Finite-Difference Algorithms for Counting Problems

Thesis by
Eric Bax

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

1998
(Submitted Novemeber 11, 1997)

© 1998
Eric Bax
All Rights Reserved

Acknowledgements

I thank my parents for their support and encouragement of my endeavors. My mother is my greatest teacher. My father has always reassured me that education speeds success. I also thank my sisters and brothers. Their inspiration and moral support are among my greatest assets. I cannot imagine making my way through life without them.

I thank my advisor, Dr. Joel Franklin. He is an excellent mentor, teacher, fellow researcher, and role model. He has given me much more than knowledge of computer science and mathematics. Through his advice and example, he has taught me much about how to work and learn. I really appreciate his emphasis on understanding not just important results but also how they were discovered and developed. Dr. Franklin has taught me to learn by looking over the shoulders of giants.

I thank Gail Stowers and everyone at the CalTech Precollege Science Initiative for rescuing me when my career was threatened by lack of funding. At CAPSI, the love of people and of education extends far beyond their stated mission of bringing inquiry-based learning to the classroom.

I thank my undergraduate advisor, Dr. Hayden Porter. I came to graduate school on his advice. I came through graduate school using what I learned from him about work and courage, as well as computer science. I also thank Dr. Douglas Rall and Dr. Marty Cook at Furman University. Dr. Rall introduced me to research. His positive attitude and encouragement of independent thinking have inspired my work to this day. Dr. Cook was always available to listen to my ideas and point me in constructive directions.

I thank the professors at MTKI in Budapest, especially Dr. Andras Recski. He gave me insights on combinatorial algorithms that have greatly influenced my work. His maxim, "If you are having a hard time, try to solve a more general problem," is the basis for the finite-difference results in this thesis. Several years after I took his course, I returned to Budapest for a few weeks. He welcomed me, read my papers, and gave me excellent advice for further research.

I thank all of my teachers through the years, including Dr. Peter Bossaerts, Dr. Mani Chandy, and Dr. Alain Martin at Caltech; and the many great teachers at Furman University and from elementary school at St. Martin de Porres through high school at Lower Richland.

I thank my many companions in learning, including Will Jowers, Todd Brandes, Michael Croan, Jin Kim, John Hughes, Mac McLaughlin, Stephanie Eller, Mimi O'Bosky, Don Wilbur, Jennifer Lahue, Zehra Cataltepe, Joseph Sill, Nicole Peill, Paolo Sivilotti, John Thornley, Adam Rifkin, Rajit Manohar, and Eve Schooler.

Finally, I thank Beth Erlanson for her encouragement and moral support. She has been a source of strength and calm throughout the process of writing this thesis. Thank you, Beth, for believing in me.

Abstract

We present a novel method to construct counting algorithms:

1. Construct a generating function in which one type of terms corresponds to the objects to be counted.
2. Apply the proper finite-difference operators to produce a formula that counts the terms.
3. Choose finite-difference parameters to reduce the computation required to evaluate the formula.

We compare this finite-difference method to two other methods, the dynamic programming method and the inclusion and exclusion method. Using the problem of counting Hamiltonian paths as an example, we show that finite-difference algorithms require only polynomial space for problems for which dynamic programming algorithms require exponential space. Also, we show that finite-difference algorithms are a generalization of inclusion and exclusion algorithms. Finite-difference algorithms have some free parameters, and inclusion and exclusion algorithms correspond to a particular setting of those parameters. Using the 0-1 matrix permanent problem as an example, we show that the finite-difference parameters can be chosen to produce finite-difference algorithms that are faster than their inclusion and exclusion counterparts.

We use the problems of counting paths by length and counting independent path sets to illustrate how the flexibility of generating functions and extensions to finite-difference operators allow the development of finite-difference algorithms for problems beyond the realm of inclusion and exclusion. Furthermore, we use the problems of sequencing, bin packing, and deadlock avoidance to demonstrate the development of finite-difference algorithms for NP-complete and #P-complete problems.

Contents

List of Figures

List of Tables

Overview of Chapters

This work focuses on creating counting algorithms by applying finite-differences to generating functions. Chapter 1 begins with an example of this process. A finite-difference algorithm is developed to count Hamiltonian paths. Then, for comparison, dynamic programming and inclusion and exclusion algorithms are developed for the same problem. In Chapter 2, finite-difference operators are formally introduced. It is proven that iterated finite-difference operators produce the number of multilinear terms in a generating function. Other operators with these properties are exhibited, and operators to count non-multilinear terms are developed. Chapter 3 contains algorithms to count paths and cycles by length. These algorithms illustrate the finishing polynomial technique, a method of transforming terms corresponding to a variety of structures into multilinear terms. Chapter 4 contains a recursive algorithm to evaluate iterated finite differences. Computational reductions based on properties of specific problem instances are outlined. In Chapter 5, algorithms are developed for several problems to illustrate the techniques introduced in earlier chapters. These problems include sequencing, bin packing, and deadlock avoidance.

The remaining chapters are devoted to finite-difference algorithms for the permanent of a 0-1 matrix. In Chapter 6, the problem is introduced and a finite-difference formula is derived. For several finite-difference parameter settings, it is proven that the expected fraction of zero-valued terms in the finite-difference formula goes to unity as matrix size increases. A simple method to reduce computation by eliminating many of the zero-valued terms from the evaluation of the formula is presented and tested. In Chapter 7, a permanent decomposition is shown to transform a single instance of the 0-1 permanent problem into a pair of permanent problem instances, each having the property that the expected fraction of nonzero-valued terms in the corresponding finite-difference formula is exponentially small. Chapter 8 contains an algorithm to compute the permanent, avoiding so many zero-valued terms that the expected running time is an exponentially small fraction of the time required to compute the permanent by the standard method. The algorithm has the drawback of requiring super-polynomial space. Methods are introduced to adjust the algorithm to trade off time and space requirements. Finally, Chapter 9 outlines methods to estimate the permanent formula and other finite-difference formulas by sampling terms at random.

Chapter 1

Introduction

To introduce the finite-difference technique, we develop an algorithm to count Hamiltonian paths. Then, for comparison, we develop algorithms using inclusion and exclusion and dynamic programming.

1.1 Finite-Difference Method

Definition 1 (Finite-Difference Operator) *The finite-difference with respect to x_j, written $D_j(u_j, v_j)$, is defined as follows:*

$$\forall u_j \neq v_j \quad D_j(u_j, v_j)f() \equiv \frac{f(x_j = u_j) - f(x_j = v_j)}{u_j - v_j} \tag{1.1}$$

where $f(x_j = u_j)$ is the function that results from setting x_j to the constant value u_j.

For example,

$$D_2(5,3)x_1x_2x_3^2 = \frac{5x_1x_3^2 - 3x_1x_3^2}{5-3} = x_1x_3^2 \tag{1.2}$$

To simplify formulas, we denote assignments to variables in $f()$ by their assigned values and omit free variables. Also, we abbreviate $D_j(u_j, v_j)$ by D_j. In the shortened notation, the definition of the finite-difference operator is written

$$D_jf() \equiv \frac{f(u_j) - f(v_j)}{u_j - v_j} \tag{1.3}$$

1.1.1 Finite-Differences and Multilinear Terms

For $P()$ a polynomial with every term of degree n or less, $D_1 \cdots D_nP()$ is the coefficient of the multilinear term $x_1 \cdots x_n$. A proof of this property is given later. Intuitively, each D_j

acts as a derivative with respect to x_j, so the composition $D_1 \cdots D_n$ finds the derivative with respect to every variable. Since the polynomial's terms have degree n or less, each term other than the multilinear term lacks some variable. Since the term is constant with respect to the missing variable, the difference operator that corresponds to the missing variable kills the term.

1.1.2 Finite-Difference Formulas

Expanding the finite-difference operators produces a formula for the multilinear term:

$$D_1 \cdots D_n P() = \frac{1}{(u_1 - v_1) \cdots (u_n - v_n)} \sum_{\mathbf{x} \in \{u_1, v_1\} \times \cdots \times \{u_n, v_n\}} (-1)^{s(\mathbf{x})} P(\mathbf{x}) \tag{1.4}$$

where $s(\mathbf{x})$ is the number of variables x_j set to v_j. Computing the formula requires 2^n evaluations of $P()$, so if each evaluation requires O(poly n) time and O(poly n) space, then the entire computation requires O(2^npoly n) time and O(poly n) space.

1.1.3 Hamiltonian Paths as the Multilinear Term of a Polynomial

Let G be a directed graph with vertex set $V = \{s, t\} \cup \{1, \ldots, n\}$. We will show that the number of s-t Hamiltonian paths in G is the coefficient of a polynomial with every term of degree n.

Let A be the adjacency matrix of the graph induced by vertex subset $\{1, \ldots, n\}$, and define a corresponding variable matrix B such that $b_{ij} = a_{ij} x_j$. Define a $1 \times n$ "entrance" vector $\mathbf{s}$:

$$s_j = \begin{cases} x_j & \text{if } G \text{ has an edge from } s \text{ to } j \\ 0 & \text{otherwise} \end{cases} \tag{1.5}$$

Define an $n \times 1$ "exit" vector $\mathbf{t}$:

$$t_i = \begin{cases} 1 & \text{if } G \text{ has an edge from } i \text{ to } t \\ 0 & \text{otherwise} \end{cases} \tag{1.6}$$

Define W to be the set of length $n+1$ s-t walks in G that contain neither vertex s nor vertex t as internal vertices. For j in $1 \ldots n$ and for each walk $w \in W$, define $m_j(w)$ to be the number of occurrences of vertex j in the walk.

Define $P() \equiv \mathbf{s}B^{n-1}\mathbf{t}$. Since adjacency matrix multiplication counts walks, $P(1, \ldots, 1) =$

$|W|$. Since each variable x_j is multiplied into terms as its vertex j is visited,

$$P_k(x_1,\dots,x_n) = \sum_{w\in W} x_1^{m_1(w)}\cdots x_n^{m_n(w)} \tag{1.7}$$

The Hamiltonian paths have $m_1(w) = \dots = m_n(w) = 1$. So the Hamiltonian paths correspond to multilinear terms. $\Box$

1.2 Inclusion and Exclusion Method

The following algorithm is adapted from a paper by Karp [24]. (For related results, see [2, 3].) Let U be the set of length $n+1$ s-t walks in G that contain neither vertex s nor vertex t as internal vertices. Let W_S be the set of walks that lack all vertices in $S \subseteq \{1,\dots,n\}$. The number of Hamiltonian paths is

$$|U - (W_{\{1\}} \cup \dots \cup W_{\{n\}})| \tag{1.8}$$

By the principle of inclusion and exclusion [29], this is equal to

$$|U| - \sum_{\{j_1\}\subseteq\{1,\dots,n\}} |W_{\{j_1\}}| + \sum_{\{j_1\,j_2\}\subseteq\{1,\dots,n\}} |W_{\{j_1\}} \cap W_{\{j_2\}}| - \dots \pm |W_{\{1\}} \cap \dots W_{\{n\}}| \tag{1.9}$$

Note that $W_{\{j_1\}} \cap \dots \cap W_{\{j_k\}} = W_{\{j_1,\dots,j_k\}}$. (Both expressions are the walks that lack all vertices in $\{j_1,\dots,j_k\}$.) By definition, $U = W_\emptyset$. So,

$$|U - (W_{\{1\}} \cup \dots \cup W_{\{n\}})| = \sum_{S\subseteq\{1,\dots,n\}} (-1)^{|S|}|W_S| \tag{1.10}$$

To count walks that lack all vertices in S, use adjacencies $\mathbf{s}$, B, and $\mathbf{t}$. Set $x_j = 0$ for $j \in S$ and $x_j = 1$ for $j \notin S$ to erase edges to forbidden vertices. Then

$$|W_S| = \mathbf{s}B^{n-1}\mathbf{t} \tag{1.11}$$

and the number of Hamiltonian paths is

$$\sum_{\mathbf{x}\in\{1,0\}^n} (-1)^{s(\mathbf{x})}\mathbf{s}B^{n-1}\mathbf{t} \tag{1.12}$$

where $s(\mathbf{x})$ is the number of variables assigned zero.

The inclusion and exclusion formula is a special case of the finite-difference formula, with parameters $u_j = 1$ and $v_j = 0$. Like the general finite-difference formula, the inclusion and exclusion formula can be computed in $O(n^2 2^n)$ time and $O(n^2)$ space.

1.3 Dynamic Programming Method

The following algorithm is adapted from a paper by Bellman [9]. (For related results, see [16].) The algorithm counts paths of successive lengths. On termination, $c_{S,j}$ is the number of s-j paths with internal vertices S. Hence, $c_{\{1,\ldots,n\},t}$ is the number of s-t Hamiltonian paths.

The adjacencies A and $\mathbf{t}$ are defined as before. Define a non-variable version of adjacency $\mathbf{s}$:

$$s_j = \begin{cases} 1 & \text{if } G \text{ has an edge from } s \text{ to } j \\ 0 & \text{otherwise} \end{cases} \tag{1.13}$$

Variable L refers to path length. The algorithm follows.

```
for j = 1 to n
    $c_{\emptyset,j} = s_j$

for L = 2 to n
    for S such that $|S| = L - 1$
        for $j \notin S$
            $c_{S,j} = \sum_{i\in\{1,\ldots,n\},i\neq j} c_{S-\{i\},i}\,a_{ij}$

$c_{\{1,\ldots,n\},t} = \sum_{i\in\{1,\ldots,n\}} c_{\{1,\ldots,n\}-\{i\},i}\,t_i$
```

The algorithm requires $O(n^2 2^n)$ time and $O(2^n n)$ space.

1.4 Comparison of Methods

The inclusion and exclusion algorithm is a special case of the finite-difference algorithm. Both algorithms sift through a larger set of walks to count the set of Hamiltonian paths. The dynamic programming algorithm is fundamentally different; it accumulates Hamiltonian paths through construction and gathering of subpaths.

The dynamic programming algorithm requires exponential space, and the other algorithms do not. Also, there are many cross-references among the computed terms, hence many messages and synchronizations are required to partition the computation among message-passing processors in a parallel computer. In contrast, the other algorithms sum over 2^n terms with no cross-references. The terms can be computed independently on separate processors. The only step that requires communication is summing over the processors.

Unlike the inclusion and exclusion formula, the finite-difference formula offers a choice of parameter settings. Some settings allow reductions in the computation of formula (1.4). For example, if every term of $P()$ has degree n, then setting $u_j = -v_j$ introduces a pairwise symmetry among the formula's terms. The terms with opposite assignments to each variable have the same value, i.e.,

$$(-1)^{s(\mathbf{x})} P(\mathbf{x}) = (-1)^{s(-\mathbf{x})} P(-\mathbf{x}) \tag{1.14}$$

Proof. If n is even, then $(-1)^{s(\mathbf{x})} = (-1)^{s(-\mathbf{x})}$ and $P(\mathbf{x}) = P(-\mathbf{x})$. If n is odd, then $(-1)^{s(\mathbf{x})} = -(-1)^{s(-\mathbf{x})}$ and $P(\mathbf{x}) = -P(-\mathbf{x})$. $\square$

The computation can be halved by computing one term of each pair and doubling the result.

As we develop finite-difference algorithms, we will explore problem-specific parameter settings that allow greater computational reductions. Also, we will extend the finite-difference framework to tackle problems beyond the domain of the inclusion and exclusion framework.

Chapter 2

Finite-Difference Operators

In this section, we show how finite-difference operators isolate terms of a polynomial. We begin with the multilinear term and proceed to other terms. For a general treatment of finite differences, consult the numerical analysis text by Hildebrand [17].

2.1 Finite-Differences and Multilinear Terms

Recall that we defined the finite-difference operator with respect to x_j as follows.

$$D_j f() \equiv \frac{f(u_j) - f(v_j)}{u_j - v_j} \text{ for } u_j \neq v_j \tag{2.1}$$

Let $P()$ be a polynomial with every term of degree n or less. Then $D_1 \cdots D_n P()$ is the coefficient of the multilinear term of $P()$.

Proof. We prove a more general result. Let L_j be a linear operator with the following property.

$$L_j x_j^p = \begin{cases} 0 & \text{if } p = 0 \\ 1 & \text{if } p = 1 \end{cases} \tag{2.2}$$

Also, assume that the operators commute: $L_i L_j = L_j L_i$.

Apply $L_1 \cdots L_n$ to $P()$.

$$L_1 \cdots L_n P() \tag{2.3}$$

Rewrite $P()$ as the sum of terms.

$$= L_1 \cdots L_n \sum_{(p_1,\ldots,p_n) \in \{0,\ldots,n\}^n, p_1+\ldots+p_n \leq n} c_{p_1,\ldots,p_n} x_1^{p_1} \cdots x_n^{p_n} \tag{2.4}$$

Apply the linear operators to each term.

$$= \sum c_{p_1,\ldots,p_n} L_1 \cdots L_n x_1^{p_1} \cdots x_n^{p_n} \tag{2.5}$$

For the multilinear term,

$$L_1 \cdots L_n x_1 \cdots x_n = L_1 \cdots L_{n-1} x_1 \cdots x_{n-1} \cdot 1 = 1 \cdots 1 = 1 \tag{2.6}$$

So the term contributes $c_{1,\ldots,1}$ to the sum.

Each non-multilinear term is constant with respect to some variable x_j. (Use the pigeonhole principle; $p_1 + \ldots + p_n \leq n$ and $(p_1, \ldots, p_n) \neq (1, \ldots, 1)$ implies $p_j = 0$ for some j.) Since the operators commute,

$$L_1 \cdots L_n x_1^{p_1} \cdots x_n^{p_n} \tag{2.7}$$

$$L_1 \cdots L_{j-1} L_{j+1} \cdots L_n L_j x_1^{p_1} \cdots x_{j-1}^{p_{j-1}} x_j^0 x_{j+1}^{p_{j+1}} \cdots x_n^{p_n} \tag{2.8}$$

$$L_1 \cdots L_{j-1} L_{j+1} \cdots L_n 0 = 0 \tag{2.9}$$

So non-multilinear terms contribute zero to the sum. Hence,

$$L_1 \cdots L_n P() = c_{1,\ldots,1} \tag{2.10}$$

The finite-difference operator D_j meets the conditions for L_j. It is linear:

$$D_j c P() = c D_j P() \tag{2.11}$$

and

$$D_j [P() + Q()] = D_j P() + D_j Q() \tag{2.12}$$

Also,

$$D_j x_j^p = \begin{cases} 0 & \text{if } p = 0 \\ 1 & \text{if } p = 1 \end{cases} \tag{2.13}$$

and

$$D_i D_j P() = D_j D_i P() \tag{2.14}$$

□

2.2 Alternative Operators

Here are some other operators that meet the conditions for L_j.

$$L_j P() = \frac{\partial}{\partial x_j} P() \tag{2.15}$$

and

$$L_j P() = \frac{3}{2} \int_{-1}^{1} P()\ x_j\ dx_j \tag{2.16}$$

We do not explore algorithms based on these operators in this paper, but they may have their uses. For example, the integral could be used to develop algorithms that estimate the multilinear term by Monte Carlo integration.

2.3 Non-Multilinear Terms

We generalize finite-difference operators to isolate non-multilinear terms. Define $P()$ as before. Let L_j^k be a commuting linear operator with the following property.

$$L_j^k x_j^p = \begin{cases} 0 & \text{if } p < k \\ 1 & \text{if } p = k \end{cases} \tag{2.17}$$

For $k_1 + \ldots + k_n = n$, $L_1^{k_1} \cdots L_n^{k_n} P()$ is the coefficient of term $x_1^{k_1} \cdots x_n^{k_n}$.

Proof. The proof is similar to the proof for multilinear terms. As before,

$$L_1^{k_1} \cdots L_n^{k_n} P() = \sum c_{p_1,\ldots,p_n} L_1^{k_1} \cdots L_n^{k_n} x_1^{p_1} \cdots x_n^{p_n} \tag{2.18}$$

For the target term,

$$L_1^{k_1} \cdots L_n^{k_n} x_1^{k_1} \cdots x_n^{k_n} = 1 \tag{2.19}$$

so the term contributes $x_1^{k_1} \cdots x_n^{k_n}$ to the sum. For other terms, $p_j < k_j$ for some x_j. So $L_j^{k_j}$ zeros the term. □

Now we develop an operator that meets the conditions for L_j^k. Define operator D:

$$DP() = \frac{1}{\Delta x}(P(x + \Delta x) - P(x)) \tag{2.20}$$

where $P(x + \triangle x)$ is $P()$ with each occurrence of x rewritten $x + \triangle x$. We show

$$D^k x^p = \begin{cases} 0 & \text{if } p < k \\ k! & \text{if } p = k \end{cases} \tag{2.21}$$

Proof. Note

$$Dx^p = px^{p-1} + \text{ lower order terms in x} \tag{2.22}$$

Assume the proposition holds for k. Then

$$D^{k+1}x^{k+1} = D^k D x^{k+1} \tag{2.23}$$

$$= D^k[(k+1)x^k + \text{ lower order terms}] = (k+1)D^k x^k = (k+1)! \tag{2.24}$$

If $p < k+1$, then

$$D^{k+1}x^p = D^k D x^p = D^k[\text{ terms of order less than k}] = 0 \tag{2.25}$$

□

Now expand $D^2P()$.

$$D^2P() = \frac{1}{(\triangle x)^2}[P(x + 2\triangle x) - 2P(x + \triangle x) + P(x)] \tag{2.26}$$

Note the binomial form. In general,

$$D^kP() = \frac{1}{(\triangle x)^k} \sum_{m=0}^{k} \binom{k}{m} (-1)^{k-m} P(x + m\triangle x) \tag{2.27}$$

To form an operator that returns 0 for x_j^p with $p < k$ and 1 for x_j^k, replace x_j by x in $P()$, apply D^k, evaluate at $x = v_j$ and $\triangle x = u_j - v_j$, and divide by $k!$.

Definition 2 (*k*th Finite-Difference Operator)

$$D_j^{(k)}P() = \frac{1}{k!}\frac{1}{(u_j - v_j)^k} \sum_{m=0}^{k} \binom{k}{m} (-1)^{k-m} P(v_j + m(u_j - v_j)) \tag{2.28}$$

Since we divide by $k!$,

$$D_j^{(k)} x_j^p = \begin{cases} 0 & \text{if } p < k \\ 1 & \text{if } p = k \end{cases} \tag{2.29}$$

Note that we can compute $D_j^{(k)}P()$ with $k+1$ evaluations of $P()$. By iteration, we can compute $D_1^{(k_1)} \cdots D_n^{(k_n)}P()$ with $(k_1+1)\cdots(k_n+1)$ evaluations of $P()$.

With the polynomial $P()$ from the introduction, $D_1^{(k_1)} \cdots D_n^{(k_n)}P()$ counts s-t walks with k_j occurrences of vertex j for $k_1 + \ldots + k_n = n$. So, for example,

$$D_1^{(\frac{n}{2})} D_2^{(\frac{n}{2})} P() \tag{2.30}$$

counts length n s-t walks with $\frac{n}{2}$ occurrences of vertices 1 and 2. The computation requires $\mathrm{O}(n^2)$ evaluations of $P()$.

The methods developed in this section isolate the coefficient of a single term in $P()$. In the next section, we show how to find the sum of a set of coefficients by manipulating the generating function $P()$. We focus on the example of counting paths and cycles by length.

Chapter 3

Counting Paths and Cycles by Length

In this chapter, we develop algorithms to count paths and cycles by length. We begin with a generating function similar to the one used to count Hamiltonian paths in the introduction. The variables correspond to vertex occurrences, and matrix multiplication counts walks of the desired length. Then, we multiply by a finishing polynomial that adds the necessary variables to turn path or cycle terms into multilinear terms.

Counting Hamiltonian paths is a special case of counting paths by length. The principle of inclusion and exclusion does not apply directly to the general problem, but the finite-difference approach produces the algorithm in a straightforward fashion. The material in this chapter was originally published as a paper in *Information Processing Letters* [6]. For an algorithm that counts paths or cycles of all lengths, see [3]. For an algorithm to list all cycles, see [31].

3.1 Paths

Let G be a directed graph with vertex set $V = \{s, t\} \cup \{1, \ldots, n\}$. We will show that the number of length k s-t paths in G is the coefficient of the multilinear term of a polynomial with every term of degree n.

Let A be the adjacency matrix of the graph induced by vertex subset $\{1, \ldots, n\}$, and define a corresponding variable matrix B such that $b_{ij} = a_{ij}x_j$. Define a $1 \times n$ "entrance" vector **s**:

$$s_j = \begin{cases} x_j & \text{if } G \text{ has an edge from } s \text{ to } j \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Define an $n \times 1$ "exit" vector **t**:

$$t_i = \begin{cases} 1 & \text{if } G \text{ has an edge from } i \text{ to } t \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Define W to be the set of length k s-t walks in G that contain neither vertex s nor vertex t as internal vertices. For j in $1 \ldots n$ and for each walk $w \in W$, define $m_j(w)$ to be the number of occurrences of vertex j in the walk.

Define $P_k(x_1, \ldots, x_n) \equiv \mathbf{s}B^{k-2}\mathbf{t}$ for $k > 1$. Since adjacency matrix multiplication counts walks, $P_k(1, \ldots, 1) = |W|$. Since each variable x_j is multiplied into terms as its vertex j is visited,

$$P_k(x_1, \ldots, x_n) = \sum_{w \in W} x_1^{m_1(w)} \cdots x_n^{m_n(w)} \tag{3.3}$$

Define the "finishing" polynomial

$$F_k(x_1, \ldots, x_n) \equiv \sum_{\{j_1, \ldots, j_k\} \subseteq \{1, \ldots, n\}} x_{j_1} \cdots x_{j_k} \tag{3.4}$$

Examine the polynomial

$$P_k(x_1, \ldots, x_n) F_{n-k+1}(x_1, \ldots, x_n) = \Big(\sum_{w \in W} x_1^{m_1(w)} \cdots x_n^{m_n(w)}\Big)\Big(\sum_{\{j_1, \ldots, j_{n-k+1}\} \subseteq \{1, \ldots, n\}} x_{j_1} \cdots x_{j_{n-k+1}}\Big) \tag{3.5}$$

Each term of $P_k()$ has degree $k-1$, and each term of $F_{n-k+1}()$ has degree $n-k+1$, so each term of the product has degree n.

For each walk $w \in W$, the polynomial $P_k()F_{n-k+1}()$ has terms:

$$x_1^{m_1(w)} \cdots x_n^{m_n(w)}\Big(\sum_{\{j_1, \ldots, j_{n-k+1}\} \subseteq \{1, \ldots, n\}} x_{j_1} \cdots x_{j_{n-k+1}}\Big) \tag{3.6}$$

If the walk is not a path, then it contains more than one occurrence of some vertex. Hence, $m_j(w) > 1$ for some $j \in \{1, \ldots, n\}$, so none of the terms in (3.6) is multilinear.

If the walk is a path, then $k-1$ of the $m_j(w)$'s are 1, and $n-k+1$ of the $m_j(w)$'s are 0. When $\{j_1, \ldots, j_{n-k+1}\}$ is the set for which $m_{j_1}(w) = \ldots = m_{j_{n-k+1}}(w) = 0$, the sum produces a multilinear term:

$$x_1^{m_1(w)} \cdots x_n^{m_n(w)} x_{j_1} \cdots x_{j_{n-k+1}} = x_1 \cdots x_n \tag{3.7}$$

By the pigeonhole principle, each of the other sets $\{j_1, \ldots, j_{n-k+1}\}$ fails to index some $m_j(w)$ that is 0. So x_j has exponent zero in:

$$x_1^{m_1(w)} \cdots x_n^{m_n(w)} x_{j_1} \cdots x_{j_{n-k+1}} \tag{3.8}$$

and the term is not multilinear. Thus, each path in W gives a single multilinear term in $P_k()F_{n-k+1}()$. So $D_1 \cdots D_n P_k() F_{n-k+1}()$ is the number of length k s-t paths in G. The formula is

$$\frac{1}{(u_1 - v_1) \cdots (u_n - v_n)} \sum_{\mathbf{x} \in \{u_1, v_1\} \times \cdots \times \{u_n, v_n\}} (-1)^{s(\mathbf{x})} P_k(\mathbf{x}) F_{n-k+1}(\mathbf{x}) \tag{3.9}$$

where $s(\mathbf{x})$ is the number of variables x_j set to v_j.

3.2 Cycles

A similar relationship holds for cycles. Let G be a directed graph with vertex set $V = \{s\} \cup \{1, \ldots, n\}$. Then the number of length k cycles on s is the coefficient of the multilinear term of a polynomial with every term of degree n.

Define A, B, and $\mathbf{s}$ as before. Define an $n \times 1$ "return" vector $\mathbf{r}$:

$$r_i = \begin{cases} 1 & \text{if } G \text{ has an edge from } i \text{ to } s \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

Define $Q_k(x_1, \ldots, x_n) \equiv \mathbf{s}B^{k-2}\mathbf{r}$. The coefficient of the multilinear term of $Q_k()F_{n-k+1}()$ is the number of length k cycles on s. So $D_1 \cdots D_n Q_k() F_{n-k+1}()$ counts these cycles. The formula is

$$\frac{1}{(u_1 - v_1) \cdots (u_n - v_n)} \sum_{\mathbf{x} \in \{u_1, v_1\} \times \cdots \times \{u_n, v_n\}} (-1)^{s(\mathbf{x})} Q_k(\mathbf{x}) F_{n-k+1}(\mathbf{x}) \tag{3.11}$$

where $s(\mathbf{x})$ is the number of variables x_j set to v_j.

3.3 Computational Issues

3.3.1 Evaluation of $F_k()$

Using dynamic programming, $F_k()$ can be evaluated in *poly*(n) time and *poly*(n) space. Define

$$f_k^m \equiv \sum_{\{j_1,\ldots,j_k\}\subseteq\{1,\ldots,n\} \text{ and } m=\max(j_1,\ldots,j_k)} x_{j_1}\cdots x_{j_k} \tag{3.12}$$

e.g., $f_2^3 = x_1x_3 + x_2x_3$. By the definition:

$$f_1^m = x_m \quad \forall\, m \in \{1,\ldots,n\} \tag{3.13}$$

$$f_k^m = \sum_{h<m} f_{k-1}^h x_m = \sum_{h=k-1}^{m-1} f_{k-1}^h x_m \tag{3.14}$$

and

$$F_k() = \sum_{m=1}^{n} f_k^m = \sum_{m=k}^{n} f_k^m \tag{3.15}$$

So $F_k()$ can be computed by the procedure:

```
initially $f_1^m = x_m \quad \forall\, m \in \{1,\ldots,n\}$
for $c = 2$ to $k$
    $\forall\, m \in \{c,\ldots,n\}\ f_c^m := \sum_{h=c-1}^{m-1} f_{c-1}^h x_m$
$F_k() := \sum_{m=k}^{n} f_k^m$
```

3.3.2 An Alternative Finishing Polynomial

The essential property of $F_{n-k+1}()$ is that its product with any term of degree $k-1$ produces a multilinear term of degree n if the degree $k-1$ term is linear in $k-1$ different variables, and it produces a non-multilinear term of degree n otherwise. Polynomials have this essential property if they meet the conditions:

- Every term has degree $n-k+1$.
- The terms of $F_{n-k+1}()$ are present, each with coefficient one.

So we define an alternative finishing polynomial:

$$\hat{F}_{n-k+1}() \equiv (x_1 + \ldots + x_n)^{n-k+1}/(n-k+1)! \tag{3.16}$$

This polynomial can be evaluated in $O(n)$ time. (The dynamic programming procedure to evaluate $F_{n-k+1}()$ requires $O(n^3)$ time.) The time required to evaluate $P_k()\hat{F}_{n-k+1}()$ or $Q_k()\hat{F}_{n-k+1}()$ is $O(n^2)$. So the algorithms to count paths and cycles have time complexity $O(n^2 2^n)$.

3.3.3 Setting u and v to Reduce Computation

If we set $\mathbf{u} = \mathbf{0}$ or $\mathbf{v} = \mathbf{0}$, then $F_{n-k+1}()$ is zero in some terms of formulas (3.9) and (3.11). Computation can be reduced by not evaluating the zero terms. For concreteness, let $\mathbf{u} = \mathbf{1}$ and $\mathbf{v} = \mathbf{0}$. In formula (3.9), $P_k(\mathbf{x})F_{n-k+1}(\mathbf{x})$ is summed over all assignments $\mathbf{x} \in \{0,1\}^n$. For a given assignment, let d be the number of elements in $\mathbf{x}$ that are assigned one. Recall that $F_{n-k+1}()$ is the sum of products over size $n-k+1$ subsets of the variables. If a subset contains a variable assigned zero, then its product is zero. If a subset contains only variables assigned one, then its product is one. So $F_{n-k+1}()$ is the number of subsets that contain only variables assigned one, i.e.,

$$F_{n-k+1}(\mathbf{x}) = \begin{cases} \binom{d}{n-k+1} & \text{if } d \geq n-k+1 \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

Hence, terms in which $\mathbf{x}$ has fewer than $n-k+1$ variables assigned one need not be evaluated. This leaves $\sum_{d \geq n-k+1} C(n,d) = \sum_{i \leq k-1} C(n,i) = O(n^{k-1})$ terms to be evaluated. The savings are substantial for counting short paths and cycles. The computation required to compute $P_k()$ or $Q_k()$ is $O(kn^2)$, so the total time complexity to count paths or cycles of length k is $O(kn^{k+1})$.

Since $P_k()F_{n-k+1}()$ and $Q_k()F_{n-k+1}()$ have only degree n terms, the computation can be halved by setting $\mathbf{u} = -\mathbf{v}$, as discussed in the introduction. Unfortunately, the $\mathbf{u} = -\mathbf{v}$ strategy cannot be combined with the $\mathbf{v} = \mathbf{0}$ strategy, because finite-differences require $u_j \neq v_j \ \forall j$. If $k > \frac{n}{2}$, then, under the $\mathbf{v} = \mathbf{0}$ strategy, more than half of the terms must be computed. So it is more efficient to use the $\mathbf{u} = -\mathbf{v}$ strategy in this case.

Under the $\mathbf{u} = -\mathbf{v}$ strategy, the computation can be reduced by using the alternative finishing polynomial $\hat{F}_{n-k+1}()$. If n is even, then set $\mathbf{u} = \mathbf{1}$ and $\mathbf{v} = -\mathbf{1}$. Assignments with equal numbers of positive and negative elements in $\mathbf{x}$ give $\hat{F}_{n-k+1}(\mathbf{x}) = 0$ since $x_1 + \ldots + x_n = 0$. The fraction of assignments with equal numbers of positive and negative elements is $\frac{C(n,\frac{n}{2})}{2^n} = O(\frac{1}{\sqrt{n}})$. If n is odd, then a similar reduction can be achieved by setting $u_1 = 2$, $v_1 = -2$, and $u_j = 1$, $v_j = -1 \ \forall\, j > 1$.

3.4 Challenges

The choice of finite-difference parameters and finishing polynomials introduces opportunities for future research. One challenge is to find parameters and finishing polynomials that reduce computation for general problem instances. For example, $\mathbf{u} = (1, -1, 1, -1, \ldots)$ and $\mathbf{v} = (0, 0, 0, 0, \ldots)$, with the proper finishing polynomial, may combine the advantages of the $\mathbf{u} = -\mathbf{v}$ setting and the $\mathbf{u} = \mathbf{1}$ and $\mathbf{v} = \mathbf{0}$ setting. Another challenge is to find simple ways to tailor $\mathbf{u}$, $\mathbf{v}$, and the finishing polynomial to specific problem instances to reduce computation. Yet another challenge is to find methods to estimate numbers of paths and cycles by using subsets of the finite-difference formulas' terms. Part of this goal is to find settings of $\mathbf{u}$ and $\mathbf{v}$ and finishing polynomials that produce terms with low variance. Another part of the goal is to identify which subsets of the terms best estimate the sum over all terms.

3.5 Extensions

The algorithms to count paths and cycles can be extended to count other structures. For example, suppose we wish to count sets of s-t paths with no common vertices, called independent path sets. Add a vertex v, with a v-j edge for each s-j edge and an i-v edge for each i-t edge. For each set of m independent paths in the original graph, there are $m!$ walks in the new graph with $m-1$ occurrences of vertex v and no more than one occurrence of each other vertex. For example, independent path set $\{s\text{-}1\text{-}2\text{-}3\text{-}t, s\text{-}4\text{-}5\text{-}t\}$ corresponds to walks s-1-2-3-v-4-5-t and s-4-5-v-1-2-3-t.

For convenience, assume there are no edges between s and t. Add to A row and column $n+1$, corresponding to vertex v. Define B as before, i.e., $b_{ij} = a_{ij}x_j$. Use k to index the sum of path lengths in the independent set. The number of independent path sets with m paths is

$$\frac{1}{m!}\sum_{k=2}^{2n} D_1 \cdots D_n D_{n+1}^{m-1} P_{k+(m-1)}() F_{n-k+1}() \tag{3.18}$$

where $F_{n-k+1+(m-1)}$ is the finishing polynomial over variables $x_1, \ldots, x_n$.

Chapter 4

Recursive Algorithms and Computational Reductions

In this chapter, we develop a recursive algorithm to compute $D_1 \cdots D_n P()$. Then we show how to produce computational reductions based on problem structure. Using the problem of counting paths by length as an example, we show that a framework to develop reductions for inclusion and exclusion algorithms [4] extends to finite-difference algorithms.

4.1 A Recursive Algorithm

The following algorithm recursively evaluates the finite-differences in the expression $D_1 \cdots D_n P()$. The problem size n and the finite-difference parameters $\mathbf{u}$ and $\mathbf{v}$ are global variables. The function call `fdiff(1,0)` evaluates $D_1 \cdots D_n P()$. Each function call `fdiff(j,x)` evaluates $D_j \cdots D_n P()$. The variable $\mathbf{x}$ accumulates assignments to $x_1, \ldots, x_n$, and $\mathbf{e}^j$ is the n-vector with value one in entry j and value zero in the other entries.

```
fdiff(j,x)
{
if $j = n + 1$ then return P(x)
else return $\frac{1}{u_j - v_j}$[fdiff($j + 1, x + u_j e^j$) − fdiff($j + 1, x + v_j e^j$)]
}
```

Each function call `fdiff(j,x)` assigns a value to x_j, and the descendant function calls assign values to $x_{j+1}, \ldots, x_n$. So the values of $x_1, \ldots, x_{j-1}$ are fixed throughout the recursion from fdiff(j,x). We refer to the values of $\mathbf{x}$ with $x_1, \ldots, x_{j-1}$ fixed as the continuations of $x_1, \ldots, x_{j-1}$.

To count paths, set $P() = P_k()F_{n-k}()$. Setting $\mathbf{u} = \mathbf{1}$ and $\mathbf{v} = \mathbf{0}$ produces the following algorithm.

```
paths(j,x)
{
if j = n + 1 then return $P_k(x)F_{n-k}(x)$
else return [paths(j + 1, x + $e^j$) − paths(j + 1, x)]
}
```

4.2 Reductions

4.2.1 Bounding and Zero-Sets

With $\mathbf{u} = \mathbf{1}$ and $\mathbf{v} = \mathbf{0}$, $P_k(\mathbf{x})$ counts length k s-t walks in the graph formed by eliminating vertices that correspond to zero-valued entries in $\mathbf{x}$. Eliminating a vertex from a graph cannot add walks, so changing a one-valued entry in $\mathbf{x}$ to zero cannot increase $P_k(\mathbf{x})$.

Since $F_{n-k}(\mathbf{x}) = \binom{d}{n-k}$, where d is the number of one-valued entries in $\mathbf{x}$, changing a one-valued entry in $\mathbf{x}$ to zero cannot increase $F_{n-k}(\mathbf{x})$.

Define $\mathbf{x}^b$ to have a given set of values in entries 1 to $j-1$ and to have ones in entries j to n. By the above arguments, $P_k(\mathbf{x}^b)F_{n-k}(\mathbf{x}^b)$ is an upper bound over continuations of $x_1^b, \ldots, x_{j-1}^b$. Compute $P_k(\mathbf{x}^b)F_{n-k}(\mathbf{x}^b)$ at the beginning of function call paths(j,x). If $P_k(\mathbf{x}^b)F_{n-k}(\mathbf{x}^b) = 0$ then return 0.

The walk polynomial $P_k(\mathbf{x})$ is zero if the vertices corresponding to zero-valued entries form an s-t cut set, for example. Hence, the reduction is more likely when the graph is sparse. The finishing polynomial $F_{n-k}(\mathbf{x})$ is zero if the number of zero-valued entries is greater than k. So the reduction is more useful when counting shorter paths.

4.2.2 Vestigial Elements

At the beginning of function call paths(j,x), remove from the graph the vertices corresponding to zero-valued $x_1, \ldots, x_{j-1}$. If no vertex in j to n is in the connected component with s and t, then the presence of vertices in j to n does not affect the number of s-t walks. So $P_k(\mathbf{x})$ is constant over all continuations.

Let c be the number of one-valued $x_1, \ldots, x_{j-1}$. Let $m = n-j+1$ be the number of "vestigial" vertices. There are $\binom{m}{i}$ continuations with i one-valued entries in $x_j, \ldots, x_n$. Since these continuations have $c+i$ one-valued entries, $F_{n-k}(\mathbf{x}) = \binom{c+i}{n-k}$. These continuations have sign $(-1)^{n-(c+i)}$. Hence, the function call can return

$$P_k(\mathbf{x}^b) \sum_{i=0}^{m} (-1)^{n-(c+i)} \binom{m}{i} \binom{c+i}{n-k} \tag{4.1}$$

4.2.3 Symmetries

Again, consider the beginning of function call $\texttt{paths(j,x)}$. Remove from the graph the vertices corresponding to zero-valued $x_1, \ldots, x_{j-1}$. Suppose vertices j to $j+m-1$ have identical adjacencies, i.e., any permutation of their labels results in the same labelled graph. Then the number of one-valued $x_j, \ldots, x_{j+m-1}$ determines $P_k(\mathbf{x})F_{n-k}(\mathbf{x})$ for continuations of $x_1, \ldots, x_{j-1}, x_j, \ldots, x_{j+m-1}$. The polynomials $P_k(\mathbf{x})$ and $F_{n-k}(\mathbf{x})$ have the same values regardless of which particular variables $x_j, \ldots, x_{j+m-1}$ are one-valued. Thus, $\texttt{paths(j + m, x)}$ returns the same value as $\texttt{paths(j + m, x}')$ if $\mathbf{x}$ and $\mathbf{x}'$ have the same number of one-valued variables corresponding to symmetric vertices.

The function call $\texttt{paths(j,x)}$ can jump ahead m levels of recursion, computing a single case of $\texttt{paths(j + m, x)}$ for each number i of one-valued entries. So the function call can be amended:

```
if j, ..., j + m − 1 are symmetric vertices
then return $\sum_{i=0}^{m}(-1)^{m-i}\binom{m}{i}$ paths(j + m, x + $e^j$ + ... + $e^{j+i-1}$)
```

A fast test for vertex symmetry is given in [4].

4.3 Ordering Finite-Difference Evaluations

The reductions depend on the order of assignment to variables $x_1, \ldots, x_n$. Some dependencies are not true restrictions; they were imposed to simplify the presentation. These dependencies can be overcome by dynamically reordering the assignments. For example, suppose vertices j and $j+3$ are symmetric. To use the symmetry reduction, swap the labels on vertices $j+1$ and $j+3$. Now j and $j+1$ are symmetric, and the reduction can be used as presented.

In general, the labels on vertices j to n may be permuted at the start of `paths(j,x)` without altering the result. To prove this, recall that `paths(j,x)` computes $D_j \cdots D_n$. Permuting labels is equivalent to reordering the evaluation of finite-difference operators. Since finite-difference operators commute, the result remains the same.

A previous paper [4] contains more details about the reductions presented here and about strategies to reduce computation by reordering the recursion.

Chapter 5

Finite-Difference Algorithms for Other Problems

5.1 Introduction

To demonstrate the methods introduced in previous chapters, we now develop finite-difference algorithms for a few other problems. Two problems, a sequencing problem and a bin packing problem, are taken from a paper by Karp [24] in which inclusion and exclusion algorithms are developed for the problems. (The paper also contains references to dynamic programming algorithms.) Another problem, involving deadlock avoidance, is taken from a paper by Gold [14].

For each problem, we derive a generating function for which the multilinear terms correspond to objects or configurations that we wish to count. Then, the counting algorithm amounts to applying the finite differences $D_1 \cdots D_n$ to the generating function. Finally, we outline algorithms for variations of the problems, including associated optimization problems and counting alternative objects or configurations.

5.2 Sequencing

5.2.1 The Problem

The sequencing problem is defined as follows [24]:

"We are given a set of tasks $T_1, T_2, \ldots, T_n$. Three integers are associated with each task T_i: a positive execution time $l(i)$, a non-negative release time $r(i)$, and a positive deadline $d(i)$. We wish to determine if there exists a one-processor schedule starting at time 0 such that the execution of each task takes place within the interval bounded by its release time and its deadline."

5.2.2 A Finite-Difference Algorithm

Define a feasible schedule for a multiset of tasks to be a sequence of disjoint open intervals with the following properties:

1. Each interval (a, b) is labelled by a task T_i such that interval length equals task execution time ($b - a = l(i)$), the beginning of the interval is no earlier than the release time ($a \geq r(i)$), and the end of the interval is no later than the deadline ($b \leq d(i)$).

2. The multiset of interval labels equals the multiset of tasks.

Define $s_{t,k}$ to be the number of feasible schedules for size k multisets of tasks such that all tasks complete by time t. We can count these feasible schedules by dynamic programming. To that end, define

$$E(t) = \{i | r(i) \leq t - l(i) \text{ and } t \leq d(i)\} \; \forall t \in \{0, \ldots, T\} \tag{5.1}$$

as in [24]. Set $E(t)$ indexes the tasks that may complete at time t without violating their time constraints. The following dynamic program computes $s_{t,k}$ for $0 \leq t \leq T$ and $0 \leq k \leq n$.

$$s_{t,0} = 1 \text{ for } t \geq 0 \tag{5.2}$$

$$s_{0,k} = 0 \text{ for } k > 0 \tag{5.3}$$

$$s_{t,k} = s_{t-1,k} + \sum_{i \in E(t)} s_{t-l(i),k-1} \text{ for } t = 1, \ldots, T \text{ and } k = 1, \ldots, n \tag{5.4}$$

Let $T = \max_i d(i)$. Then $s_{T,n}$ is the number of feasible schedules for multisets of n tasks. We wish to determine whether or not there is a feasible schedule that contains one occurrence of each task. For that purpose, define

$$s_{t,k}(\mathbf{x}) = \sum_{p_1 + \ldots + p_n = k} c_t(p_1, \ldots, p_n) x_1^{p_1} \cdots x_n^{p_n} \tag{5.5}$$

where $c_t(p_1, \ldots, p_n)$ is the number of feasible schedules that complete by time t for the multiset with p_i occurrences of task t_i for $1 \leq i \leq n$. We wish to determine whether or not

$c_T(1,\ldots,1) > 0$. Since $c_T(1,\ldots,1)$ is the multilinear term of $s_{T,n}(\mathbf{x})$,

$$c_T(1,\ldots,1) = D_1 \cdots D_n s_{T,n}(\mathbf{x}) \tag{5.6}$$

To compute $s_{T,n}(\mathbf{x})$ by dynamic programming, examine the program for $s_{t,k}$. To convert the computation of $s_{t,k}$ to a computation of $s_{t,k}(\mathbf{x})$, we must multiply by x_i when we add task t_i to schedules. The sum in (5.4) represents the addition of task T_i to schedules, so the dynamic program is:

$$s_{t,0}(\mathbf{x}) = 1 \text{ for } t \geq 0 \tag{5.7}$$

$$s_{0,k}(\mathbf{x}) = 0 \text{ for } k > 0 \tag{5.8}$$

$$s_{t,k}(\mathbf{x}) = s_{t-1,k}(\mathbf{x}) + \sum_{i\in E(t)} s_{t-l(i),k-1}(\mathbf{x})x_i \text{ for } t = 1,\ldots,T \text{ and } k = 1,\ldots,n \tag{5.9}$$

5.2.3 Complexity

The time required to compute $s_{T,n}(\mathbf{x})$ is $O(Tn^2)$, so the time required to compute $c_T(1,\ldots,1)$ by (5.6) is $O(2^nTn^2)$. The space requirement is $O(Tn)$ plus enough space to store the sum of $s_{T,n}(\mathbf{x})$ over 2^n assignments to $\mathbf{x}$.

5.2.4 Variations

Our sequencing problem is an existence problem. An associated optimization problem is to determine the minimum amount of time required to accomplish the tasks. To solve this problem, first solve the existence problem with $T = \max_i d(i)$. If there is no feasible schedule for the tasks, then the optimization problem has no solution. Otherwise, reduce T by one and resolve the existence problem. Repeat until there is no feasible schedule for the tasks. The solution to the optimization problem is the last value of T for which there is a feasible schedule. Since there is no feasible schedule in which task T_i completes before $r(i) + l(i)$, the existence problem will need to be solved at most

$$\max_i d(i) - \max_i r(i) + l(i) \tag{5.10}$$

times to solve the optimization problem.

To save computation, solve all of the existence problems using a single pass through the 2^n assignments to $\mathbf{x}$ for the finite differences and a single pass through the dynamic programming procedure for $s_{\max_i d(i),n}(\mathbf{x})$ for each assignment. In the dynamic programming procedure, we compute the values $s_{t,k}(\mathbf{x})$ corresponding to the other existence problems. Instead of summing only $s_{\max_i d(i),n}(\mathbf{x})$ over the assignments, sum $s_{\max_i r(i)+l(i),n}(\mathbf{x}), \ldots, s_{\max_i d(i),n}(\mathbf{x})$. The least T such that $c_T(1, \ldots, 1) > 0$ is the solution to the optimization problem.

To count feasible schedules for a multiset of tasks other than the complete set, use the finite-difference operators for non-multilinear terms that were introduced in Chapter 2. For example, suppose we wish to count feasible schedules for two instances of task T_1, three instances of T_2, no instances of T_3, and one instance of T_4. Examine (5.5). We wish to compute $c_T(2,3,0,1)$. By (2.29),

$$c_T(2,3,0,1) = D_1^{(2)} D_2^{(3)} D_4^{(1)} s_{T,6}(\mathbf{x}) \tag{5.11}$$

If two tasks in the original problem have identical specifications ($l(i) = l(j)$, $r(i) = r(j)$, and $d(i) = d(j)$,) then, instead of solving the problem for the complete set, solve the non-multilinear term problem for two instances of T_i and no instance of T_j. Then multiply the result by 2! since the two instances of T_i may represent either t_i followed by T_j or T_j followed by T_i. The solution is the same, and there is a reduction in computation, as outlined at the end of Chapter 2.

To count feasible schedules for all size k subsets of (distinct) tasks, use the technique of finishing polynomials that was developed in Chapter 3 to count paths and cycles by length. From (5.5), note that $s_{T,k}(\mathbf{x})$ consists of terms with coefficients that count feasible schedules for size k multisets of tasks and variables with exponents that count multiset members by task. We wish to compute the sum of coefficients over terms that correspond to multisets with k distinct tasks. These terms have k variables with nonzero exponents.

Recall the finishing polynomial (3.4):

$$F_{n-k}(\mathbf{x}) = \sum_{\{j_1,\ldots,j_{n-k}\} \subseteq \{1,\ldots,n\}} x_{j_1} \cdots x_{j_{n-k}} \tag{5.12}$$

The finishing polynomial contains one term corresponding to each size $n-k$ subset of (distinct) tasks. Consider the polynomial:

$$s_{T,k}(\mathbf{x}) F_{n-k}(\mathbf{x}) \tag{5.13}$$

For each term in $s_{T,k}(\mathbf{x})$ with k variables with nonzero exponents, there is one term in $F_{n-k}(\mathbf{x})$ that contributes the $n-k$ remaining variables to form a multilinear term. The other terms in $s_{T,k}(\mathbf{x})$ lack more than $n-k$ variables, so no term in $F_{n-k}(\mathbf{x})$ contributes enough variables to form a multilinear term. So the coefficient of the multilinear term of the product polynomial is the number of feasible schedules for subsets of k (distinct) tasks. Hence, the solution is:

$$D_1 \cdots D_n s_{T,k}(\mathbf{x}) F_{n-k}(\mathbf{x}) \tag{5.14}$$

5.3 Bin Packing

5.3.1 The Problem

The bin packing problem is defined in [24] as follows:

"An instance of the bin packing problem is specified by a positive integer bin capacity B, a positive integer m giving the number of bins, and n items, where the size of item i is given by a positive integer $s(i)$. The question is whether the set of items can be partitioned into sets $U_1, \ldots, U_m$ such that the sum of the sizes of the items in each u_j is B or less."

(For notational convenience, we use m for the number of bins; k is used in [24].)

5.3.2 A Finite-Difference Algorithm

Define a feasible packing for a multiset of items to be a sequence of m sequences of items with the following properties:

1. The items in the sequences of items form the multiset.
2. In each sequence of items, the sum of item sizes is no greater than B.
3. The sequences of items may be empty or may contain duplicates.

Define $d_{j,k}$ to be the number of sequences of k items (duplicates allowed) with sum of sizes j. We can compute $d_{j,k}$ for $0 \leq j \leq B$ and $0 \leq k \leq n$ as follows:

$$d_{0,0} = 1 \tag{5.15}$$

$$d_{0,k} = 0 \text{ for } k > 0 \tag{5.16}$$

$$d_{j,0} = 0 \text{ for } j > 0 \tag{5.17}$$

$$d_{j,k} = 0 \text{ for } j < 0 \tag{5.18}$$

$$d_{j,k} = \sum_{i=1}^{n} d_{j-s(i),k-1} \text{ for } j = 1, \ldots, B \text{ and } k = 1, \ldots, n \tag{5.19}$$

Define b_k to be the number of sequences of k (not necessarily distinct) items with sum of sizes less than B. Then $b_k = \sum_{j=0}^{B} d_{j,k}$. Now, define $t_{h,k}$ to be the number of feasible packings for multisets of k items with h (possibly empty) sequences of items. We can compute $t_{h,k}$ using the following dynamic program:

$$t_{0,0} = 1 \tag{5.20}$$

$$t_{0,k} = 0 \text{ for } k > 0 \tag{5.21}$$

$$t_{h,k} = \sum_{q=0}^{k} t_{h-1,q} b_{k-q} \text{ for } h = 1, \ldots, B \text{ and } k = 1, \ldots, n \tag{5.22}$$

Recall that there are m bins. We wish to determine whether or not $t_{m,n}$ counts a feasible packing for the full set of items $\{1, \ldots, n\}$. To that end, define

$$t_{h,k}(\mathbf{x}) = \sum_{p_1+\ldots+p_n=k} c_h(p_1, \ldots, p_n) x_1^{p_1} \cdots x_n^{p_n} \tag{5.23}$$

where $c_h(p_1, \ldots, p_n)$ is the number of feasible packings for the multiset with p_i occurrences of item i for $1 \leq i \leq n$. We wish to determine whether or not $c_m(1, \ldots, 1) > 0$. Since $c_m(1, \ldots, 1)$ is the multilinear term of $t_{m,n}(\mathbf{x})$,

$$c_m(1, \ldots, 1) = D_1 \cdots D_n t_{m,n}(\mathbf{x}) \tag{5.24}$$

To compute $t_{m,n}(\mathbf{x})$ by dynamic programming, we need to introduce variable x_i into terms when we count item i being added to a sequence. Examine (5.19). The sum corresponds to adding item i to a sequence. So insert x_i after the sum to define $d_{j,k}(\mathbf{x})$ by the dynamic program:

$$d_{0,0}(\mathbf{x}) = 1 \tag{5.25}$$

$$d_{0,k}(\mathbf{x}) = 0 \text{ for } k > 0 \tag{5.26}$$

$$d_{j,0}(\mathbf{x}) = 0 \text{ for } j > 0 \tag{5.27}$$

$$d_{j,k}(\mathbf{x}) = 0 \text{ for } j < 0 \tag{5.28}$$

$$d_{j,k}(\mathbf{x}) = \sum_{i=1}^{n} d_{j-s(i),k-1}(\mathbf{x}) x_i \text{ for } j = 1, \ldots, B \text{ and } k = 1, \ldots, n \tag{5.29}$$

Also, let $b_k(\mathbf{x}) = \sum_{j=0}^{B} d_{j,k}(\mathbf{x})$. Then $t_{h,k}(\mathbf{x})$ can be computed as follows:

$$t_{0,0}(\mathbf{x}) = 1 \tag{5.30}$$

$$t_{0,k}(\mathbf{x}) = 0 \text{ for } k > 0 \tag{5.31}$$

$$t_{h,k}(\mathbf{x}) = \sum_{q=0}^{k} t_{h-1,q} b_{k-q}(\mathbf{x}) \text{ for } h = 1, \ldots, B \text{ and } k = 1, \ldots, n \tag{5.32}$$

5.3.3 Complexity

The time required to compute $t_{m,n}(\mathbf{x})$ is $O(n^2 B)$, so the time required to solve the problem by computing the finite differences is $O(2^n n^2 B)$. The storage requirements are $O(nB)$ for the values $d_{j,k}(\mathbf{x})$, plus the storage needed to sum $w_{m,n}(\mathbf{x})$ over 2^n assignments to $\mathbf{x}$.

5.3.4 Variations

An associated optimization problem is to find the minimum number of bins required for all items. To solve this problem, solve the feasibility problem for

$$m \in \{\lceil \frac{\sum_{i=1}^{n} s(i)}{B} \rceil, \ldots, n\} \tag{5.33}$$

The value $\lceil \frac{\sum_{i=1}^{n} s(i)}{B} \rceil$ follows from the fact that the combined capacity of the bins must be at least as large as the sum of item sizes. The value n follows from the fact that if there is no feasible packing in which each item gets its own bin, then there is no feasible packing for any number of bins. The minimum value of m with a feasible solution is the solution to the optimization problem. A similar approach works for the problem of finding the minimum bin capacity B required.

To count feasible packings for multisets of items other than the complete set, $\{1, \ldots, n\}$, use the finite-difference operators for non-multilinear terms. For example, to count feasible packings for four instances of item 1 and two instances each of items 5 and 6, compute

$$c_m(4,0,0,0,2,2) = D_1^{(4)} D_5^{(2)} D_6^{(2)} t_{m,8}(\mathbf{x}) \tag{5.34}$$

As for the sequencing problem, if two (or more) items in the original bin packing problem have identical sizes, then, instead of solving the original problem for the multilinear term, solve the non-multilinear term problem for two (or more) instances of a single item with the size of the duplicated items. Once again, the answer is the same, and the computation is reduced.

To count feasible packings over all size k subsets of (distinct) items, use the finishing polynomial technique. Examine (5.23). Note that $t_{h,k}(\mathbf{x})$ is the sum of terms that have sum of exponents k. We desire the sum of the coefficients of terms with nonzero exponents on k distinct variables. Recall the finishing polynomial:

$$F_{n-k}(\mathbf{x}) = \sum_{\{j_1,\ldots,j_{n-k}\}\subseteq\{1,\ldots,n\}} x_{j_1}\cdots x_{j_{n-k}} \tag{5.35}$$

The finishing polynomial contains one term with each set of $n-k$ distinct variables. Hence, each term in $t_{m,k}(\mathbf{x})$ with k distinct variables produces one multilinear term in the polynomial:

$$t_{m,k}(\mathbf{x})F_{n-k}(\mathbf{x}) \tag{5.36}$$

So the feasible packings for subsets of k items are counted by

$$D_1\cdots D_n t_{m,k}(\mathbf{x})F_{n-k}(\mathbf{x}) \tag{5.37}$$

5.4 Deadlock Avoidance

A paper by Gold [14] analyzes a general deadlock avoidance problem and several constrained versions of the problem. We use the problem with the constraint that no partial requests are allowed. For in-depth analysis of deadlock problems, see [1, 14, 15, 19]. For a proof that the problem presented here is NP-complete, see [14].

5.4.1 The Problem

For a system with r resource types, an instance of the deadlock avoidance problem is specified by a nonnegative integer r-vector $\mathbf{a}_0$ that indicates how many units of each resource are available initially and n processes $P_1,\ldots,P_n$. Each process P_i is specified by a pair of nonnegative integer r-vectors $(\mathbf{y}_i,\mathbf{z}_i)$. If process P_i obtains the resources indicated by $\mathbf{y}_i$, then it eventually terminates and provides resources $\mathbf{z}_i$ to the available pool.

We wish to determine if there exists a sequence of processes $P_{s(1)},\ldots,P_{s(n)}$ in which each process occurs once:

$$\{s(1),\ldots,s(n)\} = \{1,\ldots,n\} \tag{5.38}$$

and no resource constraints are violated:

$$\mathbf{y}_{s(i)} \leq \mathbf{a}_0 + \sum_{j<i}(\mathbf{z}_{s(j)} - \mathbf{y}_{s(j)}) \quad \forall i \in \{1,\ldots,n\} \tag{5.39}$$

(By vector inequality $\mathbf{x} \leq \mathbf{y}$, we mean that the inequality holds for all positions.)

5.4.2 A Finite-Difference Algorithm

First, we develop a dynamic program to count sequences that obey the resource constraints (5.39) but may violate (5.38) by having no occurrences of some processes and multiple occurrences of others. Refer to these sequences as feasible sequences. Let $s_{\mathbf{a},k}$ be the number of such sequences of length k. Let m_j be the maximum units of resource j available during or after a sequence of length n:

$$m_j = [\mathbf{a}_0]_j + n[\max_i(z_{ij} - y_{ij})] \quad \forall j \in \{1, \ldots, r\} \tag{5.40}$$

Then

$$s_{\mathbf{a},0} = \begin{cases} 1 & \text{if } \mathbf{a} = \mathbf{a}_0 \\ 0 & \text{otherwise} \end{cases} \quad \forall \mathbf{a} \in \{0, m_1\} \times \ldots \times \{0, m_r\} \tag{5.41}$$

$$s_{\mathbf{a},k} = 0 \quad \forall \mathbf{a} \text{ such that } a_j < 0 \text{ for some } j \in \{1, \ldots, r\} \tag{5.42}$$

and

$$s_{\mathbf{a},k} = \sum_{i=1}^{n} s_{\mathbf{a}+\mathbf{y}_i-\mathbf{z}_i,k-1} \quad \forall \mathbf{a} \in \{0, m_1\} \times \ldots \times \{0, m_r\} \text{ and } k \in \{1, \ldots, n\} \tag{5.43}$$

The number of length n feasible sequences is:

$$\sum_{\mathbf{a} \in \{0,m_1\} \times \ldots \times \{0,m_r\}} s_{\mathbf{a},n} \tag{5.44}$$

To solve the problem, we need to count the length n feasible sequences that contain all processes. To that end, alter the dynamic program to index processes $P_1, \ldots, P_n$ by variables $x_1, \ldots, x_n$.

$$s_{\mathbf{a},0}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{a} = \mathbf{a}_0 \\ 0 & \text{otherwise} \end{cases} \quad \forall \mathbf{a} \in \{0, m_1\} \times \ldots \times \{0, m_r\} \tag{5.45}$$

$$s_{\mathbf{a},k}(\mathbf{x}) = 0 \quad \forall \mathbf{a} \text{ such that } a_j < 0 \text{ for some } j \in \{1, \ldots, r\} \tag{5.46}$$

$$s_{\mathbf{a},k}(\mathbf{x}) = \sum_{i=1}^{n} s_{\mathbf{a}+\mathbf{y}_i-\mathbf{z}_i,k-1}(\mathbf{x}) x_i \quad \forall \mathbf{a} \in \{0, m_1\} \times \ldots \times \{0, m_r\} \text{ and } k \in \{1, \ldots, n\} \tag{5.47}$$

and

$$P(\mathbf{x}) = \sum_{\mathbf{a} \in \{0,m_r\} \times \ldots \times \{0,m_r\}} s_{\mathbf{a},n}(\mathbf{x}) \tag{5.48}$$

Then $D_1 \cdots D_n P(\mathbf{x}) > 0$ if and only if there exists a sequence that meets conditions (5.38) and (5.39).

5.4.3 Complexity

The polynomial $P(\mathbf{x})$ can be evaluated in $O(n\prod_{j=1}^{r}(m_j+1))$ time, so $D_1\cdots D_nP(\mathbf{x})$ can be evaluated in $O(2^n n\prod_{j=1}^{n}(m_j+1))$ time. The computation requires $O(\prod_{j=1}^{n}(m_j+1))$ space.

5.4.4 Variations

To determine if it is possible to execute process P_i first and avoid deadlock, solve the deadlock avoidance problem for the remaining processes, with $\mathbf{z}_i - \mathbf{y}_i$ added to $\mathbf{a}_0$. To find a deadlock-free sequence of resource allocations (if one exists), first test each process to find a safe initial process. Then find a safe second process by finding a safe initial process among those remaining. Iterate to find safe processes for the subsequent positions in the ordering. Since $D_1\cdots D_nP(\mathbf{x})$ counts deadlock-free process orderings, if $D_1\cdots D_nP(\mathbf{x}) = n!$, then all process orderings are deadlock-free.

To count sequences that satisfy the resource constraints and contain multiple occurrences of some processes, use the finite-difference operators for non-multilinear terms from Chapter 2. For example, to count sequences with two occurrences of process P_1, three occurrences of process P_2, and one occurrence of P_3, evaluate

$$D_1^{(2)}D_2^{(3)}D_3^{(1)}P(\mathbf{x}) \tag{5.49}$$

where $P(\mathbf{x})$ is the generating function for sequences of 6 processes.

The same approach works for sets of processes with identical specifications. Represent each set by a single process from the set, and count sequences with as many occurrences of the process as there are members of the set. Multiply the result by the factorials of the set sizes since the occurrences of each representative process may be replaced by the processes in its set in any order to form a solution sequence for the original problem. For example,

$$2!3!D_1^{(2)}D_2^{(3)}D_3^{(1)}P(\mathbf{x}) \tag{5.50}$$

counts the sequences containing one instance of each process if the processes can be partitioned into a pair with the specifications of P_1, a triple with the specifications of P_2, and a single process with the specification of P_3. As discussed in Chapter 2, this problem transformation saves computation.

To count length k sequences that satisfy the resource constraints and contain k distinct processes, use the finishing polynomial technique. Define

$$P_k(\mathbf{x}) = \sum_{\mathbf{a} \in \{0,m_1\} \times \ldots \times \{0,m_r\}} s_{\mathbf{a},k}(\mathbf{x}) \tag{5.51}$$

and recall the finishing polynomial (3.4):

$$F_{n-k}(\mathbf{x}) = \sum_{\{j_1,\ldots,j_{n-k}\} \subseteq \{1,\ldots,n\}} x_{j_1} \cdots x_{j_{n-k}} \tag{5.52}$$

The multilinear terms of $P_k(\mathbf{x})F_{n-k}(\mathbf{x})$ correspond to sequences without repeated processes, so

$$D_1 \cdots D_n P_k(\mathbf{x}) F_{n-k}(\mathbf{x}) \tag{5.53}$$

counts these sequences.

5.5 Discussion

The examples in this chapter illustrate how to develop finite-difference algorithms for counting problems. The examples also illustrate how to develop algorithms for associated optimization problems and for variations of the counting problems. Although the techniques involving multilinear terms and finishing polynomials are illustrated separately, these techniques can be combined to produce algorithms for more complex problems. Likewise, the finishing polynomial may be altered to develop algorithms for even more problems. The key to using these techniques is understanding how they interact with one another and with the underlying generating function.

We have not considiered ways to choose finite-difference parameters in order to reduce computation for the finite-difference formulas developed in this chapter. Later, we will show that the proper choice of parameters enables dramatic computational reductions for the permanent problem. Finding such parameters (if they exist) for the problems presented here is a challenge for the future.

Chapter 6

Computing the Permanent of a 0-1 Matrix

In this chapter, we develop finite-difference algorithms to compute the permanent of a 0-1 matrix. First, we develop a finite-difference formula. Then we prove that some finite-difference parameter settings produce many zero-valued terms. Finally, we develop an algorithm that reduces computation by eliminating collections of zero-valued terms.

The permanent of an $n \times n$ 0-1 matrix is the number of ways to choose a set of one-valued entries with one entry in each row and one entry in each column. If the columns represent workers, the rows represent jobs, and one-valued entries represent worker-job compatibility, then the permanent is the number of ways to assign each worker to a job and have each job accomplished. Likewise, if the rows and columns represent the vertex partitions in a bipartite graph and one-valued entries represent edges, then the permanent is the number of complete matchings. Alternatively, if the matrix is the adjacency matrix of a directed graph, then the permanent is the number of directed cycle sets in which each vertex is on one cycle.

The permanent problem has applications to statistical physics, including problems involving monomer-dimer systems. For more information, see [30].

Computing the permanent is a #P-complete problem [32]. #P problems are the counting problems associated with NP problems [32, 13]. For example, determining whether or not a graph has a Hamiltonian cycle is an NP problem, so counting the Hamiltonian cycles is a #P problem. In this case, the existence problem is NP-complete [23, 13], and the counting problem is #P-complete [33]. In the case of the permanent, the existence problem – determining whether a bipartite graph has a complete matching – is in P [25, 10]. However, the counting problem is #P-complete.

Ryser made progress on computing the permanent by developing an inclusion and exclusion algorithm [29]. The algorithms presented here are a generalization of that inclusion and exclusion algorithm. The results in this chapter are taken from several papers [7, 8].

6.1 A Finite-Difference Formula for the Permanent

The permanent of an $n \times n$ 0-1 matrix is the number of n-sets of one-valued entries with one entry in each row and one entry in each column.

$$\text{per } A = \sum_{j_1 \cdots j_n} a_{1j_1} \cdots a_{nj_n} \tag{6.1}$$

where $j_1 \dots j_n$ is a permutation of $1 \dots n$. Suppose the rows represent women, the columns represent men, and the one-valued entries denote compatibility. Then the permanent counts the marriage arrangements in which each person has a compatible spouse.

The product of row sums counts the dating arrangements in which each woman chooses a compatible man. (Entry a_{ij} represents woman i choosing man j. Row sum i represents woman i's choices.)

$$(a_{11} + \dots + a_{1n}) \cdots (a_{n1} + \dots + a_{nn}) \tag{6.2}$$

$$= \prod_{i=1}^{n} \sum_{j=1}^{n} a_{ij} = \sum_{(j_1,\dots,j_n)\in\{1,\dots,n\}^n} a_{1j_1} \cdots a_{nj_n} \tag{6.3}$$

In some of these arrangements, several women choose the same man. The permanent terms correspond to arrangements in which each man is chosen by exactly one woman.

Define $[A(\mathbf{x})]_{ij} = [A]_{ij} x_j$, i.e., multiply each column j by x_j. Now each woman's choices are labelled.

$$P(\mathbf{x}) \equiv (a_{11}x_1 + \dots + a_{1n}x_n) \cdots (a_{n1}x_1 + \dots + a_{nn}x_n) \tag{6.4}$$

$$= \prod_{i=1}^{n} \sum_{j=1}^{n} a_{ij}x_j = \sum_{(j_1,\dots,j_n)\in\{1,\dots,n\}^n} a_{1j_1} \cdots a_{nj_n} x_{j_1} \cdots x_{j_n} \tag{6.5}$$

In the terms of the last sum, x_j occurs once for each woman who chooses man j. The permanent terms are those in which each x_j occurs exactly once. Hence, the permanent is the coefficient of the multilinear term of $P(\mathbf{x})$, and per $A = D_1 \cdots D_n P(\mathbf{x})$.

Different choices of finite-difference variables produce different formulas for the permanent.

$$\text{per } A = \frac{1}{(u_1 - v_1)\cdots(u_n - v_n)} \sum_{\mathbf{x}\in\{u_1,v_1\}\times\ldots\times\{u_n,v_n\}} (-1)^{s(\mathbf{x})} P(\mathbf{x}) \tag{6.6}$$

where $s(\mathbf{x})$ is the number of variables x_j set to v_j.

The formulas can be computed by evaluating $P(\mathbf{x})$ for each of the 2^n assignments to $\mathbf{x}$. This requires $O(2^n \text{ poly } n)$ time and $O(\text{poly } n)$ space. (The inclusion and exclusion formula by Ryser [29] is the finite-difference formula with $u_j = 1$ and $v_j = 0$ for all j.)

6.2 Zero-Valued Terms

We develop computational reductions by selecting finite-difference parameters that induce many zero-valued terms in formula (6.6). If $P(\mathbf{x})$ is zero, then the term corresponding to assignment $\mathbf{x}$ is zero. Recall that $P(\mathbf{x})$ is the product of row sums of $A(\mathbf{x})$.

$$P(\mathbf{x}) = (a_{11}x_1 + \ldots + a_{1n}x_n)\cdots(a_{n1}x_1 + \ldots + a_{nn}x_n) \tag{6.7}$$

If any row sum is zero, then $P(\mathbf{x})$ is zero. Assume A is a 0-1 matrix. Consider the finite-difference parameter setting: $u_j = 1$ and $v_j = -1$ for all j. For $\mathbf{x}$ drawn at random from $\{1, -1\} \times \ldots \times \{1, -1\}$, each row sum has expected value zero. Shortly, we will show that this zero-mean parameter setting produces many zero-valued terms. Now consider a different setting: for odd j, $u_j = 1$ and $v_j = 0$; for even j, $u_j = 0$ and $v_j = -1$. Like the previous setting, this one balances positive and negative entries in $\mathbf{x}$. But the new setting produces row sums with lower variance. We will show that this low-variance parameter setting produces even more zero-valued terms than the zero-mean setting.

Assume A is drawn at random, each entry having value one with probability $p < 1$ and having value zero with probability $q = 1 - p$. We prove that the expected fraction of nonzero-valued terms with the zero-mean parameter setting is $O(\frac{1}{\sqrt{n^{\frac{q}{p}} \log n}})$. Then we prove that the fraction goes to $O(\frac{1}{n^{\frac{q}{p}}\sqrt{\log n}})$ for the low-variance parameter setting.

Figure 6.1 compares expected fractions of zero-valued terms in the low-variance, the zero-mean, and the inclusion and exclusion formulas. The low-variance formula has the most zero-valued terms. Figure 6.2 displays expected fractions of zero-valued terms for the low-variance formula with various element probabilities p. Figure 6.3 shows that for sparse matrices, the expected fraction of zero-valued terms is nearly 100%.

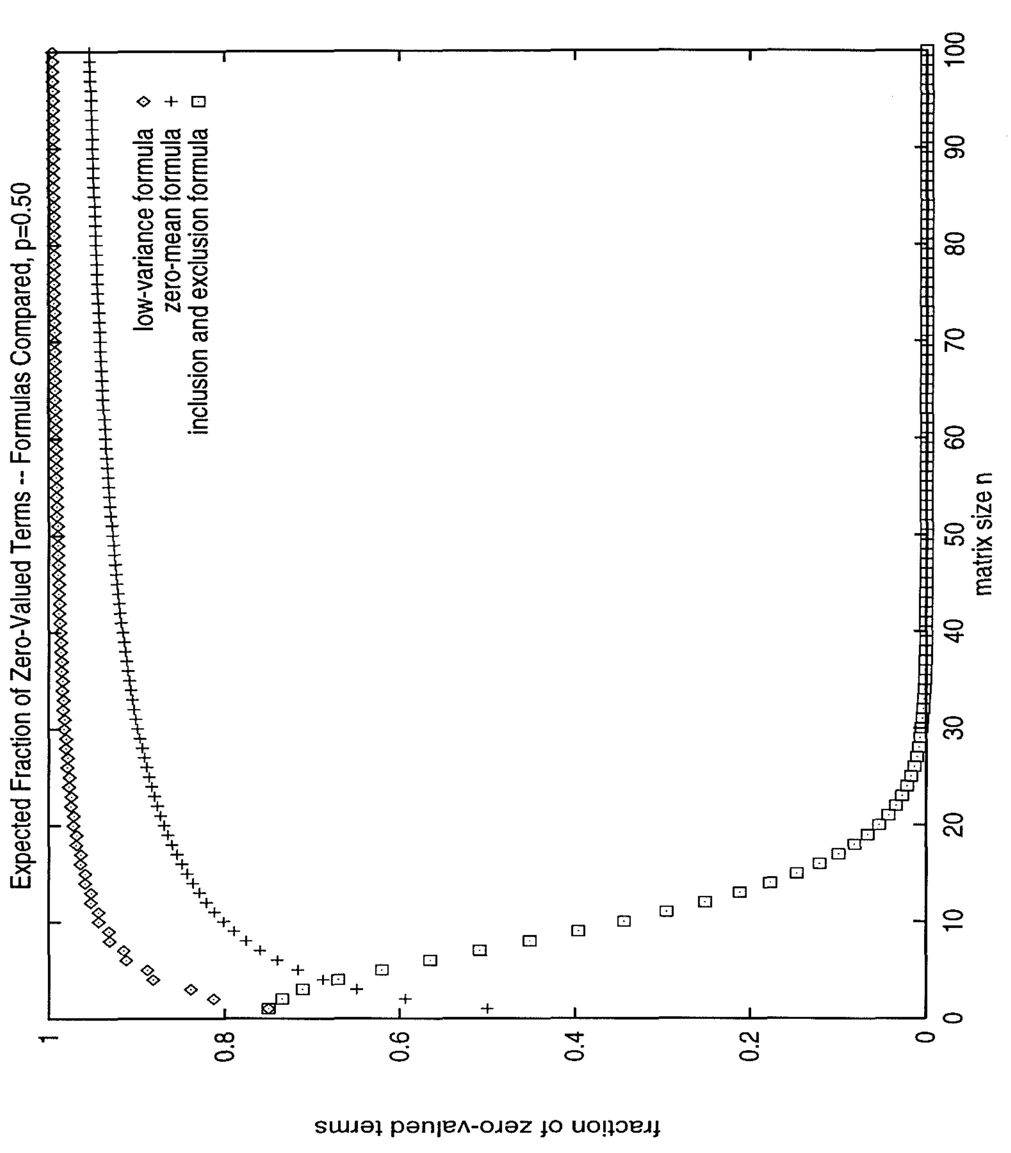

Figure 6.1: The low-variance formula has many-zero-valued terms. As matrix size increases, the expected fraction of zero-valued terms goes to 100% for the low-variance and zero-mean formulas.

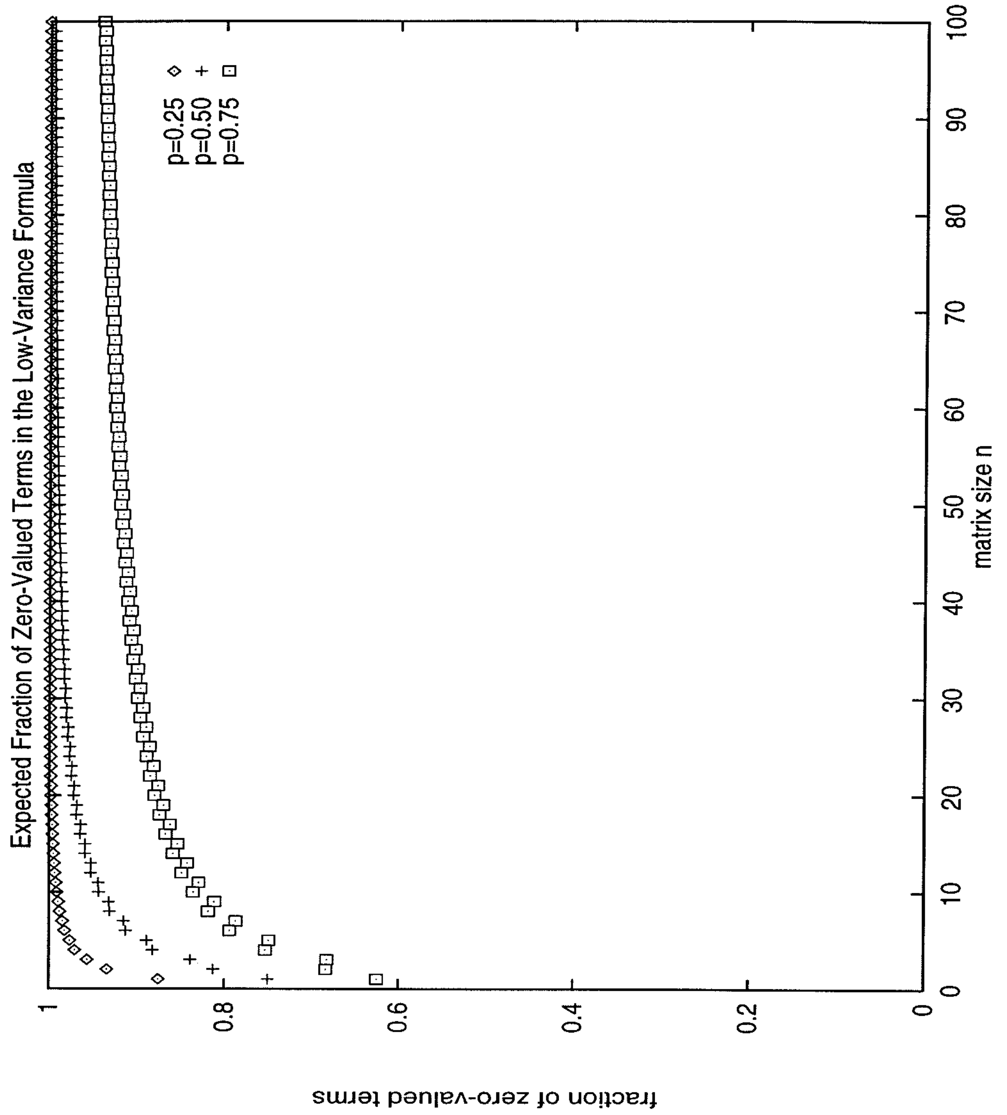

Figure 6.2: The expected fraction of zero-valued terms in the low-variance formula is shown for several matrix densities p.

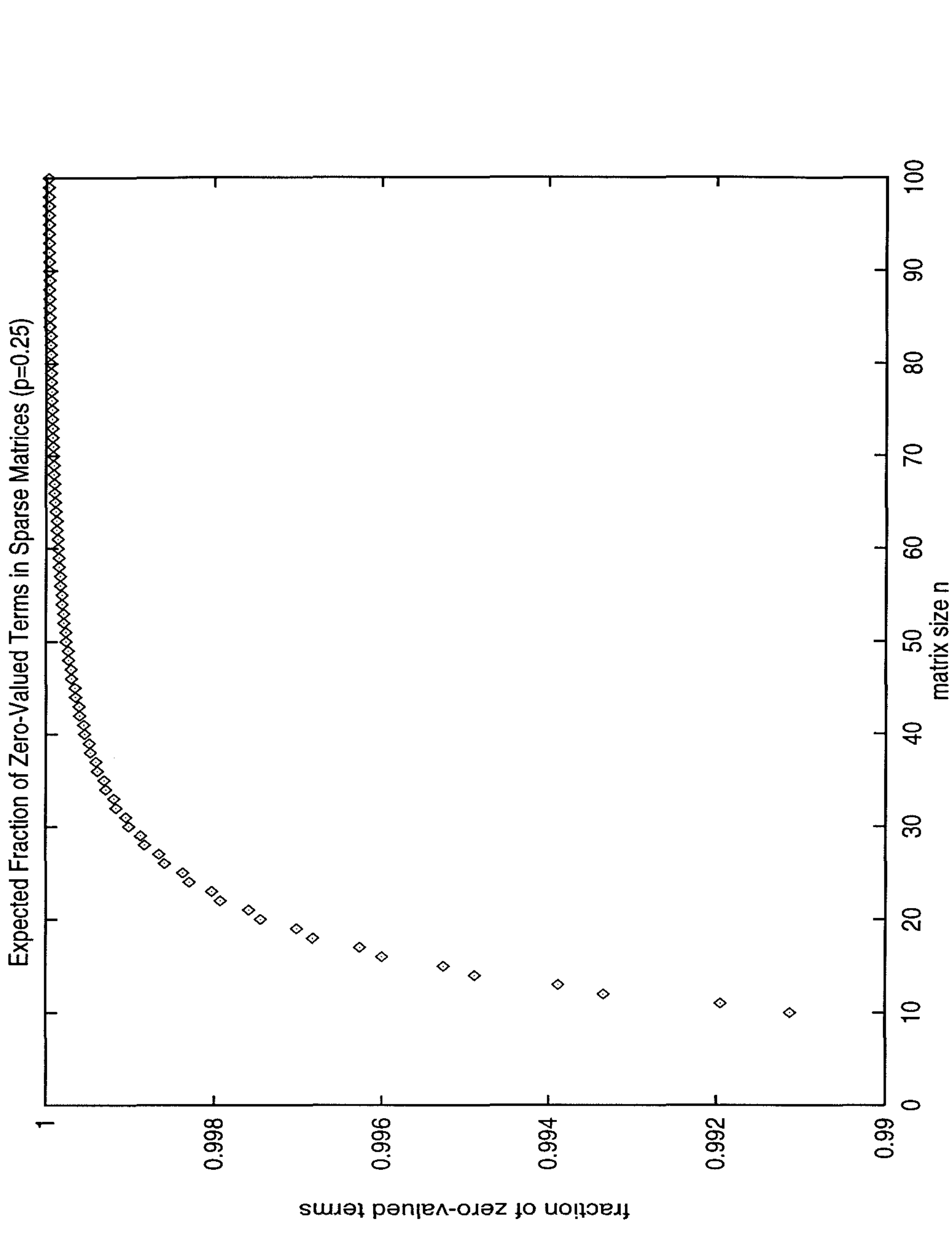

Figure 6.3: The expected fraction of zero-valued terms in the low-variance formula is nearly 100% for sparse matrices ($p = 0.25$). Note that the vertical axis varies from 99% to 100%.

6.3 Expected Fraction of Nonzero-Valued Terms in the Zero-Mean Formula

Given $\mathbf{x}$, the row sums are i.i.d. So the expected fraction of nonzero-valued terms is

$$\frac{1}{2^n} \sum_{\mathbf{x} \in \{1,-1\}^n} [1 - \Pr\{a_1 x_1 + \ldots + a_n x_n\}]^n \tag{6.8}$$

where $a_j = 1$ with probability $0 < p < 1$ and $a_j = 0$ with probability $q = 1 - p$. Let k be the number of variables x_j assigned one.

$$= \sum_{k=0}^{n} \binom{n}{k} \frac{1}{2^n} [1 - \Pr\{(a_1 + \ldots + a_k) - (a_{k+1} + \ldots + a_n) = 0\}]^n \tag{6.9}$$

To compute (6.9), note that the row sum is zero if equal numbers of $a_1, \ldots, a_k$ and $a_{k+1}, \ldots, a_n$ are one. Let ν be the number of $a_1, \ldots, a_k$ with value one, and sum over cases.

$$Pr\{a_1 + \ldots + a_k - (a_{k+1} + \ldots + a_n) = 0\} = \sum_{\nu=0}^{\min(k,n-k)} C(k,\nu) p^\nu q^{k-\nu} C(n-k,\nu) p^\nu q^{(n-k)-\nu} \tag{6.10}$$

(For a dynamic programming procedure to compute the fraction of nonzero-valued terms for general finite-difference parameters, see [7].)

To find an upper bound for (6.9) as $n \to \infty$, we will use several results from probability theory [11]. First, approximate the binomial distribution $\binom{n}{k}\frac{1}{2^n}$ by the normal distribution $\frac{2}{\sqrt{n}} g(\frac{2}{\sqrt{n}}(k - \frac{n}{2}))$, where $g(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}$. The terms in the tails of the distribution, $|k - \frac{n}{2}| > n^{\frac{1}{2}+\epsilon}$, contribute an exponentially small sum, so they can be ignored. Also, approximate the row sum by a normal distribution with mean $(2k - n)p$ and variance npq.

$$\sim \sum_{|k-\frac{n}{2}|<n^{\frac{1}{2}+\epsilon}} \frac{2}{\sqrt{n}} g(\frac{2k-n}{\sqrt{n}}) [1 - \frac{1}{\sqrt{npq}} g(\frac{(2k-n)p}{\sqrt{npq}})]^n \tag{6.11}$$

(For a detailed justification of this approximation, refer to Appendix A.) Let $t_k = \frac{2k-n}{\sqrt{n}}$ and $\Delta t = \frac{2}{\sqrt{n}}$.

$$\sim \int dt\, g(t) [1 - \frac{1}{\sqrt{npq}} g(\sqrt{\frac{p}{q}} t)]^n \tag{6.12}$$

The limits of integration may be taken as $\pm\infty$ or as $\pm 2n^\epsilon$; the same asymptotic form results.

Observe that

$$[1-\frac{1}{\sqrt{npq}}g(\sqrt{\frac{p}{q}}t)]^n \sim \exp(-\sqrt{\frac{n}{pq}}g(\sqrt{\frac{p}{q}}t))\exp(-\frac{1}{2pq}g^2(\sqrt{\frac{p}{q}}t)) \tag{6.13}$$

We can ignore the second exponential function since it is $1+O(t^2)$ for $t \to 0$. So the fraction of nonzero-valued terms is

$$\sim 2\int_0^\infty g(t)\exp(-\sqrt{\frac{n}{pq}}g(\sqrt{\frac{p}{q}}t))dt \tag{6.14}$$

Let $r = \frac{q}{p}$, and make the change of variable $w = \exp(\frac{-t^2}{2r})$.

$$\sim \sqrt{\frac{r}{\pi}}\int_0^1 w^{r-1}(\log\frac{1}{w})^{-\frac{1}{2}}e^{-\lambda w}dw \tag{6.15}$$

where $\lambda = \sqrt{\frac{n}{2\pi pq}}$.

Partition the integral into three regions, 0 to $\lambda^{-\frac{1}{2}}$, $\lambda^{-\frac{1}{2}}$ to $\frac{1}{2}$, and $\frac{1}{2}$ to 1. Note that $(\log\frac{1}{w})^{-\frac{1}{2}}$ increases from 0 to ∞ as w goes from 0 to 1. We will see that the first region dominates the integral as $\lambda \to \infty$.

Examine the first region.

$$\int_0^{\lambda^{-\frac{1}{2}}} w^{r-1}(\log\frac{1}{w})^{-\frac{1}{2}}e^{-\lambda w}dw \tag{6.16}$$

Note that $(\log\frac{1}{w})^{-\frac{1}{2}}$ is maximized at $w = \lambda^{-\frac{1}{2}}$.

$$< \sqrt{\frac{2}{\log\lambda}}\int_0^{\lambda^{-\frac{1}{2}}} w^{r-1}e^{-\lambda w}dw \tag{6.17}$$

Make the change of variable $w = \frac{u}{\lambda}$.

$$< \sqrt{\frac{2}{\log\lambda}}\lambda^{-r}\int_0^\infty u^{r-1}e^{-u}du \tag{6.18}$$

The integral is $\Gamma(r)$.

$$= \sqrt{\frac{2}{\log\lambda}}\lambda^{-r}\Gamma(r) \tag{6.19}$$

Now examine the second region.

$$\int_{\lambda^{-\frac{1}{2}}}^{\frac{1}{2}} w^{r-1}(\log\frac{1}{w})^{-\frac{1}{2}}e^{-\lambda w}dw \tag{6.20}$$

Note that $(\log\frac{1}{w})^{-\frac{1}{2}}$ is maximized at $w = \frac{1}{2}$.

$$< \frac{1}{\sqrt{\log 2}}\int_{\lambda^{-\frac{1}{2}}}^{\frac{1}{2}} w^{r-1}e^{-\lambda w}dw \tag{6.21}$$

The expression in the integral is maximized over positive numbers at $w = \frac{r-1}{\lambda}$. As $\lambda \to \infty$, $\frac{r-1}{\lambda} < \lambda^{-\frac{1}{2}}$, so the expression is maximized over the region of integration at $w = \lambda^{-\frac{1}{2}}$. Substituting the maximum value produces the bound

$$< \frac{1}{\sqrt{\log 2}}\lambda^{-\frac{1}{2}(r-1)}e^{-\lambda^{\frac{1}{2}}} \tag{6.22}$$

which is dominated by (6.19), the integral over the first region.

Examine the third region.

$$\int_{\frac{1}{2}}^{1} w^{r-1}(\log\frac{1}{w})^{-\frac{1}{2}}e^{-\lambda w}dw \tag{6.23}$$

Note that $e^{-\lambda w}$ is maximized at $w = \frac{1}{2}$.

$$< e^{-\frac{\lambda}{2}}\int_{\frac{1}{2}}^{1} w^{r-1}(\log\frac{1}{w})^{-\frac{1}{2}}dw \tag{6.24}$$

Since $(\log\frac{1}{w})^{-\frac{1}{2}}$ is $O((1-w)^{-\frac{1}{2}})$ as $w \to 1$, the integral is a finite constant. Hence, the integral over the third region is dominated by (6.19), the integral over the first region.

Substitute (6.19) into (6.15) to derive the following asymptotic upper bound for the fraction of nonzero-valued terms.

$$< \sqrt{\frac{2r}{\pi\log\lambda}}\lambda^{-r}\Gamma(r) \text{ as } \lambda \to \infty \tag{6.25}$$

Expand λ.

$$= \sqrt{\frac{4r}{\pi}}(2\pi pq)^{\frac{r}{2}}\Gamma(r)\frac{1}{\sqrt{n^r\log n}} \tag{6.26}$$

Figure 6.4 compares this asymptotic upper bound for the actual expected fraction of nonzero-valued terms. (The figure shows the corresponding lower bound for the expected fraction of zero-valued terms.)

Figure 6.4 also shows an asymptotic approximation, which is derived as follows. To evaluate (6.15), we partition the integral into three regions, 0 to $\lambda^{-\frac{1}{2}}$, $\lambda^{-\frac{1}{2}}$ to $\frac{1}{2}$, and $\frac{1}{2}$ to 1. Instead, consider the regions 0 to $\lambda^{-(1-\epsilon)}$, $\lambda^{-(1-\epsilon)}$ to $\frac{1}{2}$, and $\frac{1}{2}$ to 1 for any fixed $\epsilon > 0$. The upper bound for integral over the first region, (6.19), becomes

$$\sqrt{\frac{1}{(1-\epsilon)\log\lambda}}\lambda^{-r}\Gamma(r) \tag{6.27}$$

The upper bound for the integral over the second region, (6.22), becomes

$$\frac{1}{\sqrt{\log 2}}\lambda^{-(1-\epsilon)(r-1)}e^{-\lambda^{\epsilon}} \tag{6.28}$$

This is dominated by the first region, (6.27). The upper bound for the third region, (6.24), remains the same. It is dominated by the upper bound for the first region, (6.27). Substitute (6.27) into (6.15) and expand λ to derive the following asymptotic upper bound for the fraction of nonzero-valued terms.

$$\sqrt{\frac{2r}{(1-\epsilon)\pi}}(2\pi pq)^{\frac{r}{2}}\Gamma(r)\frac{1}{\sqrt{n^r\log n}} \tag{6.29}$$

Note that (6.26) is the special case of this bound with $\epsilon = \frac{1}{2}$. Setting $\epsilon = 0$ gives the approximation which is plotted in Figure (6.4).

6.4 Expected Fraction of Nonzero-Valued Terms in the Low-Variance Formula

First, we will exhibit a correspondence between $\mathbf{x}$ values in the low-variance formula and $\mathbf{x}$ values in the zero-mean formula. Row sums in the low-variance formula have half the expected value and, on average, half the variance of corresponding row sums in the zero-mean formula. We will alter the previous derivation according to these differences to derive an asymptotic upper bound for the fraction of nonzero-valued terms in the low-variance formula.

For simplicity, assume n is even. Given $\mathbf{x} \in \{1, -1\}^n$ (the zero-mean formula), generate the corresponding $\mathbf{x}$ in the low variance formula as follows. For odd j, if $x_j = -1$ then change it to zero; for even j, if $x_j = 1$ then change it to zero. Some examples:

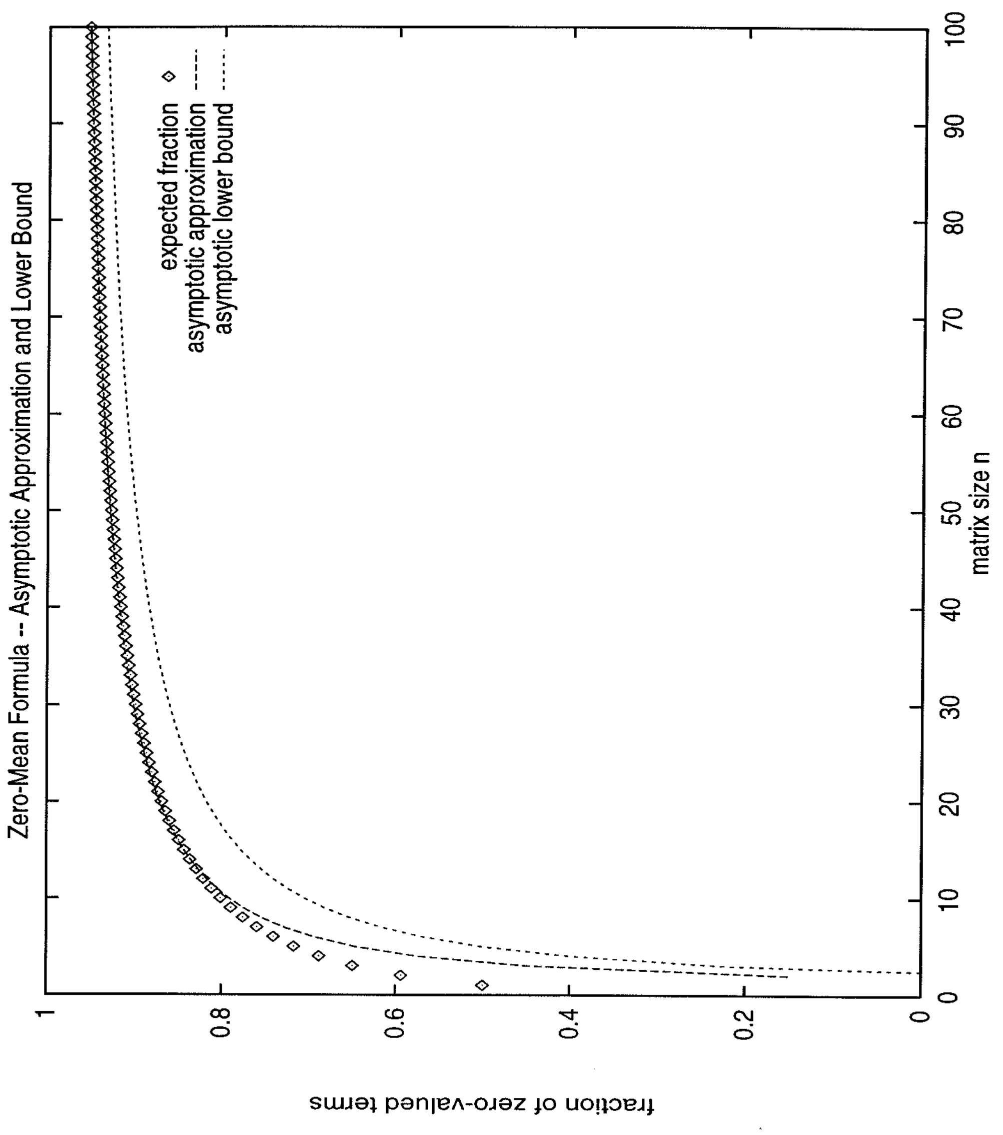

Figure 6.4: For the expected fraction of zero-valued terms in the zero-mean formula with $p = 0.50$, the asymptotic approximation and asymptotic lower bound are shown with the actual values for various matrix sizes n.

zero-mean **x**	low-variance **x**
(1,1,1,1,1,1)	(1,0,1,0,1,0)
(1,-1,-1,1,1,1)	(1,-1,0,0,1,0)

Note that the sum of entries in the low-variance **x** is half the sum in the zero-mean **x**. To prove this, we show that the sum is half for every corresponding adjacent pair (x_j, x_{j+1}). (Assume j is odd.)

zero-mean		low-variance	
(x_j, x_{j+1})	sum	(x_j, x_{j+1})	sum
-1 -1	-2	0 -1	-1
1 -1	0	1 -1	0
-1 1	0	0 0	0
1 1	2	1 0	1

Each row sum $a_1x_1 + \ldots + a_nx_n$ has half the expected value for each low-variance **x** as for the corresponding zero-mean **x**.

The number of nonzero entries in a random low-variance **x** is binomially distributed, i.e., **x** has m nonzero entries with probability $\binom{n}{m}\frac{1}{2^n}$. So the variance of row sum $a_1x_1+\ldots+a_nx_n$ is binomially distributed, with variance mpq occurring with probability $\binom{n}{m}\frac{1}{2^n}$. The terms in the tails of the distribution, $|m - \frac{n}{2}| > n^{\frac{1}{2}+\epsilon}$, are an exponentially small fraction of the terms, so they can be ignored in our derivation of the asymptotic upper bound.

To find the asymptotic upper bound for the fraction of nonzero-valued terms in the low-variance formula, we use the approximations made to analyze the zero-mean formula, but halve the mean row sum and consider only terms with variance

$$mpq \in [\frac{1}{2}(1 - \frac{2}{n^{\frac{1}{2}-\epsilon}})npq, \frac{1}{2}(1 + \frac{2}{n^{\frac{1}{2}-\epsilon}})npq] \tag{6.30}$$

Modify (6.11) to get the fraction

$$\sim \sum_{|k-\frac{n}{2}|<n^{\frac{1}{2}+\epsilon}} \frac{2}{\sqrt{n}} g(\frac{2k-n}{\sqrt{n}})[1 - \frac{1}{\sqrt{mpq}} g(\frac{\frac{1}{2}(2k-n)p}{\sqrt{mpq}})]^n \tag{6.31}$$

We bound this expression by taking the maximizing variance mpq for each term in the sum. Each term is maximized when

$$\frac{1}{\sqrt{mpq}} g(\frac{\frac{1}{2}(2k-n)p}{\sqrt{mpq}}) \tag{6.32}$$

is minimized. Let $x = \frac{1}{2}(2k-n)p$ and $\sigma = \sqrt{mpq}$. Then (6.32) has the form

$$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{x^2}{\sigma^2}} \tag{6.33}$$

This expression is minimized with respect to σ^2 when $\sigma^2 = x^2$. Also, when σ^2 is restricted to a range $[\sigma^2_{\mathrm{min}}, \sigma^2_{\mathrm{max}}]$, expression (6.33) is minimized by

$$\sigma^2 = \begin{cases} \sigma^2_{\mathrm{min}} & x^2 < \sigma^2_{\mathrm{min}} \\ x^2 & \sigma^2_{\mathrm{min}} \le x^2 \le \sigma^2_{\mathrm{max}} \\ \sigma^2_{\mathrm{max}} & \sigma^2_{\mathrm{max}} < x^2 \end{cases} \tag{6.34}$$

Hence, we bound (6.31) by partitioning into three sets of terms and assigining the worst-case value of $\sigma^2 = mpq$ for each.

$$mpq = \begin{cases} \frac{1}{2}(1-\frac{2}{n^{\frac{1}{2}-\epsilon}})npq & (\frac{1}{2}(2k-n)p)^2 < \frac{1}{2}(1-\frac{2}{n^{\frac{1}{2}-\epsilon}})npq \\ (\frac{1}{2}(2k-n)p)^2 & \frac{1}{2}(1-\frac{2}{n^{\frac{1}{2}-\epsilon}})npq \le (\frac{1}{2}(2k-n)p)^2 \le \frac{1}{2}(1+\frac{2}{n^{\frac{1}{2}-\epsilon}})npq \\ \frac{1}{2}(1+\frac{2}{n^{\frac{1}{2}-\epsilon}})npq & \frac{1}{2}(1+\frac{2}{n^{\frac{1}{2}-\epsilon}})npq < (\frac{1}{2}(2k-n)p)^2 \end{cases} \tag{6.35}$$

We will find that the first and second sets contribute exponentially small fractions to (6.31), so the third set determines the asymptotic fraction of nonzero-valued terms.

Consider the first set of terms. Examine (6.31). Note that $\frac{2}{\sqrt{n}}g(\frac{2k-n}{\sqrt{n}})$ approximates a p.d.f., so it weights each set of terms by O(1) or less. We could bound the contribution of the first set of terms by setting $mpq = \frac{1}{2}(1-\frac{2}{n^{\frac{1}{2}-\epsilon}})npq$ in the remaining part of the expression. Instead, we set $mpq = (\frac{1}{2}(2k-n)p)^2$ to achieve a looser but simpler bound.

$$[1 - \frac{1}{\frac{1}{2}(2k-n)p} g(\frac{\frac{1}{2}(2k-n)p}{\frac{1}{2}(2k-n)p})]^n \tag{6.36}$$

$$= [1 - \frac{1}{\frac{1}{2}(2k-n)p} g(1)]^n \tag{6.37}$$

Recall that $(\frac{1}{2}(2k-n)p)^2 < \frac{1}{2}(1-\frac{2}{n^{\frac{1}{2}-\epsilon}})npq$ for the first set of terms.

$$< [1 - \frac{1}{\sqrt{\frac{1}{2}(1-\frac{2}{n^{\frac{1}{2}-\epsilon}})npq}} g(1)]^n \tag{6.38}$$

The denominator is $O(\sqrt{n})$, so the expression is $O(e^{-\sqrt{n}})$. Hence, the first set of terms contributes an exponentially small fraction to the sum.

Now consider the second set of terms. As for the previous case, set $mpq = (\frac{1}{2}(2k-n)p)^2$. Once again, we get

$$= [1 - \frac{1}{\frac{1}{2}(2k-n)p} g(1)]^n \tag{6.39}$$

Recall that $(\frac{1}{2}(2k-n)p)^2 < \frac{1}{2}(1+\frac{2}{n^{\frac{1}{2}-\epsilon}})npq$ for the second set of terms.

$$< [1 - \frac{1}{\sqrt{\frac{1}{2}(1+\frac{2}{n^{\frac{1}{2}-\epsilon}})npq}} g(1)]^n \tag{6.40}$$

The denominator is $O(\sqrt{n})$, so the expression is $O(e^{-\sqrt{n}})$. Hence, the second set of terms contributes an exponentially small fraction to the sum.

Finally, consider the third set of terms. To derive an asymptotic upper bound on their contribution to the sum (6.31), assume that all terms are in the third set. Set $mpq = \frac{1}{2}(1+\frac{2}{n^{\frac{1}{2}-\epsilon}})npq$.

$$\sum_{|k-\frac{n}{2}|<n^{\frac{1}{2}+\epsilon}} \frac{2}{\sqrt{n}} g(\frac{2k-n}{\sqrt{n}})[1 - \frac{1}{\sqrt{\frac{1}{2}(1+\frac{2}{n^{\frac{1}{2}-\epsilon}})npq}} g(\frac{\frac{1}{2}(2k-n)p}{\sqrt{\frac{1}{2}(1+\frac{2}{n^{\frac{1}{2}-\epsilon}})npq}})]^n \tag{6.41}$$

Let $c = \frac{1}{2}(1+\frac{2}{n^{\frac{1}{2}-\epsilon}})$.

$$= \sum_{|k-\frac{n}{2}|<n^{\frac{1}{2}+\epsilon}} \frac{2}{\sqrt{n}} g(\frac{2k-n}{\sqrt{n}})[1 - \frac{1}{\sqrt{cnpq}} g(\frac{\frac{1}{2}(2k-n)p}{\sqrt{cnpq}})]^n \tag{6.42}$$

Note the similarity to (6.11). We follow the derivation of the asymptotic bound for the zero-mean formula. As before, let $t_k = \frac{2k-n}{\sqrt{n}}$ and $\Delta t = \frac{2}{\sqrt{n}}$.

$$\sim \int dt\, g(t)[1 - \frac{1}{\sqrt{cnpq}} g(\sqrt{\frac{p}{q}}t(\frac{1}{2\sqrt{c}}))]^n \tag{6.43}$$

Observe that

$$[1 - \frac{1}{\sqrt{cnpq}} g(\sqrt{\frac{p}{q}} t(\frac{1}{2\sqrt{c}}))]^n \tag{6.44}$$

$$\sim \exp(-\sqrt{\frac{n}{cpq}} g(\sqrt{\frac{p}{q}} t(\frac{1}{2\sqrt{c}}))\exp(\frac{1}{2cpq} g^2(\sqrt{\frac{p}{q}} t(\frac{1}{2\sqrt{c}})))) \tag{6.45}$$

As in the previous derivation, we can ignore the second exponential function since it is $1 + O(t^2)$ for $t \to 0$.

So the upper bound for the fraction of nonzero-valued terms is

$$\sim 2 \int_0^\infty g(t) \exp(-\sqrt{\frac{n}{cpq}} g(\sqrt{\frac{p}{q}} t(\frac{1}{2\sqrt{c}})))dt \tag{6.46}$$

Let $r = \frac{q}{p}$, $\lambda = \sqrt{\frac{n}{2\pi pqc}}$, and $w = \exp(\frac{-t^2}{2r})$.

$$\sim \sqrt{\frac{r}{\pi}} \int_0^1 w^{r-1} (\log \frac{1}{w})^{-\frac{1}{2}} e^{-\lambda w^{\frac{1}{4c}}} dw \tag{6.47}$$

Partition the integral by regions as in the previous derivation, and the change of variable $w = (\frac{u}{\lambda})^{4c}$ yields

$$< \sqrt{\frac{2r}{\pi \log \lambda}} \lambda^{-4cr} 4c \int_0^\infty u^{4cr-1} e^{-u} du \text{ as } \lambda \to \infty \tag{6.48}$$

The integral is $\Gamma(4cr)$.

$$\sim 4c\ \Gamma(4cr) \sqrt{\frac{2r}{\pi \log \lambda}} \lambda^{-4cr} \tag{6.49}$$

$$= 4c\ \Gamma(4cr) \sqrt{\frac{2r}{\pi(\log n - \log 2c\pi pq)}} (\frac{2c\pi pq}{n})^{2cr} \tag{6.50}$$

For $\epsilon < \frac{1}{2}$, as $n \to \infty$, $c \to \frac{1}{2}$. So the upper bound for the fraction of nonzero-valued terms goes to

$$2\ \Gamma(2r) \sqrt{\frac{2r}{\pi \log n}} (\frac{\pi pq}{n})^r \tag{6.51}$$

This can be rewritten as

$$\sqrt{\frac{8r}{\pi}} (\pi pq)^r \Gamma(2r) \frac{1}{n^r \sqrt{\log n}} \tag{6.52}$$

which is $O(n^{-r} (\log n)^{-\frac{1}{2}})$.

6.5 Eliminating Sets of Zero-Valued Terms to Speed Computation

In this section, we develop a method to speed up computation of the permanent by collecting sets of zero terms in the low-variance formula. First, we develop a recursive algorithm that eliminates sets of zero-valued terms. Then, we present a preprocessing method that encourages elimination of large sets of zero-valued terms. Finally, we present test results.

The following algorithm computes the permanent by evaluating the finite-differences in the expression per $A = D_1 \cdots D_n P(\mathbf{x})$. The matrix A, the matrix size n, and the finite difference parameters $\mathbf{u}$ and $\mathbf{v}$ are global variables. The function call **permanent(1, 0)** evaluates the permanent. Each function call permanent(j, **r**) evaluates $D_j \cdots D_n P(\mathbf{x})$. The variable $\mathbf{r}$ accumulates row sums as the variables x_j receive assignments. In each function call,

$$r_i = \sum_{k=1}^{j-1} a_{ik} x_k \tag{6.53}$$

We use $\mathbf{a}^j$ to denote column j of A.

```
permanent(j, r)
{
if $j = n + 1$ then return $\prod_{i=1}^{n} r_i$
else return $\frac{1}{u_j - v_j}$[permanent($j + 1, \mathbf{r} + u_j \mathbf{a}^j$) − permanent($j + 1, \mathbf{r} + v_j \mathbf{a}^j$)]
}
```

Each function call permanent(j, **r**) assigns a value to x_j, and the descendant function calls assign values to $x_{j+1}, \ldots, x_n$. If, for some row, the partial row sum r_i is zero, and the entries in columns j through n are all zero, then the row sum will remain zero in all descendant function calls. Hence, all descendant function calls will return zero. The following algorithm avoids these descendant function calls and returns zero instead. Each global variable f_i stores the column of the rightmost nonzero entry in row i.

```
collect_zeros(j, r)
{
if, for some row i, $f_i = j - 1$ and $r_i = 0$, then return 0
else if j = n + 1 then return $\prod_{i=1}^{n} r_i$
else return $\frac{1}{u_j - v_j}$[collect_zeros(j + 1, r + $u_j a^j$) − collect_zeros(j + 1, r + $v_j a^j$)]
}
```

The computational savings scale exponentially with the level of recursion at which zero terms are collected. The following preprocessing procedure encourages efficient collection of zero terms by permuting the columns of A to pack nonzero entries to the left. First, choose a row with a minimum number of nonzero entries. Permute the columns of A to pack the row's nonzero entries into the leftmost columns. Then choose a row with a minimum (but positive) number of nonzero entries in the remaining columns, and permute the remaining columns to pack those nonzero entries to the left. Repeat until all columns have been packed (or until columns with only zero entries remain, in which case the permanent is zero).

Tables 6.1 and 6.2 show the results of using the preprocessing procedure with function `collect_zeros` and the low-variance formula. The procedure is very effective for sparse matrices. For example, with matrix size $n = 20$ and entry probability $p = 0.25$, only $100\% - 99.25\% = 0.75\%$ of the function calls are executed, speeding up the computation by a factor of $\frac{1}{0.0075} \approx 133$. Finally, note that the fraction of function calls avoided increases as the matrix size increases.

Permuting the columns prior to executing `collect_zeros` is equivalent to permuting the finite-differences in the expression $D_1 \cdots D_n P(\mathbf{x})$. Our preprocessing procedure is equivalent to a single static reordering. The finite-differences can also be dynamically reordered, as discussed in the previous chapter. Recall that `collect_zeros(j, r)` evaluates finite-differences D_j to D_n. Hence, columns j to n may be permuted at the start of `collect_zeros(j, r)` without altering the result, assuming that the values f_i are updated to reflect changes in the rightmost nonzero entries of rows.

The following algorithm is a dynamic reordering version of `collect_zeros`. Instead of permuting columns, we reorder assignments to variables $x_1, \ldots, x_n$. The variable set `S` indexes unassigned variables. Function call `dynamic(S, r, w)` evaluates the finite-differences indexed by `S`. As in `collect_zeros`, the variable `r` accumulates row sums as the variables x_j receive assignments. In each function call,

$$r_i = \sum_{k \in S} a_{ik} x_k \tag{6.54}$$

The variable w_i performs the function that variable f_i performed in `collect_zeros`. Each

p	% zero-valued terms	% function calls eliminated
0.25	99.38	98.08
0.50	94.10	81.22
0.75	82.84	52.49

Table 6.1: Matrix size 10 – percentages of terms with value zero and percentages of function calls eliminated. Each value is the average over 100 random matrices.

p	% zero-valued terms	% function calls eliminated
0.25	99.82	99.25
0.50	97.23	85.79
0.75	87.47	56.35

Table 6.2: Matrix size 20 – percentages of terms with value zero and percentages of function calls eliminated. Each value is the average over 100 random matrices.

w_i records the number of nonzero entries in row i in columns corresponding to unassigned x_j's. When $w_i = 0$, row sum i is fixed in all descendant function calls. So if $r_i = 0$, then all descendant function calls would return 0. Initially, w_i is the sum of entries in row i. To compute the permanent, call dynamic($\{1,\ldots,n\}, \mathbf{0}, \mathrm{w}$).

```
dynamic(S, r, w)
{
if, for some row i, $w_i = 0$ and $r_i = 0$, then return 0
else if $S = \emptyset$ then return $\prod_{i=1}^{n} r_i$
else choose $j \in S$
andreturn $\frac{1}{u_j - v_j}$[dynamic($S - j, r + u_j a^j, w - a^j$) − dynamic($S - j, r + v_j a^j, w - a^j$)]
}
```

The choice of j determines the efficiency of the algorithm. Since zero-valued terms are collected when $w_i = 0$ and $r_i = 0$ for some row i, the general strategy is to choose j corresponding to a column with nonzero entries in rows with few "unassigned" nonzero elements (low w_i) and low row sums r_i. We have tested several specific methods to choose j on small matrices, and they are modestly successful.

6.6 Discussion

The finite-difference parameter settings analyzed in this chapter are not necessarily optimal over the space of random 0-1 matrices – even better settings may remain to be discovered. The settings analyzed here are certainly not optimal for all specific 0-1 matrices. A technical report [5] outlines some methods to tailor the finite-difference parameters to problem instances.

The next chapter is devoted to a procedure to increase the expected fraction of zero-valued terms. The procedure works in conjunction with the parameter settings analyzed in this chapter.

The subsequent chapter is devoted to another method to eliminate sets of zero-valued terms than the method presented in this chapter. The method presented here is simpler and easier to implement than the method developed later. Also, the method developed here springs from the framework developed in an earlier chapter on recursive algorithms for finite-difference formulas. The method developed later is more amenable to analysis and it eliminates more zero-valued terms, but it has super-polynomial space requirements.

Chapter 7

A Permanent Decomposition Increases the Expected Fraction of Zero-Valued Terms

We exploit a permanent decomposition identity to replace problem instances by pairs of problem instances that have many more zero-valued terms. In the end, we produce problem instances for which all but an exponentially small fraction of the terms have value zero.

7.1 Row Zeroings and Correlations

In the finite-difference permanent formulas analyzed in a previous chapter, the row sums are correlated. For $\mathbf{x}$ assignments with about half the entries positive and about half negative, there are likely to be several rows in $A(\mathbf{x})$ with sum zero. For $\mathbf{x}$ assignments with almost all entries positive (or negative), it is likely that no row in $A(\mathbf{x})$ has sum zero. When one row sum is zero, it is likely that several others are zero as well. Since only one zero row sum is needed to zero the term, the extra zero row sums are "wasted."

If the row zeroings were independent, there would be many more zero-valued terms, as we show in the following analytic sketch. For simplicity, assume n is a multiple of 4, A has $\frac{n}{2}$ "even" rows with $\frac{n}{2}$ one-valued entries each, and the other rows have odd numbers of one-valued entries, so they never have sum zero. Each even row has sum zero when half the entries of $\mathbf{x}$ that correspond to one-valued entries in the row are assigned $+1$. The fraction of assignments that zero each even row is

$$\frac{1}{2^{\frac{n}{2}}}\binom{\frac{n}{2}}{\frac{n}{4}} \approx \sqrt{\frac{4}{\pi}}\frac{1}{\sqrt{n}} \tag{7.1}$$

by Stirling's formula [11]. So each even row has a nonzero sum in about $1 - \frac{1}{\sqrt{n}}$ of the assignments.

If the row zeroings were independent, then the fraction of nonzero-valued terms would be about

$$(1 - \frac{1}{\sqrt{n}})^{\frac{n}{2}} \tag{7.2}$$

To find the fraction as $n \to \infty$, take the natural logarithm, approximate by a Taylor series expansion about 1, and exponentiate. The fraction of nonzero-valued terms would be

$$\approx e^{-\frac{1}{2}\sqrt{n}-\frac{1}{4}} \tag{7.3}$$

In this chapter, we show how to reduce correlations among row sums to achieve an exponentially small fraction of nonzero-valued terms.

7.2 Decomposition

Note that the permanent is linear in its columns. For example, if we define $A_{\mathbf{a}}$ as the matrix A with the first row replaced by vector $\mathbf{a}$, then

$$\text{per } A_{\mathbf{a}+\mathbf{b}} = \text{ per } A_{\mathbf{a}} + \text{ per } A_{\mathbf{b}} \tag{7.4}$$

Set $\mathbf{a}$ to the first column of A to make $A_{\mathbf{a}} = A$. Then we have a formula for the permanent of A:

$$\text{per } A = \text{ per } A_{\mathbf{a}+\mathbf{b}} - \text{ per } A_{\mathbf{b}} \tag{7.5}$$

We refer to $\mathbf{b}$ as the decomposition vector.

To compute per A, we can use formula (7.5) to compute the permanents of $A_{\mathbf{a}+\mathbf{b}}$ and $A_{\mathbf{b}}$. On the face of it, this doubles the computation. However, we can choose the decomposition vector to decrease the expected fraction of nonzero-valued terms.

Consider the worst case for correlations among row zeroings, when A is the matrix with all entries one-valued. For each assignment $\mathbf{x}$, all row sums in $A(\mathbf{x})$ are equal. Using the low-variance formula, the row sums are zero for $\frac{1}{\mathrm{O}(\sqrt{n})}$ of the assignments $\mathbf{x}$. So the fraction of nonzero-valued terms is $1 - \frac{1}{O(\sqrt{n})}$.

We will now use decomposition to compute A. For simplicity, assume n is odd. Use finite-difference parameters $u_1 = 1$, $v_1 = -1$; for odd $j > 1$, $u_j = 1$ and $v_j = 0$; for even j, $u_j = 0$ and $v_j = -1$. (These are the low-variance parameters for all columns except the first.) Let $\mathbf{b} = (-\frac{n-1}{2}, \ldots, \frac{n-1}{2})$. The row sums over columns 2 to n

$$\sum_{j=2}^{n} a_{ij} x_j \tag{7.6}$$

vary from $-\frac{n-1}{2}$ to $\frac{n-1}{2}$.

In the computation of the permanent of $A_{\mathbf{b}}$, the first column, $x_1\mathbf{b}$, has entries $-\frac{n-1}{2}$ to $\frac{n-1}{2}$. Hence, each row sum over columns 2 to n is cancelled by some element in the first column. So every term has value zero.

In the computation of the permanent of $A_{\mathbf{a}+\mathbf{b}}$, the first column has entries $-\frac{n-1}{2}+1$ to $\frac{n-1}{2}+1$ if $x_1 = +1$ and entries $-\frac{n-1}{2}-1$ to $\frac{n-1}{2}-1$ if $x_1 = -1$. For $x_1 = +1$, the term with partial row sums $-\frac{n-1}{2}$ over columns 2 to n is the only nonzero-valued term. Likewise, for $x_1 = -1$, the term with partial row sums $\frac{n-1}{2}$ is the only nonzero-valued term. So the computation has only 2 nonzero-valued terms!

As illustrated in the example, decomposition can significantly increase the fraction of zero-valued terms for matrices with many one-valued entries. In general, decomposition can be used to separate row sums for rows that have their one-valued entries in common columns. Specific decomposition strategies are a topic for future research. In the remainder of this chapter, we analyze a simple strategy – choosing the decomposition vector entries at random.

7.2.1 A Random Decomposition Strategy to Increase Row Sum Variance

Each row sum in matrix $A_{\mathbf{a}+\mathbf{b}}(\mathbf{x})$ is the sum of the row sum in the original matrix $A(\mathbf{x})$ and x_1 times the decomposition vector entry for the row. Each row sum in matrix $A_{\mathbf{b}}(\mathbf{x})$ is the sum of the partial row sum in $A(\mathbf{x})$, neglecting the first entry, and x_1 times the decomposition vector entry for the row. If we select decomposition vector entries independently at random from an integer-valued distribution that takes the shape of a zero-mean Gaussian as n increases, then the distributions (over random 0-1 matrices and random decomposition vectors) of row sums of $A_{\mathbf{a}+\mathbf{b}}(\mathbf{x})$ take the shape of Gaussians with the variance of the row sums of the original matrix $A(\mathbf{x})$ plus the variance of the decomposition vector entry distribution. The row sums of $A_{\mathbf{b}}(\mathbf{x})$ have similar distributions. Hence, this decomposition strategy increases the variance of row sums.

To specify our strategy in more detail, let us generate the entries of the decomposition vector by summing over i.i.d. Bernoulli variables:

$$b_i = (d_{i1} + \ldots + d_{im}) - (d_{im+1} + \ldots + d_{i2m}) \quad \forall i \in \{1, \ldots, n\} \tag{7.7}$$

where each d_{ij} has value one with probability w and value zero with probability $1 - w$, and m and w are values that we will choose. For matrix $A_{\mathbf{a}+\mathbf{b}}(\mathbf{x})$, the row sums have the form:

$$[(d_{i1} + \ldots + d_{im}) - (d_{im+1} + \ldots + d_{i2m})]x_1 + a_{i1}x_1 + \ldots a_{in}x_n \tag{7.8}$$

Whether x_1 is assigned positive one or negative one, the term

$$[(d_{i1} + \ldots + d_{im}) - (d_{im+1} + \ldots + d_{i2m})]x_1 \tag{7.9}$$

has mean zero and variance $2mw(1 - w)$. If k variables x_j are assigned positive one, then the sum of the remaining terms, e.g.,

$$a_{i1} + \ldots + a_{ik} - a_{ik+1} - \ldots - a_{in} \tag{7.10}$$

has mean $(2k - n)p$ and variance npq. The entire row sum has mean $(2k - n)p$ and variance $npq + 2mw(1 - w)$. As $n \to \infty$, the row sum distribution takes the shape of a Gaussian with mean $(2k - n)p$ and variance $npq + 2mw(1 - w)$. We can produce any variance greater than npq through the choice of m and w. For the remainder of this exposition, let $w = p$. Then the variance is $(n + 2m)pq$. We will control the variance through the choice of m.

For the computation of the permanent of $A_{\mathbf{a}+\mathbf{b}}$, the expected fraction of nonzero-valued terms is

$$\frac{1}{2^n} \sum_{\mathbf{x} \in \{1,-1\}^n} \prod_{i=1}^{n} [1 - \Pr\{[(d_{i1} + \ldots + d_{im}) - (d_{im+1} + \ldots + d_{i2m})]x_1 + a_{i1}x_1 + \ldots + a_{in}x_n = 0\}] \tag{7.11}$$

Note that the distribution of $[(d_{i1} + \ldots + d_{im}) - (d_{im+1} + \ldots + d_{i2m})]x_1$ is the same whether x_1 is positive one or negative one, since the distribution of $(d_{i1} + \ldots + d_{im}) - (d_{im+1} + \ldots + d_{i2m})$ is symmetric about zero.

Collect terms, letting k be the number of entries in $\mathbf{x}$ assigned positive one. Then the expected fraction of nonzero-valued terms is

$$\sum_{k=0}^{n} \frac{1}{2^n}\binom{n}{k} \prod_{i=1}^{n} [1-\Pr\{(d_{i1}+\ldots+d_{im})-(d_{im+1}+\ldots+d_{i2m})+(a_{i1}+\ldots+a_{ik})-(a_{ik+1}+\ldots+a_{in}) = 0\}] \tag{7.12}$$

Given k, the row sums are independent and identically distributed. So the expected fraction of nonzero-valued terms is

$$\sum_{k=0}^{n} \frac{1}{2^n}\binom{n}{k} [1-\Pr\{(d_1+\ldots+d_m)-(d_{m+1}+\ldots+d_{2m})+(a_1+\ldots+a_k)-(a_{k+1}+\ldots+a_n) = 0\}]^n \tag{7.13}$$

where each a_j is a random variable that has value one with probability p and zero with probability $q = 1-p$, and each d_j is a random variable that has value one with probability p and zero with probability $q = 1-p$.

The row sums of $A_{\mathbf{b}}(\mathbf{x})$ are similar to the row sums of $A_{\mathbf{a}+\mathbf{b}}(\mathbf{x})$. They have the form (7.8), except that the term $a_{i1}x_1$ is replaced by zero. Because of this, the asymptotic expected fraction of nonzero-valued terms in the computation of the permanent of $A_{\mathbf{b}}$ is slightly less than the expected fraction of nonzero-valued terms in the computation of the permanent of $A_{\mathbf{a}+\mathbf{b}}$. Hence, we focus on the fraction for $A_{\mathbf{a}+\mathbf{b}}$ and use the result as a bound on the fraction for $A_{\mathbf{b}}$.

7.2.2 Variance Increased by a Constant Multiplier

Consider the asymptotic form of the expected fraction of nonzero-valued terms (7.13) in the computation of the permanent of $A_{\mathbf{a}+\mathbf{b}}$ when we select the number m to create row sum variances $(n+2m)pq = cn$, where c is constant with respect to n. (Without decomposition, $c = pq$; with decomposition, $c > pq$.) Examine (7.13). Note that the bracketed expression is in $[0,1]$ for all k. So the term corresponding to k is no greater than $\frac{1}{2^n}\binom{n}{k}$, which is the probability that the sum of n i.i.d. Bernoulli variables is k, given that each variable takes value one with probability $\frac{1}{2}$ and zero with probability $\frac{1}{2}$. According to Feller [11], p. 193, (6.7)

$$\sum_{|k-\frac{n}{2}|>2n^{0.64}} \frac{1}{2^n}\binom{n}{k} \sim \frac{1}{\sqrt{2\pi}}\frac{1}{n^{0.14}}\exp(-\frac{1}{2}n^{0.28}) \tag{7.14}$$

So we introduce error $o(\exp(-\frac{1}{2}n^{0.28}))$ by restricting the sum to values of k such that $|k-\frac{n}{2}| \le 2n^{0.64}$. According to Feller [11], p. 184, (3.13),

$$\frac{1}{2^n}\binom{n}{k} \sim \frac{2}{\sqrt{n}}g(\frac{2k-n}{\sqrt{n}}) \text{ for } |k-\frac{n}{2}| \le 2n^{0.64} \tag{7.15}$$

where $g()$ is the standard normal, i.e., $g(z) = \frac{1}{\sqrt{2\pi}}\exp(-\frac{1}{2}z^2)$.

Note that the sum of random variables in (7.13),

$$(d_1 + \ldots + d_m) - (d_{m+1} + \ldots + d_{2m}) + (a_1 + \ldots + a_k) - (a_{k+1} + \ldots + a_n) \tag{7.16}$$

is the difference of sums over two sets of i.i.d. Bernoulli variables:

$$(d_1 + \ldots + d_m + a_1 + \ldots + a_k) - (d_{m+1} + \ldots + d_{2m} + a_{k+1} + \ldots + a_n) \tag{7.17}$$

Thus, according to the main result of Appendix A,

$$\Pr\{(d_1 + \ldots + d_m) - (d_{m+1} + \ldots + d_{2m}) + (a_1 + \ldots + a_k) - (a_{k+1} + \ldots + a_n) = 0\} \tag{7.18}$$

$$\sim \frac{1}{\sqrt{(n+2m)pq}}g(\frac{(2k-n)p}{\sqrt{(n+2m)pq}}) \tag{7.19}$$

as $n \to \infty$, for $|k-\frac{n}{2}| \le 2n^{0.64}$, with additive error $o(\exp(-\frac{1}{2}n^{0.28}))$. (Having $2m+n$ random variables instead of n random variables strengthens the result of Appendix A.)

Using tail bound (7.14) and asymptotic forms (7.15) and (7.19) in (7.13) gives the following expression for the asymptotic expected fraction of nonzero-valued terms:

$$\sum_{|k-\frac{n}{2}|\le 2n^{0.64}} \frac{2}{\sqrt{n}}g(\frac{2k-n}{\sqrt{n}})[1-\frac{1}{\sqrt{cn}}g(\frac{(2k-n)p}{\sqrt{cn}})]^n + o(\exp(-\frac{1}{2}n^{0.28})) \tag{7.20}$$

Since our result will dominate the error term $o(\exp(-\frac{1}{2}n^{0.28}))$, we disregard it in the remaining analysis.

Let $t_k = \frac{2k-n}{\sqrt{n}}$ and $\Delta t = \frac{2}{\sqrt{n}}$. Then the asymptotic expected fraction is

$$\sum_{|t_k|\le 4n^{0.14}} \Delta t \; g(t)[1-\frac{1}{\sqrt{cn}}g(\frac{pt_k}{\sqrt{c}})]^n \tag{7.21}$$

Observe that

$$[1-\frac{1}{\sqrt{cn}}g(\frac{pt}{\sqrt{c}})]^n \sim \exp(-\sqrt{\frac{n}{c}}g(\frac{pt}{\sqrt{c}}))\exp(-\frac{1}{2c}g^2(\frac{pt}{\sqrt{c}})) \tag{7.22}$$

The second exponential is no greater than one. So the asymptotic form of the expected fraction of nonzero-valued terms is

$$\leq \sum_{|t_k| \leq 4n^{0.14}} \Delta t \; g(t) \exp(-\sqrt{\frac{n}{c}} g(\frac{pt}{\sqrt{c}})) \tag{7.23}$$

Analysis similar to that for (A.30) in Appendix A shows that the following integral has the same asymptotic form as the sum above.

$$\int_{-\infty}^{\infty} dt \; g(t) \exp(-\sqrt{\frac{n}{c}} g(\frac{pt}{\sqrt{c}})) \tag{7.24}$$

By symmetry, this is equal to

$$2 \int_{0}^{\infty} g(t) \exp(-\sqrt{\frac{n}{c}} g(\frac{pt}{\sqrt{c}})) dt \tag{7.25}$$

Let $r = \frac{c}{p^2}$ and $\lambda = \sqrt{\frac{n}{2\pi c}}$. Make the change of variable $w = \exp(\frac{-t^2}{2r})$. Then we have the following bound for the asymptotic expected fraction of nonzero-valued terms:

$$\sqrt{\frac{r}{\pi}} \int_{0}^{1} w^{r-1} (\log \frac{1}{w})^{-\frac{1}{2}} e^{-\lambda w} dw \tag{7.26}$$

In the earlier analysis of the expected fraction of nonzero-valued terms without decomposition, this was shown to have asymptotic form:

$$\sqrt{\frac{2r}{\pi}} (2\pi pq)^{\frac{r}{2}} \Gamma(r) \frac{1}{\sqrt{n^r \log n}} \tag{7.27}$$

So the asymptotic expected fraction of nonzero-valued terms is

$$O(\frac{1}{\sqrt{n^{\frac{c}{p^2}} \log n}}) \tag{7.28}$$

For $n \rightarrow \infty$ and c constant with respect to n, increasing row sum variance cn through decomposition decreases the expected fraction of nonzero-valued terms.

7.2.3 Variance Increased by a Factor of n – Too Much Variance

There is a limit to how much the expected fraction of nonzero-valued terms can be decreased by increasing row sum variance through decomposition. In our previous analysis, c is constant with respect to n. Suppose instead we increase the variance multiplier with n, i.e.,

$c = n$, so that variance $(n + 2m)pq = n^2$.

Examine expression (7.13), the expected fraction of nonzero-valued terms. For simplicity, assume n is even. Note that the expression in brackets, which is the probability of a nonzero row sum, is minimum when $k = \frac{n}{2}$, i.e., when half the variables in $\mathbf{x}$ are assigned $+1$ and half are assigned -1. Hence, the expected fraction of nonzero-valued terms is

$$\geq \sum_{k=0}^{n} \frac{1}{2^n}\binom{n}{k}[1-\Pr\{(d_1+\ldots+d_m)-(d_{m+1}+\ldots+d_{2m})+(a_1+\ldots+a_{\frac{n}{2}})-(a_{\frac{n}{2}+1}+\ldots+a_n)\}]^n \tag{7.29}$$

Since k does not appear in the bracketed term, and $\frac{1}{2^n}\binom{n}{k}$ is a distribution, we can move the bracketed expression to the left of the sum and substitute one for the sum over the distribution. So the previous expression is equal to

$$[1-\Pr\{(d_1+\ldots+d_m)-(d_{m+1}+\ldots+d_{2m})+(a_1+\ldots+a_{\frac{n}{2}})-(a_{\frac{n}{2}+1}+\ldots+a_n)\}]^n \tag{7.30}$$

Use (7.19) to derive the following asymptotic form for this lower bound on the expected fraction of nonzero-valued terms. Remember that $(n + 2m)pq = n^2$.

$$[1 - \frac{1}{n}g(0)]^n \to \exp(-g(0)) = \exp(-\frac{1}{\sqrt{2\pi}}) \text{ as } n \to \infty \tag{7.31}$$

Hence, the expected fraction of nonzero-valued terms does not even go to zero as $n \to \infty$. This is worse than not using decomposition at all.

7.2.4 Variance Increased by a Factor of $\sqrt{n}$ – An Exponentially Small Fraction of Nonzero-Valued Terms

Suppose we choose m to make the variance $(n + 2m)pq = \sqrt{n}n$. Examine (7.13). Use (7.14) and (7.19) to derive the following expression for the asymptotic expected fraction of nonzero-valued terms:

$$\sum_{|k-\frac{n}{2}|\leq 2n^{0.64}} \frac{1}{2^n}\binom{n}{k}[1 - \frac{1}{\sqrt{n}n}g(\frac{(2k-n)p}{\sqrt{\sqrt{n}n}})]^n + o(\exp(-\frac{1}{2}n^{0.28}) \tag{7.32}$$

Since our result will dominate the error term $o(\exp(-\frac{1}{2}n^{0.28}))$, we disregard it in the remaining analysis. Note that the term in brackets is minimum for the largest deviations of k from $\frac{n}{2}$, i.e., $|k - \frac{n}{2}| = 2n^{0.64}$. So the asymptotic form is

$$\leq \sum_{|k-\frac{n}{2}|\leq 2n^{0.64}} \frac{1}{2^n}\binom{n}{k}[1 - \frac{1}{n^{0.75}}g(\frac{(2\cdot 2n^{0.64})p}{n^{0.75}})]^n \tag{7.33}$$

The bracketed term is independent of k, and the sum of $\frac{1}{2^n}\binom{n}{k}$ over the range of k is a partial distribution. So the expression is

$$\leq [1 - \frac{1}{n^{0.75}}g(\frac{n^{0.64}4p}{n^{0.75}})]^n = [1 - \frac{1}{n^{0.75}}g(\frac{4p}{n^{0.11}})]^n \tag{7.34}$$

As $n \to \infty$, $\frac{4p}{n^{0.11}} \to 0$. Bound this fraction by one to find an upper bound for the previous expression.

$$\leq [1 - \frac{1}{n^{0.75}}g(1)]^n \tag{7.35}$$

To find the asymptotic behavior of this expression, let $x = -\frac{1}{n^{0.75}}g(1)$. Then

$$[1 - \frac{1}{n^{0.75}}g(1)]^n = (1+x)^n \tag{7.36}$$

Exponentiate and take the natural logarithm.

$$(1+x)^n = \exp[\ln(1+x)^n] = \exp[n\ln(1+x)] \tag{7.37}$$

Take the Taylor series expansion for $\ln(1+x)$:

$$\ln(1+x) = \ln(1) + \frac{x}{1!}\ln'(1) + \frac{x^2}{2!}\ln''(1) + \frac{x^3}{3!}\ln'''(1) + \ldots \tag{7.38}$$

Note that $\ln'(y) = \frac{1}{y}$, $\ln''(y) = \frac{-1}{y^2}$, $\ln'''(y) = 2\frac{1}{y^3}$, and $\ln^{(k)}(y) = (-1)^{k+1}(k-1)!\frac{1}{y^k}$. So

$$\ln(1+x) = 0 + \frac{x}{1!}(\frac{1}{1}) + \frac{x^2}{2!}\ln''(1) + \frac{x^3}{3!}\ln'''(1) + \ldots \tag{7.39}$$

and

$$\ln(1+x) = 0 + x - \frac{1}{2}x^2 + \frac{1}{3}x^3 \pm \ldots \tag{7.40}$$

So we have:

$$\exp[n\ln(1+x)] = \exp[n(x - \frac{1}{2}x^2 + \frac{1}{3}x^3 \pm \ldots)] \tag{7.41}$$

and

$$(1+x)^n = \exp[n(x - \frac{1}{2}x^2 + \frac{1}{3}x^3 \pm \ldots)] \tag{7.42}$$

p	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9
0.5	0	0	0	0	0	0	0	0	0
0.6	-1	-1	0	0	0	0	0	1	1
0.7	-1	-1	-1	0	0	0	1	1	1
0.8	-2	-1	-1	0	0	0	1	1	2
0.9	-2	-1	-1	0	0	1	1	2	2

Table 7.1: Optimal decomposition vectors for A_{b}, with matrix size $n = 9$ and various entry probabilities p.

Undo the substitution $x = -\frac{1}{n^{0.75}}g(1)$:

$$[1 - \frac{1}{n^{0.75}}g(1)]^n = \exp[n(-\frac{1}{n^{0.75}}g(1) - \frac{1}{2}\frac{1}{n^{1.50}}g^2(1) - \frac{1}{3}\frac{1}{n^{2.25}}g^3(1) - \ldots)] \tag{7.43}$$

$$= \exp[-n^{0.25}g(1) - \frac{1}{2}\frac{1}{n^{0.50}}g^2(1) - \frac{1}{3}\frac{1}{n^{1.25}}g^3(1) - \ldots] \tag{7.44}$$

$$= \exp[-n^{0.25}g(1)]\exp[-\frac{1}{2}\frac{1}{n^{0.50}}g^2(1)]\exp[-\frac{1}{3}\frac{1}{n^{1.25}}g^3(1)]\cdots \tag{7.45}$$

In all but the first exponential, the exponents go to zero as $n \to \infty$, so these exponentials go to one. Hence, the first exponential determines the asymptotic form. So the asymptotic expected fraction of nonzero-valued terms is

$$\mathrm{O}(\exp(n^{-\frac{1}{4}}\frac{1}{\sqrt{2\pi e}})) \tag{7.46}$$

7.3 Optimal Nonrandom Decomposition Vectors

Tables 7.1, 7.2, and 7.3 show the best (nonrandom) decomposition vectors for various matrix sizes and entry probabilities. The displayed decomposition vectors produce the minimum possible expected fraction of nonzero-valued terms in the formula for A_{b} with column multipliers $u_1 = 1$ and $v_1 = -1$, and the remaining column multipliers half $u_j = 1$ and $v_j = 0$ and half $u_j = 0$ and $v_j = -1$. The optimal decomposition vectors were found through exhaustive search.

Note that the zero vector is the best decomposition vector for matrices of size $n = 9$ and $n = 11$ with entry probability $p = 0.5$. Hence, decomposition does not reduce computation in these cases. However, as matrix size and entry probability grow, the entries of the best decomposition vector increase in absolute value.

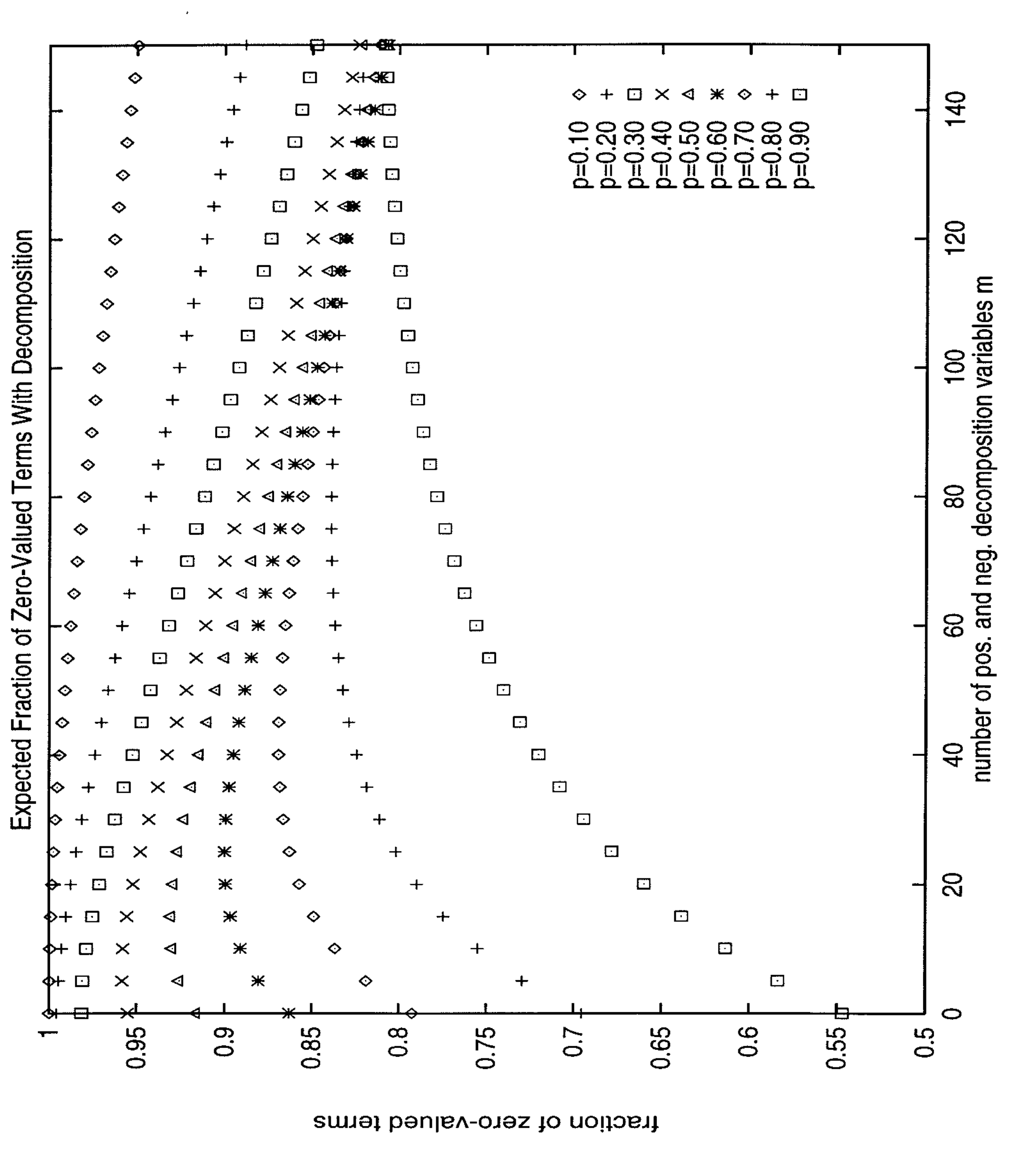

Figure 7.1: The expected fractions of zero-valued terms in decomposition matrix $A_{\mathrm{a+b}}$ are shown for matrices of size $n = 40$, various entry probabilities p, and various numbers m of pairs of decomposition variables, each with probability $w = p$. The optimal number of decomposition variables increases with expected matrix density p.

p	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_10	b_11
0.5	0	0	0	0	0	0	0	0	0	0	0
0.6	-1	-1	-1	0	0	0	0	0	1	1	1
0.7	-1	-1	-1	-1	-1	0	1	1	1	1	1
0.8	-2	-2	-1	-1	0	0	0	1	1	2	2
0.9	-3	-2	-2	-1	-1	0	0	1	1	2	2

Table 7.2: Optimal decomposition vectors for A_{b}, with matrix size $n = 11$ and various entry probabilities p.

p	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}
0.5	-1	0	0	0	0	0	0	0	0	0	0	0	1
0.6	-1	-1	-1	-1	-1	0	0	0	1	1	1	1	1
0.7	-1	-1	-1	-1	-1	-1	0	1	1	1	1	1	2
0.8	-2	-2	-2	-1	-1	0	0	0	1	1	1	2	2
0.9	-3	-2	-2	-1	-1	0	0	0	1	1	2	2	3

Table 7.3: Optimal decomposition vectors for A_{b}, with matrix size $n = 13$ and various entry probabilities p.

7.4 Discussion

The random decomposition strategies that we analyzed all have independently drawn decomposition vector entries. These strategies are effective because they use independent decomposition vector entries with large variance to overwhelm the dependencies among row sums in the original matrix, creating nearly independent row sums. A challenge for the future is to identify decomposition strategies in which the decomposition vector entries are chosen with dependencies that counteract the dependencies among the original row sums, creating nearly independent row sums with lower variance and hence more row zeroings.

Both our analysis of random decomposition vectors and our search for optimal non-random decomposition vectors focus on decomposition strategies with good average results over problem instances drawn at random. Another challenge is to develop decomposition strategies tailored to problem instances. Finding the best decomposition strategy for an instance may prove to be intractable, so there may be a tradeoff between the computation required to find a good decomposition vector and the computation saved by decomposition.

Chapter 8

A Method to Compute the Permanent that Avoids Many Zero-Valued Terms

8.1 Alternative Computation of the Permanent Formula

Recall the permanent formula:

$$\text{per } A = \sum_{\mathbf{x}\in\{-1,1\}^n} (-1)^{s(\mathbf{x})} \prod_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_j \tag{8.1}$$

where $s(\mathbf{x})$ is the number of variables x_j assigned -1. The straightforward method to compute this formula is to step through $\mathbf{x}$ assignments, compute the signed product of row sums for each assignment, and sum these values over assignments. In this chapter, we use an alternative method to compute the formula in order to avoid computing many zero-valued terms.

Focus on the product of row sums:

$$\prod_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_j \tag{8.2}$$

View vector $\mathbf{x}$ as the concatenation of "half vectors" $\mathbf{y}$ and $\mathbf{z}$, with $\frac{n}{2}$ entries each. (To simplify the exposition, assume n is even.) For a given assignment $\mathbf{x} = (y_1, \ldots, y_{\frac{n}{2}}, z_1, \ldots, z_{\frac{n}{2}})$, suppose we have already computed the row sums over the first half of the columns of $A(\mathbf{x})$:

$$\mathbf{c}(\mathbf{y}) = \sum_{j=1}^{\frac{n}{2}} a_{ij} y_j \tag{8.3}$$

and suppose we have already computed the row sums over the second half of the columns of $A(\mathbf{x})$:

$$\mathbf{d}(\mathbf{z}) = \sum_{j=\frac{n}{2}+1}^{n} a_{ij} z_{j-\frac{n}{2}} \tag{8.4}$$

Then we can compute the product of row sums by taking the product of the sums of the half-row sums.

$$\prod_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_j = \prod_{i=1}^{n} (c_i(\mathbf{y}) + d_i(\mathbf{z})) \tag{8.5}$$

If we define $s(\mathbf{y})$ to be the number of variables y_j assigned -1 and define $s(\mathbf{z})$ to be the number of variables z_j assigned -1, then $(-1)^{s(\mathbf{x})} = (-1)^{s(\mathbf{y})}(-1)^{s(\mathbf{z})}$. For notational convenience, define $c_0(\mathbf{y}) = (-1)^{s(\mathbf{y})}$ and $d_0(\mathbf{z}) = (-1)^{s(\mathbf{z})}$. Then

$$(-1)^{s(\mathbf{x})} = c_0(\mathbf{y}) d_0(\mathbf{z}) \tag{8.6}$$

Note that the space $\{-1,1\}^n$ in formula (8.1) is the cross product $\{-1,1\}^{\frac{n}{2}} \times \{-1,1\}^{\frac{n}{2}}$. Hence, the assignments $\mathbf{x} \in \{-1,1\}^n$ are equivalent to the assignments $(\mathbf{y},\mathbf{z}) \in \{-1,1\}^{\frac{n}{2}} \times \{-1,1\}^{\frac{n}{2}}$. We can use (8.5) and (8.6) to create a formula equivalent to (8.1).

$$\text{per } A = \sum_{(\mathbf{y},\mathbf{z}) \in \{-1,1\}^{\frac{n}{2}} \times \{-1,1\}^{\frac{n}{2}}} c_0(\mathbf{y}) d_0(\mathbf{z}) \prod_{i=1}^{n} (c_i(\mathbf{y}) + d_i(\mathbf{z})) \tag{8.7}$$

To compute this formula, we can first compute and store all half-row sums.

$$C = \{\mathbf{c}(\mathbf{y}) | \mathbf{y} \in \{-1,1\}^{\frac{n}{2}}\} \text{ and } D = \{\mathbf{d}(\mathbf{z}) | \mathbf{z} \in \{-1,1\}^{\frac{n}{2}}\} \tag{8.8}$$

Then the formula becomes

$$\text{per } A = \sum_{(\mathbf{c},\mathbf{d}) \in C \times D} c_0 d_0 \prod_{i=1}^{n} (c_i + d_i) \tag{8.9}$$

Now, suppose we partition the sets of half-row sum vectors according to the values in entries $1, \ldots, m$. (These are the sums over halves of the columns of $A(\mathbf{x})$ for the first m

rows. We will specify the value of m later.) Denote the partitions of C as follows:

$$C(\mathbf{s}) = \{\mathbf{c} \in C | c_1 = s_1 \text{ and } \dots \text{ and } c_m = s_m\} \quad \forall\, \mathbf{s} \in \{-\frac{n}{2}, \dots, \frac{n}{2}\}^m \tag{8.10}$$

Denote the partitions of D as follows:

$$D(\mathbf{t}) = \{\mathbf{d} \in D | d_1 = t_1 \text{ and } \dots \text{ and } d_m = t_m\} \quad \forall\, \mathbf{t} \in \{-\frac{n}{2}, \dots, \frac{n}{2}\}^m \tag{8.11}$$

Introducing the partitions into formula (8.9) produces:

$$\text{per } A = \sum_{(\mathbf{s},\mathbf{t}) \in \{-\frac{n}{2},\dots,\frac{n}{2}\}^m \times \{-\frac{n}{2},\dots,\frac{n}{2}\}^m} \quad \sum_{(\mathbf{c},\mathbf{d}) \in C(\mathbf{s}) \times D(\mathbf{t})} c_o d_o \prod_{i=1}^{n} (c_i + d_i) \tag{8.12}$$

All $\mathbf{c} \in C(\mathbf{s})$ have values $s_1, \dots, s_m$ in the first m entries. All $\mathbf{d} \in D(\mathbf{t})$ have values $t_1, \dots, t_m$ in the first m entries. So these values are constant within the second sum, and they can be moved outside the sum.

$$\text{per } A = \sum_{(\mathbf{s},\mathbf{t}) \in \{-\frac{n}{2},\dots,\frac{n}{2}\}^m \times \{-\frac{n}{2},\dots,\frac{n}{2}\}^m} \prod_{i=1}^{m} (s_i + t_i) \sum_{(\mathbf{c},\mathbf{d}) \in C(\mathbf{s}) \times D(\mathbf{t})} c_0 d_0 \prod_{i=m+1}^{n} (c_i + d_i) \tag{8.13}$$

The new product is the key to avoiding many zero-valued terms. If $s_i + t_i = 0$ for some $i \in \{1, \dots, m\}$, then the new product is zero, so there is no need to compute the second sum. In terms of the original formula (8.1), we avoid computing the products of row sums for all assignments $\mathbf{x}$ such that $A(\mathbf{x})$ has a zero row sum in the first m rows.

8.2 Algorithm

The algorithm proceeds as follows:

(1) For each $\mathbf{y} \in \{-1, 1\}^{\frac{n}{2}}$, compute:

$$\mathbf{c}(\mathbf{y}) = \sum_{j=1}^{\frac{n}{2}} a_{ij} y_j \text{ and } c_0(\mathbf{y}) = (-1)^{s(\mathbf{y})} \tag{8.14}$$

(2) Store vectors $\mathbf{c}(\mathbf{y})$ as set C.

(3) Partition set C:

$$C(\mathbf{s}) = \{\mathbf{c} \in C | c_1 = s_1 \text{ and } \dots \text{ and } c_m = s_m\} \text{ for } \mathbf{s} \in \{-\frac{n}{2}, \dots, \frac{n}{2}\}^m \tag{8.15}$$

(4) For each $\mathbf{z} \in \{-1, 1\}^{\frac{n}{2}}$, compute:

$$\mathbf{d}(\mathbf{z}) = \sum_{j=\frac{n}{2}+1}^{n} a_{ij} z_{j-\frac{n}{2}} \text{ and } d_0(\mathbf{z}) = (-1)^{s(\mathbf{z})} \tag{8.16}$$

(5) Store vectors $\mathbf{d}(\mathbf{y})$ as set D.

(6) Partition set D:

$$D(\mathbf{t}) = \{\mathbf{d} \in D | d_1 = t_1 \text{ and } \dots \text{ and } d_m = t_m\} \text{ for } \mathbf{t} \in \{-\frac{n}{2}, \dots, \frac{n}{2}\}^m \tag{8.17}$$

(7) For each pair $(\mathbf{s}, \mathbf{t}) \in \{-\frac{n}{2}, \dots, \frac{n}{2}\}^m \times \{-\frac{n}{2}, \dots, \frac{n}{2}\}^m$, examine $\prod_{i=1}^{m}(s_i + t_i)$. If it is nonzero, then do step (8).

(8) Compute

$$\prod_{i=1}^{m}(s_i + t_i) \sum_{(\mathbf{c},\mathbf{d}) \in C(\mathbf{s}) \times D(\mathbf{t})} c_0 d_0 \prod_{i=m+1}^{n} (c_i + d_i) \tag{8.18}$$

and add the result to the running total.

On completion, the running total is the permanent of A.

8.3 Analysis

The algorithm requires quite a bit of space to store multisets C and D of vectors $\mathbf{c}(\mathbf{y})$ and $\mathbf{d}(\mathbf{z})$. Since there are $2^{\frac{n}{2}}$ assignments each for $\mathbf{y}$ and $\mathbf{z}$, the algorithm requires $O(2^{\frac{n}{2}} n)$ space.

Steps (1), (2), and (3) require $O(2^{\frac{n}{2}} \text{poly } n)$ time since there are $2^{\frac{n}{2}}$ assignments $\mathbf{y}$ in $\{-1, 1\}^{\frac{n}{2}}$. Likewise, steps (4), (5), and (6) require $O(2^{\frac{n}{2}} \text{poly } n)$ time. Step (7) requires $O(n^{2m}\text{poly } n) = O(2^{2m\log_2 n}\text{poly } n)$ time, since there are $(n+1)^{2m}$ pairs $(\mathbf{s},\mathbf{t})$ in $\{-\frac{n}{2},\ldots,\frac{n}{2}\}^m \times \{-\frac{n}{2},\ldots,\frac{n}{2}\}^m$. Step (8) requires $O(\text{poly } n)$ time for each assignment $\mathbf{x}$ in $\{-1,1\}^n$ such that $A(\mathbf{x})$ has no zero row sums in the first m rows. Let $f(n,p,m)$ represent the expected fraction of assignments $\mathbf{x}$ for which $A(\mathbf{x})$ has no zero row sums in the first m rows, with the expectation over $n \times n$ matrices with each entry one with probability p and zero with probability $1-p$. Then the algorithm has expected running time:

$$\max[O(2^{\frac{n}{2}}\text{poly } n), O(2^{2m\log_2 n}\text{poly } n), O(2^n f(n,p,m)\text{poly } n)] \tag{8.19}$$

If we set $m = \frac{n}{4\log_2 n}$, then the center expression is equal to the first expression. In this case, we avoid computing terms with zero row sums in the first $\frac{n}{4\log_2 n}$ rows. The running time is:

$$\max[O(2^{\frac{n}{2}}\text{poly } n), O(2^n f(n,p,\frac{n}{4\log_2 n})\text{poly } n)] \tag{8.20}$$

In the previous chapter, we analyzed $f(n,p,n)$, the expected fraction of assignments $\mathbf{x}$ for which $A(\mathbf{x})$ has no zero row sums. Later in this chapter, we analyze the expected value of $f(n,p,m)$ for $m < n$.

8.4 Variations

The same basic steps can be used to construct an algorithm using $(-1,0)$ and $(0,1)$ column multipliers instead of $(-1,1)$ multipliers. If the first $\frac{n}{2}$ columns have multipliers $(0,1)$ and the remaining columns have multipliers $(-1,0)$, then the range of each entry in $\mathbf{s}$ is $\{0,\ldots,\frac{n}{2}\}$ and the range of each entry in $\mathbf{t}$ is $\{-\frac{n}{2},\ldots,0\}$. This reduces the computation in step (7) from $O(n^{2m}\text{poly } n)$ to $O((\frac{n}{2})^{2m}\text{poly } n)$.

The procedures also work with decomposition. If all columns have multipliers $(-1,+1)$, and the decomposition vector is $\mathbf{b}$, then the range of s_i is $\{b_i - \frac{n}{2},\ldots,b_i+\frac{n}{2}\} \cup \{-b_i - \frac{n}{2},\ldots,-b_i+\frac{n}{2}\}$. The range of each entry s_i has at most $2n+2$ values, so the computation required for step (7) increases to $O((2n)^m n^m \text{poly } n) = O(2^{m+2m\log_2 n}\text{poly } n)$. If only the initial (decomposition) column has multipliers $(-1,1)$, and the remaining multipliers are $(0,1)$ for the first half of the columns and $(-1,0)$ for the second half, then the range of each entry s_i has no more than $n+2$ elements, and the range of each entry t_i has $\frac{n}{2}+1$ elements. In this case, step (7) requires $O(n^m(\frac{n}{2})^m\text{poly } n)$ computation.

Computation can be saved by making a list of those partitions $C(\mathbf{s})$ and $D(\mathbf{t})$ that are nonempty and computing step (7) only for pairs $(\mathbf{s},\mathbf{t})$ that correspond to nonempty pairs of partitions. The algorithm will collect zero-valued terms more effectively if the first m

rows can each have zero sums in $A(\mathbf{x})$. Hence, if the column multipliers are $(-1,1)$, then permute the rows of A so that as many as possible of the first m rows have even numbers of elements.

Computation and space can be saved by "merging" duplicated partial row sums. If C contains duplicates, then keep only one and store the number of duplicates as a coefficient. When the duplicated vector $\mathbf{c}$ is used to find the product of row sums, multiply the result by the coefficient. If C contains vectors $\mathbf{c}$ and $\mathbf{c}'$ that are identical except for opposite signs $(c_0 = -c_0')$, then their terms will cancel in the computation, so they can be eliminated from C. (The same rules apply to duplicates and opposites in D.)

The storage requirement can be reduced as follows. Compute and store partial row sums over less than half of the columns. Partition these partial row sum vectors according to the first m entries, as we did with the vectors $\mathbf{c}(\mathbf{y})$ in the original algorithm. For each assignment to the x_j's corresponding to the remaining columns, compute the partial row sum vector $\mathbf{d}$. Find the partitions such that none of the first m partial row sum vector entries are the opposite of the corresponding entries of $\mathbf{d}$. For each $\mathbf{c}$ in these partitions, compute the product of row sums using $\mathbf{c}$ and $\mathbf{d}$.

For example, if we use the first $\frac{n}{4}$ columns to form partial row sums $\mathbf{c}$, then the algorithm proceeds as follows:

(1)' For each $\mathbf{y} \in \{-1,1\}^{\frac{n}{4}}$, compute

$$\mathbf{c}(\mathbf{y}) = \sum_{j=1}^{\frac{n}{4}} a_{ij} y_j \text{ and } c_0(\mathbf{y}) = (-1)^{s(\mathbf{y})} \tag{8.21}$$

(2)' Store vectors $\mathbf{c}(\mathbf{y})$ as set C.

(3)' Partition set C:

$$C(\mathbf{s}) = \{\mathbf{c} \in C | c_1 = s_1 \text{ and } \ldots \text{ and } c_m = s_m\} \text{ for } \mathbf{s} \in \{-\frac{n}{2}, \ldots, \frac{n}{2}\}^m \tag{8.22}$$

(4)' For each $\mathbf{z} \in \{-1,1\}^{\frac{3n}{4}}$, compute

$$\mathbf{d}(\mathbf{z}) = \sum_{j=\frac{n}{4}+1}^{n} a_{ij} z_{j-\frac{n}{4}} \text{ and } d_0(\mathbf{z}) = (-1)^{s(\mathbf{z})} \tag{8.23}$$

and let $\mathbf{t} = (d_1, ..., d_m)$.

(5)' For each $\mathbf{s} \in \{-\frac{n}{2}, \dots, \frac{n}{2}\}^m$, examine $\prod_{i=1}^{m}(s_i+t_i)$. If it is nonzero, then do step (6)'.

(6)' Compute

$$\prod_{i=1}^{m}(s_i + t_i) \sum_{(\mathbf{c},\mathbf{d}) \in C(\mathbf{s}) \times D(\mathbf{t})} c_0 d_0 \prod_{i=m+1}^{n} (c_i + d_i) \tag{8.24}$$

and add the result to the running total.

The storage requirements are determined by step (2)'. Since C consists of one vector for each $\mathbf{y} \in \{-1,1\}^{\frac{n}{4}}$, the space required is $O(2^{\frac{n}{4}}\text{poly } n)$, a reduction from the original algorithm by a factor of $2^{-\frac{n}{4}}$.

There is a tradeoff of space for time. We do not store and partition the vectors $\mathbf{d}(\mathbf{z})$ as in the original algorithm, so we must step through these vectors one at a time (step (5)') instead of computing for entire partitions at once (step (7)). Hence, the time required for steps (4)' and (5)' is $O(2^{\frac{3n}{4}} n^m \text{poly } n) = O(2^{\frac{3n}{4}+m\log_2 n}\text{poly } n)$. (Compare to $O(2^{2m\log_2 n}\text{poly } n)$ for step (7) in the original algorithm.)

The expected running time for step (6)' is $O(2^n f(n,p,m)\text{poly } n)$. Hence, the algorithm has expected running time

$$max[O(2^{\frac{3n}{4}+m\log_2 n}\text{poly } n), O(2^n f(n,p,m)\text{poly } n)] \tag{8.25}$$

For example, if we set $m = \frac{n}{8\log_2 n}$, then the expected running time is

$$max[O(2^{\frac{7n}{8}}\text{poly } n), O(2^n f(n,p,m)\text{poly } n)] \tag{8.26}$$

8.5 Expected Fraction of Terms Computed by the Algorithm – f(n,p,m)

In this section, we find the asymptotic form (as n increases) of the expected fraction of the 2^n assignments $\mathbf{x}$ for which $A(\mathbf{x})$ has no zero row sum among the first m rows when we use column multipliers $(-1, 1)$ and decomposition. We show that, for $m = \frac{n}{4\log_2 n}$, this fraction determines the expected running time of our algorithm, and the expected running time is an exponentially small fraction of the 2^npoly n time required to evaluate the permanent formula (8.1) directly.

In a previous chapter, we showed that if we use a random decomposition strategy to increase row sum variances to $\sqrt{n}n$, then the expected fraction of nonzero-valued terms is $O(\exp(-n^{\frac{1}{4}}\frac{1}{\sqrt{2\pi e}}))$. The fraction of nonzero-valued terms is the fraction of assignments $\mathbf{x}$ for which $A(\mathbf{x})$ has nonzero row sums in all n rows. In the previous chapter, we showed that this fraction has an asymptotic form bounded by (7.35):

$$[1 - \frac{1}{n^{0.75}}g(1)]^n \tag{8.27}$$

The fraction of terms computed by the algorithm is a similar quantity. It is the fraction of assignments $\mathbf{x}$ for which $A(\mathbf{x})$ has nonzero row sums in the first m rows. Review the derivation of the previous expression. Note that the n in the exponent is the only reference to the number of rows that have nonzero sums in the fraction of terms counted by the expression. All other n's refer to the number of columns in A. Hence, an upper bound for the asymptotic form of $f(n, p, m)$, the fraction of terms computed by our algorithm, is:

$$[1 - \frac{1}{n^{0.75}}g(1)]^m \tag{8.28}$$

Following the derivation in the previous chapter from (7.35) to (7.43) , we find that

$$[1 - \frac{1}{n^{0.75}}]^m = \exp[m(-\frac{1}{n^{0.75}}g(1) - \frac{1}{2}\frac{1}{n^{1.50}}g^2(1) - \frac{1}{3}\frac{1}{n^{2.25}}g^3(1) - \ldots)] \tag{8.29}$$

Let $m = \frac{n}{4\log_2 n}$:

$$= \exp[\frac{n}{4\log_2 n}(-\frac{1}{n^{0.75}}g(1) - \frac{1}{2}\frac{1}{n^{1.50}}g^2(1) - \frac{1}{3}\frac{1}{n^{2.25}}g^3(1) - \ldots)] \tag{8.30}$$

$$= \exp[-\frac{n^{0.25}}{\log_2 n}\frac{g(1)}{4} - \frac{1}{2}\frac{1}{n^{0.50}\log_2 n}\frac{g^2(1)}{4} - \frac{1}{3}\frac{1}{n^{1.25}\log_2 n}\frac{g^3(1)}{4} - \ldots] \tag{8.31}$$

$$= \exp[-\frac{n^{0.25}}{\log_2 n}\frac{g(1)}{4}]\exp[-\frac{1}{2}\frac{1}{n^{0.50}\log_2 n}\frac{g^2(1)}{4}]\exp[-\frac{1}{3}\frac{1}{n^{1.25}\log_2 n}\frac{g^3(1)}{4}]\cdots \tag{8.32}$$

The first exponential dominates. In the others, the exponents go to zero as $n \to \infty$, so the exponentials go to one. Hence, $f(n, p, \frac{n}{4\log_2 n})$, the asymptotic expected fraction of terms computed by the algorithm, is

$$O(\exp[-\frac{n^{\frac{1}{4}}}{4\log_2 n}\frac{1}{\sqrt{2\pi e}}]) \tag{8.33}$$

We can use this result to analyze the expected running time of the algorithm. Earlier, we showed that steps (1) through (6) require $O(2^{\frac{n}{2}}\text{poly } n)$ time. With decomposition, we found that step (7) requires $O(2^{m+2m\log_2 n}\text{poly } n)$ time. With $m = \frac{n}{4\log_2 n}$, this is $O(2^{\frac{n}{4\log_2 n}+\frac{n}{2}}\text{poly } n)$. Also, we found that step (8) requires $O(2^n f(n,p,m)\text{poly } n)$ expected running time. From (8.33), we can see that this step determines the expected running time, which is

$$O(2^n \exp[-\frac{n^{\frac{1}{4}}}{4\log_2 n}\frac{1}{\sqrt{2\pi e}}]\text{poly } n) \tag{8.34}$$

This is smaller than the $O(2^n\text{poly } n)$ required to evaluate the permanent formula directly, by a factor of

$$\exp[-\frac{n^{\frac{1}{4}}}{4\log_2 n}\frac{1}{\sqrt{2\pi e}}] \tag{8.35}$$

8.6 Discussion

The result above can be extended to matrices with entries in $\{-1, 0, 1\}$. Specifically, examine the possibility of applying the algorithm developed in this chapter to a $\{-1, 0, 1\}$ matrix, with decomposition to make row sum variances $\sqrt{n}n$ and column multipliers $(-1, 1)$. The range of possible row sums

$$\sum_{j=1}^{n} a_{ij}x_j \tag{8.36}$$

is the same as for 0-1 matrices, so the data structures and procedures of the algorithm for 0-1 matrices work for $\{-1, 0, 1\}$ matrices as well.

Now consider whether the expected computational reduction achieved for 0-1 matrices is also achieved for $\{-1, 0, 1\}$ matrices. As in the analysis for 0-1 matrices, assume the entries of the $\{-1, 0, 1\}$ matrices are drawn i.i.d. to compute expectations. Note that the reduction in computation relies on small expected values of row sums (8.36) and the possibility of achieving row sum variance $\sqrt{n}n$ through decomposition.

Recall that the expected value of a row sum in a 0-1 matrix, given k variables x_j assigned +1, is:

$$\mu_k = (2k - n)p \tag{8.37}$$

where p is the probability of each entry taking value 1. Since the computational reduction proof applies for $p \in [0,1]$, it applies for $|\mu_k|$ as large as $|2k-n|$. For $\{-1,0,1\}$ matrices, let μ' be the expected value of each entry. Then the expected value of a row sum, given k variables x_j assigned $+1$, is:

$$\mu_k = (2k-n)\mu' \tag{8.38}$$

Note that $|\mu'| \leq 1$. Hence, $|\mu_k| \leq |2k-n|$, and the expected row sums are in the range covered by the proof.

Since decomposition can only increase row sum variance, the proof applies only if the row sum variance without decomposition is less than $\sqrt{n}n$. Let $(\sigma')^2$ be the variance of each entry. Note that the row sum variance without decomposition is $n(\sigma')^2$. The maximum entry variance is achieved by the distribution

$$a_{ij} = \begin{cases} -1 & \text{with probability } \frac{1}{2} \\ 0 & \text{with probability } 0 \\ 1 & \text{with probability } \frac{1}{2} \end{cases} \tag{8.39}$$

In this case, $(\sigma')^2 = 1$, and the row sum variance without decomposition is n, which is in the range covered by the proof.

Chapter 9

Estimation of Finite-Difference Formulas by Sampling

In this chapter, we examine methods to estimate the value of a finite-difference formula by evaluating a subset of the terms. While our examples and computations involve the finite-difference formula for the permanent, many of the results extend to finite-difference formulas in general.

We begin by analyzing methods to sample terms at random from the entire formula. The mean of the sampled terms is an estimator for the mean of all terms in the formula. We consider the use of different finite-difference parameters to improve the estimation by decreasing the variance among terms.

Next, we consider a method in which we partition the terms, then sample within each partition to estimate the sums over partitions separately. If we can partition the terms such that terms within each partition have similar values, then this method gives better estimates with fewer samples.

We also outline a method to sample blocks of terms. The blocks are generated by choosing an assignment to the finite-difference variables at random, then listing all assignments that can be formed by varying the values of a subset of the finite-difference variables. The set of terms that corresponds to this set of assignments is the block of terms. For the permanent generating function, the sum of a block of terms can be computed more efficiently than the sum of the same number of terms sampled at random. Furthermore, summing a block of terms computes a portion of the finite-difference sieve, producing cancellations among terms in the generating function.

9.1 Sampling Terms from the Entire Formula

9.1.1 Formula and Notation

Recall the finite-difference formula for the permanent:

$$\text{per } A = \sum_{\mathbf{x} \in \{u_1, v_1\} \times \ldots \times \{u_n, v_n\}} (-1)^{s(\mathbf{x})} \frac{1}{(u_1 - v_1) \cdots (u_n - v_n)} P(\mathbf{x}) \tag{9.1}$$

where $s(\mathbf{x})$ is the number of variables x_j assigned v_j, and the generating function is

$$P(\mathbf{x}) = \prod_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_j \tag{9.2}$$

Also, note that the generating function may be rewritten:

$$P(\mathbf{x}) = \sum_{\mathbf{p} \in Q} c(\mathbf{p}) x_1^{p_1} \cdots x_n^{p_n} \tag{9.3}$$

where Q is the set of integer-valued vectors defined as follows:

$$Q = \{\mathbf{p} \geq 0 | p_1 + \ldots + p_n = n\} \tag{9.4}$$

In this chapter, we refer to terms of the finite-difference formula (9.1) as finite-difference terms or simply as terms. We refer to terms of the generating function (9.3) as generating-function terms.

9.1.2 Estimation by Sampling

In this section, we consider the following method to estimate formula (9.1). Select m finite-difference variable assignments $\mathbf{x}_1, \ldots, \mathbf{x}_m$ independently and uniformly at random from $\{u_1, v_1\} \times \ldots \times \{u_n, v_n\}$. For assignment $\mathbf{x}_i$, define X_i to be 2^n times the term corresponding to $\mathbf{x}_i$:

$$X_i = 2^n (-1)^{s(\mathbf{x}_i)} \frac{1}{(u_1 - v_1) \cdots (u_n - v_n)} P(\mathbf{x}_i) \tag{9.5}$$

The estimate of (9.1) is the average of the random variables $X_1, \ldots, X_m$:

$$\frac{X_1 + \ldots + X_m}{m} \tag{9.6}$$

The estimate is unbiased, i.e., the expected value of the estimate is the value of the finite-difference formula. To see this, note that the expected value of a term drawn uniformly at random from a sum is the value of the sum divided by the number of terms. Hence,

$$EX_i = 2^n E[(-1)^{s(\mathbf{x}_i)} \frac{1}{(u_1 - v_1)\cdots(u_n - v_n)} P(\mathbf{x}_i)] = 2^n \frac{1}{2^n} \text{ per } A = \text{ per } A \tag{9.7}$$

So,

$$E[\frac{X_1 + \ldots + X_m}{m}] = \frac{m \text{ per } A}{m} = \text{per } A \tag{9.8}$$

Intuitively, the estimate is likely to be better when the terms of the finite-difference formula have similar values. For example, in the extreme case that all terms have the same value, the estimate is exact regardless of the terms chosen for the sample. We place this intuition into a mathematical context using the central limit theorem, as derived in Feller [11], p. 244.

Note that random variables $X_1, \ldots, X_m$ are i.i.d., with means $\mu = EX_i = \text{per } A$ and variances:

$$\sigma^2 = E(X_i^2) - (EX_i)^2 \tag{9.9}$$

$$= \frac{1}{2^n} \sum_{\mathbf{x} \in \{u_1, v_1\} \times \ldots \times \{u_n, v_n\}} [2^n (-1)^{s(\mathbf{x})} \frac{1}{(u_1 - v_1)\cdots(u_n - v_n)} P(\mathbf{x})]^2 - (\text{per } A)^2 \tag{9.10}$$

By the central limit theorem, as sample size m increases, the distribution of the average

$$\frac{X_1 + \ldots + X_m}{m} \tag{9.11}$$

takes the shape of a Gaussian distribution with mean μ and variance $\frac{\sigma^2}{m}$. Hence, the distribution of

$$\frac{\frac{X_1 + \ldots + X_m}{m} - \mu}{\frac{\sigma}{\sqrt{m}}} \tag{9.12}$$

takes the shape of a standard normal distribution with p.d.f.

$$g(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} \tag{9.13}$$

Since $\mu = \text{per } A$,

$$\Pr\{\frac{X_1 + \ldots + X_m}{m} - \text{per } A < \beta \frac{\sigma}{\sqrt{m}}\} \rightarrow \int_{z=-\infty}^{\beta} \frac{1}{\sqrt{2\pi}} e^{\frac{1}{2}z^2} dz \text{ as } m \rightarrow \infty \tag{9.14}$$

Thus, lower variance σ^2 among finite-difference terms makes a better estimate more likely.

9.1.3 Choosing Finite-Difference Parameters to Reduce Variance Among Terms

We now examine how different finite-difference parameters $(\mathbf{u}, \mathbf{v})$ influence the variance among terms in the finite-difference formula. Recall that

$$\sigma^2 = E(X_i^2) - (EX_i)^2 \tag{9.15}$$

The term $(EX_i)^2$ is the square of per A, so it is constant with respect to the finite-difference parameters. To reduce the variance, we must reduce the other term, $E(X_i^2)$.

From (9.10),

$$E(X_i^2) = \frac{1}{2^n} \sum_{\mathbf{x} \in \{u_1,v_1\} \times \ldots \times \{u_n,v_n\}} [2^n(-1)^{s(\mathbf{x})} \frac{1}{(u_1 - v_1) \cdots (u_n - v_n)} P(\mathbf{x})]^2 \tag{9.16}$$

We now expand this expression to determine the influence of the finite-difference parameters.

$$E(X_i^2) = \frac{1}{2^n} \sum_{\mathbf{x} \in \{u_1,v_1\} \times \ldots \times \{u_n,v_n\}} 2^{2n}(-1)^{2s(\mathbf{x})} [\frac{1}{(u_1 - v_1) \cdots (u_n - v_n)}]^2 [P(\mathbf{x})]^2 \tag{9.17}$$

Simplify.

$$E(X_i^2) = 2^n [\frac{1}{(u_1 - v_1) \cdots (u_n - v_n)}]^2 \sum_{\mathbf{x} \in \{u_1,v_1\} \times \ldots \times \{u_n,v_n\}} [P(\mathbf{x})]^2 \tag{9.18}$$

Expand $p(\mathbf{x})$ by (9.3).

$$E(X_i^2) = 2^n [\frac{1}{(u_1 - v_1) \cdots (u_n - v_n)}]^2 \sum_{\mathbf{x} \in \{u_1,v_1\} \times \ldots \times \{u_n,v_n\}} [\sum_{\mathbf{p} \in Q} c(\mathbf{p}) x_1^{p_1} \cdots x_n^{p_n}]^2 \tag{9.19}$$

Expand the squared sum.

$$E(X_i^2) = 2^n [\frac{1}{(u_1 - v_1) \cdots (u_n - v_n)}]^2 \sum_{\mathbf{x} \in \{u_1,v_1\} \times \ldots \times \{u_n,v_n\}} \sum_{(\mathbf{p},\mathbf{q}) \in Q^2} c(\mathbf{p}) c(\mathbf{q}) x_1^{p_1+q_1} \cdots x_n^{p_n+q_n} \tag{9.20}$$

Let $t_j = p_j + q_j$. Swap the order of summation.

$$E(X_i^2) = 2^n \sum_{(\mathbf{p},\mathbf{q}) \in Q^2} c(\mathbf{p}) c(\mathbf{q}) [\frac{1}{(u_1 - v_1) \cdots (u_n - v_n)}]^2 \sum_{\mathbf{x} \in \{u_1,v_1\} \times \ldots \times \{u_n,v_n\}} x_1^{t_1} \cdots x_n^{t_n} \tag{9.21}$$

The coefficients $c(\mathbf{p})$ and $c(\mathbf{q})$ are determined by the problem instance, and they are constant with respect to the finite-difference parameters. For example, for the permanent of a 0-1 matrix, $c(\mathbf{p})$ is the coefficient of $x_1^{p_1} \cdots x_n^{p_n}$ in the product of row sums:

$$\prod_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_j \tag{9.22}$$

Hence, $c(\mathbf{p})$ is the number of sets of one-valued entries with one entry from each row and p_j entries from each column j.

The finite-difference parameters influence the variance through effects on the following expression from (9.21):

$$[\frac{1}{(u_1 - v_1) \cdots (u_n - v_n)}]^2 \sum_{\mathbf{x} \in \{u_1, v_1\} \times \ldots \times \{u_n, v_n\}} x_1^{t_1} \cdots x_n^{t_n} \tag{9.23}$$

The value of the sum is 2^n times the expected value of a term corresponding to an assignment $\mathbf{x}$ drawn uniformly at random from $\{u_1, v_1\} \times \ldots \times \{u_n, v_n\}$.

$$\sum_{\mathbf{x} \in \{u_1, v_1\} \times \ldots \times \{u_n, v_n\}} x_1^{t_1} \cdots x_n^{t_n} = 2^n E[x_1^{t_1} \cdots x_n^{t_n}] \tag{9.24}$$

Since the assignments to the variables in $\mathbf{x}$ are independent under this distribution,

$$\sum_{\mathbf{x} \in \{u_1, v_1\} \times \ldots \times \{u_n, v_n\}} x_1^{t_1} \cdots x_n^{t_n} = 2^n E(x_1^{t_1}) \cdots E(x_n^{t_n}) \tag{9.25}$$

Variable x_j takes values u_j and v_j with equal probability.

$$\sum_{\mathbf{x} \in \{u_1, v_1\} \times \ldots \times \{u_n, v_n\}} x_1^{t_1} \cdots x_n^{t_n} = 2^n (\frac{u_1^{t_1} + v_1^{t_1}}{2}) \cdots (\frac{u_n^{t_n} + v_n^{t_n}}{2}) \tag{9.26}$$

Simplify.

$$\sum_{\mathbf{x} \in \{u_1, v_1\} \times \ldots \times \{u_n, v_n\}} x_1^{t_1} \cdots x_n^{t_n} = (u_1^{t_1} + v_1^{t_1}) \cdots (u_n^{t_n} + v_n^{t_n}) \tag{9.27}$$

Parameters $(\mathbf{u}, \mathbf{v}) = (\mathbf{1}, -\mathbf{1})$

Suppose we set $u_j = 1$ and $v_j = -1$ for all j. Then

$$[\frac{1}{(u_1 - v_1) \cdots (u_n - v_n)}]^2 = \frac{1}{2^{2n}} \tag{9.28}$$

Recall (9.27). Note that

$$(1^{t_j} + (-1)^{t_j}) = \begin{cases} 0 & \text{if } t_j \text{ is odd} \\ 2 & \text{if } t_j \text{ is even} \end{cases} \tag{9.29}$$

Hence,

$$\sum_{\mathbf{x}\in\{u_1,v_1\}\times\ldots\times\{u_n,v_n\}} x_1^{t_1}\cdots x_n^{t_n} = \begin{cases} 0 & \text{if } t_j \text{ is odd for some } j \\ 2^n & \text{if } t_j \text{ is even for all } j \end{cases} \tag{9.30}$$

So, if we define

$$S = \{(\mathbf{p},\mathbf{q}) \in Q^2 | t_j \text{ is even for all } j\} \tag{9.31}$$

then, by substitution into (9.21),

$$E(X_i^2) = 2^n \sum_{(\mathbf{p},\mathbf{q})\in S} c(\mathbf{p})c(\mathbf{q})\frac{1}{2^{2n}} \cdot 2^n \tag{9.32}$$

Simplify.

$$E(X_i^2) = \sum_{(\mathbf{p},\mathbf{q})\in S} c(\mathbf{p})c(\mathbf{q}) \tag{9.33}$$

Parameters $(\mathbf{u},\mathbf{v}) = (\mathbf{1},\mathbf{0})$

Suppose we set $u_j = 1$ and $v_j = 0$ for all j. Then

$$[\frac{1}{(u_1 - v_1)\cdots(u_n - v_n)}]^2 = 1 \tag{9.34}$$

Recall (9.27). Note that

$$(1^{t_j} + 0^{t_j}) = \begin{cases} 1 & \text{if } t_j > 0 \\ 2 & \text{if } t_j = 0 \end{cases} \tag{9.35}$$

For a given $x_1^{t_1}\cdots x_n^{t_n}$, let k be the number of exponents with value greater than zero. Then

$$\sum_{\mathbf{x}\in\{u_1,v_1\}\times\ldots\times\{u_n,v_n\}} x_1^{t_1}\cdots x_n^{t_n} = 2^{n-k} \tag{9.36}$$

If we define

$$S_k = \{(\mathbf{p},\mathbf{q}) \in Q^2 | \mathbf{p}+\mathbf{q} \text{ has } k \text{ entries greater than zero}\} \tag{9.37}$$

then, by substitution into (9.21),

$$E(X_i^2) = 2^n \sum_{k=0}^{n} \sum_{(\mathbf{p},\mathbf{q})\in S_k} c(\mathbf{p})c(\mathbf{q})2^{n-k} \tag{9.38}$$

Simplify.

$$E(X_i^2) = 2^n \sum_{k=0}^{n} 2^{n-k} \sum_{(\mathbf{p},\mathbf{q})\in S_k} c(\mathbf{p})c(\mathbf{q}) \tag{9.39}$$

Parameters $(u_j, v_j) = (1, 0)$ and $(0, -1)$

Suppose n is even, and we set $u_j = 1$ and $v_j = 0$ for $j \in \{1, \ldots, \frac{n}{2}\}$ and $u_j = 0$ and $v_j = -1$ for $j \in \{\frac{n}{2}+1, \ldots, n\}$. Then

$$[\frac{1}{(u_1 - v_1)\cdots(u_n - v_n)}]^2 = 1 \tag{9.40}$$

Recall (9.27). Note that

$$(1^{t_j} + 0^{t_j}) = \begin{cases} 1 & \text{if } t_j > 0 \\ 2 & \text{if } t_j = 0 \end{cases} \tag{9.41}$$

and

$$(0^{t_j} + (-1)^{t_j}) = \begin{cases} -1 & \text{if } t_j > 0 \text{ and } t_j \text{ is odd} \\ 1 & \text{if } t_j > 0 \text{ and } t_j \text{ is even} \\ 2 & \text{if } t_j = 0 \end{cases} \tag{9.42}$$

Let k be the number of exponents t_j with value greater than zero. Let d be the number of exponents t_j with value greater than zero and $j \in \{\frac{n}{2}+1, \ldots, n\}$. Then

$$\sum_{\mathbf{x}\in\{u_1,v_1\}\times\ldots\times\{u_n,v_n\}} x_1^{t_1}\cdots x_n^{t_n} = (-1)^d 2^{n-k} \tag{9.43}$$

Define

$$S_{k,d} = \{(\mathbf{p},\mathbf{q}) \in Q^2 | t_j > 0 \text{ for } k \text{ entries in } \mathbf{p}+\mathbf{q} \text{ and } t_j > 0 \text{ for } d \text{ entries with } j \in \{\tfrac{n}{2}+1, \ldots, n\}\} \tag{9.44}$$

By substitution into (9.21),

$$E(X_i^2) = 2^n \sum_{k=0}^{n} \sum_{d=0}^{k} \sum_{(\mathbf{p},\mathbf{q})\in S_{k,d}} c(\mathbf{p})c(\mathbf{q})(-1)^d 2^{n-k} \tag{9.45}$$

Simplify.

$$E(X_i^2) = 2^n \sum_{k=0}^{n} 2^{n-k} \sum_{d=0}^{k} (-1)^d \sum_{(\mathbf{p},\mathbf{q}) \in S_{k,d}} c(\mathbf{p})c(\mathbf{q}) \tag{9.46}$$

The variance for these finite-difference parameters is no greater than the variance for the previous parameters. To see this, compare (9.46) to (9.39). Note that sets $S_{k,d}$ partition set S_k, so the sums involve the same pairs of generating-function terms. The difference is that the factor $(-1)^d$ in (9.46) allows some cancellation to occur.

9.1.4 Complex Parameters

So far, we have analyzed the variance for the original inclusion and exclusion parameters ($(\mathbf{u}, \mathbf{v}) = (\mathbf{1}, \mathbf{0})$) and two sets of parameters that we used earlier to increase the number of zero-valued terms in the finite-difference formula. We have shown that one of these sets has variance at least as low as the inclusion and exclusion parameters for every instance of the 0-1 matrix permanent problem. By intuition, the introduction of many zero-valued terms reduces the variance among terms as long as many large terms are not introduced as well. This is the case for our parameters. However, there must be a limit to this approach. Consider the extreme case in which the permanent is nonzero, and every term except one has value zero. Unless $O(2^n)$ sample terms are evaluated, it is very likely that all sample terms have value zero, so the estimate is likely to be zero regardless of the value of the permanent. Instead of introducing as many zero-valued terms as possible, good finite-difference parameters for estimation should make all terms have nearly equal values.

Consider complex parameters. The generating function is the product of row sums. Complex multiplication is equivalent to scaling and rotation. The rotation produces smoothing among finite-difference terms by reducing correlations among generating-function terms. This motivation is an appeal to intuition, but the intuition is supported by the following analysis and test results.

To use complex parameters, first note that (9.1) holds true for complex $\mathbf{u}$ and $\mathbf{v}$. Now redefine the sample variable from (9.5):

$$X_i = Re(2^n (-1)^{s(\mathbf{x}_i)} \frac{1}{(u_1 - v_1) \cdots (u_n - v_n)} P(\mathbf{x}_i)) \tag{9.47}$$

(Note that this reduces to the previous definition for real $\mathbf{u}$ and $\mathbf{v}$.) Once again, the estimate is

$$\frac{X_1 + \ldots + X_m}{m} \tag{9.48}$$

As before, the estimate is unbiased. To see this, note that the expected value of the real part of a term drawn uniformly at random from a sum of complex numbers is the value of the real part of the sum divided by the number of terms. Hence,

$$EX_i = 2^n E[Re((-1)^{s(\mathbf{x}_i)} \frac{1}{(u_1 - v_1)\cdots(u_n - v_n)} P(\mathbf{x}_i))] = 2^n \frac{1}{2^n} Re(\text{per } A) = \text{per } A \quad (9.49)$$

So,

$$E[\frac{X_1 + \ldots + X_m}{m}] = \frac{m \text{ per } A}{m} = \text{per } A \quad (9.50)$$

The variance among terms is still

$$\sigma^2 = E(X_i^2) - (EX_i)^2 \quad (9.51)$$

Once again, only $E(X_i^2)$ is affected by the choice of parameters. From (9.47),

$$E(X_i^2) = \frac{1}{2^n} \sum_{\mathbf{x} \in \{u_1, v_1\} \times \ldots \times \{u_n, v_n\}} [Re(2^n (-1)^{s(\mathbf{x})} \frac{1}{(u_1 - v_1)\cdots(u_n - v_n)} P(\mathbf{x}))]^2 \quad (9.52)$$

Following the derivation of (9.21), we get:

$$E(X_i^2) = 2^n \sum_{(\mathbf{p},\mathbf{q}) \in Q^2} c(\mathbf{p}) c(\mathbf{q}) Re([\frac{1}{(u_1 - v_1)\cdots(u_n - v_n)}]^2 \sum_{\mathbf{x} \in \{u_1, v_1\} \times \ldots \times \{u_n, v_n\}} x_1^{t_1} \cdots x_n^{t_n}) \quad (9.53)$$

From (9.27),

$$\sum_{\mathbf{x} \in \{u_1, v_1\} \times \ldots \times \{u_n, v_n\}} x_1^{t_1} \cdots x_n^{t_n} = (u_1^{t_1} + v_1^{t_1}) \cdots (u_n^{t_n} + v_n^{t_n}) \quad (9.54)$$

Parameters $(\mathbf{u}, \mathbf{v}) = (\mathbf{i}, -\mathbf{i})$

Suppose we set $u_j = i$ and $v_j = -i$ for all j. Then

$$[\frac{1}{(u_1 - v_1)\cdots(u_n - v_n)}]^2 = (\frac{1}{2i})^{2n} = (-\frac{1}{2}i)^{2n} = \frac{1}{2^{2n}}(-1)^n \quad (9.55)$$

Recall (9.54). Note that

$$(i^{t_j} + (-i)^{t_j}) = \begin{cases} 0 & \text{if } t_j \text{ is odd} \\ 2i^{t_j} & \text{if } t_j \text{ is even} \end{cases} \quad (9.56)$$

Hence,

$$\sum_{\mathbf{x}\in\{u_1,v_1\}\times\ldots\times\{u_n,v_n\}} x_1^{t_1}\cdots x_n^{t_n} = \begin{cases} 0 & \text{if } t_j \text{ is odd for some } j \\ 2^n i^{2n} = 2^n(-1)^n & \text{if } t_j \text{ is even for all } j \end{cases} \tag{9.57}$$

So, if we define

$$S = \{(\mathbf{p},\mathbf{q}) \in Q^2 | t_j \text{ is even for all } j\} \tag{9.58}$$

then, by substitution into (9.53),

$$E(X_i^2) = 2^n \sum_{(\mathbf{p},\mathbf{q})\in S} c(\mathbf{p})c(\mathbf{q})\frac{1}{2^{2n}}(-1)^n \cdot 2^n(-1)^n \tag{9.59}$$

Simplify.

$$E(X_i^2) = \sum_{(\mathbf{p},\mathbf{q})\in S} c(\mathbf{p})c(\mathbf{q}) \tag{9.60}$$

This is identical to the result for parameters $(\mathbf{u},\mathbf{v}) = (\mathbf{1},-\mathbf{1})$. So far, complex parameters yield no improvement. To take proper advantage of complex parameters, we must use a variety of parameter values.

Parameters $(u_j, v_j) = (i, -i)$ and $(1, -1)$

Suppose n is even, and we set $u_j = i$ and $v_j = -i$ for $j \in \{1, \ldots, \frac{n}{2}\}$ and $u_j = 1$ and $v_j = -1$ for $j \in \{\frac{n}{2}+1, \ldots, n\}$. Then

$$[\frac{1}{(u_1 - v_1)\cdots(u_n - v_n)}]^2 = (\frac{1}{2i})^n \frac{1}{2^n} = \frac{1}{2^{2n}}(\frac{1}{i})^n \tag{9.61}$$

Since $\frac{1}{i} = -i$ and n is even, we have:

$$[\frac{1}{(u_1 - v_1)\cdots(u_n - v_n)}]^2 = \frac{1}{2^{2n}} i^n \tag{9.62}$$

Recall (9.54). Note that

$$(i^{t_j} + (-i)^{t_j}) = \begin{cases} 0 & \text{if } t_j \text{ is odd} \\ 2i^{t_j} & \text{if } t_j \text{ is even} \end{cases} \tag{9.63}$$

and

$$(1^{t_j} + (-1)^{t_j}) = \begin{cases} 0 & \text{if } t_j \text{ is odd} \\ 2 & \text{if } t_j \text{ is even} \end{cases} \tag{9.64}$$

Hence,

$$\sum_{\mathbf{x}\in\{u_1,v_1\}\times\ldots\times\{u_n,v_n\}} x_1^{t_1}\cdots x_n^{t_n} = \begin{cases} 0 & \text{if } t_j \text{ is odd for some } j \\ 2^n i^{t_1+\ldots+t_{\frac{n}{2}}} & \text{if } t_j \text{ is even for all } j \end{cases} \tag{9.65}$$

Define

$$S_0 = \{(\mathbf{p},\mathbf{q}) \in Q^2 | t_j \text{ is even for all } j \text{ and } t_1 + \ldots + t_{\frac{n}{2}} \bmod 4 = 0\} \tag{9.66}$$

and

$$S_2 = \{(\mathbf{p},\mathbf{q}) \in Q^2 | t_j \text{ is even for all } j \text{ and } t_1 + \ldots + t_{\frac{n}{2}} \bmod 4 = 2\} \tag{9.67}$$

Note that $t_1 + \ldots + t_{\frac{n}{2}} \bmod 4$ is either 0 or 2 if all t_j are even. Thus, S_0 and S_2 partition the pairs of generating function terms corresponding to terms $x_1^{t_1}\cdots x_n^{t_n}$ that have nonzero value in (9.65). Pairs in S_0 have

$$\sum_{\mathbf{x}\in\{u_1,v_1\}\times\ldots\times\{u_n,v_n\}} x_1^{t_1}\cdots x_n^{t_n} = 2^n \tag{9.68}$$

Pairs in S_2 have

$$\sum_{\mathbf{x}\in\{u_1,v_1\}\times\ldots\times\{u_n,v_n\}} x_1^{t_1}\cdots x_n^{t_n} = -2^n \tag{9.69}$$

Substitution into (9.53) gives:

$$E(X_i^2) = 2^n[\sum_{(\mathbf{p},\mathbf{q})\in S_0} c(\mathbf{p})c(\mathbf{q})Re(\frac{1}{2^{2n}} i^n 2^n) + \sum_{(\mathbf{p},\mathbf{q})\in S_2} c(\mathbf{p})c(\mathbf{q})Re(\frac{1}{2^{2n}} i^n (-2)^n)] \tag{9.70}$$

Since n is even, i^n is real. If $n \bmod 4 = 0$, then $i^n = 1$, and we have:

$$E(X_i^2) = \sum_{(\mathbf{p},\mathbf{q})\in S_0} c(\mathbf{p})c(\mathbf{q}) - \sum_{(\mathbf{p},\mathbf{q})\in S_2} c(\mathbf{p})c(\mathbf{q}) \tag{9.71}$$

If $n \bmod 4 = 2$, then $i^n = -1$, and we have:

$$E(X_i^2) = \sum_{(\mathbf{p},\mathbf{q})\in S_2} c(\mathbf{p})c(\mathbf{q}) - \sum_{(\mathbf{p},\mathbf{q})\in S_0} c(\mathbf{p})c(\mathbf{q}) \tag{9.72}$$

Compare this to result (9.33), for parameters $(\mathbf{u},\mathbf{v}) = (\mathbf{i},-\mathbf{i})$. Note that S_0 and S_2 partition S. Hence, the variance with the present parameters can be no greater then the variance with the previous parameters. So the present parameters are superior. Next, we consider parameters with even more variation over the unit circle.

Parameters (u, v) from the Roots of Unity

Let r be the complex number $\frac{1}{2n}$ of the circumference around the unit circle from 1, counterclockwise. Note that $r^{2n} = 1$. Suppose we set $u_j = r^j$ and $v_j = r^{j+n} = -u_j$ for all $j \in \{1, \ldots, n\}$. Then

$$[\frac{1}{(u_1 - v_1) \cdots (u_n - v_n)}]^2 = [\frac{1}{2^n} \cdot \frac{1}{r^{1+\ldots+n}}]^2 = \frac{1}{2^{2n}} \frac{1}{r^{(n)(n+1)}} \tag{9.73}$$

Since $r^n = -1$:

$$[\frac{1}{(u_1 - v_1) \cdots (u_n - v_n)}]^2 = \frac{1}{2^{2n}}(-1)^{n+1} \tag{9.74}$$

Recall (9.54). Note that

$$((r^j)^{t_j} + (-r^j)^{t_j}) = \begin{cases} 0 & \text{if } t_j \text{ is odd} \\ 2(r^j)^{t_j} & \text{if } t_j \text{ is even} \end{cases} \tag{9.75}$$

Hence,

$$\sum_{x \in \{u_1, v_1\} \times \ldots \times \{u_n, v_n\}} x_1^{t_1} \cdots x_n^{t_n} = \begin{cases} 0 & \text{if } t_j \text{ is odd for some } j \\ 2^n \prod_{j=1}^n (r^j)^{t_j} & \text{if } t_j \text{ is even for all } j \end{cases} \tag{9.76}$$

Define

$$S = \{(\mathbf{p}, \mathbf{q}) \in Q^2 | t_j \text{ is even for all } j\} \tag{9.77}$$

Substitute into (9.53):

$$E(X_i^2) = 2^n \sum_{(\mathbf{p},\mathbf{q}) \in S} c(\mathbf{p})c(\mathbf{q})Re(\frac{1}{2^{2n}}(-1)^{n+1}2^n \prod_{j=1}^{n} (r^j)^{t_j}) \tag{9.78}$$

Simplify.

$$E(X_i^2) = \sum_{(\mathbf{p},\mathbf{q}) \in S} c(\mathbf{p})c(\mathbf{q})Re((-1)^{n+1} \prod_{j=1}^{n} (r^j)^{t_j}) \tag{9.79}$$

This is similar to the previous result (9.71), except that instead of simply adding or subtracting $c(\mathbf{p})c(\mathbf{q})$ for each $(\mathbf{p}, \mathbf{q}) \in S$, these values are multiplied by coefficients

$$Re((-1)^{n+1} \prod_{j=1}^{n} (r^j)^{t_j}) \tag{9.80}$$

Note that the complex value in parentheses is on the unit circle, so its real part is less than or equal to one. For most pairs $(\mathbf{p}, \mathbf{q}) \in S$, the real part is less than one in absolute value, so the contribution to the present result has a smaller absolute value than the contribution to the previous result. The following computations show that this produces lower variances for matrices with all one-valued entries.

9.1.5 Computations

Table 9.1 shows the values of $E(X_i^2)$ for matrices of various sizes having all one-valued entries. Results are displayed for the following finite-difference parameter settings:

- real parameters – $(\mathbf{u}, \mathbf{v}) = (\mathbf{1}, -\mathbf{1})$
- alternating parameters – $(u_j, v_j) = (i, -i)$ and $(1, -1)$
- root parameters – $(u_j, v_j) = (r^j, r^{j+n})$

The results show that the roots give the least variance, followed by the alternating parameters. The real parameters give the most variance.

For reference, the values of (per $A)^2 = (n!)^2$ are listed for each matrix size n. Subtracting this value from $E(X_i^2)$ gives the variance. The size of the permanent gives some idea of the distribution of relative error achieved by sampling. Recall that we are trying to predict the mean of a distribution with mean equal to the permanent. Through sampling, the variance goes to the variance indicated by the table divided by the number of samples.

Matrices with all one-valued entries were used in these computations because they have the maximum permanents over 0-1 matrices. Hence, these computations give some idea of how variances scale with permanents in the case of the largest permanents. Similar computations can be carried out to find the expected variances for different parameters over random 0-1 matrices with i.i.d. Bernoulli entries, i.e., for the matrix distributions considered in the analysis of fractions of zero-valued terms. Given a specific problem instance, the variance among finite-difference terms can be estimated by sampling, and it can be bounded by finding upper and lower bounds on term values, then computing the variance for the worst-case distribution with these bounds. Also, Hoeffding's inequality [18] may be used to derive probabilistic bounds on the effectiveness of estimation by sampling.

9.2 Sampling Terms by Type

In this section, we consider methods to estimate the finite-difference formula (9.1) by separately estimating sums over partitions of the terms and adding the estimates to obtain

n	real	alternating	roots	(per A)^2
2	8.0e0	4.0e0	4.0e0	4.0e0
3	1.8e2	6.1e1	4.8e1	3.6e1
4	8.3e3	1.2e3	1.1e3	5.8e2
5	6.3e5	5.3e4	4.0e4	1.4e4
6	7.1e7	2.5e6	2.1e6	5.2e5
7	1.1e10	2.2e8	1.6e8	2.5e7
8	2.4e12	2.0e10	1.5e10	1.6e9
9	6.4e14	3.0e12	1.9e12	1.3e11
10	2.2e17	4.4e14	2.9e14	1.3e13
11	9.0e19	9.9e16	5.4e16	1.6e15
12	4.5e22	2.2e19	1.2e19	2.3e17
13	2.6e25	6.8e21	3.3e21	3.9e19
14	1.8e28	2.1e24	1.0e24	7.6e21
15	1.4e31	8.8e26	3.5e26	1.7e24
16	1.3e34	3.6e29	1.4e29	4.4e26

Table 9.1: $E(X_i^2)$ for matrices with all one-valued entries. For each parameter setting, the variance of randomly sampled terms is the difference between the entry for the setting and the entry in the column on the right. Among the settings tested, the root parameters produce the minimum variance.

an estimate of the entire sum. This method is more effective than random sampling from the entire set of terms if the variances among terms in the partitions are lower than the variance among all terms. Thus, the goal is to partition the terms into sets of terms with similar values.

9.2.1 Parameters $(\mathbf{u}, \mathbf{v}) = (\mathbf{1}, -\mathbf{1})$

Consider the case in which the finite-difference parameters are $(\mathbf{u}, \mathbf{v}) = (\mathbf{1}, -\mathbf{1})$. Partition the assignments $\mathbf{x} \in \{1, -1\}^n$ according to the number of variables x_j assigned $+1$:

$$P_k = \{\mathbf{x} | \mathbf{x} \text{ has } k \text{ entries with value } +1\} \;\; \forall k \in \{0, \ldots, n\} \tag{9.81}$$

Define the partial sum:

$$S_k = \sum_{\mathbf{x} \in P_k} (-1)^{s(\mathbf{x})} \frac{1}{2^n} P(\mathbf{x}) \;\; \forall k \in \{0, \ldots, n\} \tag{9.82}$$

Note that the finite-difference formula is equal to $S_0 + \ldots + S_n$. The analogous random variable to X_i in the previous section is

$$X_{k,i} = \binom{n}{k}(-1)^{s(\mathbf{x}_i)}\frac{1}{2^n}P(\mathbf{x}_i) \tag{9.83}$$

for $\mathbf{x}_i$ chosen uniformly at random from the $\binom{n}{k}$ assignments $\mathbf{x}$ in P_k. The sampling estimate

$$\frac{X_{k,1} + \ldots + X_{k,m}}{m} \tag{9.84}$$

has expected value S_k.

For a random 0-1 matrix, it is likely that:

- Most terms in the sums S_k for k near 0 have large negative values.
- Most terms in the sums S_k for k near n have large positive values.
- Most terms in the sums S_k for k near $\frac{n}{2}$ have relatively small values.

Intuitively, it would seem that the extreme sums, S_k for k near 0 and n, would have the largest variances among terms simply because they have terms with the largest absolute values. This intuition is shown to be true by random tests on the computer. (Of course, S_0 and S_n themselves are exceptions – these "sums" each have one term, so the variances among their terms are 0.) Table 9.2 shows the averages, over 1000 randomly generated matrices of size $n = 16$ and density $p = 0.50$, of the means and standard deviations over terms in the sums S_k. The standard deviations are smallest over the sets of terms with k near $\frac{n}{2}$ – the sets with the lion's share of the terms in the formula. The exceptions are the sets with $k = 0$ and $k = n$. These sets each contain a single term, so the standard deviation is 0.

Fortunately, these extreme sums have the fewest terms. We can calculate the most extreme sums exactly by computing all terms instead of sampling. (Recall that sum S_k has $\binom{n}{k}$ terms.) For the remaining sums, samples can be allocated so that sums with higher variances are sampled more heavily. The variances among terms within sums can be estimated either by sampling or by bounding through computing bounds on the terms.

9.2.2 Parameters $(\mathbf{u}, \mathbf{v}) = (\mathbf{1}, \mathbf{0})$

Now consider finite-difference parameters $(\mathbf{u}, \mathbf{v}) = (\mathbf{1}, \mathbf{0})$. Partition the assignments $\mathbf{x} \in \{1, 0\}^n$ as follows:

$$P_k = \{\mathbf{x} | \mathbf{x} \text{ has } k \text{ entries with value } 1\} \quad \forall k \in \{0, \ldots, n\} \tag{9.85}$$

k	mean	standard deviation
0	2.8e14	0.0e0
1	-3.3e13	1.9e13
2	2.8e12	2.9e12
3	-1.5e11	2.6e11
4	4.3e9	1.4e10
5	-4.3e7	3.9e8
6	5.7e4	7.1e6
7	2.9e2	1.9e5
8	-1.4e1	3.3e4
9	2.9e2	1.9e5
10	5.7e4	7.1e6
11	-4.3e7	3.9e8
12	4.3e9	1.4e10
13	-1.5e11	2.6e11
14	2.8e12	2.9e12
15	-3.3e13	1.9e13
16	2.8e14	0.0e0

Table 9.2: The means and standard deviations over the terms with k entries of value 1 in $\mathbf{x}$ for finite-difference parameters $(\mathbf{u}, \mathbf{v}) = (\mathbf{1}, -\mathbf{1})$. The figures shown are averages over 1000 randomly generated matrices with size $n = 16$ and entry probability $p = 0.50$.

and define sums

$$S_k = \sum_{\mathbf{x} \in P_k} (-1)^{s(\mathbf{x})} P(\mathbf{x}) \quad \forall k \in \{0, \ldots, n\} \tag{9.86}$$

As k increases, the values of terms in S_k and the variances among terms in S_k increase. Unfortunately, the sums with the smallest variances do not contain the most terms, and the sums with the most terms have relatively large variances. Table 9.3 shows the averages, over 1000 randomly generated matrices of size $n = 16$ and density $p = 0.50$, of the means and standard deviations over terms in the sums S_k. The absolute values of the means and the standard deviations increase with k, except that the standard deviation for $k = 16$ is 0 since there is only one term of this type.

9.2.3 Parameters $(1, 0)$ and $(0, -1)$

Consider finite-difference parameters $(1, 0)$ and $(0, -1)$. Assume n is even. Let $u_j = 1$ and $v_j = 0$ for $j \in \{1, \ldots, \frac{n}{2}\}$ and $u_j = 0$ and $v_j = -1$ for $j \in \{\frac{n}{2} + 1, \ldots, n\}$. Partition the

k	mean	standard deviation
0	0.0e0	0.0e0
1	0.0e0	0.0e0
2	6.9e4	5.2e5
3	-4.4e7	2.0e8
4	4.3e9	1.3e10
5	-1.5e11	3.5e11
6	2.8e12	5.1e12
7	-3.3e13	4.9e13
8	2.8e14	3.5e14
9	-1.8e15	1.9e15
10	9.9e15	8.8e15
11	-4.6e16	3.4e16
12	1.8e17	1.1e17
13	-6.6e17	3.4e17
14	2.2e18	8.5e17
15	-6.5e18	1.7e18
16	1.8e19	0.0e0

Table 9.3: The means and standard deviations over the terms with k entries of value 1 in $\mathbf{x}$ for finite-difference parameters $(\mathbf{u}, \mathbf{v}) = (\mathbf{1}, \mathbf{0})$. The figures shown are averages over 1000 randomly generated matrices with size $n = 16$ and entry probability $p = 0.50$.

assignments $\mathbf{x} \in \{1, 0\}^{\frac{n}{2}} \times \{0, -1\}^{\frac{n}{2}}$ as follows:

$$P_{k,d} = \{\mathbf{x} | \mathbf{x} \text{ has } k \text{ entries with value } +1 \text{ and } d \text{ entries with value } -1 \ \forall (k,d) \in \{1, \ldots, \frac{n}{2}\}^2\} \tag{9.87}$$

and define sums:

$$S_{k,d} = \sum_{\mathbf{x} \in P_{k,d}} (-1)^{s(\mathbf{x})} P(\mathbf{x}) \quad \forall (k,d) \in \{1, \ldots, \frac{n}{2}\}^2 \tag{9.88}$$

Sums $S_{k,d}$ with small k and d have terms with small absolute values and small variances among terms, because many of the variables x_j are assigned 0. Sums $S_{k,d}$ with k and d nearly equal also tend to have terms with small absolute values, because the effects of the $+1$ and -1 assignments to x_j's tend to cancel each other, making row sums small. This parameter setting has the desirable property that terms with large absolute values tend to be confined to sums with few terms. Tables 9.4 and 9.5 show the averages, over 1000 randomly generated matrices of size $n = 16$ and density $p = 0.50$, of the means and standard deviations over terms in the sums $S_{k,d}$.

(k,d)	0	1	2	3	4	5	6	7	8
0	0.0e0	0.0e0	8.0e4	-4.7e7	4.4e9	-1.5e11	2.8e12	-3.3e13	2.8e14
1	0.0e0	0.0e0	6.0e1	2.9e4	-9.6e6	7.2e8	-2.2e10	3.9e11	-4.4e12
2	2.5e3	-1.6e1	-3.3e-1	2.2e2	-1.2e3	-2.0e6	1.6e8	-5.4e9	9.6e10
3	-1.4e6	4.0e3	3.4e2	3.6e2	5.3e1	-9.5e3	-9.5e5	9.6e7	-2.9e9
4	7.1e7	-7.3e5	5.6e2	-4.6e2	-9.0e2	8.6e3	4.1e4	-2.2e4	2.0e7
5	-3.1e9	9.4e7	-1.0e6	-3.6e4	7.5e4	-8.9e3	1.1e4	-4.2e5	9.7e6
6	1.1e11	-6.4e9	2.1e8	-1.0e6	-2.8e6	4.7e5	2.4e5	-4.8e4	-3.7e6
7	-4.1e12	3.5e11	-2.0e10	6.6e8	-1.3e6	-2.4e6	-2.2e6	9.0e5	4.4e6
8	2.7e14	-3.1e13	2.6e12	-1.4e11	3.5e9	3.9e8	-1.0e8	1.2e7	5.7e6

Table 9.4: The means over terms with k entries of value $+1$ and d entries of value -1 in $\mathbf{x}$ with finite-difference parameters (1,0) and (0,-1). The number of $+1$ entries corresponds to the row, and the number of -1 entries corresponds to the column. The figures shown are averages over 1000 randomly generated matrices with size $n = 16$ and entry probability $p = 0.50$.

(k,d)	0	1	2	3	4	5	6	7	8
0	0.0e0	0.0e0	3.3e5	1.4e8	8.8e9	2.1e11	2.6e12	1.9e13	0.0e0
1	0.0e0	0.0e0	1.2e3	7.9e5	1.1e8	4.9e9	1.1e11	1.4e12	1.2e13
2	1.3e4	3.1e2	1.7e2	3.0e4	1.7e6	9.7e7	2.9e9	5.2e10	6.0e11
3	1.0e7	2.2e5	2.5e4	3.6e4	2.3e5	4.2e6	1.3e8	2.5e9	3.0e10
4	5.9e8	2.2e7	1.0e6	1.7e5	2.7e5	1.1e6	8.7e6	1.1e8	1.6e9
5	2.3e10	1.3e9	6.6e7	5.6e6	2.7e6	1.3e6	3.8e6	2.8e7	1.8e8
6	5.8e11	4.9e10	3.4e9	2.4e8	6.2e7	1.9e7	8.2e6	9.5e6	5.0e7
7	1.1e13	1.2e12	1.1e11	8.5e9	7.5e8	1.6e8	4.5e7	1.7e7	2.1e7
8	0.0e0	2.0e13	3.1e12	3.4e11	2.9e10	3.1e9	7.9e8	8.0e7	0.0e0

Table 9.5: The standard deviations over terms with k entries of value $+1$ and d entries of value -1 in $\mathbf{x}$ with finite-difference parameters (1,0) and (0,-1). The number of $+1$ entries corresponds to the row, and the number of -1 entries corresponds to the column. The figures shown are averages over 1000 randomly generated matrices with size $n = 16$ and entry probability $p = 0.50$.

9.3 Sampling Blocks of Terms

In this section, we consider a method to sample blocks of terms instead of sampling single terms. By a block of terms, we mean a set of terms in which a subset of the finite-difference variables are assigned all possible combinations of parameter values. Let $S \subseteq \{1, \ldots, n\}$ index the variables x_j to be varied in the block. Let $\mathbf{x}'$ be an assignment in the block. Define the block sum:

$$B_{\mathbf{x}',S} = \sum_{\mathbf{x} | x_j = x'_j \ \forall j \notin S \text{ and } x_j \in \{u_j, v_j\} \ \forall j \in S} (-1)^{s(\mathbf{x})} \frac{1}{(u_1 - v_1) \cdots (u_n - v_n)} P(\mathbf{x}) \tag{9.89}$$

This is the sum over a subset of $2^{|S|}$ of the terms in the finite-difference formula (9.1).

The random variable for sampling blocks of terms that is analogous to X_i for sampling single terms is:

$$Y_i = 2^{n-|S|} B_{\mathbf{x}_i, S_i} \tag{9.90}$$

where $\mathbf{x}_i$ is chosen uniformly at random from $\{u_1, v_1\} \times \ldots \times \{u_n, v_n\}$, and S_i may be a fixed subset of $\{1, \ldots, n\}$ or a subset chosen at random, independently of $\mathbf{x}_i$. In either case, the estimate

$$\frac{Y_1 + \ldots + Y_m}{m} \tag{9.91}$$

has expected value equal to the value of the finite-difference formula.

A block of terms $B_{\mathbf{x}',S}$ can be computed more efficiently than the sum over a random sample of the same number of terms. To illustrate, assume $(\mathbf{u}, \mathbf{v}) = (\mathbf{1}, -\mathbf{1})$. Use variables r_i to accumulate row sums and use variable s to compute the sign $(-1)^{s(\mathbf{x})} = x_1 \cdots x_n$. Initially, compute partial row sums over columns with fixed values of x_j:

$$r_i := \sum_{j \notin S} a_{ij} x'_j \quad \forall i \in \{1, \ldots, n\} \tag{9.92}$$

and compute the "partial" sign using the fixed values of x_j:

$$s := \prod_{j \notin S} x'_j \tag{9.93}$$

Next, define the following procedure to assign values to variable x_j's, accumulate row sums and the sign, and compute the product of row sums for complete assignments. Set V indexes the unassigned variables. Use $\mathbf{a}^j$ to denote column j of matrix A.

```
block(V, r, s)
{
if V = ∅ then return s · $\prod_{i=1}^{n} r_i$
let j be some member of V
return block(V − j, r + $a^j$, s) + block(V − j, r − $a^j$, −s)
}
```

The function call block(S, r, s), with the values of $\mathbf{r}$ and s from (9.92) and (9.93), computes $B_{\mathbf{x}',S}$. This method requires $O(n(n-|S|)+2^{|S|}n)$ time, while straightforward evaluation of the products of row sums in the terms of the block requires $O(2^{|S|}n^2)$ time.

Because the block sum evaluates a portion of the finite-difference sieve, many of the generating function terms are zeroed. To illustrate, suppose that the finite-difference parameters are $(\mathbf{u},\mathbf{v}) = (\mathbf{1},-\mathbf{1})$. Substitute the parameter values into the block sum (9.89):

$$B_{\mathbf{x}',S} = \sum_{\mathbf{x}|x_j=x'_j\ \forall j\notin S \text{ and } x_j\in\{1,-1\}\ \forall j\in S} (-1)^{s(\mathbf{x})}\frac{1}{2^n}P(\mathbf{x}) \tag{9.94}$$

Note that

$$(-1)^{S(\mathbf{x})} = x_1\cdots x_n \tag{9.95}$$

Make this substitution, and expand $P(\mathbf{x})$ according to (9.3).

$$B_{\mathbf{x}',S} = \sum_{\mathbf{x}|x_j=x'_j\ \forall j\notin S \text{ and } x_j\in\{1,-1\}\ \forall j\in S} \frac{1}{2^n}x_1\cdots x_n \sum_{\mathbf{p}\in Q} c(\mathbf{p})x_1^{p_1}\cdots x_n^{p_n} \tag{9.96}$$

Permute sums and simplify.

$$B_{\mathbf{x}',S} = \frac{1}{2^n}\sum_{\mathbf{p}\in Q} c(\mathbf{p}) \sum_{\mathbf{x}|x_j=x'_j\ \forall j\notin S \text{ and } x_j\in\{1,-1\}\ \forall j\in S} x_1^{p_1+1}\cdots x_n^{p_n+1} \tag{9.97}$$

For a given term in the second sum, suppose p_j is even for some $j \in S$. Then p_j+1 is odd, so the second sum is zero because x_j is positive and negative in symmetric halves of the assignments in the block. Thus, if we define

$$R = \{\mathbf{p}\in Q | p_j \text{ is odd } \forall j \in S\} \tag{9.98}$$

then

$$B_{\mathbf{x}',S} = \frac{1}{2^n} \sum_{\mathbf{p} \in R} c(\mathbf{p}) \sum_{\mathbf{x} | x_j = x'_j \ \forall j \notin S \text{ and } x_j \in \{1,-1\} \ \forall j \in S} x_1^{p_1+1} \cdots x_n^{p_n+1} \tag{9.99}$$

since the generating function terms indexed by $P - R$ are zeroed.

9.4 Discussion

Recently, several algorithms have been developed to estimate the permanent of a 0-1 matrix [20, 21, 22, 28]. The ultimate goal is to develop a fully polynomial randomized approximation scheme (FPRAS), i.e., a randomized algorithm that takes as input an $n \times n$ 0-1 matrix A and a real number $\epsilon > 0$, and in time polynomial in both n and $\frac{1}{\epsilon}$ produces output $Y(A, \epsilon)$ with the property:

$$\Pr\{(1-\epsilon) \text{ per } A \le Y(A,\epsilon) \le (1+\epsilon) \text{ per } A\} \ge \frac{3}{4} \tag{9.100}$$

Karmarkar, Karp, Lipton, Lovász, and Luby [22] designed a randomized algorithm for which

$$\Pr\{(1+\epsilon)^{-1} \text{ per } A \le Y(A,\epsilon) \le (1+\epsilon) \text{ per } A\} \ge 1 - \delta \tag{9.101}$$

The algorithm has time complexity $O(\text{poly}(\delta, \epsilon, n) 2^{\frac{n}{2}})$, which is about the square root of the $O(n^2 2^n)$ time required to compute the permanent using Ryser's algorithm [29]. More recently, Jerrum and Vazirani [21] have developed an approximation algorithm with time complexity $O(\text{poly}(\delta, \epsilon, n) 2^{\sqrt{n} \log^2 n})$. Furthermore, there are polynomial time algorithms for several restricted classes of 0-1 matrices. For more information, see [28], Ch. 11.

The approximation methods developed in this chapter do not fit into the FPRAS framework. There are no guarantees regarding the ratio of our estimators to the permanent, for reasons involving algorithm design and theory. The FPRAS-type algorithms [21, 22] operate by sampling permanent terms at random or evaluating determinants, which involve signed permanent terms. Our sampling procedures evaluate products of row sums, which may have many terms that are not permanent terms. These extra terms may overwhelm the permanent terms by any ratio, since even if the permanent is zero, the product of row sums may be nonzero. The FPRAS-type algorithms approximate by samples or "signed" samples of the objects we wish to count; our procedures approximate by sampling terms from a sieve formula that involves many objects besides those that we wish to count. Hence, the FPRAS-type algorithms must be designed specifically to sample the desired objects, while our procedures apply to general finite-difference formulas.

There are theoretical limits on the types of problems that can be tackled by FPRAS. Intuitively, to sample objects at random in polynomial time, we must be able to generate the objects in polynomial time. If the existence problem associated with our counting problem is NP-complete, then we do not know how to generate the objects in polynomial time. Formally, let $N(A)$ be the number that we wish to compute, given instance A with size poly(n). (For example, in (9.100), $N(A) = \text{per } A$ and A is an $n \times n$ 0-1 matrix.) The existence of an FPRAS for $N(A)$ with output $Y(A, \epsilon)$ means that

$$\Pr\{(1-\epsilon)N(A) \leq Y(A,\epsilon) \leq (1+\epsilon)N(A)\} \geq \frac{3}{4} \tag{9.102}$$

and $Y(A, \epsilon)$ can be computed in time polynomial in n and $\frac{1}{\epsilon}$. Set $\epsilon = \frac{1}{2}$. Then

$$\Pr\{\frac{1}{2}N(A) \leq Y(A,\frac{1}{2}) \leq \frac{3}{2}N(A)\} \geq \frac{3}{4} \tag{9.103}$$

and $Y(A, \frac{1}{2})$ can be computed in time polynomial in n. Note that the FPRAS identifies whether or not $N(A) = 0$ with probability at least $\frac{3}{4}$. If $N(A) = 0$, then $Y(A, \frac{1}{2}) = 0$ with probability at least $\frac{3}{4}$. If $N(A) \geq 1$, then $Y(A, \frac{1}{2}) > 0$ with probability at least $\frac{3}{4}$. Hence, the FPRAS solves the existence problem associated with $N(A)$ with probability at least $\frac{3}{4}$. For example, if $N(A)$ is the problem of counting Hamiltonian cycles in the graph with adjacency matrix A, then the FPRAS solves the Hamiltonian cycle existence problem with probability at least $\frac{3}{4}$. Existence problems having this type of solution constitute the class BPP. Since the Hamiltonian cycle existence problem is NP-complete, an FPRAS for the Hamiltonian cycle problem would imply that all NP problems are in BPP. This is considered unlikely. Unless it is true, there is no FPRAS for a counting problem that corresponds to an NP-complete existence problem. This argument is adapted from [28], Ch. 11.

For even stronger results on the difficulty of approximating some specific problems, refer to the paper by Zuckerman [35]. There, it is shown that unless NP=P, the permanent of a matrix with entries in $\{-1, 0, 1\}$ and with a positive permanent cannot be accurately approximated by any polynomial time procedure.

A paper by Linial and Nisan [27] contains some results on approximating the size of a union of sets by the sum over a subset of the terms in the inclusion and exclusion formula. They show that if the sizes of the sets and the sizes of intersections of K or fewer of the sets are known, and if $K \geq \Omega(\sqrt{n})$, then the size of the union can be estimated to within a multiplicative factor of $1 \pm e^{-\Omega(K/\sqrt{n})}$.

Conclusion

In this work, we develop and demonstrate the finite-difference technique to produce counting algorithms:

1. Construct a generating function in which one type of terms corresponds to the objects to be counted.

2. Apply the proper finite-difference operators to produce a formula that counts the terms.

3. Choose finite-difference parameters to reduce the computation required to evaluate the formula.

Finite-difference algorithms have advantages over other counting algorithms. Finite-difference algorithms require less storage space than their dynamic programming counterparts. Also, finite-difference algorithms have fewer cross-references than dynamic programming algorithms, which is an advantage for parallel computation on message-passing multicomputers. Finite-difference algorithms are a generalization of inclusion and exclusion algorithms. The free parameters of finite-difference formulas can be chosen to produce algorithms that are faster than their inclusion and exclusion counterparts.

In combinatorics, generating functions are used to count the structures in a fixed superstructure [34] or to solve a recursion [26]. For these applications, a fixed generating function is produced, and the desired coefficients are deduced through analysis. In this work, a class of generating functions is developed for each problem. Given a problem instance, a method to compute the corresponding generating function is produced, and the desired coefficient is determined through evaluation of finite-differences. Generating functions are usually reserved for the realm of proof, but this work applies them to the realm of computation.

Appendix A

Gaussian Approximations for Sums and Differences Over Two Sets of Bernoulli Variables

At several points in the preceeding chapters, (6.11), (7.19), and (7.32), we have approximated the distribution of the difference between sums over two sets of Bernoulli variables by Gaussians. These differences have the form:

$$x_1 + \ldots + x_k - x_{k+1} - \ldots - x_n \tag{A.1}$$

where the variables are i.i.d.

We need approximations for which the multiplicative errors can be made arbitrarily small by increasing n and for which the additive error is $o(e^{-\frac{1}{2}n^{0.28}})$. The approximations are needed only for k such that

$$|k - \frac{n}{2}| < c_1 n^{c_2} \tag{A.2}$$

where c_1 and c_2 are constants, with $0 < c_1 < \infty$ and $0 < c_2 < 1$. The approximations need only be asymptotic. They need not hold for all n, but for any given multiplicative error tolerance greater than zero, there must be some N such that the approximations are within the multiplicative and additive error tolerances for all $n \geq N$.

This appendix supplies approximations in which for every multiplicative error tolerance greater than zero, there exists a pair K and K' such that for all $k \geq K$ and $n - k \geq K'$, the approximations are within the multiplicative and additive error tolerances. The approximations have additive error $o(e^{-\frac{1}{2}k^{0.32}}) + o(e^{-\frac{1}{2}(n-k)^{0.32}})$.

To show that this appendix supplies sufficient approximations, first consider the additive error. Note that

$$|k - \frac{n}{2}| < c_1 n^{c_2} \text{ implies } |(n-k) - \frac{n}{2}| < c_1 n^{c_2} \tag{A.3}$$

So

$$k > \frac{n}{2} - c_1 n^{c_2} \text{ and } n - k > \frac{n}{2} - c_1 n^{c_2} \tag{A.4}$$

As $n \to \infty$, these lower bounds go to $\frac{n}{2}$ because $c_2 < 1$. So, for k restricted so that $|k - \frac{n}{2}| < c_1 n^{c_2}$, there is some value of N such that for all $n \geq N$,

$$e^{-\frac{1}{2}k^{0.32}} + e^{-\frac{1}{2}(n-k)^{0.32}} < e^{-\frac{1}{2}n^{0.28}} \tag{A.5}$$

Hence, the additive error bound supplied by this appendix is sufficient.

Now consider the multiplicative error. Since we need approximations only for k such that

$$|k - \frac{n}{2}| < c_1 n^{c_2} \text{ and } |(n-k) - \frac{n}{2}| < c_1 n^{c_2} \tag{A.6}$$

we need approximations for k and $n - k$ with the following ratios:

$$\frac{\frac{n}{2} - c_1 n^{c_2}}{\frac{n}{2} + c_1 n^{c_2}} < \frac{k}{n-k} < \frac{\frac{n}{2} + c_1 n^{c_2}}{\frac{n}{2} - c_1 n^{c_2}} \tag{A.7}$$

Since c_1 is finite and $c_2 < 1$, the bounding ratios go to 1 as $n \to \infty$. Hence, for any lower bound less than 1 and upper bound greater than 1, for all n sufficiently large, all pairs $(k, n-k)$ that meet condition A.6 have a ratio within our bounds. To be specific, choose lower bound $\frac{1}{2}$ and upper bound 2. Then our approximations are sufficient if they are within error tolerances for all $(k, n-k)$ such that

$$\frac{1}{2} < \frac{k}{n-k} < 2 \tag{A.8}$$

The appendix supplies approximations within error tolerances for all $k \geq K$ and $n-k \geq K'$, where K and K' depend on the error tolerances. By taking $N = 3\max(K, K')$, we ensure that for all $n \geq N$, for all pairs $(k, n-k)$ that meet condition A.8, $k \geq K$ and $n - k \geq K'$. For example, given $K = 10$ and $K' = 10$, if $N \geq 30$ then all pairs $(k, n-k)$ such that

$$\frac{1}{2} < \frac{k}{n-k} < 2 \tag{A.9}$$

have $k \geq 10$ and $n - k \geq 10$. Since we can choose N such that, for all $n \geq N$, the approximations supplied by this appendix meet the error tolerance requirements, this appendix supplies sufficient asymptotic approximations.

To use notation from Feller [11], in which k plays a different role, we use n in the role of k and n' in the role of $n-k$ in the remainder of this appendix. Thus, n here corresponds to $n+n'$ in the remainder of this appendix, and K and K' here correspond to $N(\epsilon)$ and $N'(\epsilon)$ in the remainder of this appendix.

We use the following process to prove that a Gaussian approximation is valid for the difference A.1. First, we show that a Gaussian approximation is valid for the distribution of the sum over two sets of independent Bernoulli variables with identical distributions within each set, but possibly different distributions between the sets. Then, we show that the distribution of the difference between two sets can be transformed to the distribution of the sum over two sets.

A.1 The Sum Over Two Sets of Bernoulli Variables

Suppose we have two sets of independent Bernoulli variables that are identically distributed within each set, and we want to estimate the distribution of the sum of the random variables by a Gaussian. Let n be the number of variables in the first set. Let $0 < p < 1$ be the probability that each variable in the first set takes value one, and let $q = 1 - p$ be the probability that each variable takes value zero. Similarly, let n' be the number of variables in the second set. Let $0 < p' < 1$ be the probability that each variable in the second set takes value one, and let $q' = 1 - p'$ be the probability that each variable in the second set takes value zero.

Denote the first set of variables $\{x_1, \ldots, x_n\}$. Denote the second set of variables $\{y_1, \ldots, y_{n'}\}$. Define b_s to be the probability that the sum has value s:

$$b_s = \Pr\{x_1 + \ldots + x_n + y_1 + \ldots + y_{n'} = s\} \tag{A.10}$$

Define B_s to be the Gaussian p.d.f. with the same mean and variance as the sum over the two sets of bernoulli variables.

$$B_s = \frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{npq + n'p'q'}} e^{-\frac{1}{2}\frac{[s-(np+n'p')]^2}{npq+n'p'q'}} \tag{A.11}$$

We will prove the following:

Theorem 1 *For every $\epsilon > 0$, there exist $N(\epsilon)$ and $N'(\epsilon)$ such that for all $n \geq N(\epsilon)$ and*

$n' \geq N'(\epsilon)$, for all $s \in \{0, \ldots, n+n'\}$,

$$(1-\epsilon)B_s + o(e^{-\frac{1}{2}n^{0.32}}) + o(e^{-\frac{1}{2}(n')^{0.32}}) < b_s < (1+\epsilon)B_s + o(e^{-\frac{1}{2}n^{0.32}}) + o(e^{-\frac{1}{2}(n')^{0.32}}) \quad \text{(A.12)}$$

Proof. We will use results from Feller [11] for the sum over a single set of i.i.d. variables to derive results for the sum over a pair of sets. Following the notation in Feller [11], Ch. VII, let m be the unique integer

$$m = np + \delta \text{ with } -q < \delta \leq p \quad \text{(A.13)}$$

and let m' be the unique integer

$$m' = n'p' + \delta' \text{ with } -q' < \delta' \leq p' \quad \text{(A.14)}$$

The values m and m' index the central terms of the binomial distributions generated by sums over the sets of variables. Denote the terms of the distributions as follows:

$$a_k = b(m+k; n, p) = \binom{n}{m+k} p^{m+k} q^{n-(m+k)} \quad \text{(A.15)}$$

i.e., the probability that the first set of variables sums to $m+k$ is denoted by a_k. Also,

$$a'_{k'} = b(m'+k'; n', p') = \binom{n'}{m'+k'} (p')^{m'+k'} (q')^{n'-(m'+k')} \quad \text{(A.16)}$$

The following lemma is proved in Feller [11], p. 184:

Lemma 1 *If $n \to \infty$ and k is constrained to an interval $|k| < K_n$ such that $K_n^3/n^2 \to 0$, then for every $\epsilon > 0$ and n sufficiently large,*

$$1 - \epsilon < \frac{a_k}{hg(kh)} < 1 + \epsilon \quad \text{(A.17)}$$

where $h = \frac{1}{\sqrt{npq}}$ and $g()$ is the normal distribution, i.e., $g(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}$. Hence, for sufficiently small k,

$$a_k \sim hg(kh) \quad \text{(A.18)}$$

Define B'_s to be the Gaussian p.d.f. with mean $m+m'$ and the same variance as B_s.

$$B'_s = \frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{npq + n'p'q'}} e^{-\frac{1}{2}\frac{[s-(m+m')]^2}{npq+n'p'q'}} \quad \text{(A.19)}$$

To prove the theorem, we will prove two more lemmas. The first states that B'_s approximates b_s, and the second states that B_s approximates B'_s.

Lemma 2 *For every $\epsilon > 0$ there exist $N(\epsilon)$ and $N'(\epsilon)$ such that for all $n \geq N(\epsilon)$ and $n' \geq N'(\epsilon)$, for all $s \in \{0, \ldots, n+n'\}$,*

$$(1-\epsilon)B'_s + o(e^{-\frac{1}{2}n^{0.32}}) + o(e^{-\frac{1}{2}(n')^{0.32}}) < b_s < (1+\epsilon)B_s + o(e^{-\frac{1}{2}n^{0.32}}) + o(e^{-\frac{1}{2}(n')^{0.32}}) \quad \text{(A.20)}$$

Proof of Lemma. Note that

$$b_s = \sum_{(k,k')|m+k+m'+k'=s \text{ and } 0 \leq m+k \leq n \text{ and } 0 \leq m'+k' \leq n'} a_k a'_{k'} \quad \text{(A.21)}$$

To simplify notation, denote the set of summation by S:

$$S = \{(k,k')|m+k+m'+k' = s \text{ and } 0 \leq m+k \leq n \text{ and } 0 \leq m'+k' \leq n'\} \quad \text{(A.22)}$$

Separate the sum into terms for which k and k' are small enough that the Gaussian approximations hold for a_k and $a'_{k'}$ and other terms. The approximations are valid for $|k| \leq n^{0.66}$ and $|k'| \leq (n')^{0.66}$.

$$b_s = \sum_{(k,k') \in S \ | \ |k| \leq n^{0.66} \text{ and } |k'| \leq (n')^{0.66}} a_k a'_{k'} + \sum_{(k,k') \in S \ | \ |k'| > n^{0.66} \text{ or } |k'| > (n')^{0.66}} a_k a'_{k'} \quad \text{(A.23)}$$

Examine the second sum. Since $a_k \leq 1$ and $a'_{k'} \leq 1$ for all values of k and k', the second sum is no greater than

$$\sum_{|k| > n^{0.66}} a_k + \sum_{|k'| > (n')^{0.66}} a'_{k'} \quad \text{(A.24)}$$

According to Feller [11], p. 193, (6.7), these sums are

$$\text{o}(e^{-\frac{1}{2}n^{0.32}}) \text{ and } \text{o}(e^{-\frac{1}{2}(n')^{0.32}}) \quad \text{(A.25)}$$

Hence,

$$b_s = \sum_{(k,k') \in S \ | \ |k| \leq n^{0.66} \text{ and } |k'| \leq (n')^{0.66}} a_k a'_{k'} + \text{o}(e^{-\frac{1}{2}n^{0.32}}) + \text{o}(e^{-\frac{1}{2}(n')^{0.32}}) \quad \text{(A.26)}$$

Now use the Gaussian approximations for a_k and $a'_{k'}$, with $h = \frac{1}{\sqrt{npq}}$ and $h' = \frac{1}{\sqrt{n'p'q'}}$.

$$b_s = [\sum_{(k,k')\in S \ | \ |k|\leq n^{0.66} \text{ and } |k'|\leq (n')^{0.66}} hg(kh)h'g(k'h')]E_1 + o(e^{-\frac{1}{2}n^{0.32}}) + o(e^{-\frac{1}{2}(n')^{0.32}}) \tag{A.27}$$

where E_1 is the error made in the sum through the use of Gaussian approximations. Using (A.17), we see that for every $\epsilon > 0$ and n and n' sufficiently large, for all (k, k') with $|k| \leq n^{0.66}$ and $|k'| \leq (n')^{0.66}$,

$$(1-\epsilon)^2 < \frac{a_k a'_{k'}}{hg(kh)h'g(k'h')} < (1+\epsilon)^2 \tag{A.28}$$

Since a_k, $a'_{k'}$, $hg(kh)$, and $h'g(k'h')$ are all nonnegative for all (k, k'), we have

$$(1-\epsilon)^2 < E_1 < (1+\epsilon)^2 \tag{A.29}$$

Now estimate the sum by an integral.

$$\sum_{(k,k')\in S \ | \ |k|\leq n^{0.66} \text{ and } |k'|\leq (n')^{0.66}} hg(kh)h'g(k'h') = [\int_{k=-\infty}^{\infty} hg(kh)h'g((s-m-m'-k)h')dk]E_2+E_3 \tag{A.30}$$

where E_2 is the error due to estimating the sum by the integral over the range $|k| \leq n^{0.66}$ and $|k'| \leq (n')^{0.66}$, and E_3 is the error due to extending the range of k (and thus k') to $(-\infty, +\infty)$.

The integral itself is the value at $s-m-m'$ of the convolution of two zero-mean Gaussians with variances npq and $n'p'q'$. So the integral is the value at $s - m - m'$ of a zero-mean Gaussian with variance $npq + n'p'q'$:

$$\frac{1}{\sqrt{npq+n'p'q'}}\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}\frac{(s-(m+m'))^2}{npq+n'p'q'}} \tag{A.31}$$

This is the value at s of a Gaussian with mean $m + m'$ and variance $npq + n'p'q'$. So, error terms aside, the distribution of the sum of the two sets of Bernoulli variables can be approximated by a Gaussian with mean within two of the sum of the means (since $|(m + m') - (np + n'p')| \leq 2$) and with variance equal to the sum of the variances.

Consider E_2, the error due to estimating the sum

$$\sum_{(k,k')\in S| \ |k|\leq n^{0.66} \text{ and } |k'|\leq (n')^{0.66}} hg(kh)h'g(k'h') \tag{A.32}$$

by the integral

$$\int_{k=\max(-n^{0.66},-(n')^{0.66})}^{\min(n^{0.66},(n')^{0.66})+1} hg(kh)h'g(k'h')dk \tag{A.33}$$

where $k' = s - m - m' - k$.

Denote by $E_2(k)$ the ratio of a single term in the sum to the integral over the corresponding region.

$$E_2(k) = \frac{hg(kh)h'g(k'h')}{\int_{x=0}^{1} hg((k+x)h)h'g((k'-x)h')dx} \tag{A.34}$$

Note that

$$\frac{hg(kh)h'g(k'h')}{\max_{x\in[0,1]} hg((k+x)h)h'g((k'-x)h')} \leq E_2(k) \leq \frac{hg(kh)h'g(k'h')}{\min_{x\in[0,1]} hg((k+x)h)h'g((k'-x)h')} \tag{A.35}$$

Examine the bounding fractions for $x \in [0,1]$.

$$\frac{hg(kh)h'g(k'h')}{hg((k+x)h)h'g((k'-x)h')} = \frac{g(kh)g(k'h')}{g((k+x)h)g((k'-x)h)} \tag{A.36}$$

Expand the Gaussians and subtract exponents to divide.

$$e^{-\frac{1}{2}h^2[k^2-(k+x)^2]-\frac{1}{2}(h')^2[(k')^2-(k'-x)^2]} \tag{A.37}$$

Simplify.

$$e^{-\frac{1}{2}h^2[-2kx-x^2]-\frac{1}{2}(h')^2[2k'x-x^2]} \tag{A.38}$$

Recall that $h = \frac{1}{\sqrt{npq}}$ and $h' = \frac{1}{\sqrt{n'p'q'}}$.

$$e^{-\frac{1}{2}\frac{1}{npq}[-2kx-x^2]-\frac{1}{2}\frac{1}{n'p'q'}[2k'x-x^2]} \tag{A.39}$$

For $|k| \leq n^{0.66}$ and $|k'| \leq (n')^{0.66}$, the exponent goes to zero as n and n' increase, so the exponential goes to one. Thus, for any $\epsilon > 0$, for all $x \in [0,1]$, for all n and n' sufficiently large, and for $|k| \leq n^{0.66}$ and $|k'| \leq (n')^{0.66}$,

$$1 - \epsilon < e^{-\frac{1}{2}\frac{1}{npq}[-2kx-x^2]-\frac{1}{2}\frac{1}{n'p'q'}[2k'x-x^2]} < 1 + \epsilon \tag{A.40}$$

Hence, for every $\epsilon > 0$ and n and n' sufficiently large, and for $|k| \leq n^{0.66}$ and $|k'| \leq (n')^{0.66}$,

$$1 - \epsilon < E_2(k) < 1 + \epsilon \tag{A.41}$$

Now consider the entire sum and integral. Since both the sum terms and the integrals over corresponding regions are all positive, the previous result for separate terms and regions implies the following. For every $\epsilon > 0$ and n and n' sufficiently large, and for $|k| \leq n^{0.66}$ and $|k'| \leq (n')^{0.66}$,

$$1 - \epsilon < E_2 < 1 + \epsilon \tag{A.42}$$

Finally, examine E_3, the error due to extending the ranges of k and k' in the approximating integral. Since $hg(kh) \leq 1$ and $h'g(k'h') \leq 1$, E_3 is no greater than

$$2\int_{k>n^{0.66}} hg(kh)dk + 2\int_{k'>(n')^{0.66}} h'g(k'h')dk' \tag{A.43}$$

According to the tail bound from Feller [11], p. 175, (1.7),

$$\int_{k>n^{0.66}} hg(kh)dk < \frac{1}{n^{0.66}} g(n^{0.66}h) = o(e^{-\frac{1}{2}n^{0.32}}) \tag{A.44}$$

Hence,

$$E_3 = o(e^{-\frac{1}{2}n^{0.32}}) + o(e^{-\frac{1}{2}(n')^{0.32}}) \tag{A.45}$$

Combining the results for the integral and errors in (A.27) and (A.30) completes the proof of the lemma. □

To complete the proof of the theorem, we show that B_s approximates B'_s when B'_s is larger than the additive error term allowed by the theorem.

Lemma 3 *For every $\epsilon > 0$, there exist $N(\epsilon)$ and $N'(\epsilon)$ such that for all $n \geq N(\epsilon)$ and $n' \geq N'(\epsilon)$, for all $s \in \{0, \ldots, n+n'\}$,*

$$(1-\epsilon) < \frac{B'_s}{B_s} < (1+\epsilon) \text{ if } B'_s > e^{-\frac{1}{2}n^{0.32}} \text{ or } B_s > e^{-\frac{1}{2}n^{0.32}} \tag{A.46}$$

Proof of Lemma. Recall that

$$B'_s = \frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{npq + n'p'q'}} e^{-\frac{1}{2}\frac{[s-(m+m')]^2}{npq+n'p'q'}} \tag{A.47}$$

and

$$B_s = \frac{1}{\sqrt{2\pi}} \frac{1}{\sqrt{npq + n'p'q'}} e^{-\frac{1}{2}\frac{[s-(np+n'p')]^2}{npq+n'p'q'}} \tag{A.48}$$

Let $\sigma^2 = npq + n'p'q'$. Then

$$\frac{B'_s}{B_s} = e^{-\frac{1}{2}\frac{1}{\sigma^2}[(s-(m+m'))^2 - (s-(np+n'p'))^2]} \tag{A.49}$$

We will show that both an upper bound and a lower bound for the ratio go to 1 as either n or n' increases, given that

$$B'_s > e^{-\frac{1}{2}n^{0.32}} \text{ or } B_s > e^{-\frac{1}{2}n^{0.32}} \tag{A.50}$$

Recall that $|m - np| \leq 1$ and $|m' - n'p'| \leq 1$. Hence,

$$|(m + m') - (np + n'p')| \leq 2 \tag{A.51}$$

Without loss of generality, assume $s \geq m + m'$. Then decreasing $np + n'p'$ decreases B_s, so $np + n'p' = m + m' - 2$ maximizes the ratio. So we have the upper bound:

$$\frac{B'_s}{B_s} \leq e^{-\frac{1}{2}\frac{1}{\sigma^2}[(s-(m+m'))^2-(s-(m+m'-2))^2]} \tag{A.52}$$

The upper bound is equal to

$$e^{\frac{2}{\sigma}\frac{s-(m+m')}{\sigma}} e^{\frac{2}{\sigma^2}} \tag{A.53}$$

We can use $np + n'p' = m + m' - 2$ to derive the equivalent upper bound:

$$e^{\frac{2}{\sigma}\frac{s-(np+n'p')}{\sigma}} e^{-\frac{2}{\sigma^2}} \tag{A.54}$$

We will show that condition (A.50) implies that this upper bound goes to 1 as $n \to \infty$. Assume n is large enough that $\frac{1}{\sqrt{2\pi}\sigma} < 1$. Then

$$B'_s > e^{-\frac{1}{2}n^{0.32}} \tag{A.55}$$

implies

$$e^{-\frac{1}{2}\frac{(s-(m+m'))^2}{\sigma^2}} > e^{-\frac{1}{2}n^{0.32}} \tag{A.56}$$

So

$$(\frac{s-(m+m')}{\sigma})^2 < n^{0.32} \tag{A.57}$$

and

$$|\frac{s-(m+m')}{\sigma}| < n^{0.16} \tag{A.58}$$

Likewise,

$$B_s > e^{-\frac{1}{2}n^{0.32}} \tag{A.59}$$

implies

$$e^{-\frac{1}{2}\frac{(s-(np+n'p'))^2}{\sigma^2}} > e^{-\frac{1}{2}n^{0.32}} \tag{A.60}$$

So

$$(\frac{s-(np+n'p')}{\sigma})^2 < n^{0.32} \tag{A.61}$$

and

$$|\frac{s-(np+n'p')}{\sigma}| < n^{0.16} \tag{A.62}$$

Combining results, we see that condition (A.50) implies (A.58) or (A.62).

Substituting (A.58) into (A.53) produces the upper bound:

$$e^{\frac{2n^{0.16}}{\sigma}} e^{\frac{2}{\sigma^2}} \tag{A.63}$$

Since $\sigma = \sqrt{npq + n'p'q'}$, both exponents go to 0 as either n or n' increase, and the bound goes to 1. Likewise, substituting (A.62) into (A.54) produces the upper bound:

$$e^{\frac{2n^{0.16}}{\sigma}} e^{-\frac{2}{\sigma^2}} \tag{A.64}$$

Once again, the exponents go to 0, so the bound goes to 1 as either n or n' increase.

Now we derive a lower bound for the ratio of B'_s to B_s. Examine (A.49). Without loss of generality, assume $s \geq m+m'$. Then increasing $np+n'p'$ increases B_s until $s-(np+n'p') = 0$. For now, assume $s - (m + m') \geq 2$, so that $s - (np + n'p') = 0$ does not occur within the range of possible values for $np + n'p'$ given by $|(m + m') - (np + n'p')| \leq 2$. Then $np + n'p' = m + m' + 2$ minimizes the ratio. So we have the lower bound:

$$\frac{B'_s}{B_s} \leq e^{-\frac{1}{2}\frac{1}{\sigma^2}[(s-(m+m'))^2-(s-(m+m'+2))^2]} \tag{A.65}$$

The lower bound is equal to

$$e^{-\frac{2}{\sigma}\frac{s-(m+m')}{\sigma}} e^{\frac{2}{\sigma^2}} \tag{A.66}$$

We can use $np + n'p' = m + m' + 2$ to derive the equivalent lower bound:

$$e^{-\frac{2}{\sigma}\frac{s-(np+n'p')}{\sigma}} e^{-\frac{2}{\sigma^2}} \tag{A.67}$$

Note that lower bounds (A.66) and (A.67) are similar to upper bounds (A.53) and (A.54). As for the upper bounds, assume that condition (A.50) holds. Then (A.58) or (A.62) holds. Substitute (A.58) into (A.66) or substitute (A.62) into (A.67) to derive a lower bound in which the exponents go to 0 as either n or n' increase, so the lower bound goes to 1.

Now consider the case $s - (m + m') < 2$. The maximum possible value of B_s is achieved by $np + n'p'$ such that $s - (np + n'p') = 0$, since B_s is a Gaussian. Maximizing B_s minimizes the ratio. So we have the lower bound:

$$\frac{B'_s}{B_s} \leq e^{-\frac{1}{2}\frac{1}{\sigma^2}[(s-(m+m'))^2-0^2]} \tag{A.68}$$

Since $s - (m + m') < 2$, this bound is at least

$$e^{-\frac{2}{\sigma^2}} \tag{A.69}$$

Since σ increases as either n or n' increase, the exponent goes to 0, and the lower bound goes to 1. □

To prove the theorem, combine the last two lemmas. Since B_s approximates B'_s, and B'_s approximates b_s, B_s approximates b_s. □

A.2 The Difference Between Two Sets of Bernoulli Variables

We need to apply this result to distributions of the form:

$$x_1 + \ldots + x_n - y_1 - \ldots - y_{n'} \tag{A.70}$$

where the variables are all independent, the x_j's are identically distributed, and the y_j's are identically distributed. To fix notation, let

$$x_j = \begin{cases} 1 & \text{with probability } p \\ 0 & \text{with probability } q = 1 - p \end{cases} \tag{A.71}$$

and let

$$y_j = \begin{cases} 1 & \text{with probability } p' \\ 0 & \text{with probability } q' = 1 - p' \end{cases} \tag{A.72}$$

Our result applies to sums of Bernoulli variables, so the subtracted variables may cause concern. We can alleviate this concern as follows. Replace each $-y_j$ by $-1 + y'_j$, where y'_j

is the Bernoulli variable

$$y'_j = \begin{cases} 1 & \text{with probability } q' = 1 - p' \\ 0 & \text{with probability } p' \end{cases} \tag{A.73}$$

Note that $-1 + y'_j$ has the same distribution as $-y_j$. So the sum can be rewritten as

$$x_1 + \ldots + x_n + y'_1 + \ldots + y'_{n'} - n' \tag{A.74}$$

This is the sum of two sets of Bernoulli variables and a single constant translation term. Our theorem applies to the sum over the sets of variables. To account for the translation term, translate the index s in b_s by $-n'$.

Bibliography

[1] T. Araki, Y. Sugiyama, T. Kasami, and J. Okui, Complexity of the deadlock avoidance problem, *Proc. 2nd IBM Symp. on Mathematical Foundations of Computer Science*, IBM Japan, Tokyo 229-252.

[2] E. Bax, Inclusion and exclusion algorithm for the Hamiltonian path problem, *Inform. Process. Lett.*, 27 (4) (1993) 203-207.

[3] E. Bax, Algorithms to count paths and cycles, *Inform. Process. Lett.*, 52 (1994) 249-252.

[4] E. Bax, Recurrence-based reductions for inclusion and exclusion algorithms applied to #P problems, CalTech-CS-TR-96-01.

[5] E. Bax, Tailoring the permanent formula to problem instances, CalTech-CS-TR-96-17.

[6] E. Bax and J. Franklin, A finite-difference sieve to count paths and cycles by length, *Inform. Process. Lett.*, 60 (1996) 171-176.

[7] E. Bax and J. Franklin, A finite-difference sieve to compute the permanent, CalTech-CS-TR-96-04.

[8] E. Bax and J. Franklin, A permanent formula with many zero-valued terms, *Inform. Process. Lett.*, 63 (1997) 33-39.

[9] R. Bellman, Dynamic programming treatment of the travelling salesman problem, *J. Assoc. Comput. Mach.*, 9 (1962) 61-63.

[10] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North-Holland 1976.

[11] W. Feller, *An Introduction to Probability Theory and Its Applications*, John Wiley and Sons, Inc. 1968.

[12] J. Franklin, *Methods of Mathematical Economics* pp.68-79, Springer-Verlag New York, Inc. 1980.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability – A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[14] E. M. Gold, Deadlock prediction: easy and difficult cases, *SIAM J. Comput.*, 7, 320-336.

[15] A. N. Habermann, Prevention of system deadlocks, *Comm. ACM*, 12, 373-377, 385.

[16] M. Held and R. M. Karp, A dynamic programming approach to sequencing problems, *J. Soc. Indust. Appl. Math*, 10 (1962) 196-210.

[17] F. B. Hildebrand, *Introduction to Numerical Analysis*, McGraw-Hill 1956.

[18] W. Hoeffding, Probability inequalities for sums of bounded random variables, *Am. Stat. Assoc. J.*, (1963):13-30.

[19] R. C. Holt Some deadlock properties of computer systems, *ACM Computing Surveys*, 4, 179-196.

[20] M. R. Jerrum and A. Sinclair, Approximating the permanent, *SIAM Journal on Computing*, 18(6):1149-1178, December 1989.

[21] M. Jerrum and U. Vazirani, A mildly exponential approximation algorithm for the permanent, *Algorithmica*, 16(1996):392-401.

[22] N. Karmarkar, R. Karp, R. Lipton, L. Lovász, and M. Luby, A Monte Carlo algorithm for estimating the permanent, *SIAM Journal on Computing*, 22(2):284-293, April 1993.

[23] R. M. Karp, Reducibility among combinatorial problems, in R. E. Miller and J. W. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 85-103.

[24] R. M. Karp, Dynamic programming meets the principle of inclusion and exclusion, *Oper. Res. Lett.*, 1 (2) (1982) 49-51.

[25] H. W. Kuhn, The Hungarian method for the assignment problem, *Naval Res. Logist. Quart.*, 2 (1955) 83-97.

[26] J. H. van Lint and R. M. Wilson, *A Course in Combinatorics*, Cambridge University Press 1992.

[27] N. Linial and N. Nisan, Approximate inclusion-exclusion, *Combinatorica*, 10 (4) (1990) 349-365.

[28] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press 1995, pp. 315-329.

[29] H. J. Ryser, *Combinatorial Mathematics*, The Mathematical Association of America 1963, Ch. 2.

[30] A. Sinclair, *Algorithms for Random Generation and Counting: A Markov Chain Approach*, Birkhäuser, Boston 1993.

[31] R. Tarjan, Enumeration of the elementary circuits of a directed graph, *SIAM J. Comput.*, 2 (3) (1973) 211-216.

[32] L. G. Valiant, The complexity of computing the permanent, *Theoretical Computer Science*, 8(1979):189-201.

[33] L. G. Valiant, The complexity of enumeration and reliability problems, *SIAM J. Comput.*, 8 (3) (1979) 410-421.

[34] H. Wilf, *Generating Functionology*, Academic Press, Boston 1994.

[35] D. Zuckerman, On unapproximable versions of NP-complete problems, *SIAM J. Comput.*, 25 (6) (1996) 1293-1304.