting by 4 bits. This means that 4 of the 6 agreeing bits are gone, and only the 2 least significant bits are now shared.

Definition 2.2.3. When the result of a computation is no longer correct in its k most significant bits, we say that the computation *loses k bits of precision.*

The loss of precision can be expressed in terms of the 2-adic valuation function.

Definition 2.2.4. The 2-adic valuation function, v_2, is given by

$$v_2(n) = \max\{v \in \mathbb{Z} : 2^v \mid n\}$$

if n is even, and $v_2(n) = \infty$ if n is odd.

Definition 2.2.5 (Alternate Definition for Loss of Precision)**.** Let a be the correct result of a computation over $\mathbb{Z}$ and b the obtained result over $\mathbb{Z}/2^M\mathbb{Z}$. Then the loss of precision of the computation is equal to $v_2(a - b)$.

The divisions by even numbers that show up in the Toom-Cook interpolation formulas can result in a loss of precision. We are interested in finding general forms for the three sets of interpolation formulas described above, so we can rigorously prove how many bits are lost.

2.3 Toom-Cook Interpolation Formulas

In this section, we introduce methods of solving the linear system of equations. As is common in the literature, we say *Toom-n* to refer to the specific member Toom-Cook family of algorithms where the polynomials are split into n pieces. Also note that the Karatsuba algorithm can be viewed as Toom-2. We focus on three sets of interpolation formulas to solve the system of linear equations.

2.3.1 The Matrix Formulas

The *matrix formulas* come from viewing polynomial evaluation as matrix multiplication. Observe that

$$\begin{pmatrix} r(-(n-2)) \\ r(-(n-3)) \\ \vdots \\ r(n-1) \\ r(\infty) \end{pmatrix} = \begin{pmatrix} 1 & -(n-2) & \dots & (-(n-2))^{2n-3} & (-(n-2))^{2n-2} \\ 1 & -(n-3) & \dots & (-(n-3))^{2n-3} & (-(n-3))^{2n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & n-1 & \dots & (n-1)^{2n-3} & (n-1)^{2n-2} \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix} \begin{pmatrix} r_0 \\ r_1 \\ \vdots \\ r_{2n-3} \\ r_{2n-2} \end{pmatrix}$$

Inverting this matrix yields a set of formulas that express the coefficients of $r(X)$ in terms of the evaluation points. As an example, consider the matrix formulas for Toom-4:

$$\begin{aligned} r_0 &= r(0) \\ r_6 &= r(\infty) \\ r_4 &= \frac{1}{4!}(r(-2) - 4r(-1) + 6r(0) - 4r(1) + r(2) - 120r(\infty)) \\ r_2 &= \frac{1}{4!}(-r(-2) + 16r(-1) - 30r(0) + 16r(1) - r(2) + 96r(\infty)) \\ r_5 &= \frac{1}{5!}(-r(-2) + 5r(-1) - 10r(0) + 10r(1) - 5r(2) + r(3) - 360r(\infty)) \\ r_3 &= \frac{1}{5!}(-5r(-2) - 5r(-1) + 50r(0) - 70r(1) + 35r(2) - 5r(3) + 1800r(\infty)) \\ r_1 &= \frac{1}{5!}(6r(-2) - 60r(-1) - 40r(0) + 120r(1) - 30r(2) + 4r(3) - 1440r(\infty)). \end{aligned}$$

2.3.2 The Natural Formulas

The *natural formulas* could be derived when solving the set of equations by hand. For Toom-4, the natural formulas are

$$
\begin{aligned}
r_0 &= r(0) \\
r_6 &= r(\infty) \\
r_4 &= \frac{1}{4!}(-4(r(1)+r(-1)) + (r(2)+r(-2)) + 6r_0 - 120r_6) \\
r_2 &= \frac{1}{2!}((r(1)+r(-1)) - 2r_0 - 2r_4 - 2r_6) \\
r_5 &= \frac{1}{5!}(5r(1) - 4r(2) + r(3) - 2r_0 + 2r_2 - 22r_4 - 478r_6) \\
r_3 &= \frac{1}{3!}(-2r(1) + r(2) + r_0 - 2r_2 - 14r_4 - 30r_5 - 62r_6) \\
r_1 &= r(1) - r_0 - r_2 - r_3 - r_4 - r_5 - r_6.
\end{aligned}
$$

2.3.3 The Efficient Formulas

The *efficient formulas* store the results of some computations as temporary variables to avoid redundant operations. We are not claiming that they are the most efficient set of formulas that could exist: we simply mean to convey that they were designed to, and indeed do, achieve efficiency gains over the other two sets of formulas. As a trade-off, we will see that they lose more bits of precision. Here are the Toom-4

efficient formulas:

$$
\begin{aligned}
r_0 &= r(0) \\
r_6 &= r(\infty) \\
E_1 &= \frac{r(1) + r(-1)}{2} - r_0 - r_6 \\
r_4 &= \left(\left(\frac{r(2) + r(-2)}{2} - r_0 - 64 r_6 \right) /4 - E_1 \right) /3 \\
r_2 &= E_1 - r_4 \\
O_1 &= \frac{r(1) - r(-1)}{2} \\
O_2 &= \left(\frac{r(2) - r(-2)}{4} - O_1 \right) /3 \\
O_3 &= \left(\frac{r(3) - r_0 - 9 r_2 - 81 r_4 - 729 r_6}{3} - O_1 \right) /8 \\
r_5 &= (O_3 - O_2)/5 \\
r_3 &= O_2 - 5 r_5 \\
r_1 &= O_1 - r_3 - r_5 .
\end{aligned}
$$

2.4 Summary of Results

For each the three sets of Toom-Cook interpolation formulas we are considering (matrix, natural, and efficient), two things need to be shown:

- The exact form of the Toom-n formulas for all n.
- The loss of 2-adic precision of using those formulas, for all n.

For the efficient formulas, we present a proof of the general Toom-n formulas, but we

do not have a precise conjecture for the loss of precision. For the natural formulas, we conjecture a general set of formulas that depends on two combinatorial identities. We present a proof of the first identity, and some work towards the proof of the second. We also conjecture that the natural formulas lose the same number of bits as the matrix formulas. For the matrix formulas, we do not conjecture a general set of Toom-n formulas, but we prove the number of bits of precision that they lose.

Chapter 3

The Efficient Formulas

In this chapter, we give a closed form for the set of efficient Toom-n interpolation formulas and prove the correctness of the closed form. To do so, we introduce a family of triangles and show how to solve systems of linear equations that satisfy certain conditions.

3.1 A Family of Triangles

For any positive integer $b \geq 1$, we define $t_b(n, k)$ to be a triangle given by the recurrence relation

$$t_b(n, k) = t_b(n-1, k-1) + (k + b - 2)^2 t_b(n-1, k) \tag{3.1}$$

and the base cases $t_b(n, 1) = 1$, $t_b(n, n) = 1$, and $t_b(n, k) = 0$ if $k < 1$ or $k > n$. Note that indexing starts at 1, so $t_b(1, 1)$ is the top point of the triangle. We refer to equation 3.1 as the "triangle recurrence."

The $b = 2$ triangle is described on OEIS [27]. It looks like

$$\begin{array}{ccccccc} & & & 1 & & & \\ & & 1 & & 1 & & \\ & 1 & & 5 & & 1 & \\ 1 & & 21 & & 14 & & 1 \\ 1 & 85 & & 147 & & 30 & 1 \end{array}$$

The subsequent triangles follow a similar pattern, with larger integers. The entries in these triangles show up in the general efficient formulas.

For brevity, we define

$$c(k, b) = 2(k-1)b + (k-1)^2.$$

This c stands for "coefficient", because the values of $c(k, b)$ comprise many of the coefficients in the efficient formulas. To prove the efficient formulas, we first show that, for all $b \geq 2$, for all $n \geq 1$, and for all $k \geq 1$,

$$t_b(n, k+1)c(k, b) = t_{b+1}(n, k) - t_b(n, k).$$

Proposition 3.1.1. *For all $b \geq 1$, for all $n \geq 1$, and for all $k \geq 1$,*

$$t_b(n, k+1)c(k, b) = t_{b+1}(n, k) - t_b(n, k).$$

Proof. We begin by handling some special cases. If $k > n$, then $t_b(n, k+1) = 0$ and $t_{b+1}(n,k) - t_b(n,k) = 0 - 0 = 0$. If $k = n$, then $t_b(n, n+1) = 0$ and $t_{b+1}(n,n) - t_b(n,n) = 1 - 1 = 0$. Finally, if $k = 1$, we observe that $c(1,b) = 0$ and we have $t_{b+1}(n,1) - t_b(n,1) = 1 - 1 = 0$.

When $1 < k < n$, we proceed by strong induction on n. By the previous special cases, the recurrence holds for $n = 1$ and $n = 2$. For the inductive hypothesis, suppose that

$$t_b(j, k+1)c(k,b) = t_{b+1}(j,k) - t_b(j,k)$$

for $j = 1, 2, \ldots, n-1$. We will use this and the triangle recurrence (equation 3.1). By the triangle recurrence where k is replaced by $k+1$, we have

$$t_b(n, k+1) = t_b(n-1,k) + (k+b-1)^2 t_b(n-1, k+1).$$

Applying the inductive hypothesis with $j = n-1$ yields

$$t_b(n-1, k+1)c(k,b) = t_{b+1}(n-1,k) - t_b(n-1,k).$$

Using the triangle recurrence where b is replaced by $b+1$, we obtain

$$(k+b-1)^2 t_{b+1}(n-1,k) = t_{b+1}(n,k) - t_{b+1}(n-1,k-1).$$

By the inductive hypothesis where k is replaced by $k-1$ and $j = n-1$ (or trivially if $k = 2$ since $c(1,b) = 0$),

$$t_{b+1}(n-1,k-1) = c(k-1,b)t_b(n-1,k) + t_b(n-1,k-1).$$

Finally, using the triangle recurrence, we have

$$t_b(n,k) = t_b(n-1,k-1) + (k+b-2)^2 t_b(n-1,k).$$

Putting all of these equations together,

$$\begin{aligned}
t_b(n,k+1)c(k,b) &= c(k,b)(t_b(n-1,k) + (k+b-1)^2 t_b(n-1,k+1)) \\
&= c(k,b)t_b(n-1,k) + (k+b-1)^2(t_{b+1}(n-1,k) - t_b(n-1,k)) \\
&= 2(k-1)bt_b(n-1,k) + (k-1)^2 t_b(n-1,k) \\
&\quad + (k+b-1)^2 t_{b+1}(n-1,k) - ((k-1)+b)^2 t_b(n-1,k) \\
&= (-b^2)t_b(n-1,k) + (k+b-1)^2 t_{b+1}(n-1,k) \\
&= (-b^2)t_b(n-1,k) + t_{b+1}(n,k) - t_{b+1}(n-1,k-1) \\
&= (-b^2)t_b(n-1,k) + t_{b+1}(n,k) - c(k-1,b)t_b(n-1,k) \\
&\quad - t_b(n-1,k-1) \\
&= (-b^2)t_b(n-1,k) + t_{b+1}(n,k) \\
&\quad - 2(k-2)bt_b(n-1,k) - (k-2)^2 t_b(n-1,k) - t_b(n-1,k-1) \\
&= t_{b+1}(n,k) - (k+b-2)^2 t_b(n-1,k) - t_b(n-1,k-1) \\
&= t_{b+1}(n,k) - t_b(n,k).
\end{aligned}$$

□

3.2 Solving a General System of Equations

Let $n \geq 5$ be a positive integer, and let $T_1, T_2, \ldots, T_{n-2}$ be constants. Suppose we have $n-2$ variables $x_1, x_2, \ldots, x_{n-2}$, and a system of $n-2$ equations given as follows:

$$T_1 = x_1 + x_2 + \cdots + x_{n-2} \tag{3.2}$$

and, for $b = 2, 3, \ldots, n-2$,

$$T_b = x_2 + t_b(3,2)x_3 + t_b(4,2)x_4 + \cdots + t_b(n-2,2)x_{n-2}. \tag{3.3}$$

We give an algorithm to solve the system of equations. As a notational note, we say "equation (b)" to refer to the equation with T_b on the left side.

3.2.1 Overview of the Solution

We proceed by introducing layers of equations, where each layer consists of equations that have each eliminated one variable from the previous layer. Layer 1 is defined to be equations (2) through $(n-2)$, and equation 3.2 can be thought of as layer 0. Observe that subtracting consecutive equations from layer 1 will cancel the x_2 out, yielding an equation that contains only the variables $x_3, \ldots, x_{n-2}$. We then divide both sides of that equation by the coefficient on x_3 to obtain an equation in layer 2. In this way, layer 2 consists of $n-3$ linear equations in the variables $x_3, \ldots, x_{n-2}$

such that the coefficient of x_3 is 1. In general, layer L consists of $n - L - 1$ linear equations that contain the variables $x_{L+1}, \ldots, x_{n-2}$ such that coefficient of x_{L+1} is 1. Layer $L + 1$ is given by subtracting the $n - L - 2$ pairs of consecutive equations in layer L (thus eliminating x_{L+1}), and then dividing each of the resulting equations by the coefficient of x_{L+2}.

Once we reach layer $n-3$, it contains only two equations that depend on x_{n-3} and x_{n-2}. We directly solve for x_{n-2}. Knowing x_{n-2}, we plug this back into an equation from the layer $n - 3$ and solve for x_{n-3}. Now that we know x_{n-2} and x_{n-3}, we plug them back into an equation from layer $n-4$ (which depended only on x_{n-4}, x_{n-3} and x_{n-2}) and solve for x_{n-4}. We continue back through the layers, finding one more x_i from each layer by substituting in the variables we have already solved for. Since we can choose any equation from each layer to substitute into, we choose the first equation from each layer.

For convenience, we give a method of indexing the equations in each layer. For each layer, the first equation is indexed by $b = 2$. We define, for all $n \geq 1, L \geq 1$ and $b \geq 2$,

$$\text{ind}(n, L, b) = (L - 1)n - \frac{(L + 1)(L + 2)}{2} + 3 + b, \tag{3.4}$$

and $\text{ind}(n, 0, 2) = 1$. This expression gives the index of the equation corresponding to b in layer L. Here is an illustration of how the index function works:

$$\text{Layer } 1$$

$$\text{ind}(n, 1, 2) = 2$$

$$\text{ind}(n, 1, 3) = 3$$

$$\vdots$$

$$\text{ind}(n, 1, n-2) = n-2$$

$$\text{Layer } 2$$

$$\text{ind}(n, 2, 2) = n-1$$

$$\text{ind}(n, 2, 3) = n$$

$$\vdots$$

$$\text{ind}(n, 2, n-3) = 2n-5$$

$$\text{Layer } 3$$

$$\text{ind}(n, 3, 2) = 2n-4$$

$$\vdots$$

3.2.2 Explicitly Stating the Formulas

We claim that the system of equations can be solved by the following formulas: for L from 2 to $n-4$, for b from 2 to $n-1-L$,

$$T_{\text{ind}(n,L,b)} = \frac{T_{\text{ind}(n,L-1,b+1)} - T_{\text{ind}(n,L-1,b)}}{c(L,b)},$$

and

$$x_{n-2} = \frac{T_{\mathrm{ind}(n,n-4,3)} - T_{\mathrm{ind}(n,n-4,2)}}{c(n-4,2)},$$

and for i from $n-3$ down to 1,

$$x_i = T_{\mathrm{ind}(n,i-1,2)} - \sum_{j=i+1}^{n-2} t_2(j,i)x_j.$$

When $n = 8$, assume that we are given $T_1, \ldots, T_6$, as they are constants appearing in the original equations which we seek to solve. Here is what the system of equations looks like:

Layer 2

$$T_7 = (T_3 - T_2)/5$$

$$T_8 = (T_4 - T_3)/7$$

$$T_9 = (T_5 - T_4)/9$$

$$T_{10} = (T_6 - T_5)/11$$

Layer 3

$$T_{11} = (T_8 - T_7)/12$$

$$T_{12} = (T_9 - T_8)/16$$

$$T_{13} = (T_{10} - T_9)/20$$

Layer 4

$$T_{14} = (T_{12} - T_{11})/21$$

$$T_{15} = (T_{13} - T_{12})/27$$

Variables

$$x_6 = (T_{15} - T_{14})/32$$

$$x_5 = T_{14} - 55x_6$$

$$x_4 = T_{11} - 30x_5 - 627x_6$$

$$x_3 = T_7 - 14x_4 - 147x_5 - 1408x_6$$

$$x_2 = T_2 - 5x_3 - 21x_4 - 85x_5 - 341x_6$$

$$x_1 = T_1 - x_2 - x_3 - x_4 - x_5 - x_6.$$

3.2.3 Proving the Correctness of the Formulas

In order for the algorithm to be correct, we need $T_{\text{ind}(n,L,b)}$, which is defined by

$$T_{\text{ind}(n,L,b)} = \frac{T_{\text{ind}(n,L-1,b+1)} - T_{\text{ind}(n,L-1,b)}}{c(L,b)},$$

to also satisfy

$$T_{\text{ind}(n,L,b)} = \sum_{j=L+1}^{n-2} t_b(j, L+1)x_j. \tag{3.5}$$

This will guarantee that the $n - L - 1$ linear equations in layer L depend only the variables $x_{L+1}, \ldots, x_{n-2}$ and that the coefficient of x_{L+1} is 1. This will also confirm that the formula

$$x_i = T_{\text{ind}(n,i-1,2)} - \sum_{j=i+1}^{n-2} t_2(j, i)x_j \tag{3.6}$$

holds, since that is what we obtain when we solve the first equation in layer $i-1$ for x_i.

We will use the result of Proposition 3.1.1: for all $n \geq 5, b \geq 2, L \geq 1$, we have

$$t_b(n-1, L+1) = \frac{t_{b+1}(n-1, L) - t_b(n-1, L)}{c(L,b)}.$$

Proposition 3.2.1. *Recall the numbers $t_b(n,k)$ defined in equation 3.1 and the numbers* $\text{ind}(n,L,b)$ *defined in equation 3.4. For all $n \geq 5, L \geq 1$, and $b \geq 2$,*

$$T_{\text{ind}(n,L,b)} = \sum_{j=L+1}^{n-2} t_b(j, L+1)x_j.$$

Proof. We use induction on L. By our fundamental assumption about the system of equations, the proposition holds for $L = 1$. For our inductive hypothesis, suppose

that the proposition holds for $L = k - 1$. We will show that it holds for $L = k$. Since the expression for each T is linear, the coefficient on x_j in the expression for $T_{\text{ind}(n,L,b)}$ depends only on the coefficient on x_j in $T_{\text{ind}(n,L-1,b)}$ and $T_{\text{ind}(n,L-1,b+1)}$. By the inductive hypothesis and Proposition 3.1.1, we have

$$\begin{aligned}
T_{\text{ind}(n,k,b)} &= \frac{T_{\text{ind}(n,k-1,b+1)} - T_{\text{ind}(n,k-1,b)}}{c(k,b)} \\
&= \frac{\sum_{j=k}^{n-2} t_{b+1}(j,k)x_j - \sum_{j=k}^{n-2} t_b(j,k)x_j}{c(k,b)} \\
&= \sum_{j=k}^{n-2} x_j \frac{t_{b+1}(j,k) - t_b(j,k)}{c(k,b)} \\
&= \sum_{j=k}^{n-2} x_j t_b(j,k+1) \\
&= t_b(k,k+1)x_k + \sum_{j=k+1}^{n-2} x_j t_b(j,k+1) \\
&= \sum_{j=k+1}^{n-2} x_j t_b(j,k+1).
\end{aligned}$$

□

3.3 Application to the Toom-Cook Algorithm

One step of the Toom-n algorithm requires us to find the coefficients $r_0, r_1, \ldots, r_{2n-2}$ of a degree $2n-2$ polynomial $r(x)$, given that we know $r(-(n-2)), r(-(n-3))$,

$\ldots, r(-1), r(0)$, $r(1), \ldots, r(n-1)$, and $r(\infty)$ (where we recall that $r(\infty)$ is defined to be the leading coefficient of r). It immediately follows that $r_0 = r(0)$ and $r_{2n-2} = r(\infty)$. We start by looking at the even coefficients: $r_2, r_4, \ldots, r_{2n-4}$. This is $n-2$ variables, so we need $n-2$ equations. First, we define

$$\begin{aligned} E_1 &= \frac{r(1) + r(-1)}{2} - r_0 - r_{2n-2} \\ &= r_2 + r_4 + \cdots + r_{2n-4}. \end{aligned}$$

Then, for b from 2 to $n-2$, define

$$E_b = \left(\left(\frac{r(b) + r(-b)}{2} - r_0 - b^{2n-2} r_{2n-2} \right) / b^2 - E_1 \right) / (b^2 - 1). \tag{3.7}$$

Now we need a couple of lemmas.

Lemma 3.3.1. *Let $b \geq 2$ be an integer. For all integers $k \geq 1$,*

$$\frac{b^{2k} - 1}{b^2 - 1} = b^{2k-2} + b^{2k-4} + \cdots + b^2 + 1.$$

Proof. We shall use induction on k. For the base case,

$$\frac{b^2 - 1}{b^2 - 1} = 1.$$

For the inductive hypothesis, suppose that the formula holds for a given k. We will

show that it holds for $k+1$. Then

$$\begin{aligned} b^{2k+2}-1 &= b^{2k+2}-b^{2k}+b^{2k}-1 \\ &= b^{2k}(b^2-1)+b^{2k}-1 \\ &= (b^2-1)(b^{2k}+b^{2k-2}+\cdots+b^2+1). \end{aligned}$$

□

This next lemma is about the $t_b(n,k)$ triangles.

Lemma 3.3.2. *For all integers $b \geq 2$ and for all integers $j \geq 2$,*

$$t_b(j,2) = \sum_{i=0}^{j-2} b^{2i}.$$

Proof. We shall proceed by induction on j. For the base case, observe that

$$t_b(2,2) = t_b(1,1) + (2+b-2)^2 t_b(1,2) = 1+0 = 1.$$

For our inductive hypothesis, assume that

$$t_b(j,2) = \sum_{i=0}^{j-2} b^{2i}$$

for some $j \geq 2$. Then

$$\begin{aligned} t_b(j+1,2) &= t_b(j,1) + (2+b-2)^2 t_b(j,2) \\ &= 1 + b^2(1 + b^2 + \cdots + b^{2j-4}) \\ &= 1 + b^2 + b^4 + \cdots + b^{2j-2} \\ &= \sum_{i=0}^{j+1-2} b^{2i}. \end{aligned}$$

□

Now, we can show what we want: that the even Toom-n coefficients satisfy a system of equations of the form studied in this chapter, and hence can be obtained with the method derived above.

Lemma 3.3.3. *Recall that E_b was defined in equation 3.7. For b from 2 to $n-2$,*

$$E_b = r_4 + t_b(3,2)r_6 + t_b(4,2)r_8 + \cdots + t_b(n-2,2)r_{2n-4}.$$

Proof. Using the two previous lemmas,

$$\begin{aligned}
E_b &= \left(\left(\frac{r(b)+r(-b)}{2} - r_0 - b^{2n-2}r_{2n-2}\right)/b^2 - E_1\right)/(b^2-1) \\
&= \left(\left(\frac{2r_0+2b^2r_2+\cdots+2b^{2n-2}}{2} - r_0 - b^{2n-2}r_{2n-2}\right)/b^2 - E_1\right)/(b^2-1) \\
&= \Big((b^2r_2+b^4r_4+\cdots+b^{2n-4}r_{2n-4})/b^2 - E_1\Big)/(b^2-1) \\
&= \Big((b^2-1)r_4+(b^4-1)r_6+\cdots+(b^{2n-6}-1)r_{2n-4}\Big)/(b^2-1) \\
&= r_4+(b^2+1)r_6+\cdots+\left(\sum_{j=0}^{n-4} b^{2j}\right)r_{2n-4} \\
&= r_4+t_b(3,2)r_6+t_b(4,2)r_8+\cdots+t_b(n-2,2)r_{2n-4}.
\end{aligned}$$

□

So we can now solve for $r_2, r_4, \ldots, r_{2n-4}$ using equation 3.6.

Now, we know $r_0, r_2, \ldots, r_{2n-2}$ and we need to find $r_1, r_3, \ldots, r_{2n-3}$, which is $n-1$ variables. We define

$$\begin{aligned}
O_1 &= \frac{r(1)-r(-1)}{2} \\
&= r_1+r_3+\cdots+r_{2n-3}.
\end{aligned}$$

For $b = 2, \ldots, n-2$, we define

$$\begin{aligned}
O_b &= \left(\frac{r(b) - r(-b)}{2b} - O_1 \right) / (b^2 - 1) \\
&= \left(\frac{2br_1 + 2b^3 r_3 + \cdots + 2b^{2n-3} r_{2n-3}}{2b} - O_1 \right) / (b^2 - 1) \\
&= \Big(r_1 + b^2 r_3 + \cdots + b^{2n-4} r_{2n-3} - O_1 \Big) / (b^2 - 1) \\
&= \Big((b^2 - 1) r_3 + (b^4 - 1) r_5 + \cdots + (b^{2n-4} - 1) r_{2n-3} \Big) / (b^2 - 1) \\
&= r_3 + (b^2 + 1) r_5 + \cdots + \left(\sum_{j=0}^{n-3} b^{2j} \right) r_{2n-3} \\
&= r_3 + t_b(3,2) r_5 + t_b(4,2) r_7 + \cdots + t_b(n-1,2) r_{2n-3}.
\end{aligned}$$

But we need one more equation to match form of equations 3.2 and 3.3, since we have $n-1$ variables. We get the last equation as follows:

$$\begin{aligned}
O_{n-1} &= \left(\frac{r(n-1) - \sum_{i=0}^{n-1} (n-1)^{2i} r_{2i}}{n-1} - O_1 \right) / \Big((n-1)^2 - 1 \Big) \\
&= \left(\frac{(n-1) r_1 + \cdots + (n-1)^{2n-3} r_{2n-3}}{n-1} - O_1 \right) / \Big((n-1)^2 - 1 \Big) \\
&= \Big(((n-1)^2 - 1) r_3 + ((n-1)^4 - 1) r_5 + \cdots + ((n-1)^{2n-4} - 1) r_{2n-3} \Big) / \Big((n-1)^2 - 1 \Big) \\
&= r_3 + \Big((n-1)^2 + 1 \Big) r_5 + \cdots + \left(\sum_{j=0}^{n-3} (n-1)^{2j} \right) r_{2n-3}.
\end{aligned}$$

And now the odd coefficients can be found using equation 3.6 with n replaced by $n+1$.

Chapter 4

The Natural Formulas

4.1 Introduction

We conjecture a general set of natural formulas that relies on two combinatorial identities:

$$\sum_{j=1}^{m}(-1)^{m+j}\binom{2m}{m-j}j^{2m} = (2m)!/2 \tag{4.1}$$

and

$$\sum_{j=1}^{m}(-1)^{m+j}\binom{2m}{m-j}j^{i} = 0 \tag{4.2}$$

if $i = 2, 4, 6, \ldots, 2m-2$. We provide a combinatorial proof for the first identity, and a proof using Gosper's algorithm for the $i = 2, 4, \ldots, 50$ cases of the second identity.

We note that a similar pair of identities are known to be true:

$$\sum_{j=0}^{m}\binom{m}{j}(m-j)^{m}(-1)^{j} = m!$$

and, for all $i = 1, 2, \ldots, m-1$,

$$\sum_{j=0}^{m} \binom{m}{j} (m-j)^i (-1)^j = 0.$$

These appear as exercises in [1, p. 89-90], and they can be proven with an elementary combinatorial argument. We hoped to find an analogous combinatorial argument for the new identities, but we have not been successful.

4.2 Conjecturing the Formulas

The objective is to find an efficient and generalizable method of solving the system of linear equations. Recall that, for all a, $r(a)$ and r_a are really degree $2d/n$ polynomials in x, but we leave off the functional notation for brevity. Since $r(0)$ and $r(\infty)$ are just the first and last coefficients, respectively, we know that $r_0 = r(0)$ and $r_{2n-2} = r(\infty)$. We use some notation to display the conjectured formulas for the even-indexed coefficients (r_{2m}) and the odd-indexed coefficients (r_{2m-1}). Let

$$e_j = (-1)^{m+j} \binom{2m}{m-j}$$

in the r_{2m} formulas and let

$$o_j = (-1)^{m+j} \binom{2m-1}{m-j} \frac{2j}{m+j}$$

in the r_{2m-1} formulas. We present the following conjecture for the formula of r_i for any i satisfying $1 \leq i \leq 2m-3$.

Conjecture 4.2.1. *Let r be a polynomial of degree $2n-2$ with coefficients r_i and denote the evaluation of r at the point $x=a$ by $r(a)$. Then for all $m=1,2,3,\ldots,n-2$,*

$$r_{2m} = \frac{1}{(2m)!}\left(\sum_{j=1}^{m}\Big(e_j\big(r(j)+r(-j)\big)\Big) - \left(\sum_{k=m+1}^{n-1}\sum_{j=1}^{m}(2e_j j^{2k})r_{2k} + \sum_{j=1}^{m}2e_j r_0\right)\right)$$

and for all $m=1,2,3,\ldots,n-1$,

$$r_{2m-1} = \frac{1}{(2m-1)!}\left(\sum_{j=1}^{m}o_j r(j) - \left(\sum_{k=0}^{n-1}\sum_{j=1}^{m}(o_j j^{2k})r_{2k} + \sum_{k=m+1}^{n-1}\sum_{j=1}^{m}(o_j j^{2k-1})r_{2k-1}\right)\right).$$

As an example, consider the case when $n=4$. Assume that we have computed $r(0)$, $r(\pm1)$, $r(\pm2)$, $r(3)$, and $r(\infty)$. According to our conjecture, we should have

$$r_0 = r(0)$$

$$r_6 = r(\infty)$$

$$r_4 = \frac{1}{24}(-4(r(1)+r(-1)) + (r(2)+r(-2)) + 6r_0 - 120r_6)$$

$$r_2 = \frac{1}{2}(r(1)+r(-1)-2r_0-2r_4-2r_6)$$

$$r_5 = \frac{1}{120}(5r(1)-4r(2)+r(3)-2r_0+2r_2-22r_4-478r_6)$$

$$r_3 = \frac{1}{6}(-2r(1)+r(2)+r_0-2r_2-14r_4-30r_5-62r_6)$$

$$r_1 = r(1)-r_0-r_2-r_3-r_4-r_5-r_6.$$

In this case, the correctness of these formulas can easily be verified by substituting in for the values of $r(a)$ and simplifying.

4.3 Stating the Identities

This proof of the conjecture relies on identities 4.1 and 4.2:

$$2\sum_{j=1}^{m}(-1)^{m+j}\binom{2m}{m-j}j^{2m} = (2m)!$$

and, for all $i = 2, 4, \ldots, 2m-2$,

$$2\sum_{j=1}^{m}(-1)^{m+j}\binom{2m}{m-j}j^{i} = 0.$$

Proposition 4.3.1. *If identities 4.1 and 4.2 both hold, then Conjecture 2.1 holds.*

Proof. We begin by considering the even case. Suppose identities 4.1 and 4.2 hold. Then

$$\sum_{j=1}^{m}(2e_j j^{2m}) = 2\sum_{j=1}^{m}(-1)^{m+j}\binom{2m}{m-j}j^{2m} = (2m)!$$

and

$$\sum_{j=1}^{m}(2e_j j^{i}) = 2\sum_{j=1}^{m}(-1)^{m+j}\binom{2m}{m-j}j^{i} = 0$$

for $i = 0, 2, \ldots, 2m-2$. Since, for any j,

$$r(j) = r_0 + r_1 j + \cdots + r_{2n-2}j^{2n-2},$$

we have

$$r(j) + r(-j) = 2r_0 + 2r_2 j^2 + \cdots + 2r_{2n-2}j^{2n-2}$$

and

$$e_j(r(j) + r(-j)) = 2e_j r_0 + 2e_j r_2 j^2 + \cdots + 2e_j r_{2n-2}j^{2n-2}.$$

For $m = 0, 1, \ldots, n-2$, we can isolate the r_{2m} term to obtain

$$\begin{aligned} 2e_j r_{2m} j^{2m} &= e_j(r(j) + r(-j)) - 2e_j r_0 - \cdots - 2e_j r_{2m-2} j^{2m-2} - 2e_j r_{2m+2} j^{2m+2} \\ &\quad - \cdots - 2e_j r_{2n-2} j^{2n-2} \\ &= e_j(r(j) + r(-j)) - \sum_{k=0}^{m-1} 2e_j r_{2k} j^{2k} - \sum_{k=m+1}^{n-1} 2e_j r_{2k} j^{2k}. \end{aligned}$$

Summing over j from 1 to m on both sides, this becomes

$$\sum_{j=1}^{m} (2e_j j^{2m}) r_{2m} = \sum_{j=1}^{m} e_j(r(j) + r(-j)) - \sum_{k=0}^{m-1} \sum_{j=1}^{m} (2e_j j^{2k}) r_{2k} - \sum_{k=m+1}^{n-1} \sum_{j=1}^{m} (2e_j j^{2k}) r_{2k}.$$

Apply identities 4.1 and 4.2 to obtain

$$r_{2m} = \frac{1}{(2m)!} \left(\sum_{j=1}^{m} \Big(e_j \big(r(j) + r(-j) \big) \Big) - \left(\sum_{k=m+1}^{n-1} \sum_{j=1}^{m} (2e_j j^{2k}) r_{2k} + \sum_{j=1}^{m} 2e_j r_0 \right) \right),$$

as desired.

We consider the odd case. For $m = 1, 2, \ldots, n-1$,

$$\sum_{j=1}^{m} o_j r(j) = \sum_{j=1}^{m} \sum_{k=0}^{2n-2} o_j r_k j^k = \sum_{k=0}^{2n-2} \sum_{j=1}^{m} o_j r_k j^k.$$

We split this summation into four pieces: when k is even, when k is odd and less than

$2m-1$, when $k=2m-1$, and when k is odd and greater than $2m-1$. This yields

$$\sum_{j=1}^{m} o_j r(j) = \sum_{k=0}^{n-1}\sum_{j=1}^{m} o_j r_{2k} j^{2k} + \sum_{k=1}^{m-1}\sum_{j=1}^{m} o_j r_{2k-1} j^{2k-1} + \sum_{j=1}^{m} o_j r_{2m-1} j^{2m-1} + \sum_{k=m+1}^{n-1}\sum_{j=1}^{m} o_j r_{2k-1} j^{2k-1} \tag{4.3}$$

For any integer k,

$$\begin{aligned}
\sum_{j=1}^{m} o_j j^{2k-1} &= \sum_{j=1}^{m} (-1)^{m-j} \binom{2m-1}{m-j} \frac{2j}{m+j} j^{2k-1} \\
&= \frac{m}{m} \sum_{j=1}^{m} (-1)^{m-j} \frac{(2m-1)!}{(m-j)!(m+j-1)!} \frac{2j}{m+j} j^{2k-1} \\
&= \frac{1}{m} \sum_{j=1}^{m} (-1)^{m-j} \frac{(2m)!}{(m-j)!(m+j)!} j^{2k} \\
&= \frac{1}{m} \sum_{j=1}^{m} (-1)^{m-j} \binom{2m}{m-j} j^{2k}.
\end{aligned}$$

Applying identities 4.1 and 4.2, this implies that

$$\sum_{j=1}^{m} o_j j^{2k-1} = \frac{1}{m} \frac{(2m)!}{2} = (2m-1)!$$

if $k=m$ and

$$\sum_{j=1}^{m} o_j j^{2k-1} = 0$$

if $k = 1, 2, \ldots, m-1$. So equation 4.3 becomes

$$\sum_{j=1}^{m} o_j r(j) = \sum_{k=0}^{n-1}\sum_{j=1}^{m} o_j r_{2k} j^{2k} + r_{2m-1}(2m-1)! + \sum_{k=m+1}^{n-1}\sum_{j=1}^{m} o_j r_{2k-1} j^{2k-1}.$$

Solving for r_{2m-1} yields the desired formula. □

4.4 Proof of Identity 4.1

In this section, we show that

$$2\sum_{j=1}^{m}(-1)^{m-j}\binom{2m}{m-j}j^{2m} = (2m)!$$

4.4.1 Preliminary Definitions

We begin by presenting some definitions.

Definition 4.4.1 (Permutations)**.** A *permutation* of the set $X = \{1, 2, 3, \ldots, n\}$ is a bijection from X to itself.

We represent permutations by writing their range in order. For example, if $\sigma(1) = 2, \sigma(2) = 4, \sigma(3) = 3$, and $\sigma(4) = 1$, then we write $\sigma = (2, 4, 3, 1)$.

Definition 4.4.2 (Inverse Descents)**.** Suppose σ is a permutation of $\{1, 2, \ldots, n\}$. For any integer $i \in \{1, 2, \ldots, n-1\}$, we say σ has an *inverse descent* at the element i if $\sigma^{-1}(i) > \sigma^{-1}(i+1)$.

For example, if $\sigma = (2, 4, 3, 1)$, then σ has an inverse descent at 1 since $\sigma^{-1}(1) = 4$ and $\sigma^{-1}(2) = 1$. Additionally, σ has an inverse descent at 3, but not at 2. An inverse descent happens whenever $i + 1$ appears before i in our representation of σ.

Definition 4.4.3 (IDes)**.** Given a permutation σ, IDes(σ) denotes the number of inverse descents of σ. In mathematical notation,

$$\text{IDes}(\sigma) = \left|\left\{i \in \{1, 2, \ldots, 2m-1\} \mid i+1 \text{ occurs to the left of } i \text{ in } \sigma\right\}\right|.$$

Definition 4.4.4 (Reverse of a Permutation)**.** Let $\sigma = (a_1, a_2, \ldots, a_n)$ be a permutation. Then the *reverse* of σ, denoted $\hat{\sigma}$, is the permutation $(a_n, a_{n-1}, \ldots, a_2, a_1)$.

For example, if $\sigma = (2, 4, 3, 1)$ then $\hat{\sigma} = (1, 3, 4, 2)$.

For our proof, we will only deal with permutations of an even number of elements.

Definition 4.4.5 (S_{2m})**.** For $m \geq 1$, we define S_{2m} to be the set of permutations of $\{1, 2, \ldots, 2m\}$.

Suppose $\sigma \in S_{2m}$. It is clear that $\mathrm{IDes}(\sigma) = 2m-1-\mathrm{IDes}(\hat{\sigma})$, since $\hat{\sigma}$ reverses the order of every pair. From this, we see that $\mathrm{IDes}(\sigma) \leq m-1 \iff \mathrm{IDes}(\hat{\sigma}) > m-1$. Since mapping a permutation to its reverse is an involution and no permutation is its own reverse, exactly $(2m)!/2$ of the permutations of $\{1, 2, \ldots, 2m\}$ have $m-1$ or fewer inverse descents.

Definition 4.4.6 ($\overline{S_{2m}}$)**.** Let $\overline{S_{2m}}$ be the set of permutations $\sigma \in S_{2m}$ such that $\mathrm{IDes}(\sigma) \leq m-1$.

Dividing both sides of identity 4.1 by 2, we obtain

$$\frac{(2m)!}{2} = \sum_{j=1}^{m} (-1)^{m+j} \binom{2m}{m-j} j^{2m}.$$

This suggests that the identity may be proven by establishing a bijection between $\overline{S_{2m}}$ and another set of cardinality $\sum_{j=1}^{m} (-1)^{m+j} \binom{2m}{m-j} j^{2m}$. This set will be the set of so-called "minimal" strings of length $2m$ with characters in $\{1, 2, \ldots, m\}$. The main work of this section is to define these strings, after which counting them will be straightforward.

4.4.2 Results About Strings

Definition 4.4.7 (Strings)**.** Let $\{1, 2, \ldots, k\}^{2m}$ denote the set of strings of length $2m$ made from the characters 1 through k.

We represent strings with parentheses and commas, the same notation as we use for permutations.

Definition 4.4.8 (Indexing Strings)**.** For any $s \in \{1, 2, \ldots, k\}^{2m}$ and for any $i \in \{1, 2, \ldots, 2m\}$, let s_i be the ith character of s (with indexing starting at 1).

We now define a function which maps strings in $\{1, 2, \ldots, m\}^{2m}$ to permutations in S_{2m}.

Definition 4.4.9 (Standardization)**.** Let the *standardization* map, $A : \{1, 2, \ldots, m\}^{2m} \to \overline{S_{2m}}$, be defined as follows: Given a string s, define $n_1, n_2, \ldots, n_m$ such that s has n_i copies of the character i. Then $A(s)$ is obtained by replacing the n_1 copies of 1 from left to right with $1, 2, \ldots, n_1$, replacing the n_2 copies of 2 from left to right with $n_1+1, n_1+2, \ldots, n_1+n_2$, and in general, replacing the n_i copies of i from left to right with $n_1 + n_2 + \cdots + n_{i-1} + 1, n_1 + n_2 + \ldots, +n_{i-1} + 2, \ldots, n_1 + n_2 + \cdots + n_{i-1} + n_i$. Since s has at most m distinct characters, the image of s has at most $m - 1$ inverse descents and is contained in $\overline{S_{2m}}$.

This definition was used in [25]. Helpful descriptions of standardization can be found in [13] and [19]. As an example, when $m = 4$, we have

$$A((4, 1, 1, 2, 2, 4, 3, 2)) = (7, 1, 2, 3, 4, 8, 6, 5).$$

Note that this map is not one-to-one, since

$$A((3,1,1,1,1,3,2,1)) = (7,1,2,3,4,8,6,5).$$

So A^{-1} is not a well-defined function. Instead, we define $B : \overline{S_{2m}} \to \{1,2,\ldots,2m\}^{2m}$ that maps σ to an element of the inverse image of σ under A by the *inverse standardization algorithm*.

Algorithm 1 Inverse Standardization Algorithm

INPUT: A permutation $\sigma \in \overline{S_{2m}}$.
OUTPUT: A string $s \in \{1,2,\ldots,m\}^{2m}$ such that $A(s) = \sigma$.

```
Set previous_index to be 0.
Set current_character to be 1.
for each integer i from 1 to 2m do
    Let j be the index of i in σ.
    if j < previous_index then
        current_character = current_character +1.
    end if
    Set s_j to be current_character.
    Set previous_index to be j.
end for
```

For example, $B((7,1,2,3,4,8,6,5)) = (3,1,1,1,1,3,2,1)$. Let $B(\sigma) = s$, where s is the result of applying the inverse standardization algorithm to σ. By the definition of this algorithm, it is clear that $A(B(\sigma)) = \sigma$ (so B is a right inverse of A). This algorithm also tells us something about strings that map to a given σ: the number of inverse descents in a permutation σ forces any string which standardizes to σ to have a certain number of distinct characters.

Proposition 4.4.10. *If σ is a permutation of $\{1,2,\ldots,2m\}$ and $s \in \{1,2,\ldots,2m\}^{2m}$ such that $A(s) = \sigma$, then the number of distinct characters in s is at least $IDes(\sigma)+1$.*

Proof. Let σ be a permutation of $\{1, 2, \ldots, 2m\}$ and let $s \in \{1, 2, \ldots, 2m\}^{2m}$ such that $A(s) = \sigma$. Suppose $\sigma^{-1}(i_1) > \sigma^{-1}(i_1 + 1)$, $\sigma^{-1}(i_2) > \sigma^{-1}(i_2 + 1)$, $\ldots$, $\sigma^{-1}(i_{\mathrm{IDes}(\sigma)}) > \sigma^{-1}(i_{\mathrm{IDes}(\sigma)} + 1)$, with $i_j < i_{j+1}$ for all j. For any j from 1 to $\mathrm{IDes}(\sigma)$, $s_{\sigma^{-1}(i_j+1)}$ must be strictly greater than $s_{\sigma^{-1}(i_j)}$. Otherwise, $\sigma^{-1}(i_j + 1)$ would be less than $\sigma^{-1}(i_j)$ since $A(s) = \sigma$. This contradicts the assumption that $\sigma^{-1}(i_j) > \sigma^{-1}(i_j + 1)$. So we have

$$s_{\sigma^{-1}(i_1)} < s_{\sigma^{-1}(i_1+1)} \leq s_{\sigma^{-1}(i_2)} < s_{\sigma^{-1}(i_2+1)} \leq \cdots < s_{\sigma^{-1}(i_{\mathrm{IDes}(\sigma)})}.$$

So s has at least $\mathrm{IDes}(\sigma) + 1$ distinct characters. □

Note that the inverse standardization algorithm only adds in a new character at each inverse descent of the permutation. Hence if a permutation σ has $m - 1$ or fewer inverse descents, then $B(\sigma) \in \{1, 2, \ldots, m\}^{2m}$. Since B is a right inverse of A, we have found a string in $\{1, 2, \ldots, m\}^{2m}$ that maps to σ under A. The inverse of this statement also holds, since a string with m characters can only map to a permutation with $m - 1$ or fewer inverse descents. So we have the following lemma:

Lemma 4.4.11. *For any permutation $\sigma \in S_{2m}$, $\sigma \in \overline{S_{2m}}$ if and only if there exists some $s \in \{1, 2, \ldots, m\}^{2m}$ such that $A(s) = \sigma$.*

By the lemma above, every permutation in $\overline{S_{2m}}$ has a preimage under A, which is given by some string in $\{1, 2, \ldots, m\}^{2m}$.

4.4.3 Incrementing Strings

We now introduce the idea of an *increment*. Given a string s of length $2m$ and an index j with $1 \leq j \leq 2m$, the idea of an increment of s at the index j is to find the

string t such that $A(s) = A(t)$, $t_j = s_j + 1$, and for every other index i, $t_i = s_i$ if possible and $t_i = s_i + 1$ otherwise.

For example, consider the string $s = (2, 1, 2, 2, 1, 1, 2, 2)$ when $m = 4$. Then

$$A(s) = (4, 1, 5, 6, 2, 3, 7, 8).$$

If we increment s at $j = 1$, then we must replace the first 2 with a 3: $(3, 1, 2, 2, 1, 1, 2, 2)$. But this does not map to the same permutation under A, so we will have to add 1 to other entries. In this situation, we will be forced to replace every 2 with a 3, but every 1 can remain. We introduce this notation: the increment of string s at index i is s inc i. So in our example,

$$(2, 1, 2, 2, 1, 1, 2, 2) \text{ inc } 1 = (3, 1, 3, 3, 1, 1, 3, 3).$$

Definition 4.4.12 (Incrementing a String)**.** Let s be a string. Then we define its increment at the jth position, s inc j, by

$$(s \text{ inc } j)_i = \begin{cases} \begin{cases} s_i & \text{if } s_i \leq s_j \\ s_i + 1 & \text{if } s_i > s_j \end{cases} & \text{if } i < j \\ \begin{cases} s_i & \text{if } s_i < s_j \\ s_i + 1 & \text{if } s_i \geq s_j \end{cases} & \text{if } i \geq j \end{cases}$$

Since we are only considering strings in $\{1, 2, \ldots, m\}^{2m}$, the increment map is undefined on s if and only if s contains the character m. If we try to increment at some

index j where $s_j = m$, the result will be $m+1$ at that index, and if we try to increment at an index j where $s_j < m$, then the m will still get replaced with an $m+1$. But if no m appears in s, then s inc j will be a valid element of $\{1, 2, \ldots, 2m\}^{2m}$ for any index j.

For our proof, we need to learn how to invert an increment. We define s inc ^{-1}j to be the string t such that t inc $j = s$, if such a t exists. In what situations would an inverse increment not exist? Obviously, we cannot subtract 1 from 1, because 0 is not an allowed character. Also, the incrementing function will result in strings that have a certain structure. This gives us a criteria to identify which strings can be inverse incremented.

Remark 4.4.13. *Suppose $t = s$ inc j. First, let i be an index less than j. We claim that $t_i \neq s_j + 1$. By the definition of the incrementing function, either $t_i = s_i$ or $t_i = s_i + 1$. If $s_i = s_j$, then the incrementing function will not change it, meaning $t_i = s_j$. If $s_i = s_j+1$, then the incrementing function will add one to it. So $t_i = s_j+2$. This means that no instance of $s_j + 1$ will occur to the left of the index j in the string t. A similar argument shows that no instance of s_j will occur to the right of j in t. We refer to these absences of certain characters as "gaps."*

So we observe that a string which has been incremented at a given index will have certain properties. The converse also holds: if all three of the conditions in Remark 4.4.13 are true of t at the index j, then there does exist a string $s \in \{1, 2, \ldots, m\}^{2m}$ such that $t = s$ inc j. This gives us the following proposition:

Proposition 4.4.14. *Let s be a string. Then s inc ^{-1}j exists if and only if*

1. $s_j \neq 1$

2. *for all* $i < j, s_i \neq s_j$

3. *for all* $i > j$, $s_i \neq s_j - 1$.

Here is the rule to compute an inverse increment:

$$(s \text{ inc } {}^{-1}j)_i = \begin{cases} \begin{cases} s_i & \text{if } s_i \leq s_j \\ s_i - 1 & \text{if } s_i > s_j \end{cases} & \text{if } i < j \\ \begin{cases} s_i & \text{if } s_i < s_j \\ s_i - 1 & \text{if } s_i \geq s_j \end{cases} & \text{if } i \geq j \end{cases}$$

Definition 4.4.15 (Minimal Strings)**.** We define a string s to be *minimal* if, for all j, s inc ^{-1}j does not exist.

We will need a couple of theorems here. Before we can state them, we present some definitions and lemmas.

Definition 4.4.16. Let s be a string. Then we define $\overline{s}$ to be the string that consists of the entries of s sorted in weakly increasing order.

Recall that n_i is defined to be the number of occurrences of i in s (with context making it clear which string n_i is referring to).

Lemma 4.4.17. *Let s be a string. If $s_j = i$ and*

$$\overline{s} = \underbrace{1, 1, \ldots, 1}_{n_1}, \ldots, \underbrace{i, i, \ldots, i}_{n_i}, \ldots, \underbrace{m, m, \ldots, m}_{n_m},$$

then

$$\overline{s \ inc \ j} = \underbrace{1, 1, \ldots, 1}_{n_1}, \ldots, \underbrace{i, \ldots, i, i+1, \ldots, i+1}_{n_i}, \underbrace{i+2, \ldots, i+2}_{n_{i+1}}, \ldots, \underbrace{m+1, \ldots, m+1}_{n_m}.$$

Proof. By the definition of the increment of a string at j, every instance of i to the right of s_j gets increased by 1, and every entry greater than i also gets increased by 1. But every instance of i to left of s_j remains unchanged. □

As a reminder, the standardization map A maps strings to permutations by tracing through the entries in weakly increasing order, from left to right. So if $A(s) = A(t)$ for strings s and t, then s and t have the same relative ordering of their elements. As an illustration, consider $s = (1, 3, 2, 3)$ and $t = (1, 5, 4, 6)$. They both map to $(1, 3, 2, 4)$ under A, and $s_i \leq s_j$ if and only if $t_i \leq t_j$ for any indices i and j.

Lemma 4.4.18. *Let s and t be strings. If $\overline{s} = \overline{t}$ and $A(s) = A(t)$, then $s = t$.*

Proof. Since $A(s) = A(t)$, when s and t are both sorted in weakly increasing order, the resulting permutation of the entries will be the same (the index of s_i in $\overline{s}$ must be the same as the index of t_i in $\overline{t}$). Since $\overline{s} = \overline{t}$, it must be that $s_i = t_i$ for all i. So $s = t$. □

Theorem 4.4.19 (Increments Commute)**.** *Let s be a string. Then $(s \ inc \ j) \ inc \ k = (s \ inc \ k) \ inc \ j$.*

Proof. Without loss of generality, suppose position j is to the left of position k. Suppose $s_j = d$ and $s_k = e$. By Lemma 4.4.17, in both $\overline{(s \text{ inc } j) \text{ inc } k}$ and $\overline{(s \text{ inc } k) \text{ inc } j}$, every character after that instance of e has 2 added to it, and every character between that instance of d but before that instance of e has 1 added to it. There-

fore, $\overline{(s \text{ inc } j) \text{ inc } k} = \overline{(s \text{ inc } k) \text{ inc } j}$. Since $A((s \text{ inc } j) \text{ inc } k) = A((s \text{ inc } k) \text{ inc } j)$, Lemma 4.4.18 implies that $(s \text{ inc } j) \text{ inc } k = (s \text{ inc } k) \text{ inc } j$. □

Theorem 4.4.20. *If s can be inverse incremented at two distinct indices j and k, then there exists some t such that t inc $j = s$ inc ^{-1}k and t inc $k = s$ inc ^{-1}j. As a result, if a string can be inverse incremented in two different locations, then it can be inverse incremented twice.*

Proof. Without loss of generality, suppose $s_j < s_k$. We begin by showing that $s \text{ inc } ^{-1}j$ can be inverse incremented at k. When we compute $s \text{ inc } ^{-1}j$, all instances of s_j are replaced by $s_j - 1$ and all instances of s_k are replaced by $s_k - 1$. Additionally, since $s_k - 1 \geq s_j$, all instances of $s_k - 1$ are replaced by $s_k - 2$. This guarantees that k is the first instance of $s_k - 1$ in $s \text{ inc } ^{-1}j$. And we know that there are no instances of $s_k - 2$ to the right of k in $s \text{ inc } ^{-1}j$ since s has no occurrences of $s_k - 1$ to the right of k. So $s \text{ inc } ^{-1}j$ can be inverse incremented at k.
Now we will show that $s \text{ inc } ^{-1}k$ can be inverse incremented at j. Since $s_j < s_k$, all instances of s_j are unchanged, so j is still the first occurrence of s_j in $s \text{ inc } ^{-1}k$. Also, since s contains no instances of $s_j - 1$ to the right of j, $s \text{ inc } ^{-1}k$ also contains no instances of $s_j - 1$ to the right of j. So $s \text{ inc } ^{-1}k$ can be inverse incremented at j.
Now will show that $(s \text{ inc } ^{-1}j) \text{ inc } ^{-1}k = (s \text{ inc } ^{-1}k) \text{ inc } ^{-1}j$. Compared to $\overline{s}$, both $\overline{(s \text{ inc } ^{-1}j) \text{ inc } ^{-1}k}$ and $\overline{(s \text{ inc } ^{-1}k) \text{ inc } ^{-1}j}$ will have 2 subtracted from every entry after s_k, and 1 subtracted from every entry after s_j but before s_k. So $\overline{(s \text{ inc } ^{-1}j) \text{ inc } ^{-1}k} = \overline{(s \text{ inc } ^{-1}k) \text{ inc } ^{-1}j}$. By Lemma 3.5, $(s \text{ inc } ^{-1}j) \text{ inc } ^{-1}k = (s \text{ inc } ^{-1}k) \text{ inc } ^{-1}j$. □

Let s be a string, and choose r different indices to increment it at. Then this resulting string has r different strings directly incrementing to it. But could there be

more strings that map to it? We have found that the answer is no, as long as the original string is minimal.

Lemma 4.4.21. *A string s is minimal if and only if $s = B(A(s))$.*

Proof. ($\Rightarrow$) Suppose s is minimal. Then for every index i, either $s_i = 1$, or s_i appears to the left of i, or $s_i - 1$ appears to the right of i. Let $c > 1$ be a character in s. Consider the leftmost location, i, of c. Since we cannot inverse increment s at that index, there must be an instance of $c-1$ to the right of i. So whenever s contains c, s also contains $c-1$: if r is the maximum character occurring in s, then s contains all of $1, 2, \ldots, r$. For all $c > 1$, the first occurrence of c in s must be in one-to-one correspondence with the inverse descents of $A(s)$. Suppose s has n_c occurrences of the character c. So the standardization map starts by putting down $1, 2, 3, \ldots, n_1$ where the 1's are, from left to right. After the last 1, it goes back to the first 2, and puts down $n_1 + 1, n_1 + 2, \ldots, n_1 + n_2$ where the 2's are, from left to right. Then it goes back and puts $n_1 + n_2 + 1$ where the first 3 is, and so on. Now suppose we are computing $B(A(s))$. We will trace through $1, 2, \ldots, n_1$ and put 1's in those spots. Then $n_1 + 1$ occurs to the left, so we put a 2 in that spot, and so on. The result is that $B(A(s)) = s$.
($\Leftarrow$). Suppose $s = B(A(s))$. By the definition of B, any entry c of $B(A(s))$ is either equal to 1, or it is the first occurrence of c (in which case it is the result of an inverse descent, so $c-1$ occurs to the right), or it is a later occurrence of c (in which case another c is to the left of it). In any situation, $B(A(s))$ cannot be inverse incremented at c. So s is a minimal string. □

Every permutation σ has a single string s such that $s = B(\sigma)$. So there is exactly one minimal string in the set $A^{-1}(\sigma)$. And we know that, if a string can be inverse

incremented in two different places, then it can be inverse incremented twice. Suppose a minimal string and some other string can both be incremented to make s. Then it would have to be possible to inverse increment the minimal string by Theorem 3.7, which is a contradiction. So if a minimal string increments to make s, no other strings can be incremented to make s. If a string is 1 increment above a minimal string, then exactly one string increments to it. Suppose s is two distinct increments above a minimal string, a. Then there exist distinct t_1 and t_2 both one increment above a and both one increment below s. Could any other strings increment to s? Since a is the only minimal string in $A^{-1}(A(s))$, any string that increments to s must ultimately inverse increment to a. Recall that incrementing a string always increases the maximal character by 1, so if a string is k increments above its minimal string, then its maximal character is k higher than the maximal character of the minimal string. Likewise, an inverse increment always decreases the maximal character by 1.

Lemma 4.4.22. *For any string s, if $j \neq k$, then s inc $j \neq s$ inc k.*

Proof. Without loss of generality, suppose $j < k$. If $s_j \leq s_k$, then $(s \text{ inc } j)_j = s_j + 1$ and $(s \text{ inc } k)_j = s_j$. If $s_j > s_k$, then $(s \text{ inc } j)_k = s_k$ and $(s \text{ inc } k)_k = s_k + 1$. □

Lemma 4.4.23. *If t is a minimal string and $s = ((t \text{ inc } j_1) \text{ inc } j_2) \ldots \text{ inc } j_r$, then s can be inverse incremented at j_i for all i from 1 to r, and at no other indices.*

Proof. We will prove this by induction on r. When $r = 1$, $s = t$ inc j_1. Since t is minimal and s is a single increment above t, only t can increment to s. If s can be inverse incremented at some other index k, then we would have t inc $j_1 = t$ inc k, contradicting the previous lemma.

For the inductive step, suppose that for some positive integer r, if t is a minimal

string and $s = ((t \text{ inc } j_1) \text{ inc } j_2) \ldots \text{ inc } j_r$, then s can be inverse incremented at j_i for all i from 1 to r, and at no other indices. So consider some index k such that $k \neq j_i$ for all i. Either $s_k = 1$, or s_k occurs to the left of k, or $s_k - 1$ occurs to the right of k. Now suppose we increment s at j_{r+1} for some index $j_{r+1} \neq k$.

Case 1: If $s_k < s_{j_{r+1}}$, then all instances of s_k and $s_k - 1$ do not get changed by incrementing s at j_{r+1}, and we still cannot inverse increment at k.

Case 2: Suppose $s_k = s_{j_{r+1}}$. If position k is to the right of position j_{r+1}, then both s_k and $s_{r_{j+1}}$ get incremented, and we cannot then inverse increment at k. Otherwise, position k is to the left of position j_{r+1}, and we will consider the three subcases separately. If some instance of s_k appears to the left of k, then both s_k and that other instance will not be incremented. So we still can't inverse increment at k. If some instance of $s_k - 1$ appears to the right of k, then neither will get incremented, so we still can't inverse increment at k. Finally, if $s_k = 1$, then this 1 will not get incremented because it is to the left of j_{r+1}. So we can still not inverse increment at k.

Case 3: Suppose $s_k > s_{j_{r+1}}$. Then s_k will have 1 added to it, making $s_k + 1$ at position k. If an instance of s_k is to the left of k, it will also have 1 added to it. If an instance of $s_k - 1$ appears to the right of k, then either $s_k - 1 > s_j$ and it gets 1 added to it, or $s_j = s_k - 1$, and s_j itself gets 1 added to it to make an instance of s_k appear to the right of position k. And s_k could not be 1, since s_k is strictly than $s_{j_{r+1}}$, which is at least 1. In any situation, s inc j_{r+1} cannot be inverse incremented at k. □

So this lemma tells us that if s is two increments above minimal a (meaning $s = (a \text{ inc } j \text{ inc } k)$), then only a inc j and a inc k increment to s. If $s = ((a \text{ inc } j_1) \text{ inc } j_2) \text{ inc } j_3$, then only three strings increment directly to s.

4.4.4 Inclusion-Exclusion Proof

Theorem 4.4.24. *For all positive integers m,*

$$2\sum_{j=1}^{m}(-1)^{m+j}\binom{2m}{m-j}j^{2m} = (2m)!$$

Proof. We will count the number of permutations σ such that $\mathrm{IDes}(\sigma) \leq m-1$. As defined earlier, this is the set $\overline{S_{2m}}$, which has size $(2m)!/2$. This is equivalent to counting the number of minimal strings we can make of length $2m$ from $\{1, 2, \ldots, m\}$. There are m^{2m} total strings. Many of those are not minimal. We know that a string is not minimal if and only if it can be inverse incremented. A string s can be inverse incremented if and only another string can be incremented to make s. A string can be incremented if and only if the string does not contain any m's. So if we take every string from $\{1, 2, \ldots, m-1\}^{2m}$ and increment them at each of their $2m$ indices, we will definitely cover every non-minimal string.

So we subtract off $(2m)(m-1)^{2m}$. But we subtracted off too much, because some strings can be inverse incremented in more than one place. A string can be inverse incremented in two places if and only if it can be inverse incremented twice (at different indices). A string s can be inverse incremented twice (at different indices) if and only if there is some other string that can be incremented at two different indices to make s. So we take all strings that can be incremented twice (those which do not use m or $m-1$) and increment them in two distinct indices, and we will get ev-

ery string who can be inverse incremented in two places. So we add on $\binom{2m}{2}(m-2)^{2m}$.

But some strings can get double-inverse-incremented to more than one string. Those are the strings who can be inverse incremented in at least 3 different indices. A string can be inverse incremented in three distinct indices if it is the triple increment of some other string that doesn't use $m, m-1, m-2$. So we can definitely get rid of all the duplicates by subtracting off $\binom{2m}{3}(m-3)^{2m}$. And this is going to keep happening, until we have to either add on or subtract off all strings that can be inverse incremented $m-1$ times. These are made from taking a string using only 1's and incrementing it in $m-1$ indices. There is only one string made from all 1's: it is $1, 1, 1, \ldots, 1$, and it is minimal. Since this string is minimal, any string made from incrementing it at $m-1$ distinct indices cannot be inverse incremented anywhere else. And since increments commute, this means the string with all 1's is the unique $(m-1)-$degree inverse increment of these strings. So we have not over-counted this time. This shows that

$$\sum_{j=1}^{m}(-1)^{m+j}\binom{2m}{m-j}j^{2m} = (2m)!/2.$$

□

4.5 Work Towards a Proof of Identity 4.2

Recall that identity 4.2 says

$$\sum_{j=1}^{m}(-1)^{m+j}\binom{2m}{m-j}j^i = 0$$

for $i = 2, 4, \ldots, 2m-2$. In this section, we demonstrate how to prove small cases of the identity (like $i = 2$, $i = 4$, etc...) first using Gosper's algorithm, and also using an inductive polynomial argument.

4.5.1 Gosper's Algorithm

Gosper's algorithm seeks to find a closed formula for the sum $\sum_{j=1}^{n} t_j$ by finding another term z_j such that $t_j = z_{j+1} - z_j$ for all indices j [23, p. 73-75]. When the algorithm succeeds, one can use a telescoping series to simplify a summation as follows:

$$\sum_{j=1}^{n} t_j = z_{n+1} - z_1.$$

We use a Mathematica implementation of Gosper's algorithm from [22]. We demonstrate the result of Gosper's algorithm on our identity when $i = 2$. Let

$$z_j = -\frac{(j-1)(j+m)}{2j(m-1)}(-1)^{j+m}\binom{2m}{m-j}j^2.$$

Observe that

$$\begin{aligned}
z_{j+1} - z_j &= -\frac{(j)(j+1+m)}{2(j+1)(m-1)}(-1)^{j+1+m}\binom{2m}{m-j-1}(j+1)^2 \\
&\quad + \frac{(j-1)(j+m)}{2j(m-1)}(-1)^{j+m}\binom{2m}{m-j}j^2 \\
&= \frac{j(j+1+m)(-1)^{m+j}(2m)!(j+1)^2}{2(j+1)(m-1)(m-j-1)!(m+j+1)!} \\
&\quad + \frac{(j-1)(j+m)(-1)^{j+m}(2m)!j^2}{2j(m-1)(m-j)!(m+j)!} \\
&= \frac{j^2(m-j)(-1)^{m+j}(2m)!(j+1)^2}{2j(j+1)(m-1)(m-j)!(m+j)!} \\
&\quad + \frac{(j+1)(j-1)(m+j)(-1)^{m+j}(2m)!j^2}{2j(j+1)(m-1)(m-j)!(m+j)!} \\
&= (-1)^{m+j}\binom{2m}{m-j}\left(\frac{j^2(m-j)(j+1)^2 + (j+1)(j-1)(m+j)j^2}{2j(j+1)(m-1)}\right) \\
&= (-1)^{m+j}\binom{2m}{m-j}\left(\frac{j(2mj-2j)}{2m-2}\right) \\
&= (-1)^{m+j}\binom{2m}{m-j}j^2,
\end{aligned}$$

which implies that

$$\sum_{j=1}^{m}(-1)^{m+j}\binom{2m}{m-j}j^2 = \sum_{j=1}^{m}(z_{j+1} - z_j) = z_{m+1} - z_1 = 0 - 0 = 0.$$

We tested this up to $i = 50$ on Mathematica, and the identity was true every time. But when we input $i = 2m-2, i = 2m-4$, or $i = 2m-2k$ for any integer k, the algorithm failed to find z_j.

4.5.2 Polynomial Induction

We now present a second possible proof technique for identity 4.2. This proof will be done by induction. We make the substitution $k = m - j$ to rewrite the identity as

$$\sum_{k=0}^{m-1} (-1)^k \binom{2m}{k} (m-k)^i = 0$$

for all $i = 2, 4, \ldots, 2m-2$. For any positive integers m and i, we define

$$P_{m,i}(x) = \sum_{k=0}^{m-1} (-1)^k \binom{2x}{k} (x-k)^i.$$

Since $P_{m,i}(m) = \sum_{k=0}^{m-1} (-1)^k \binom{2m}{k} (m-k)^i$, it suffices to show that $(x-m)$ is a factor of $P_{m,i}(x)$. We start by considering $i = 2$.

Lemma 4.5.1. *For all integers $m \geq 2$,*

$$P_{m,2}(x) = (-1)^{m-1} \frac{2^{m-1}}{(m-1)!} (x-m)(x-(m-1)) \frac{\prod_{j=0}^{m-1} (x - j/2)}{x-1}.$$

Proof. For the base case, let $m = 2$.

$$\begin{aligned} P_{2,2}(x) &= \sum_{k=0}^{1} (-1)^k \binom{2x}{k} (x-k)^2 \\ &= \binom{2x}{0}(x-0)^2 - \binom{2x}{1}(x-1)^2 \\ &= x^2 - 2x(x-1)^2 \\ &= -2x(x-2)(x-1/2) \\ &= (-1)^1 \frac{2^1}{1!}(x-2)(x-1)\frac{\prod_{j=0}^{1}(x-j/2)}{x-1}, \end{aligned}$$

as desired. For the inductive hypothesis, we assume that the proposition holds for some m. We now show that it also holds for $m+1$. By the definition of $P_{m,i}(x)$,

$$P_{m+1,2}(x) = P_{m,2}(x) + (-1)^m \binom{2x}{m}(x-m)^2.$$

Using the inductive hypothesis and simplifying, we obtain

$$\begin{aligned}
P_{m+1,2}(x) &= (-1)^{m-1}\frac{2^{m-1}}{(m-1)!}(x-m)(x-(m-1))\frac{\prod_{j=0}^{m-1}(x-j/2)}{x-1} \\
&\quad + (-1)^m\binom{2x}{m}(x-m)^2 \\
&= (-1)^{m-1}\frac{2^{m-1}}{(m-1)!}(x-m)(x-(m-1))\frac{\prod_{j=0}^{m-1}(x-j/2)}{x-1} \\
&\quad + (-1)^m\frac{(x-m)^2}{m!}\prod_{j=0}^{m-1}(2x-j) \\
&= \left((-1)^m\frac{x-m}{(2x-2)m!}\prod_{j=0}^{m-1}(2x-j)\right)(-m(x-m+1)+(x-m)(2x-2)) \\
&= \left((-1)^m\frac{x-m}{(2x-2)m!}\prod_{j=0}^{m-1}(2x-j)\right)(2x-m)(x-(m+1)) \\
&= (-1)^m\frac{2^m}{m!}(x-(m+1))(x-m)\frac{\prod_{j=0}^{m}(x-j/2)}{x-1},
\end{aligned}$$

as desired. □

Since $(x-m)$ is a factor of $P_{m,2}(x)$, this gives us an alternate proof that the identity is true in the $i=2$ case. We applied the same method to $i=4$, and similarly found a proof. We omit this, since it is nearly identical to the previous proof. The only difference is that there is more algebra.

Lemma 4.5.2. *For all integers $m \geq 3$,*

$$P_{m,4}(x) = (-1)^{m-1}\frac{2^{m-1}}{(m-1)!}(x-m)(x-(m-1))Q_4(x)\frac{\prod_{j=0}^{m-1}(x-j/2)}{(x-2)(x-1)},$$

where

$$Q_4(x) = x^3 - (2m)x^2 + (m^2 + m - 2)x - (m^2 - m).$$

We conjecture that there is a general form for these polynomials.

Conjecture 4.5.3. *For all positive even integers i and for all integers $m \geq i - 1$,*

$$P_{m,i}(x) = (-1)^{m-1} \frac{2^{m-1}}{(m-1)!}(x - m)(x - (m - 1))Q_i(x) \frac{\prod_{j=0}^{m-1}(x - j/2)}{\prod_{j=0}^{i/2}(x - j)},$$

where $Q_i(x)$ is a degree $3(i - 1)/2$ polynomial whose coefficients are polynomials of degree $i - 2$ or less in the variable m.

We have not been able to conjecture an exact formula for the coefficients of $Q_i(x)$.

Chapter 5

The Matrix Formulas

In this chapter, we seek a general expression for the matrix Toom-n formulas. As stated in Chapter 2, this is done by viewing polynomial evaluation as matrix multiplication, and finding the inverse of the matrix. While we do not find a general formula for the inverse, the process of trying to derive it leads us to a proof of the loss of precision of the matrix formulas.

5.1 A Review of Relevant Properties of Matrices

In this section, we discuss the process of inverting a matrix, Vandermonde matrices, and their determinants.

We now review how to invert a matrix, V. First, we need the matrix of minors, M. The entry $M_{a,b}$ is equal to the determinant of the submatrix of V formed by

removing the ath column and the bth row from V (with indexing starting at 1). Then we compute the cofactor matrix, C, given by $C_{a,b} = (-1)^{a+b} M_{a,b}$. The inverse of the original matrix is given by

$$V^{-1} = \frac{1}{\det(V)} C^T.$$

The most substantial step in this process is computing the matrix of minors. So if we can easily find determinants of submatrices of V, inverting is easy.

Now we discuss *Vandermonde* matrices. A matrix V is said to be Vandermonde if, for some real numbers $c_1, \ldots, c_n$,

$$V = \begin{pmatrix} 1 & c_1 & c_1^2 & \ldots & c_1^{m-1} \\ 1 & c_2 & c_2^2 & \ldots & c_2^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & c_m & c_m^2 & \ldots & c_m^{m-1} \end{pmatrix}.$$

It is well-known that the corresponding determinant is given by

$$\det(V) = \prod_{1 \leq k < j \leq m} (c_j - c_k).$$

Computing the matrix of minors of V is less straightforward, because the result of removing a column is no longer a Vandermonde matrix. However, a formula for the determinants of the submatrices is given in [24]. That work uses the notation

$$S_k(c_1, \ldots, c_m) = \sum_{1 \leq i_1 < \cdots < i_k \leq m} c_{i_1} c_{i_2} \ldots c_{i_k},$$

with

$$S_0(c_1, \ldots, c_m) = 1.$$

Suppose the matrix V_b is given by

$$V_b = \begin{pmatrix} 1 & c_1 & c_1^2 & \ldots & c_1^{b-1} & c_1^{b+1} & \ldots & c_1^{m-1} \\ 1 & c_2 & c_2^2 & \ldots & c_2^{b-1} & c_2^{b+1} & \ldots & c_2^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 1 & c_m & c_m^2 & \ldots & c_m^{b-1} & c_m^{b+1} & \ldots & c_m^{m-1} \end{pmatrix}.$$

Then its determinant is given by the expression

$$\det(V_b) = S_{m-b}(c_1, \ldots, c_m) \prod_{1 \leq k < j \leq m} (c_j - c_k).$$

Finally, recall that for any matrix M, if the ath row of M consists of a 1 in the bth entry and 0's everywhere else, then the determinant is given by

$$\det(M) = (-1)^{a+b} M_{a,b}.$$

5.2 Work on Inverting the Toom-n Matrix

In this section, we apply properties of Vandermonde matrices to work toward an expression for the inverse of the Toom-n matrix. Recall that the Toom-n matrix is

defined to be

$$V(n) = \begin{pmatrix} 1 & -(n-2) & \dots & (-(n-2))^{2n-3} & (-(n-2))^{2n-2} \\ 1 & -(n-3) & \dots & (-(n-3))^{2n-3} & (-(n-3))^{2n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & n-1 & \dots & (n-1)^{2n-3} & (n-1)^{2n-2} \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix}.$$

We note that monomial anti-symmetric functions could provide another direction of future work finding the inverse.

5.2.1 Computing the Determinant of $V(n)$

First, we note that this is nearly a Vandermonde matrix: only the bottom row is not of the correct form. But the presence of a single 1 in the midst of 0's implies that the determinant of $V(n)$ is given by

$$\det(V(n)) = (-1)^{(2n-1)+(2n-1)} V(n)_{2n-1,2n-1}.$$

Let $sV(n)$ denote the submatrix formed by removing the $(2n-1)$th row and the $(2n-1)$th column. This submatrix is a Vandermonde matrix, where $m = 2n-2$ and

$c_i = i - (n-1)$. So

$$\begin{aligned}
\det(V(n)) &= \det(sV(n)) \\
&= \prod_{1 \le k < j \le 2n-2} (c_j - c_k) \\
&= \prod_{1 \le k < j \le 2n-2} (j - (n-1) - (k - (n-1))) \\
&= \prod_{j=1}^{2n-2} \prod_{k=1}^{j-1} (j-k) \\
&= \prod_{j=1}^{2n-2} (j-1)! \\
&= 0! \times 1! \times \cdots \times (2n-3)!
\end{aligned}$$

This product of factorials is called a superfactorial, and it is denoted by $(2n-3)\$$. For reference, the first few values are

n	$n\$$
0	1
1	1
2	2
3	12
4	288
5	24560
6	24883200

So the overall determinant of $V(n)$, which we have already stated is 1 times the

determinant of $sV(n)$, is

$$\det(V(n)) = (2n-3)\$.$$

5.2.2 Determinants of Submatrices of $V(n)$

In order to invert $V(n)$, we need the determinant of the submatrix of $V(n)$ with the ath row and the bth column removed, for all $a, b \in \{1, 2, \ldots, 2n-2\}$. One special case is

$$V(n)_{2n-2,2n-2} = (2n-3)\$,$$

since removing the last column and the last row leaves $sV(n)$. The other special case is that when we remove the last column, and some row other than the bottom row. In this case,

$$V(n)_{a,2n-2} = 0$$

since the bottom row would then be entirely 0's.

For any value of $b \neq 2n-2$, removing the bth column will cause the new submatrix to no longer be a Vandermonde matrix since a power will be absent. So we need something different to compute $V(n)_{a,b}$ when $b \neq 2n-2$. There is a special case: if $a = 2n-2$, then the last column will still contribute to the determinant because the bottom row will not be there. For now, let us assume $a \neq 2n-2$ and $b \neq 2n-2$. In this case, the bottom row will still be all 0's with a 1 at the right side, implying that

$$V(n)_{a,b} = sV(n)_{a,b}.$$

So let $c_1, c_2, \ldots, c_{2n-3} = -(n-2), -(n-3), \ldots, a-n, a-n+2, \ldots, n-1$. Then

$$\begin{aligned} V(n)_{a,b} &= sV(n)_{a,b} \\ &= S_{2n-2-b}(c_1, \ldots, c_{2n-3}) \prod_{1 \le k < j \le 2n-3} (c_j - c_k). \end{aligned}$$

Note that subscript on the S is $2n-2-b$ because the $(b-1)$th power is absent from the rows of the matrix, and $2n-3-(b-1) = 2n-2-b$.

We now address the special case when $a = 2n-2$ and $b \neq 2n-2$. In this situation, $c_1, c_2, \ldots, c_{2n-3} = -(n-2), -(n-3), \ldots, n-1$. None of the rows are omitted. Also, compared to the expression for $V(n)_{a,b}$ when both a and b were not equal to $2n-2$, the upper limit of the product indices is $2n-2$ instead of $2n-3$ because the last column is not omitted. So

$$\begin{aligned} V(n)_{2n-2,b} &= S_{2n-2-b}(c_1, \ldots, c_{2n-3}) \prod_{1 \le k < j \le 2n-2} (c_j - c_k) \\ &= S_{2n-2-b}(c_1, \ldots, c_{2n-3}) \det(V(n)) \\ &= S_{2n-2-b}(c_1, \ldots, c_{2n-3})(2n-3)\$. \end{aligned} \tag{5.1}$$

We need to find a general expression for sum and product components of those formulas. For brevity, we introduce the notation

$$\sum(a,b) = S_{2n-2-b}(c_1, \ldots, c_{2n-3})$$

and

$$\prod(a) = \prod_{1 \le k < j \le 2n-3} (c_j - c_k),$$

where the c_i's are determined by the context. Both expressions depend on a, because a determines which c_i's are included. The next section will discuss our progress on both.

5.3 Sums and Products

In this section, we derive a general formula for $\prod(a)$ and provide an explanation as to why $\sum(a, b)$ is more challenging.

5.3.1 A General Expression for the Product

As some helpful notation, we let $a^\star = a - (n-1)$. This converts from the row index to the number being exponentiated in that row of $V(n)$. The values for $\prod(a)$, for $n = 4$, are shown in the following table.

a	$a^\star$	$\prod(a)$
1	-2	288
2	-1	1440
3	0	2880
4	1	2880
5	2	1440
6	3	288

Observe that these values are all divisible by 288, which is a superfactorial number.

Furthermore, dividing them by 288 yields 1, 5, 10, 10, 5, 1, which is a row of Pascal's triangle. This leads us to the following result.

Proposition 5.3.1. *Let $n > 2$ be a positive integer, let $a \in \{1, 2, \ldots, 2n-2\}$, and let $a^\star = a - (n-1)$. Let $c_1, \ldots, c_{2n-3} = -(n-2), \ldots, a^\star - 1, a^\star + 1, \ldots, n-1$. Then*

$$\prod_{1 \le k < j \le 2n-3} (c_j - c_k) = \binom{2n-3}{a-1} (2n-4)\$.$$

Proof. Let $d_1, d_2, \ldots, d_{2n-2} = -(n-2), -(n-3), \ldots, n-1$. From the calculation of $\det(V(n))$, we know that

$$\prod_{1 \le k < j \le 2n-2} (d_j - d_k) = (2n-3)\$.$$

Since the list of c_i's is equal to the list of d_i's with $a^\star$ removed, we can divide $(2n-3)\$$ by the contributions from $a^\star$:

$$\prod_{1 \le k < j \le 2n-3} (c_j - c_k) = \frac{(2n-3)\$}{\left(\prod_{c=-(n-2)}^{(a^\star-1)} (a^\star - c)\right)\left(\prod_{c=a^\star+1}^{n-1} (c - a^\star)\right)}.$$

These two products involving $a^\star$ can be simplified to

$$\prod_{c=-(n-2)}^{(a^\star-1)} (a^\star - c) = (a^\star + n - 2)!$$

and

$$\prod_{c=a^\star+1}^{n-1} (c - a^\star) = (n - 1 - a^\star)!.$$

Substituting these in and rearranging, we obtain

$$\begin{aligned}\prod_{1\le k<j\le 2n-3}(c_j-c_k) &= \frac{(2n-3)\$}{(a^\star+n-2)!(n-1-a^\star)!}\\ &= \frac{(2n-4)\$(2n-3)!}{(a^\star+n-2)!(n-1-a^\star)!}\\ &= (2n-4)\$\binom{2n-3}{a^\star+n-2}\\ &= (2n-4)\$\binom{2n-3}{a-1}.\end{aligned}$$

□

5.3.2 Work Towards a General Expression for the Sum

We have not found a general formula for $\sum(a,b)$. Recall that

$$\sum(a,b) = S_{2n-2-b}(c_1,\ldots,c_{2n-3}) = \sum_{1\le i_1<\cdots<i_k\le 2n-2-b}(c_{i_1}c_{i_2}\ldots c_{i_k}),$$

where $c_1,\ldots,c_{2n-3} = -(n-2),\ldots,a^\star-1,a^\star+1,\ldots,n-1$. The following table contains $\sum(a,b)$ in the (a,b)th entry, for $n=4$:

$a^\star \backslash b$	1	2	3	4	5	6
-2	0	-6	-5	5	5	1
-1	0	-12	-16	-1	4	1
0	12	4	-15	-5	3	1
1	0	12	-8	-7	2	1
2	0	6	-1	-7	1	1
3	0	4	0	-5	0	1

We also present the table for $n = 5$:

$a^\star \backslash b$	1	2	3	4	5	6	7	8
-3	0	48	28	-56	-35	7	7	1
-2	0	72	54	-71	-60	-2	6	1
-1	0	144	180	-16	-65	-9	5	1
0	-144	-36	196	49	-56	-14	4	1
1	0	-144	108	88	-39	-17	3	1
2	0	-72	18	89	-20	-18	2	1
3	0	-48	4	64	-5	-17	1	1
4	0	-36	0	49	0	-14	0	1

There are some visible patterns, but we do not have a formula to compute $\sum(a, b)$ in general. Without this, we cannot discover a general expression for the inverse of the Toom-n matrix. However, we make two observations about the values of $\sum(a, b)$ that will be sufficient to prove the loss of precision of the matrix formulas:

- Every value of $\sum(a, b)$ is an integer, since it is a sum of products of integers.

- For all n and for all a, $\sum(a, 2n-2) = 1$ since $S_0(c_1, \ldots, c_{2n-3}) = 1$ by definition.

5.4 Loss of Precision

5.4.1 Preliminary Discussion about Losing Precision

In this section, we use the progress of the previous section to prove the loss of precision of the matrix formulas. Recall that for an integer x, $v_2(x)$ denotes the 2-adic valuation of x. This is also equal to the largest power of 2 dividing x. We also introduce some notation: for any integer $n \geq 2$, $L(n)$ denotes the loss of precision of the matrix Toom-n formulas. Recall that a bit of precision is lost when a division by 2 is performed. This effect has the potential to stack if an expression that has already lost bits is then divided by another power of 2. The loss of precision is easier to trace in the matrix formulas (as compared to the natural and efficient formulas) because all of the coefficients (r_0, $\ldots$, r_{2n-2}) are calculated explicitly in terms of the $r(a)$ values. To illustrate this point, compare the equation for r_2 in the Toom-4 matrix formulas:

$$r_2 = \frac{1}{4!}\Big(- r(-2) + 16r(-1) - 30r(0) + 16r(1) - r(2) + 96r(\infty)\Big)$$

to that of the Toom-4 natural formulas:

$$r_2 = \frac{1}{2!}\Big((r(1) + r(-1)) - 2r_0 - 2r_4 - 2r_6\Big).$$

The latter depends on r_0, r_4, and r_6, while the former does not. This is significant, because the computation of r_4 has already lost precision.

Because the coefficients r_i and r_j are computed independently of each other in the matrix formulas, the loss of precision is equal to the largest power of 2 that appears in the denominator of any entry in the Toom-n matrix. For example, the Toom-4 matrix formulas lose 3 bits of precision because the largest power of 2 that appears in a denominator 2^3. This is caused by the denominators 4! and 5!, which factor as $(2^3)(3)$ and $(2^3)(15)$, respectively.

We now describe the entries of the Toom-n matrix. The transposing and the multiplication by $(-1)^{a+b}$ have no effect on the largest power of 2 appearing in a denominator, so we will disregard those steps. Therefore, we only need to look at $M_{a,b}/\det(V(n))$ as a and b range from 1 to $2n-2$. There are four cases to consider.

If $a = b = 2n-2$, then

$$\frac{M_{2n-2,2n-2}}{(2n-3)\$} = \frac{(2n-3)\$}{(2n-3)\$} = 1.$$

If $a \neq 2n-2$ and $b = 2n-2$, then

$$\frac{M_{a,2n-2}}{(2n-3)\$} = \frac{0}{(2n-3)\$} = 0.$$

If $a \neq 2n-2$ and $b \neq 2n-2$, then

$$\begin{aligned}\frac{M_{a,b}}{(2n-3)\$} &= \frac{\prod(a)\sum(a,b)}{(2n-3)\$} \\ &= \frac{\binom{2n-3}{a-1}(2n-4)\$\sum(a,b)}{(2n-3)\$} \\ &= \frac{\binom{2n-3}{a-1}\sum(a,b)}{(2n-3)!} \\ &= \frac{\sum(a,b)}{(a-1)!(2n-2-a)!}.\end{aligned}$$

If $a = 2n-2$ and $b \neq 2n-2$, then by equation 5.1,

$$\begin{aligned}\frac{M_{a,b}}{(2n-3)\$} &= \frac{(2n-3)\$\sum(a,b)}{(2n-3)\$} \\ &= \sum(a,b).\end{aligned}$$

Since the first two cases have no denominator and $\sum(a,b)$ is always an integer, the loss of precision is less than or equal to the largest power of 2 dividing $(a-1)!(2n-2-a)!$, as a ranges from 1 to $2n-2$.

Proposition 5.4.1. $L(n) \leq \max\left\{v_2((a-1)!(2n-2-a)!)\right\}_{a=1}^{2n-2}$.

So we seek the maximum value of $v_2((a-1)!(2n-2-a)!)$ as a ranges from 1 to $2n-2$.

5.4.2 Proving the Loss of Precision

In this section, we prove that the matrix Toom-n formulas lose $v_2((2n-4)!)$ bits of precision.

Proposition 5.4.2. *Let $n > 2$. Then* $\max\left\{v_2((a-1)!(2n-2-a)!)\right\}_{a=1}^{2n-2} = v_2((2n-4)!)$.

Proof. Let $f(a) = v_2((a-1)!(2n-2-a)!)$. This function is symmetric about $a = n - 1/2$, since

$$\begin{aligned} f(n-1/2-x) &= v_2((n-1/2-x-1)!(2n-2-(n-1/2-x))!) \\ &= v_2((n-2+1/2+x)!(n-1/2-x-1)!) \\ &= v_2((n-1/2+x-1)!(n-2+1/2-x)!) \\ &= v_2((n-1/2+x-1)!(2n-2-(n-1/2+x))!) \\ &= f(n-1/2+x). \end{aligned}$$

So we can consider only $a = 1, 2, \ldots, n-1$. We now demonstrate that $f(1) \geq f(a)$ for all a in that range. We present three helpful facts:

- $f(1) = v_2((0)!(2n-3)!) = v_2((2n-3)!)$.
- Suppose $i, j, k \in \mathbb{Z}$ such that $i + j = k$. Then $j = k - i$. So $i!j! = i!(k-i)!$, and $i!(k-i)! \mid k!$ since binomial coefficients are always integers.
- Suppose $x, y \in \mathbb{Z}$ such that $x \mid y$. Then $v_2(x) \leq v_2(y)$.

For any $a \in 1, 2, \ldots, n-1$, $(a-1) + (2n-2-a) = 2n-3$. This implies that

$$v_2((a-1)!(2n-2-a)!) \leq v_2((2n-3)!).$$

So $f(a) \leq f(1)$ for $a = 1, 2, \ldots, n-1$. Thus,

$$\max\left\{v_2((b+n-2)!(n-b-1)!)\right\}_{a=1}^{2n-2} = f(1) = v_2((2n-3)!) = v_2((2n-4)!),$$

as desired. □

The next result follows.

Corollary 5.4.3. *Let $n > 2$ and let L denote the loss of precision of the Toom-n matrix formulas. Then $L(n) \leq v_2((2n-4)!)$.*

So we know that the largest power of 2 that a denominator could have is $v_2((2n-4)!)$. This will occur for certain when $a = 1$ and $a = 2n-2$, since the denominator is $(2n-3)!$ and $v_2((2n-3)!) = v_2((2n-4)!)$. The loss of precision could be less, if the numerator $\sum(a, b)$ is even whenever $a = 1$ or $a = 2n-2$. However, it will always be the case that some corresponding value of $\sum(a, b)$ is odd.

Proposition 5.4.4. *Let $n > 2$ and let L denote the loss of precision of the Toom-n matrix formulas. Then $L(n) \geq v_2((2n-4)!)$.*

Proof. By the definition of $S_k(c_1, \ldots, c_m)$, $S_0 = 1$. Therefore, $\sum(2n-2, 0) = 1$ for all n. By the previous result, the entry in the inverse matrix corresponding to that will be 1 divided by $(2n-3)!$. The loss of precision due to that term will be $v_2((2n-4)!)$. □

This proves the loss of precision.

Theorem 5.4.5. *Let $n > 2$ and let L denote the loss of precision of the Toom-n matrix formulas. Then $L(n) = v_2((2n-4)!)$.*

Chapter 6

Loss of Precision

In this chapter, we conjecture the loss of precision of the natural and efficient formulas based on data from simulations. We also discuss future approaches to tracking the loss of precision.

6.1 Simulating Loss of Precision

We used a Python program to simulate the loss of precision for each of the interpolation methods. The program performs multiplication on many random polynomials in $(\mathbb{Z}/2^m\mathbb{Z})[x]$, and records the largest number of bits that were lost. The results are shown in the following table.

n	Matrix	Natural	Efficient
3	1	1	1
4	3	3	4
5	4	4	4
6	7	7	8
7	8	8	9
8	10	10	11
9	11	11	11
10	15	15	16
11	16	16	17
12	18	18	19
13	19	19	20
14	22	22	23
15	23	23	24

The first observation we make is that the data for the matrix formulas agrees with the result of the previous section: $L(n) = v_2((2n-4)!)$. Next, we observe that the natural formulas appear to lose the same number of bits as the matrix formulas.

Conjecture 6.1.1. *For all $n \geq 3$, the loss of precision of the natural formulas is equal to the loss of precision of the matrix formulas.*

Finally, we observe that the efficient formulas usually lose one extra bit of precision compared to the matrix formulas, but they have the same loss for $n = 3, 5, 9$. We do not have a conjecture for exactly when this happens.

6.2 Steps Toward Proving Loss of Precision

In this section, we present methods that have the potential to be used to prove the loss of precision for any set of Toom interpolation formulas.

6.2.1 Analyzing Loss of Precision in the Natural Formulas

In this section, we explain why determining the loss of precision of the natural formulas is much harder than it was for the matrix formulas. Suppose, for example, that we want to compute the loss of precision of the natural Toom-4 formulas. Recall that

these formulas are

$$
\begin{aligned}
r_0 &= r(0) \\
r_6 &= r(\infty) \\
r_4 &= \frac{1}{4!}(-4(r(1)+r(-1)) + (r(2)+r(-2)) + 6r_0 - 120r_6) \\
r_2 &= \frac{1}{2!}((r(1)+r(-1)) - 2r_0 - 2r_4 - 2r_6) \\
r_5 &= \frac{1}{5!}(5r(1) - 4r(2) + r(3) - 2r_0 + 2r_2 - 22r_4 - 478r_6) \\
r_3 &= \frac{1}{3!}(-2r(1) + r(2) + r_0 - 2r_2 - 14r_4 - 30r_5 - 62r_6) \\
r_1 &= r(1) - r_0 - r_2 - r_3 - r_4 - r_5 - r_6.
\end{aligned}
$$

If we assume that the loss of precision always accumulates as more divisions by 2 are performed on expressions that have already lost precision, we will overestimate the true loss. It is true that r_4 loses 3 bits of precision since the term $r(2)/24$ has 2^3 in the denominator. When we say that an expression, like r_4 loses 3 bits of precision, we mean that if we multiplied the same polynomials with Toom-4 in $\mathbb{Z}[x]$, then both versions of r_4 are congruent to each other modulo 2^{m-3}, and no higher power of 2. And r_2 also loses 3 bits, since the equation for r_2 contains the term $2r_4/2$. But one might expect r_5 to lose a total of 5 bits, because it contains $(2r_2 - 22r_4)/5! = (r_2 - 11r_4)/60$. However, it turns out that the expression $2r_2 - 22r_4$ regains some of the precision that r_2 and r_4 had lost. In total, the natural Toom-4 formulas do lose 3 bits of precision. This can be seen by carefully tracing through each step and representing the variables entirely in terms of $r(-2)$, $r(-1)$, ..., $r(3)$, $r(\infty)$. This process is extremely tedious,

and does not provide a general method for proving the loss of precision for all n.

6.2.2 Tracking p-adic Precision

A possible method of proving the loss of precision due to a set of interpolation formulas is by applying the results of Caruso, Roe, and Vaccon [4]. This work describes how to propagate p-adic precision though computations. It can be applied to functions that fit certain conditions. Since the set of Toom-n formulas is a linear function from $(\mathbb{Z}/2^m\mathbb{Z})^{2n-1}$ to $(\mathbb{Z}/2^m\mathbb{Z})^{2n-1}$, the results could be applied to determine the loss of precision. Caruso, Roe, and Vaccon also created a Sage package to track p-adic precision in practice [5]. Further work applying their results has the potential to prove the loss of precision for any given set of Toom-n interpolation formulas.

Chapter 7

Comparing Implementations of Toom-Cook in Practice

In this chapter, we start by describing how the overhead costs of splitting and interpolation stages of Toom-n multiplication outweigh the asymptotic complexity for small degrees. Then, we show how different n-values can be combined, and present graphs comparing different interpolation and decomposition strategies.

7.1 Thresholds

When first introduced, it was mentioned that the Toom-Cook algorithm is recursive. When multiplying with Toom-n, one could split the original polynomials into n parts, and to multiply those smaller polynomials, recursively split them into n parts. For large degrees, many rounds of Toom-n may need to be performed. In this chapter, we use Toom-n to refer to a single iteration of Toom-n, and Toom-$\underbrace{n - \cdots - n}_{k}$ to refer to k recursive iterations.

When the degree is small enough, schoolbook multiplication performs better than Toom-Cook. This is due to the extra work of splitting and interpolating the polynomials. Although these steps do not affect the complexity, they significantly increase the runtime for smaller degrees in practice. For any n, we can plot the average CPU time to determine the degree where a single iteration of Toom-n becomes faster than schoolbook multiplication. We refer to this intersection point as the *threshold* of Toom-n versus schoolbook. This has also been called a "crossover point" in [29, pp. 221-247]. First, we determine the threshold between Toom-2 and the schoolbook algorithm.

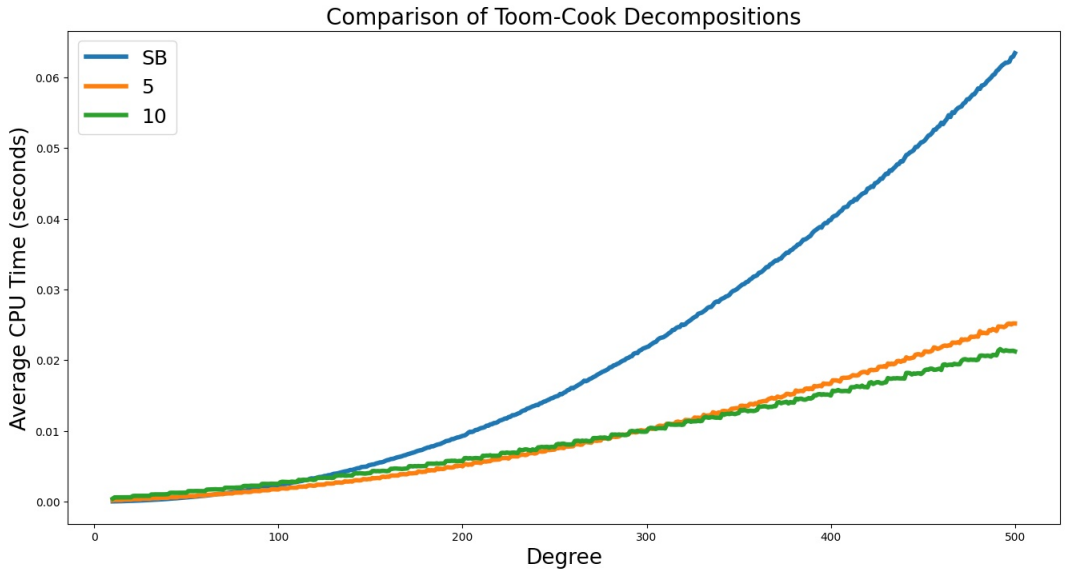
The first observation we make is that the graph of Toom-2 is not smooth. This is because, if the degree is even (meaning the number of terms is odd), then our implementation pads the polynomials with an extra 0 to make the number of terms even. In general, the graph of Toom-n will approximate a step function that jumps up after every multiple of n. Next, we see that the threshold is near to 45. This

means that schoolbook is faster when the degree is less than 45, and Toom-2 is faster when the degree is greater than 45. The exact value of the threshold depends on the implementation. For highly-optimized Toom-2 implementations, this threshold can be closer to 30 [9] or even 20 [15].

We can make an analogous plot for Toom-3.

This shows that the threshold for Toom-3 is close to 50, a little higher than it was for Toom-2.

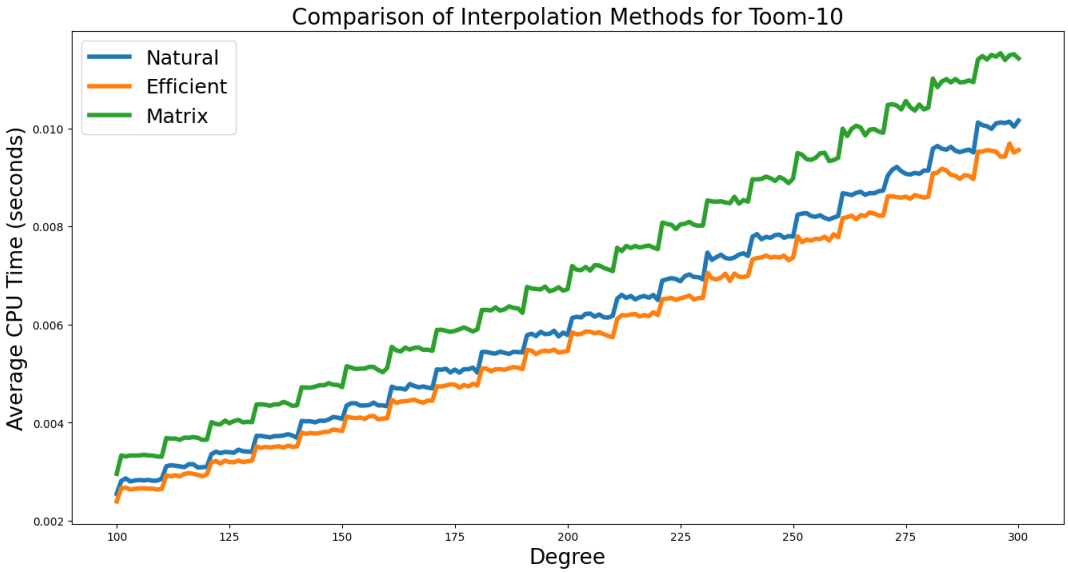
We can also compare thresholds between Toom-n_1 and Toom-n_2 for distinct n_1 and n_2. If $n_1 < n_2$, then Toom-n_2 is asymptotically faster than Toom-n_1, but also has a higher overhead for small degrees. The following graph compares schoolbook, Toom-5, and Toom-10.



This shows that the threshold between Toom-5 and Toom-10 is around 300.

7.2 Comparing Interpolation Methods

We have claimed that the efficient interpolation formulas ought to be faster than the natural and matrix formulas. In addition, we expect the matrix formulas to be the slowest, because they do not save any results of computations to avoid duplicating calculations. We have a graph to support these expectations.



This shows that the matrix formulas are significantly slower than the other two, and that the efficient formulas are a tiny bit faster than the natural formulas. We made many graphs like this for different values of n and different ranges of degrees. They all look similar to this one.

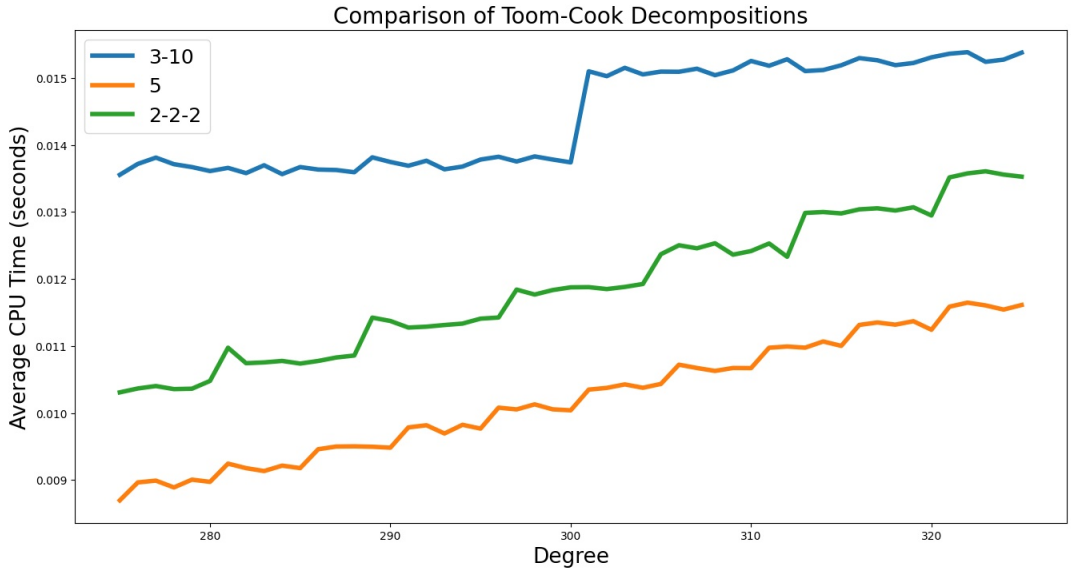
7.3 Choosing Decompositions in Theory

7.3.1 Introducing Decompositions

When using the Toom-Cook algorithm recursively, there is no requirement that n has to remain the same. One could start by splitting the original polynomials with Toom-3, then recursively split the smaller polynomials with Toom-10, and then multiply these polynomials using schoolbook. We would call this Toom-3-10. The term *decomposition* refers to the order which different values of n are recursively applied. For example, Toom-3-10, Toom-5, and Toom-2-2-2 are different decompositions that

could be used to multiply polynomials.

The most important property of a decomposition is how small it makes the polynomials that are handled by schoolbook multiplication. In general, Toom-n_1-n_2-...-n_k splits each original degree d polynomial into n_1 parts, and then splits those into n_2 parts, and so on. After the final split, the small polynomials to be multiplied by schoolbook how have degree $d/(n_1 n_2 \dots n_k)$. For example, suppose we are multiplying degree 300 polynomials. Then Toom-3-10 would create degree $300/(3 \times 10) = 10$ polynomials, Toom-5 would create degree $300/5 = 60$ polynomials, and Toom-2-2-2 would create degree $300/(2 \times 2 \times 2) = 38$ polynomials (rounding up). This is evidence that Toom-3-10 might not be the most efficient of those three algorithms, since 10 not is close to the threshold where schoolbook stops being the fastest (45). Toom-3-10 would also not be optimal because it applies Toom-10 to degree 100 polynomials (recall that Toom-10 is slower than Toom-5 until around degree 300). This can be seen in the following graph.

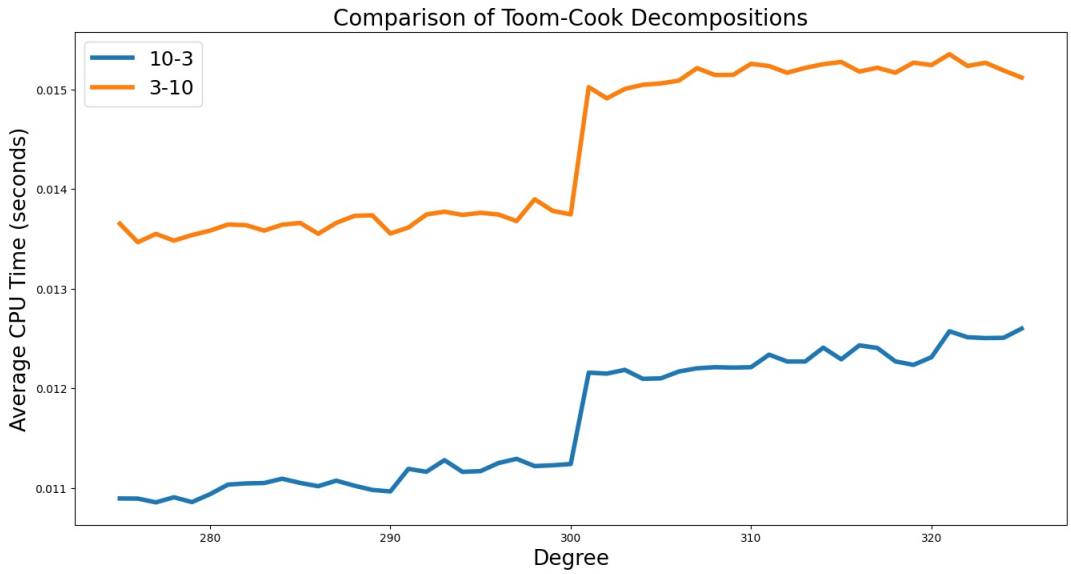


In general, we say the *division* of the decomposition given by Toom-n_1-n_2-...-n_k is equal to $n_1 \times n_2 \times \cdots \times n_k$.

For any degree d, there should be a decomposition that performs best at that degree. Let t denote the threshold where Toom-2 becomes faster than schoolbook for the given implementation. The most likely candidates will be those decompositions whose divisions are close to d/t, since this will ensure that schoolbook is not being applied to degrees that are too large, and that Toom-Cook is not being applied to degrees that are too small.

7.3.2 Ordering of Decompositions

In the last section, we considered Toom-3-10. Toom-10-3 has the same division (because both decompositions result in degree 10 polynomials being multiplied by schoolbook), but the following graph shows that it is faster than Toom-3-10.



This is because more splitting (higher n) is better asymptotically but has a higher overhead cost. It is always faster to have the decomposition in decreasing order. From now on, we only consider decompositions Toom-n_1-n_2-...-n_k where $n_1 \geq n_2 \geq \ldots n_k$.

7.4 Comparing Decompositions in Practice

In this section, we use the NTRU cryptosystem as an example to determine the ideal decomposition for a given degree. All interpolation will be done using the natural formulas.

7.4.1 Precision of Decompositions

Since multiplication is being done in $(\mathbb{Z}/2^m\mathbb{Z})[x]$, we have to keep track of the loss of precision. We make two major assumptions:

- Our conjectured loss of precision from Chapter 6 is correct.
- Loss of precision is additive in Toom decompositions.

Based on many simulations, we have much confidence in our conjectured loss of precision. Also based on simulations and intuition, we believe that the precision adds up. For example, performing Toom-6-4 loses $7 + 3 = 10$ bits of precision. This makes sense, because the smaller polynomials that Toom-4 multiplies lose 3 bits of precision. Then Toom-6 interpolates based on this lower-precision polynomials, and loses

7 more bits. We do not have a proof of this, but we note that [15] agrees.

Given some integer m such that we are working over $(\mathbb{Z}/2^m\mathbb{Z})[x]$ and some larger integer M based on the data type used to store coefficients (like $M = 16$ for 16-bit ints), we can only use decompositions that lose $M - m$ or fewer bits of precision. This means that the fastest decomposition for a certain degree in $\mathbb{Z}[x]$ might not give correct results.

7.4.2 Decompositions for NTRU

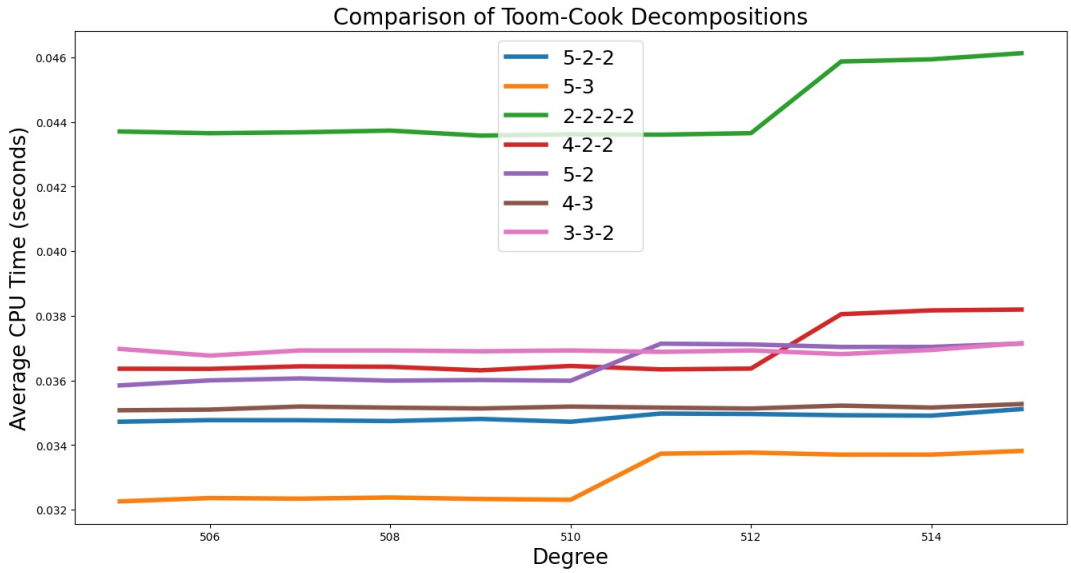
We consider three specific NTRU parameter sets [6], listed as (N, m) pairs (where $N - 1$ is the polynomial degree and $q = 2^m$ determines the coefficient ring $\mathbb{Z}/q\mathbb{Z}$):

1. $(509, 11)$.
2. $(677, 11)$.
3. $(821, 12)$.

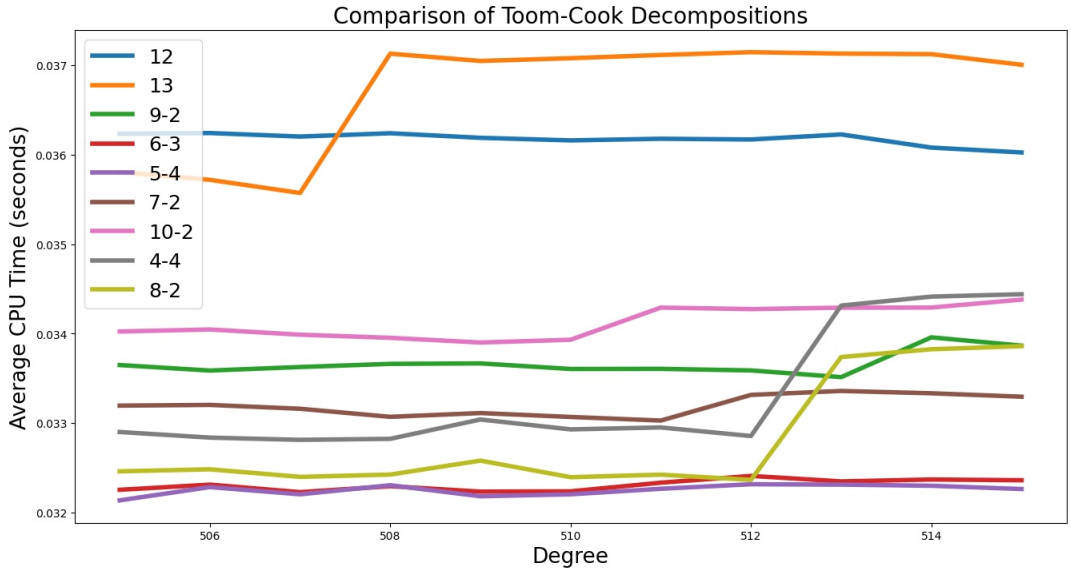
As noted in [15], the current NTRU implementations that use 16-bit ints are restricted to using combinations of Toom-2, Toom-3, and Toom-4 due to the loss of precision. We also note that Toom-5 can be used. We seek to determine how much multiplication could be sped up if we used 32-bit ints, thus allowing higher-order Toom decompositions.

For parameter set 1, we can afford to lose up 5 bits of precision with 16-bit ints,

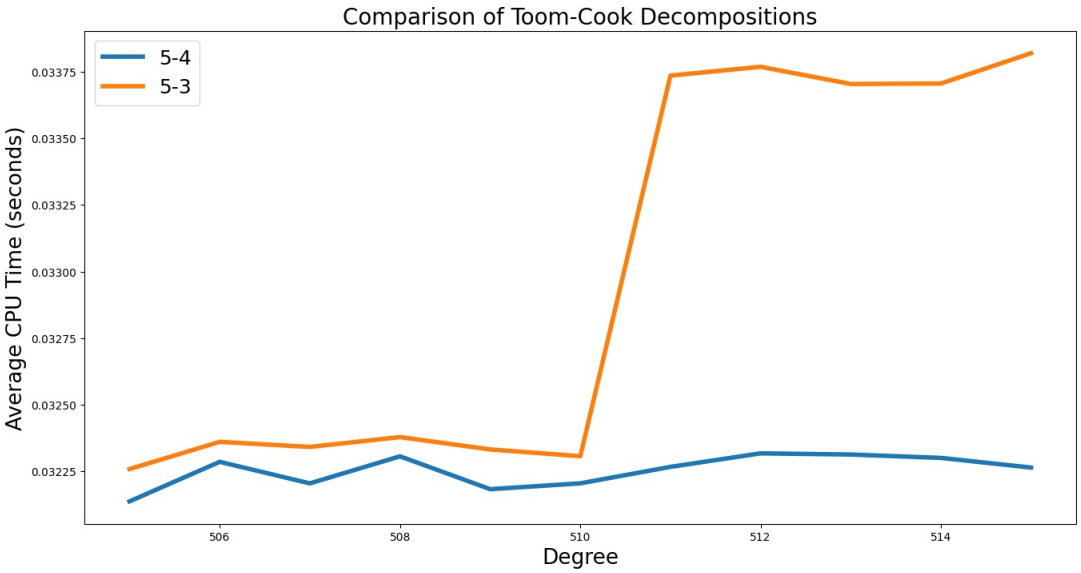
and 21 bits of precision with 32-bit ints. We begin with a plot of the most reasonable decompositions that give a correct result with 16-bit ints:



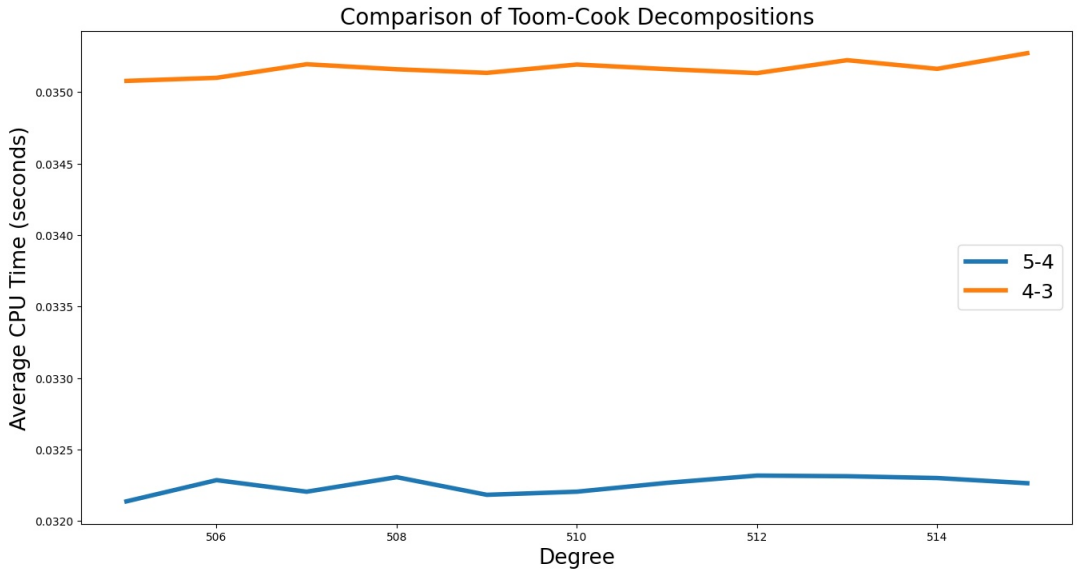
This shows that Toom-5-3 (which loses exactly 5 bits of precision) is the fastest for degree 508 polynomials. For the decompositions that give a correct result with 32-bit ints, we have



In this case, Toom-5-4 (which loses 7 bits of precision) is the fastest for degree 508 polynomials. Comparing these directly yields the following plot:



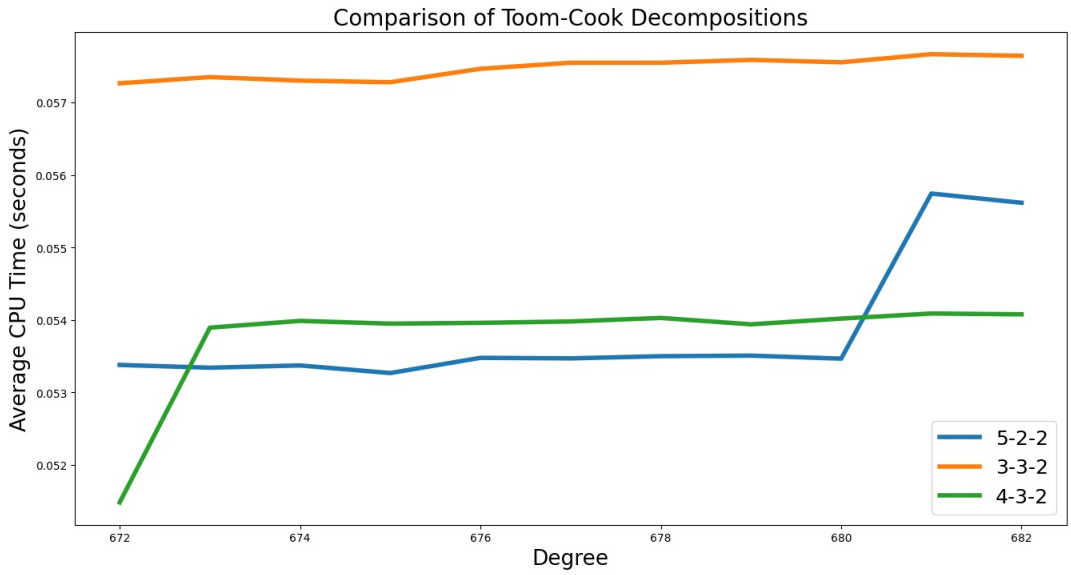
The saving in this case is very minimal. However, current implementations only make use of Toom-4 and Toom-3. Comparing Toom-4-3 to Toom-5-4 is more dramatic:



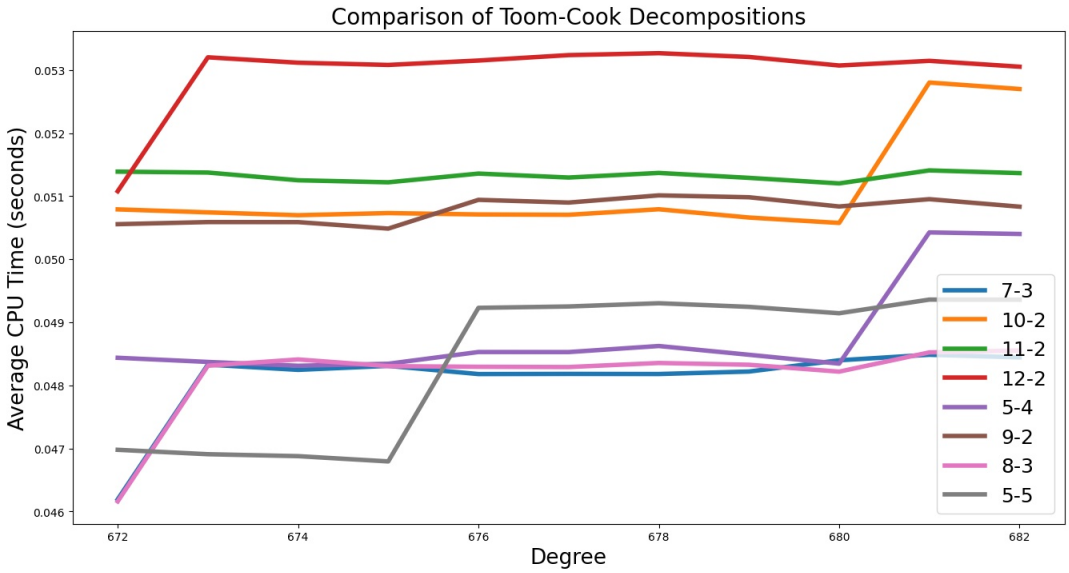
When the degree of the polynomial is 508, the running time of Toom-4-3 is about 1.1

times the running time Toom-5-4.

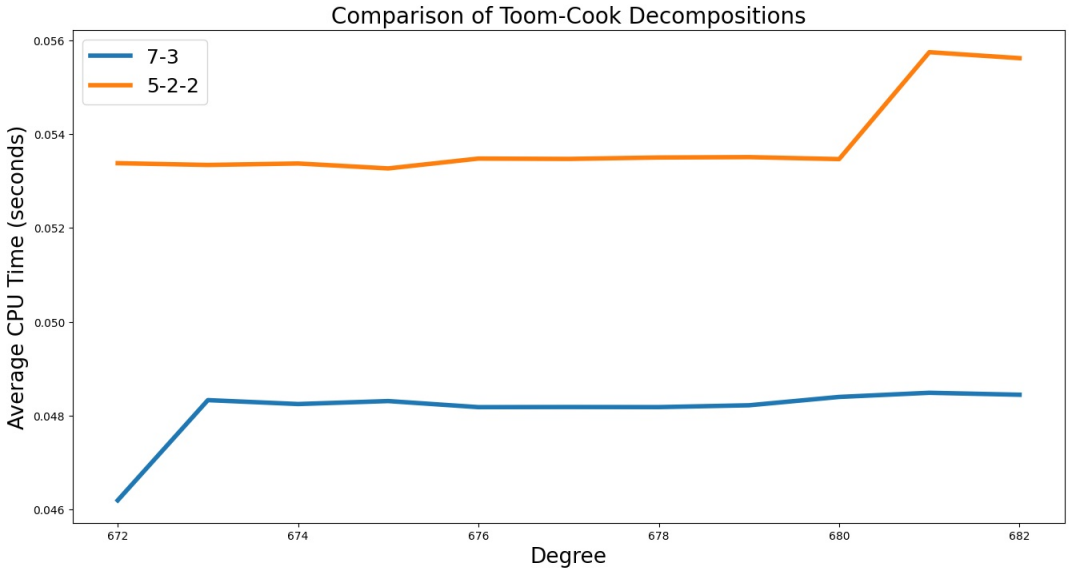
For parameter set 2, we can also afford to lose up 5 bits of precision with 16-bit ints, and 21 bits of precision with 32-bit ints. We begin with a plot of the most reasonable decompositions that give a correct result with 16-bit ints:



This shows that Toom-5-2-2 (which loses 4 bits of precision) is the fastest for degree 676 polynomials. For the decompositions that give a correct result with 32-bit ints, we have

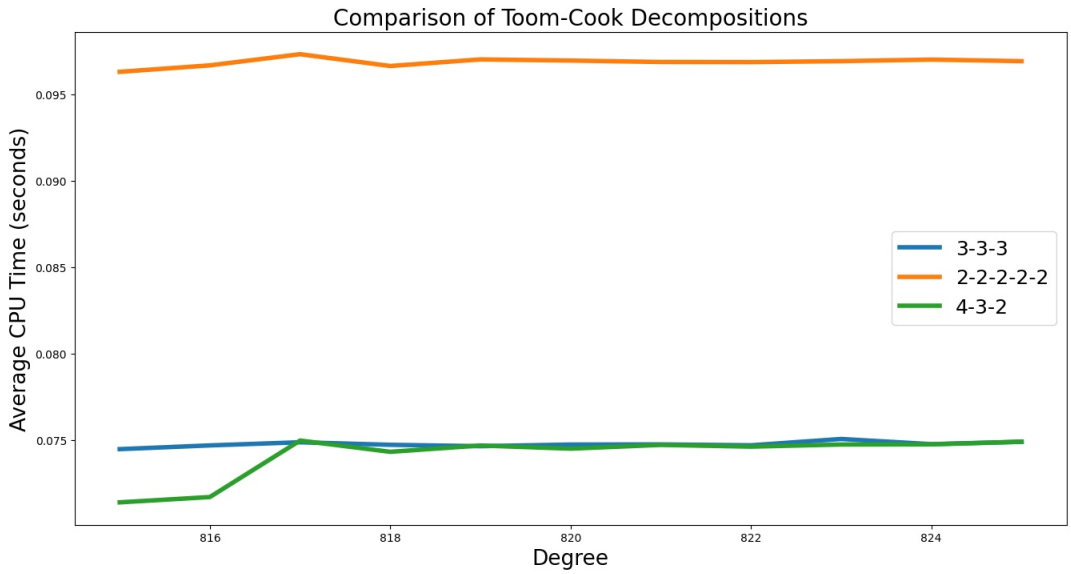


In this case, Toom-7-3 (which loses 9 bits of precision) is the fastest for degree 676 polynomials. Comparing these directly yields the following plot:

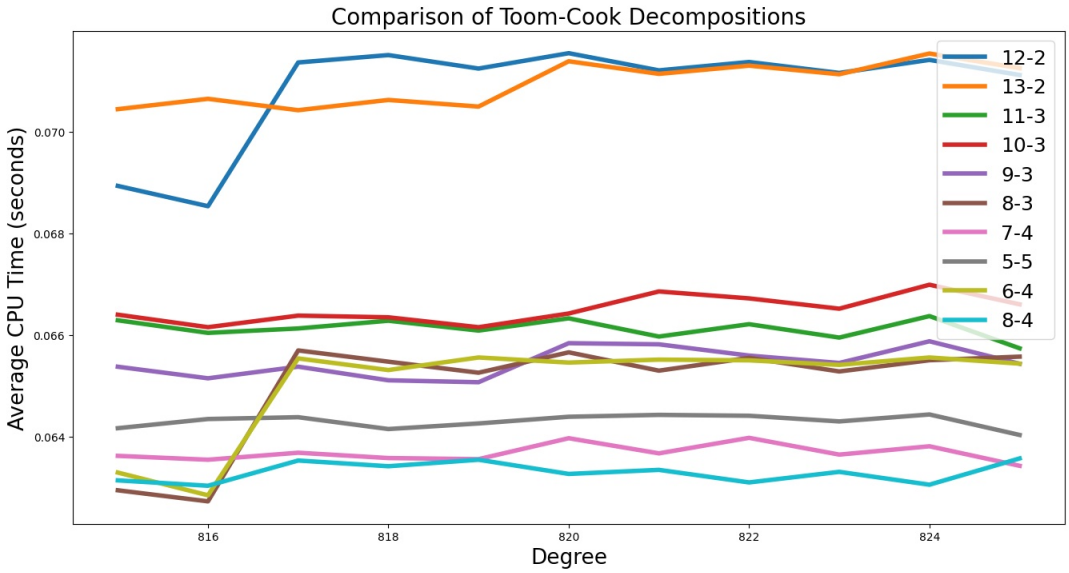


As with parameter set 1, the fastest 16-bit algorithm takes 1.1 times as long to run as the fastest 32-bit algorithm.

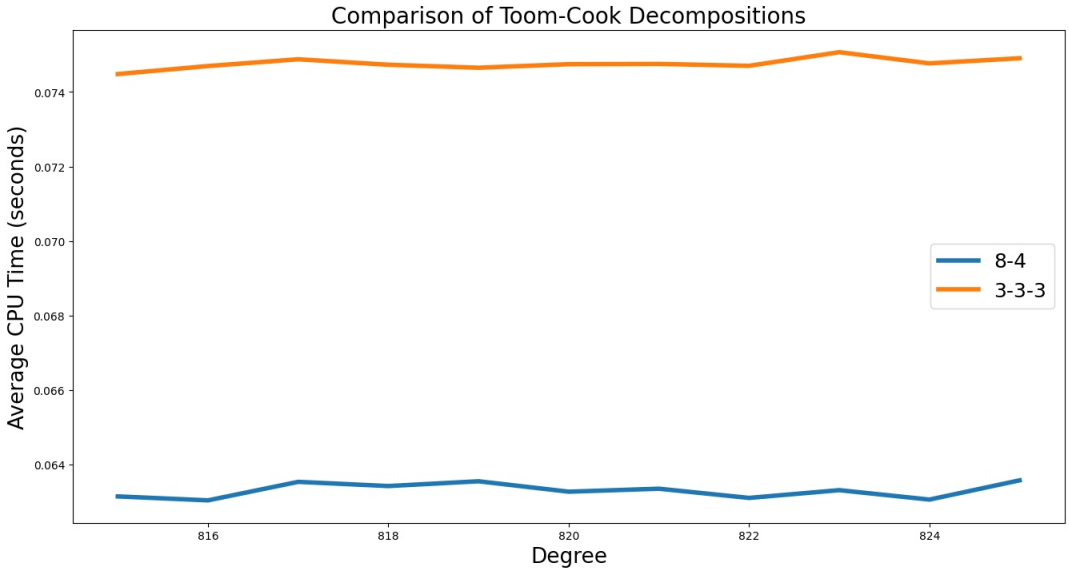
For parameter set 4, we can only afford to lose up 4 bits of precision with 16-bit ints, and 20 bits of precision with 32-bit ints. We begin with a plot of the most reasonable decompositions that give a correct result with 16-bit ints:



This shows that Toom-4-3-2 (which loses exactly 4 bits of precision, provided we use the natural formulas) and Toom-3-3-3 (which loses only 3 bits of precision) are virtually tied for being the fastest for degree 820 polynomials. Since simplicity is preferred when possible, we select Toom-3-3-3 as the winner. For the decompositions that give a correct result with 32-bit ints, we have



In this case, Toom-8-4 (which loses 13 bits of precision) is the fastest for degree 820 polynomials. Comparing these directly yields the following plot:



This time, the fastest 16-bit algorithm takes 1.17 times as long to run as the fastest 32-bit algorithm. We conjecture, that as N increases, the ratio of the fastest 16-bit decomposition's running time to that of the fastest 32-bit decomposition will increase.

We conclude that using 32-bit ints instead of 16-bit ints could speed up polynomial multiplication for NTRU. However, the implementations of schoolbook multiplication for small polynomials are highly optimized for 16-bit ints, and switching to 32-bit ints could cancel out any benefits.

Bibliography

[1] Arthur Benjamin and Jennifer Quinn. *Proofs that Really Count: The Art of Combinatorial Proof*, volume 27 of *Dolciani Mathematical Expositions*. The Mathematical Association of America, 2003.

[2] Marco Bodrato. Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0. In Claude Carlet and Berk Sunar, editors, *WAIFI'07 proceedings*, volume 4547 of *LNCS*, pages 116–133. Springer, June 2007. http://bodrato.it/papers/#WAIFI2007.

[3] Marco Bodrato and Alberto Zanoni. Integer and polynomial multiplication: towards optimal Toom-Cook matrices. In *ISSAC '07: Proceedings of the 2007 international symposium on Symbolic and algebraic computation*, pages 17–24, July 2007.

[4] Xavier Caruso, David Roe, and Tristan Vaccon. Tracking p-adic precision. *LMS Journal of Computation and Mathematics*, 17:274–294, 2014.

[5] Xavier Caruso, David Roe, and Tristan Vaccon. ZpL: a p-adic precision pacakge. In *Proceedings of ACM Conference*, 2018.

[6] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, and Zhenfei Zhang. NTRU: Algorithm specifications and supporting documentation. URL: https://ntru.org/f/ntru-20190330.pdf.

[7] C.-M. M. Chung, V. Hwang, M. J. Kannwischer, G. Seiler, C.-J. Shih, and B.-Y. Yang. NTT multiplication for NTT-unfriendly rings,. *Cryptology ePrint Archive*, Report 1397, 2020.

[8] Stephen A. Cook. *On the minimum computation time of functions*. PhD thesis, Harvard University, 1966.

[9] Wei Dai, William Whyte, and Zhenfei Zhang. Optimizing polynomial convolution for NTRUEncrypt. *IEEE Transactions on Computers*, 02 2018.

[10] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Rrederik Vercauteren. Saber: Algorithm specification and supporting documentation. URL: https://csrc.nist.gov/projects/.

[11] Srijit Dutta, Debjyoti Bhattacharjee, and Anupam Chattopadhyay. Quantum circuits for Toom-Cook multiplication. *Physical Review A*, 98, July 2018.

[12] M. Furer. Faster integer multiplication. In *Proceedings of the Thirty-Ninth ACM Symposium on Theory of Computing, STOC 2007*, pages 57–66, 2007.

[13] Ira M. Gessel and Christophe Reutenauer. Counting permutations with given cycle structure and descent set. *Journal of Combinatorial Theory*, 64:189–215, 1993.

[14] David Harvey and Joris van der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics, Princeton University, Department of Mathematics, In press.*

[15] Matthias J. Kannwischer, Joost Rijneveld, and Peter Schwabe. Faster multiplication in $\mathbf{Z}_{2^m}[x]$ on cortex-m4 to speed up NIST PQC candidates. In *Applied Cryptography and Network Security*, Lecture Notes in Computer Science, pages 281–301. Springer-Verlag Berlin Heidelberg, 2019.

[16] A. A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, pages 595–596, 1963.

[17] Donald E. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, 1998.

[18] M.J. Kronenburg. Toom-Cook multiplication: Some theoretical and practical aspects. February 2016. URL: https://arxiv.org/abs/1602.02740.

[19] Nicholas Loehr and Gregory Warrington. Quasisymmetric and Shur expansions of cycle index polynomials. *Discrete Mathematics*, 342(1):113–127, 2019.

[20] Partha Maji and Robert Mullins. On the reduction of computational complexity of deep convolutional neural networks. *Entropy*, 20, April 2018.

[21] Jose Maria Bermudo Mera, Angshuman Karmakar, and Ingrid Verbauwhede. Time-memory trade-off in Toom-Cook multiplication: an application to module-lattice based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2:222–244, 2020.

[22] Marko Petkovsek. Mathematica implementation of Gosper's algorithm. URL: https://www.math.upenn.edu/~wilf/progs.html.

[23] Marko Petkovsek, Herbert Wilf, and Doron Zeilberger. *A=B*. A K Peters, Ltd., 1997. URL: https://www.math.upenn.edu/~wilf/AeqB.pdf.

[24] E. A. Rawashdeh. A simple method for finding the inverse matrix of Vandermonde matrix. *Matematicki Vesnik*, 71:207–213, September 2019.

[25] C. Schensted. Longest increasing and decreasing subsequences. *Canadian journal of mathematics*, 13:179–191, 1961.

[26] A. Schonhage and V. Strassen. Schnelle multiplikation grosser zahlen. *Computing (Arch. Elektron. Rechnen)*, 7:281–292, 1971.

[27] N.J.A. Sloane. Eulerian number, Sequence A036969. URL: https://oeis.org/A036969.

[28] Andrei Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics Doklady*, 3:714–716, 1963.

[29] Joachim von zur Gathen and Jurgen Gerhard. *Modern Computer Algebra*.

[30] Alberto Zanoni. Toom-Cook 8-way for long integers multiplication. In *11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2009.

[31] Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, and William Whyte. NTRUEncrypt: Algorithm specification and supporting documentation. URL: https://csrc.nist.gov/projects/.