EXPLORATIONS IN COMPLEX NETWORK THEORY: INTERFEROMETER NETWORKS AND THE APPLICATION OF GENERATING FUNCTIONS TO MATRICES

by

Benjamin A. Krawciw

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Computational and Applied Mathematics).

Golden, Colorado

Date _____

Signed: _____

Benjamin A. Krawciw

Signed: _____

Dr. Cecilia Diniz Behn
Thesis Advisor

Signed: _____

Dr. Lincoln D. Carr
Thesis Advisor

Golden, Colorado

Date _____

Signed: _____

Dr. Gerrald Greivel
Professor and Department Head
Department of Applied Mathematics and Statistics

ABSTRACT

Complex network theory examines the structures that arise from systems like neurons in the brain, members of societies, and pages on the world wide web. I push the field forward in two distinct areas: I address the current inability of complex network theory to handle networks weighted with complex numbers; and I introduce a novel way of expressing networks through the generating function technique. The interest in networks based on complex numbers is motivated by a desire to extend complex network theory to problems in physics which involve traveling signals with phase, such as waves traveling through networks, interfering constructively and destructively depending on their phase at the point of interaction. My new analysis of this class of networks centers around the case of interferometry. I call these complex-valued networks *interferometer networks*. The work combining generating functions and matrices is motivated by the success generating functions have had in treating large sequences with recursive structure. Large networks can be similarly constructed as a sequence with recursion, making generating functions a possible tool for examining networks.

I explore interferometer networks with structures very different to traditional interferometers. Most interferometers compare the phases of only two beams of light. They function by counting the difference in the number of cycles of light waves, which occur at a rate of 1 cycle per 2π radians of phase shift. For more complicated interferometers, I prove that they still only produce 1 cycle per 2π radians of phase shift as long as the phase shift occurs at only one site in the network. However, if this phase shift is split up over multiple places in the network, this limitation no longer holds. I produce an interferometer model, called the *N-stage skew-cycle interferometer*, which breaks this limitation. The N-stage skew-cycle interferometer serves as a demonstration that large changes in output phase can arise from small changes in inputs.

To incorporate the traditional tools of complex network theory, I take two well-known measures in complex network theory, path length and clustering, and generalize them to interferometer networks. These measures are extended to the complex numbers, creating versions that account for the constructive and destructive interference of waves travelling over the network. I call my newly-defined measures *apparent path strength* and *interferometric clustering*.

There are instances of interferometer networks for which apparent path strength is undefined, especially in the case of feedback loops. This chapter identifies a real-world case where this can arise: the cavity of a Fabry-Perot interferometer. Subsequently, I prove that unique signal solutions exist and are bounded if the ℓ_1 norm of the weighted adjacency matrix is less than one. This allows the interferometer network research to continue with a guarantee that my newly-defined measures exist and are bounded.

I explore the small-world effect in the context of interferometer networks. The small world effect is a phenomenon that occurs in real-world networks. In the language of clustering and path length, small-world networks tend to have high clustering and short path lengths. This behavior is quantified by a measure called the *small-world coefficient*. I adapt a model for this behavior to produce interferometer networks. Then, I computationally test those networks using my newly-defined measures. I found that the small-world coefficient, adapted for interferometer networks, depends heavily on the presence of phase in the networks. The small-world coefficient computed with the generalized measures ranges from slightly lower than the small-world coefficient computed with real-valued network measures to several times higher, demonstrating that interferometric network measures are necessary to capture the behavior of the model. This result concludes the investigations into interferometer networks.

Subsequently, I explore the application of generating functions to adjacency matrices. I create two representations of networks: the *array generating function* and the *transformation operator*. The array generating function stores the entries of an adjacency matrix on a power series of x and y, where powers of x correspond to rows and powers of y correspond to columns. The transformation operator acts on the coefficients of a generating function the same way that matrices act on the entries of a column vector. After defining the array generating function and the transformation operator, I establish some of their properties and demonstrate how to convert between adjacency matrices, array generating functions, and transformation operators. The result of this work is the creation of new ways of expressing networks, which allows convenient ways of generating networks and performing calculations on them.

Finally, I discuss the discoveries made throughout the thesis and consider what those results mean for future network research. Interferometer networks, with their associated interferometric measures, have potential applications in quantum random walks, condensed matter models, complex-valued neural networks, and network analysis of alternating-current circuits. Particular problems include analytically modeling the small-world coefficient of the small-world interferometer model and demonstrating the applicability of interferometric measures by applying them to an existing data set. The application of generating functions to matrices likewise creates opportunities for new research: using array generating functions to compute network measures, using generating function techniques to find matrix decompositions, and using repeated applications of the transformation operator to create a generating function that enumerates path lengths on an unlabeled network.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

A matrix with a single entry at row k, column l . . . . . Δ^{kl}

A vector b . . . . . $\vec{b}$

The (complex or real number) weighted adjacency matrix . . . . . W

The ℓ_1 norm of an expression (vector or matrix), $[\cdot]$ . . . . . $\|[\cdot]\|_1$

The clustering coefficient at node j . . . . . C_j

The coefficient of x^m in the power series $f(x)$ . . . . . $[x^m]f(x)$

The complex conjugate of a complex number z . . . . . $z^\dagger$

The constant source term supplying vertices in an interferometer network . . . . . $\vec{S}$

The end of a proof . . . . . $\square$

The entry on row i, column j of a matrix B . . . . . B_{ij}

The entry on row i, column j of a matrix expression $\cdot$ . . . . . $[\cdot]_{i,j}$

The error of a variable x . . . . . δx

The formal power series antiderivative . . . . . $\hat{I}$

The formal power series derivative . . . . . $\hat{D}$

The matrix of apparent path lengths of a network . . . . . l_P

The matrix of apparent path strengths of a network . . . . . P

The maximum value of a set A . . . . . $\max_{x \in A} x$

The nth entry of a sequence a . . . . . a_n

The signals traveling around an interferometer network . . . . . $\vec{E}$

The supremum of a set A . . . . . $\sup_{x \in A} x$

The unweighted adjacency matrix . . . . . A

$f(x) = \sum_{n=0}^{\infty} a_n x^n$ is the ordinary power series (OPS) generating function of the sequence $\{a_n\}$ . . . . . $\{a_n\} \overset{OPS}{\longleftrightarrow} f(x)$

LIST OF ABBREVIATIONS

Apparent Path Length . . . . . APL

Apparent Path Strength . . . . . APS

Array Generating Function . . . . . AGF

Ordinary Power Series . . . . . OPS

Strongest Path Length . . . . . SPL

Strongest Path Strength . . . . . SPS

The N-stage Skew-Cycle Interferometer . . . . . NSCI

Transformation Operator . . . . . TO

ACKNOWLEDGMENTS

First and foremost, I am thankful for the cadre of mentors I have gathered over my time at Mines: Dr. Lincoln Carr, Dr. Cecilia Diniz Behn, and Dr. Antwan Clark. I owe a tremendous debt of gratitude to Dr. Lincoln Carr, who snatched me up into research during a group advising session during my freshman year, and has driven me to succeed at every stage of research. I am likewise indebted to Dr. Cecilia Diniz Behn, who has been a tremendous advocate and advisor throughout my time in the math department, and even before then. I would like to express my gratefulness for Dr. Antwan Clark, who, without any previous connection to me, found me out of the blue and brought me to the Laboratory for Physical Sciences. I had a great time exploring network theory deeply with him during the summer of 2022. Additionally, I thank Dr. Dinesh Mehta for serving as the chair of my thesis committee.

I am personally grateful for my parents, who have nurtured and provided for me from birth through college. I will never be able to repay that debt, but I suppose that is the nature of unconditional love. I am likewise grateful to the Navigators, who have been like a family to me during my time at the Colorado School of Mines. May I never lose the vision of knowing Christ, making him known, and helping others do the same.

This research was funded by National Science Foundation Grant DCCF 1839232. Also, I acknowledge the Colorado School of Mines High Performance Computing resources (`https://ciarc.mines.edu/hpc/`) made available for conducting the research reported in this thesis.

CHAPTER 1

INTRODUCTION

Complex network theory has been used to describe large interacting systems in diverse contexts including sociology [1, 2], the analysis of technological networks like electrical grids [3] and the internet [4, 5], and neuroscience [6, 7]. I seek to expand the field in two broad ways: first, in Chapters 2-5, I extend complex network theory to networks based on complex numbers, which I call *interferometer networks*. Second, in Chapter 6, I apply the generating function technique to matrices, which serves as a novel way to express networks. In this Chapter, I introduce the language of complex networks, interferometer networks, and generating functions, then I outline the rest of the thesis.

Before I begin to discuss the new discoveries, I must introduce the language of complex network theory [8]. A network is a set of objects called vertices, which are connected to one another by edges. These edges can be directed, going either from vertex A to vertex B, or they can be undirected, connecting A and B in no particular order. Networks are often depicted with vertices as dots and edges as lines between the dots. Directed edges are drawn as arrows. An example of a directed network is shown in Figure 1.1.

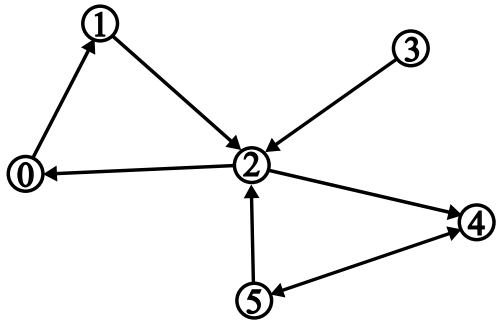

Figure 1.1 An example of a directed network. The vertices in the network are indexed with natural numbers. The vertices are connected by arrows, denoting directed edges.

If we index the vertices with distinct numbers, we can represent the edges in the network with an adjacency matrix. In an adjacency matrix, the position of the row of an entry corresponds to the vertex an edge is entering, the position of the column of an entry represents the index of the vertex the edge is coming from, and an entry of 1 denotes the existence of an edge, while an entry of 0 denotes the absence of

an edge. The adjacency matrix for the network in Figure 1.1 is

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \tag{1.1}$$

Networks where edges simply exist or do not exist are called *binary networks*. Networks that have edges of varying connection strength have *weights*, which, traditionally, are real numbers. The weight often quantifies the strength of a connection, but it can also quantify the distance between two vertices, or any other number that one may want to associate with an edge. These weights can be stored in a weighted adjacency matrix, usually called W instead of A, where entries are real numbers instead of just 1 or 0. The number 0 is still usually associated with the absence of an edge.

Now that I have introduced the basic notation of networks, I can clarify the meaning of a complex network. Complex networks are networks that have too many vertices to examine by mere inspection, have edges that are too disordered to fit into simple patterns like rings and lattices, yet also have too much structure to fit into a completely random network model. This field is necessitated by the structure of real-world systems like societies and the brain, which are large and structured, but also variable. Thus, the field simultaneously involves problems of computing with large systems, defining measures to quantify structural properties, and the use of statistical analysis to examine variability in these systems.

Current network-theoretic explorations of interfering systems are limited by the need to exclude phase information so that those networks can be quantified by existing real-valued measures. Three especially pertinent examples of works that face this limitation are the "Complex-network description of thermal quantum states in the Ising spin chain" [9], the "Real-space visualization of quantum phase transitions by network topology" [10], and "Detecting quantum critical points in the t-t' Fermi-Hubbard model via complex network theory" [11]. In all of these works, a quantum problem is examined, which exhibits interfering complex numbers, but the phase information stored by those complex numbers is thrown away so that the results can fit into traditional network measures. Other phase-based problems include interferometry, the Aharonov-Bohm Effect [12], and the "Quantum Interference Effect Transistor" [13]. To accommodate these problems, network models and network measures need to be created with complex-number edge weights. After these generalizations are introduced, they can analyze optical and quantum systems of sizes that were previously intractable while also accounting for interference effects. The tools we develop here have uses in modeling complex optical systems, detecting phase transitions in statistical mechanics, and quantifying decoherence in quantum computers.

To adapt complex network theory to complex-valued problems, I begin by considering the physical problem of interferometry. In an interferometer, light from a source is split into multiple paths. Each path travels differently, and accumulates its own distinct phase. At other points in the interferometer, the distinct paths of light are recombined. Since light takes the form of waves, these paths with their distinct phases create waves that do not align perfectly with each other. When they are added together, they interfere, meaning that the light observed at the output of an interferometer does not necessarily have the combined strength of all the beams that combine. The example of the Michelson interferometer [14], which was used to attempt to measure a difference in the speed of light in two orthogonal directions, is depicted in Figure 1.2. The measurement takes place by having light split into the paths at a half-silvered mirror. On each path, the light encounters a regular mirror, the light reflects back into the half-silvered mirror, and the light recombines. The intensity of the light leaving the half-silvered mirror is measured. The difference in travel time causes a difference in the phase of the light beams when they recombine.

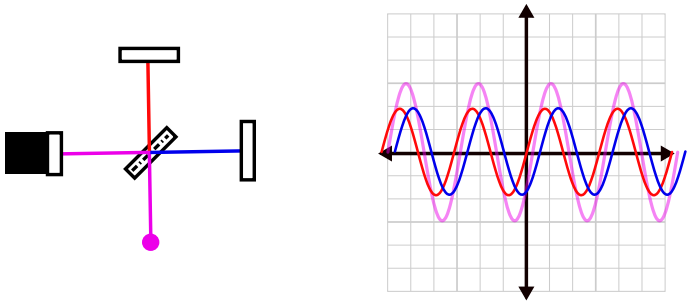

Figure 1.2 Left: the Michelson interferometer, which measures the difference in light's travel time between two paths (red and blue). The left is the light source. The center of the interferometer is a half-silvered mirror. Each of the two paths includes a regular mirror. The bottom is the observer. Right: two waves of differing phase being added together, illustrating interference.

I define interferometer networks as follows: interferometer networks are directed networks with edges weighted by a complex number. The weighted adjacency matrix W contains these complex edge weights. Each vertex has an associated value, corresponding to some kind of signal. In the case of interferometer networks, this signal is the electric field strength at the vertex. The vertex indexed at i has a signal value E_i. The signal vector $\vec{E}$ contains the signals at each vertex. The signal E_i is the sum of two inputs: signals traveling over edges to the vertex and a constant source term. The incoming edges carry a signal equal to the edge weight W_{ij} times the incident vertex's signal E_j. The constant source terms, S_i for each vertex i, are contained in a source vector $\vec{S}$. In total, this produces Equation 1.2.

$$E_i = S_i + \sum_j W_{ij} E_j. \tag{1.2}$$

The entire system is then described by a matrix equation, Equation 1.3, the vertex signal equation:

$$\vec{E} = W\vec{E} + \vec{S}. \tag{1.3}$$

To demonstrate the creation of an interferometer network, we start with a well-known type of interferometer: the Sagnac interferometer [15]. This interferometer measures the speed of its rotation by splitting a beam of light into two paths. The first beam travels in a clockwise loop. The other beam travels the same loop, but counter-clockwise. When the Sagnac interferometer spins, the rotation makes one of the directions of travel a shorter distance than the other. These differences in distance create a difference in the number of wavelengths traveled by each beam, which results in a difference of phase. When the beams are recombined, the interference between the two beams allows the angular velocity to be calculated.

Example 1.1. The Sagnac interferometer as an interferometer network

Consider the Sagnac interferometer. I depict the Sagnac interferometer as an interferometer network in Figure 1.3, then I use the vertex signal equation (Equation 1.3) to solve for the observed electric field strength at the output. The parameters for the Sagnac interferometer are the wavenumber k, the speed of light c, the radius of the interferometer loop r, and the interferometer's angular velocity ω.

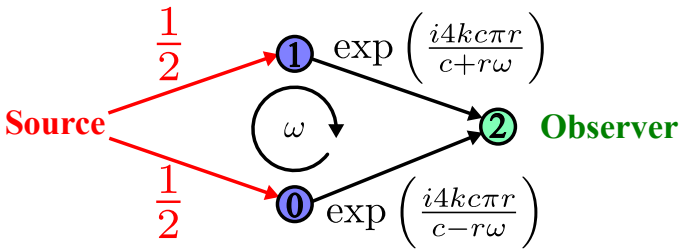

Figure 1.3 The Sagnac interferometer expressed as an interferometer network. On the network diagram, the source is indicated with red text and lines, the blue vertices are intermediary vertices, and the green vertex is the observer vertex. The parameters for are the wavenumber k, the speed of light c, the radius of the interferometer loop r, and the interferometer's angular velocity ω.

The Sagnac interferometer network's weighted adjacency matrix is expressed by Equation 1.4.

$$W = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \exp\left(\frac{i4kc\pi r}{c+r\omega}\right) & \exp\left(\frac{i4kc\pi r}{c-r\omega}\right) & 0 \end{bmatrix}. \tag{1.4}$$

The constant source supplied to the network is represented by the vector in Equation 1.5.

$$\vec{S} = \begin{bmatrix} 1/2 \\ 1/2 \\ 0 \end{bmatrix}. \tag{1.5}$$

When we solve the vertex signal equation (Equation 1.3) for the vertex electric field strength vector $\vec{E}$, we get the vector in Equation 1.6.

$$\vec{E} = \begin{bmatrix} 1/2 \\ 1/2 \\ \frac{1}{2}\left(\exp\left(\frac{i4kc\pi r}{c+r\omega}\right) + \exp\left(\frac{i4kc\pi r}{c-r\omega}\right)\right) \end{bmatrix}. \tag{1.6}$$

This matches the established output of the Sagnac interferometer [15], which can be expressed as Equation 1.7.

$$E_{\text{out}} = \frac{1}{2}\left(\exp\left(\frac{i4kc\pi r}{c+r\omega}\right) + \exp\left(\frac{i4kc\pi r}{c-r\omega}\right)\right). \tag{1.7}$$

Although the definition of interferometer networks is inspired by the specific problem of interferometry, its simple form is quite versatile. Interferometer networks can be adapted to other complex-valued signal transfer problems. Recontextualizing the interferometer network formalism primarily involves reinterpreting or slightly changing the vertex signal equation (Equation 1.3) to match the problem. The vertex signal equation can be easily adapted to model the time evolution of state vectors in quantum walks [16–18]; inputs, states, and observables in complex-valued observability and controlability problems [19, 20]; and the matrix analysis of node voltages in alternatring-current circuits with complex impedance [21, 22].

Now that I have defined interferometer networks, the next important mathematical idea to define is the *generating function* technique. Generating functions encode a sequence of numbers in the coefficients of a power series [23]. Generating functions traditionally find use in combinatorics and probability theory [24]. Notably, generating functions are used to prove the central limit theorem [25]. To illustrate their usefulness in addressing large sequences with recursion, we will find an explicit formula for the n^{th} Fibonacci number using generating functions.

Example 1.2. Finding the n^{th} term of the Fibonacci sequence The Fibonacci sequence is defined by a recursion relation and the first two terms.

$$F_0 = 0,\ F_1 = 0. \tag{1.8}$$

For all $n > 0$,

$$F_{n+1} = F_n + F_{n-1}. \tag{1.9}$$

We will find an explicit expression for the n^{th} term of the Fibonacci sequence using a generating function (per a derivation adapted from [23]).

First, we will define the generating function for this series. Then, we will solve for it by using the recursion relation.

$$F(x) = \sum_{n=0}^{\infty} F_n x^n = 0 + x + x^2 + 2x^3... \tag{1.10}$$

Since the recursion relation holds for each individual term, it also holds for the sum of terms times x^{n+1}.

$$\sum_{n=1}^{\infty} F_{n+1} x^{n+1} = \sum_{n=1}^{\infty} F_n x^{n+1} + \sum_{n=1}^{\infty} F_{n-1} x^{n+1}. \tag{1.11}$$

$$\Rightarrow \sum_{n=2}^{\infty} F_n x^n = x \sum_{n=1}^{\infty} F_n x^n + x^2 \sum_{n=0}^{\infty} F_n x^n. \tag{1.12}$$

$$\Rightarrow F(x) - x = xF(x) + x^2 F(x). \tag{1.13}$$

Therefore,

$$F(x) = \frac{x}{1 - x - x^2}. \tag{1.14}$$

This simple rational expression has encoded the entire Fibonacci sequence. Now, to find the explicit expression for each term, we need to find the coefficient of each power of x. We will start by splitting the denominator with the quadratic formula, then by performing partial fraction decomposition.

$$F(x) = \frac{x}{1 - x - x^2} \tag{1.15}$$

$$= \frac{x}{-\left(x + \frac{1+\sqrt{5}}{2}\right)\left(x + \frac{1-\sqrt{5}}{2}\right)} \tag{1.16}$$

$$= \frac{x}{\left(1 - x\frac{1+\sqrt{5}}{2}\right)\left(1 - x\frac{1-\sqrt{5}}{2}\right)} \tag{1.17}$$

$$= \frac{1}{\sqrt{5}}\left[\frac{1}{1 - x\left(\frac{1+\sqrt{5}}{2}\right)} - \frac{1}{1 - x\left(\frac{1-\sqrt{5}}{2}\right)}\right]. \tag{1.18}$$

This is still a pair of rational functions, and we need a power series. However, we have manipulated this expression into a form similar to that of the geometric series [24], which takes the form

$$\frac{1}{1 - z} = \sum_{n=0}^{\infty} z^n. \tag{1.19}$$

With this form in mind, we can express the generating function as

$$F(x) = \frac{1}{\sqrt{5}}\left[\sum_{n=0}^{\infty}\left(x\frac{1+\sqrt{5}}{2}\right)^n - \sum_{n=0}^{\infty}\left(x\frac{1-\sqrt{5}}{2}\right)^n\right]. \tag{1.20}$$

Finally, taking the coefficients from both power series together, we see

$$F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]. \tag{1.21}$$

This process of taking a recursive sequence, treating it as the coefficients of a power series, and performing computations on that power series is the essence of generating functions.

In Chapter 6, I adapt the generating function technique to apply to adjacency matrices, then I explore the properties of these new objects. The motivation for this is that large networks can be constructed from smaller networks through a recursive algorithm. This process of growing a network from a smaller one resembles the kind of series the generating function technique is useful for.

The novelty of the approach we take to generating functions and matrices arises from presenting generating function objects as alternatives to matrices. The precursors to this work include other multivariate generating functions, which are similar to the array generating function, but do not exist to encode the entries of matrices [26, 27], and previous work employing generating functions to study the properties of matrices, including the distribution of Toeplitz matrix eigenvalues [28], the generation of normal Toeplitz matrices [29], and the use of the transfer matrix method for ennumerating graph walks [30]

Now, I outline the rest of the thesis. Chapter 2 explores the possibility of creating interferometers more sensitive than basic interferometers measuring the difference between only two paths. Chapter 3 explains the well-known network measures of path length and clustering, then adapts those measures for interferometer networks. Chapter 4 explores amplification in signals due to feedback loops and describes how to limit that feedback. Chapter 5 visits the small-world effect, which describes how real-world networks simultaneously form tight clusters while still having short paths between distant parts of the network [31]. I apply the newly adapted measures to a version of small-world networks created for interferometers. The small-world effect looks very different for interferometer networks; destructive interference due to large phase causes the breakdown of signal transfer between neighbors and distant parts of the network, while constructive interference due to small phases can strengthen signal transfer greatly. The results in Chapters 3 and 5 arise from joint work with my advisors, Cecilia Diniz Behn and Lincoln Carr. A manuscript reporting these results is in preparation and will be submitted to the *Journal of Physics: Complexity*. Chapter 6 departs from interferometer networks. Instead, I develop a new technique for expressing networks using an existing technique called *generating functions*. This work was undertaken with Antwan Clark at the Laboratory for Physical Sciences. Here, I define two new ways of expressing networks: the *array generating function* and the *transformation operator*, then I prove some of their basic properties, show how to convert between them, and give examples of using them to describe networks. Finally, Chapter 7 discusses the results, further connections to the rest of the field, and ways to build upon this work in future research.

CHAPTER 2

INTERFEROMETER LIMITATION AND THE N-STAGE SKEW-CYCLE INTERFEROMETER (NSCI)

In its simplest form, an interferometer splits a beam of light, adds a phase to one of the paths, and compares the two paths at an observer. This simple scheme is depicted in Figure 2.1. The signal at the observer is written out in Equation 2.2.

$$E_{\text{observer}} = \frac{1}{2}\left(E_{\text{source}} + e^{i\phi}E_{\text{source}}\right) \tag{2.1}$$

$$= e^{i\phi/2}E_{\text{source}}\cos(\phi/2). \tag{2.2}$$

For each applied phase shift of 2π, this interferometer produces one fringe (a signal of zero). To take a measurement, the phase would be applied gradually, and the fringes would be counted. Well-known interferometers like the Michelson interferometer [14] or the Sagnac interferometer [15] can be expressed in the form of this archetypal interferometer, where the particulars of the problem merely change the expression for the phase ϕ. One of the first research questions that intrigued me was, "For an applied phase ϕ, is there an interferometer that produces more than one fringe per 2π phase shift?" My initial conjecture was that there was not such an interferometer. I called this conjecture "interferometer limitation." I proved a special case of this–the case where phase can only be applied to one edge (Figure 2.2)–but I found an interesting counterexample for when the measured phase is allowed to be split up and applied over many edges. In this chapter, I will give the proof of the special case, then I will explain the counterexample for the general case, which I have called the N-stage skew-cycle interferometer (NSCI).

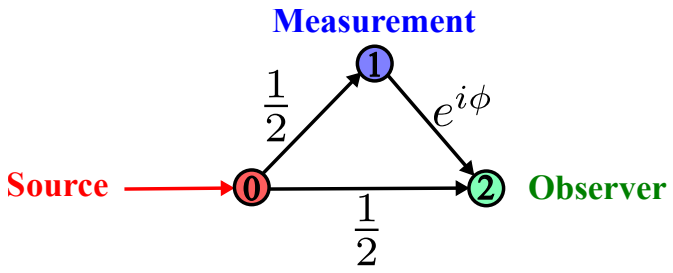

Figure 2.1 The archetypal interferometer, with a source, splitting, an applied phase, and recombination at an observer

For a certain type of interferometer, no improvement can be made over the archetypal interferometer of Figure 2.1. To be more specific, I will prove that any interferometer network with only one variable edge phase cannot produce more than one fringe per 2π applied phase shift. This proof will have two steps. First, I will prove that this holds true when the input to the variable edge is fixed. Second, I will extend this to the general case, when the input to the variable edge is not fixed.

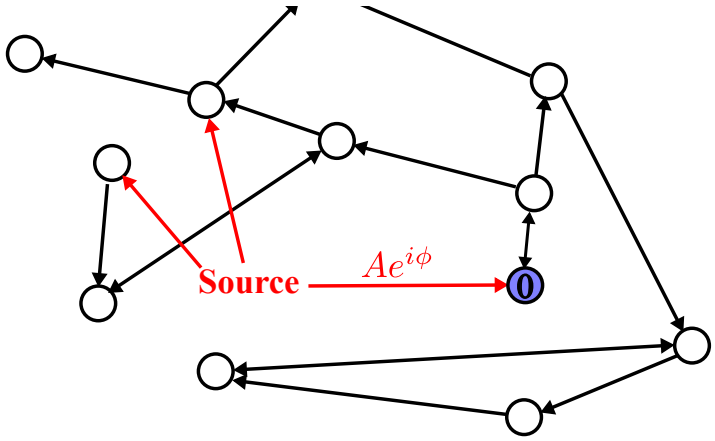

Figure 2.2 Example of a network where only one edge has a variable phase, ϕ, and the input to this variable edge is held fixed. For convenience, the destination of this variable edge is indexed at zero. Theorem 2.1 applies to networks like these.

Theorem 2.1 (Interferometer Limitation for Single Edge with Fixed Source)**.** *Suppose we have an interferometer network with n vertices and m edges. This network has a constant complex weighted adjacency matrix W, but the zeroth term of the source vector $\vec{S}$ is allowed to rotate such that*

$$S_0 = Ae^{i\phi}. \tag{2.3}$$

Also assume that the vertex-signal equation (Equation 1.3) has a unique solution for all source vectors. Then, the signal vector takes the form

$$E_j = A\left[(I-W)^{-1}\right]_{j,0} e^{i\phi} + B_j, \tag{2.4}$$

where

$$B_j = \sum_{k=1}^{n-1} \left[(I-W)^{-1}\right]_{j,k} S_k. \tag{2.5}$$

The result of this theorem is that each vertex signal changes like a constant times $e^{i\phi}$ plus another constant. For an interferometer that improves upon the archetypal interferometer, we would want the vertex signals to change like $e^{ig\phi}$, where $g > 1$. So, interferometers of the form described in Theorem 2.1 do

not contradict the interferometer limitation hypothesis.

Proof. Since we have assumed that the vertex-signal equation (Equation 1.3) has a solution for every source vector $\vec{S}$, the fundamental theorem of invertible matrices [32, 172] tells us that Equation 1.3 is invertible.

$$\vec{E} = (I - W)^{-1} \vec{S}. \tag{2.6}$$

Writing the matrix multiplication in index notation, this becomes

$$E_j = \sum_{k=0}^{n-1} \left[(I - W)^{-1}\right]_{j,k} S_k \tag{2.7}$$

$$= \left[(I - W)^{-1}\right]_{j,0} S_0 + \sum_{k=1}^{n-1} \left[(I - W)^{-1}\right]_{j,k} S_k. \tag{2.8}$$

Equation 2.4 arises when we make the substitutions $S_0 = Ae^{i\phi}$, $B_j = \sum_{k=1}^{n-1} \left[(I - W)^{-1}\right]_{j,k} S_k$. □

Now, I must extend this to the case where the variable edge is not attached to the fixed source, but occurs within the network itself, as a part of the adjacency matrix W.

Theorem 2.2 (Interferometer Limitation for Single Edge)**.** *Suppose we have an interferometer network with n vertices and m edges. The network has a fixed source vector $\vec{S}$. One of the edges in the network is variable, with edge weight $Ae^{i\phi}$. The rest of the edge weights are fixed, with the whole network having adjacency matrix W. Without loss of generality, assume that the edge comes from vertex 0 and goes to vertex 1, such that $W_{1,0} = Ae^{i\phi}$. Finally, also assume that the vertex-signal equation (Equation 1.3) has a unique solution for every source vector $\vec{S}$. Then, the vertex signals take the form*

$$E_0 = \frac{1}{se^{i\phi} + t}, \tag{2.9}$$

$$E_{j\neq 0} = \frac{u_j e^{i\phi} + v_j}{se^{i\phi} + t}, \tag{2.10}$$

where s, t, u_j, v_j are constant complex numbers.

Proof. The idea of this proof is to transform the vectors in the vertex-signal equation to artificially fix the signal at the source vertex of the variable edge. Then, we can apply the previous theorem. We must first lay out two cases for our network: If $E_0 = 0$, the input to the variable edge is also zero, and the outputs cannot change as the phase shifts. This case is trivial. If $E_0 \neq 0$, the variable edge is allowed to influence the network, and we can also divide everything in the vertex signal equation (Equation 1.3) by E_0. For this second case, we can begin manipulating the vertex signal equation (Equation 1.3).

To aid in our algebraic manipulation, we will use partitioned matrix notation to split up the elements of the vertex signal equation (Equation 1.3) into smaller pieces so we can rearrange them later. First, in

Equation 2.11, we partition the vector $\frac{1}{E_0}\vec{E}$ into the scalar 1 and a new vector $\vec{F}$ with one fewer entry than $\vec{E}$.

$$\left[\begin{array}{c|c} 1 & \\ \hline \vec{F} & \end{array}\right] = \frac{1}{E_0}\left[\begin{array}{c|c} E_0 & \\ \hline E_{j\neq 0} & \end{array}\right] = \frac{1}{E_0}\vec{E}. \tag{2.11}$$

Note the use of partitioned matrix notation; a solid line splits a matrix into sub-matrices, and splits a vector into sub-vectors. In Equation 2.12, we also partition the weighted adjacency matrix W into the scalar w_{00}, the row vector $\vec{b}^T$, the column vector $\vec{a}$, and the matrix H.

$$\left[\begin{array}{c|c} w_{00} & \vec{b}^T \\ \hline \vec{a} & H \end{array}\right] = W. \tag{2.12}$$

Finally, we partition the vector $\frac{1}{E_0}\vec{S}$ in Equation 2.13 into the scalar S_0/E_0 and the vector $\vec{V}$.

$$\left[\begin{array}{c|c} S_0/E_0 & \\ \hline \vec{V} & \end{array}\right] = \frac{1}{E_0}\left[\begin{array}{c|c} S_0 & \\ \hline S_{j\neq 0} & \end{array}\right] = \frac{1}{E_0}\vec{S}. \tag{2.13}$$

The vertex-signal equation (Equation 1.3) has a unique solution, by assumption. This implies that Equations 2.14 through 2.17 also have a unique solution.

$$\frac{1}{E_0}\vec{E} = W\frac{1}{E_0}\vec{E} + \frac{1}{E_0}\vec{S} \tag{2.14}$$

$$\Rightarrow \left[\begin{array}{c|c} 1 & \\ \hline \vec{F} & \end{array}\right] = \left[\begin{array}{c|c} w_{00} & \vec{b}^T \\ \hline \vec{a} & H \end{array}\right]\left[\begin{array}{c|c} 1 & \\ \hline \vec{F} & \end{array}\right] + \left[\begin{array}{c|c} S_0/E_0 & \\ \hline \vec{V} & \end{array}\right] \tag{2.15}$$

$$\Rightarrow 1 = w_{00} + \vec{b}^T\vec{F} + \frac{S_0}{E_0}, \tag{2.16}$$

$$\vec{F} = \vec{a} + H\vec{F} + \vec{V}. \tag{2.17}$$

Rearranging Equations 2.16 and 2.17, they become Equations 2.18 and 2.19.

$$\frac{1}{E_0} = (S_0 - 1)\frac{1}{E_0} + \vec{b}^T\vec{F} + w_{00} - 1, \tag{2.18}$$

$$\vec{F} = \vec{V}\frac{1}{E_0} + H\vec{F} + \vec{a}. \tag{2.19}$$

We re-express these as partitioned matrices in Equation 2.20.

$$\left[\begin{array}{c|c} 1/E_0 & \\ \hline \vec{F} & \end{array}\right] = \left[\begin{array}{c|c} S_0 - 1 & \vec{b}^T \\ \hline \vec{V} & H \end{array}\right]\left[\begin{array}{c|c} 1/E_0 & \\ \hline \vec{F} & \end{array}\right] + \left[\begin{array}{c|c} w_{00} - 1 & \\ \hline \vec{a} & \end{array}\right]. \tag{2.20}$$

Notice that this takes the form of the vertex signal equation (Equation 1.3). By dividing everything by E_0, we have artificially fixed vertex 0, making it a part of the constant source, but we have made the source

vary with E_0. In this new matrix equation, the variable edge weight $w_{1,0}$ is a part of the vector $\vec{a}$. So, this is now the case laid out in Theorem 2.1. Theorem 2.1 allows us to express the solution to Equation 2.20 as

$$\begin{bmatrix} 1/E_0 \\ \hline \\ \vec{F} \\ \\ \end{bmatrix} = \begin{bmatrix} se^{i\phi} + t \\ \hline \\ \vec{u}e^{i\phi} + \vec{v} \\ \\ \end{bmatrix}, \tag{2.21}$$

where $s, t \in \mathbb{C}$ and $\vec{u}, \vec{v} \in \mathbb{C}^{n-1}$ are all constant. This allows us to reach the final form of our solution.

$$E_0 = \frac{1}{se^{i\phi} + t}, \tag{2.22}$$

$$E_{j\neq 0} = \frac{u_j e^{i\phi} + v_j}{se^{i\phi} + t}. \tag{2.23}$$

□

The result of Theorem 2.2 gives us an expression for the vertex signals of a general interferometer with a single variable edge phase. We still need to interpret this result to see if interferometer limitation holds for the general case. We shall start by trying to understand the geometry of E_0, then we will build off of that to examine the geometry of every other E_j. The conclusions from Theorem 2.1, that interferometer limitation holds, would still hold for Theorem 2.2 if Equations 2.9 and 2.10 also traced a circle in the complex plane with period 2π. This is because the signal would still only take its minimum value once every 2π radians, never creating more fringes than the archetypal interferometer. Thus, it is natural to attempt to show that Equations 2.9 and 2.10 trace out circles. This intuition is further encouraged by manually plotting the results for a few values of s, t, u_j, and v_j on a graphing calculator.

For E_0, the first step is to find the center of the circle. Since the center lies at the midpoint of any diameter, the first step is to find two points on opposite sides of the circle. The form of the denominator in Equation 2.9, reveals that $|E_0|$ is maximized and minimized at the values of ϕ where t and $se^{i\phi}$ align. If E_0 does, indeed, trace a circle, the center ought to lie at the midpoint between these points. A depiction of this setup is show in Figure 2.3. The extremal points lie at

$$p_1 = \frac{1}{t - |s|\frac{t}{|t|}}, \tag{2.24}$$

$$p_2 = \frac{1}{t + |s|\frac{t}{|t|}}. \tag{2.25}$$

The proposed center lies at

$$c = \frac{1}{2}\left(\frac{1}{t - |s|\frac{t}{|t|}} + \frac{1}{t + |s|\frac{t}{|t|}}\right). \tag{2.26}$$

We can manipulate this into a more convenient form,

$$c = \frac{t^*}{|t|^2 - |s|^2}. \tag{2.27}$$

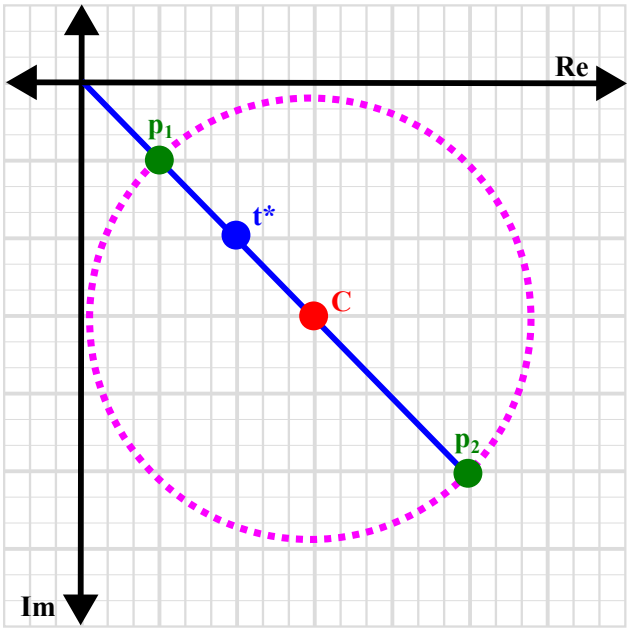

Figure 2.3 Depiction of the method used to find the center of the circle traced by E_0. The points p_1 and p_2 are the extremal values of E_0, drawn in green. The point t^* is colinear to these points and drawn in blue. The proposed center c of the circle lies at the midpoint between p_1 and p_2, and it is marked in red. The circle we hope to show E_0 traces over the different values of ϕ is drawn in magenta.

Now, we check to see that a circular form arises by subtracting this center point (Equation 2.27) from E_0.

$$E_0 - c = \frac{1}{se^{i\phi} + t} - \frac{t^*}{|t|^2 - |s|^2}. \tag{2.28}$$

After manipulating Equation 2.28 algebraically, it becomes

$$E_0 = \frac{s}{|s|^2 - |t|^2} \frac{\left|se^{i\phi} + t\right|^2}{\left(se^{i\phi} + 1\right)^2} e^{i\phi}. \tag{2.29}$$

The expression $\frac{s}{|s|^2 - |t|^2}$ is constant with respect to ϕ. Let us call it R.

$$R = \frac{s}{|s|^2 - |t|^2}. \tag{2.30}$$

The expression $\frac{\left|se^{i\phi} + t\right|^2}{\left(se^{i\phi} + t\right)^2} e^{i\phi}$ has magnitude 1. We can define a function $\theta(\phi)$ such that

$$e^{i\theta(\phi)} = \frac{\left|se^{i\phi} + t\right|^2}{\left(se^{i\phi} + t\right)^2} e^{i\phi}. \tag{2.31}$$

Thus, we have finally shown that E_0 can be expressed in the form of Equation 2.32.

$$E_0 = Re^{i\theta(\phi)} + c. \tag{2.32}$$

Equation 2.32 could break interferometer limitation if θ rotated faster than ϕ, such that more fringes were made over the course of the entire circle. However, we can rule out this possibility by recalling two things: First, each point on $E_0 = \frac{1}{se^{i\phi}+t}$ corresponds to one unique value of $se^{i\phi} + t$. Second, E_0 is continuous, meaning that θ cannot skip around to cover each point on E_0 by completing several rotations. Thus, though the instantaneous rate of change for θ may be above that of ϕ, they must complete the same number of total cycles. So, interferometer limitation holds for vertex 0 in particular.

We can show that interferometer limitation holds for all other vertices too. We use Equation 2.32 to express E_j as

$$\begin{aligned}
E_{j\neq 0} &= \frac{u_j e^{i\phi} + v_j}{se^{i\phi} + t} && (2.33)\\
&= \frac{\frac{u_j}{s}\left[se^{i\phi} + t - \left(t - \frac{sv_j}{u_j}\right)\right]}{se^{i\phi} + t} && (2.34)\\
&= \frac{u_j}{s} + \frac{t - \frac{sv_j}{u_j}}{se^{i\phi} + t} && (2.35)\\
&= \frac{u_j}{s} + \left(t - \frac{sv_j}{u_j}\right)\left(Re^{i\theta(\phi)} + c\right) && (2.36)\\
&= R\left(t - \frac{sv_j}{u_j}\right)e^{i\theta(\phi)} + \frac{u_j}{s} + c\left(t - \frac{sv_j}{u_j}\right). && (2.37)
\end{aligned}$$

Once again, Equation 2.37 is a circle that completes a full rotation once every time ϕ goes from 0 to 2π. Interferometer limitation holds whenever there is only one variable edge in a network. For an interferometer to be more sensitive to changes in ϕ than the archetypal interferometer, it cannot have only one variable edge weight phase.

The proofs for Theorems 2.1 and 2.2 give hints for how to break interferometer limitation. I proved that a single variable edge cannot do it, so I must split the variation across multiple edges. With one variable edge, I was able to trace circles in the complex plane such that certain parts of the circle traced arc length faster with respect to ϕ, but it was not enough to accumulate more total cycles, since other parts of the circle would compensate with slower accumulation of arc length. The idea for the N-stage skew cycle interferometer (NSCI) is that each of the N phases has a variable edge with a small phase shift, ϕ/N. Each skew-cycle stage takes advantage of tracing a circle with irregular arc length speed. This is done by only tracing the part of the circle that is fastest; since each stage only rotates through ϕ/N, there is no need to also use the part of the circle that accumulates arc length more slowly. Accumulating over N stages, this produces an output that oscillates quickly. The diagram of an NSCI is shown in Figure 2.4.

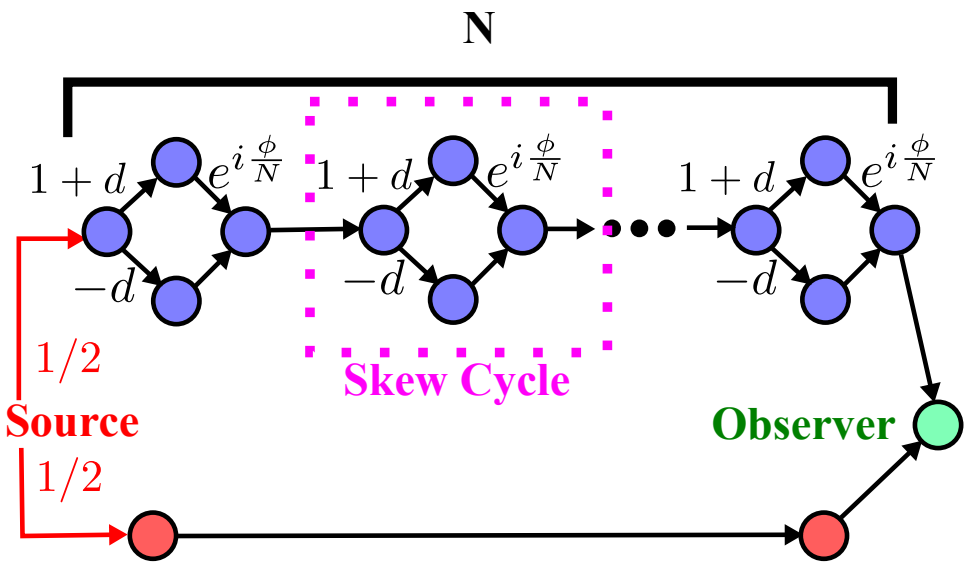

Figure 2.4 Diagram of the NSCI. A source (red) feeds two paths: one directly to the observer (green) and another through N stages (blue), each acting as a skew cycle. An individual skew cycle is highlighted by boxing it in magenta.

I will make an approximation now, and I will justify it later. Let us assume for now that, at each skew-cycle stage,

$$\frac{E_{\text{out}}}{E_{\text{in}}} \approx \exp\left(i\frac{(1+d)\phi}{N}\right). \tag{2.38}$$

Then, after N stages, the output should be

$$E_{\text{out}} \approx E_{\text{in}} e^{i(1+d)\phi}. \tag{2.39}$$

The result at the observer will be

$$E_{\text{obs}} = E_{\text{source}}\frac{1}{2}\left(1+e^{i(1+d)\phi}\right), \tag{2.40}$$

$$|E_{\text{obs}}| = |E_{\text{source}}|\left|\cos\left(\frac{(1+d)\phi}{2}\right)\right| \tag{2.41}$$

Now, I must justify the approximation in Equation 2.38 by examining an individual skew-cycle stage more closely. Figure 2.5 shows an individual skew cycle, with its output plotted on an argand diagram. The actual output is a circle of radius $(1+d)$, centered at $-d$. However, for small θ, the output is approximated by a circle of radius 1 centered at 0. The arc length along the original circle is $(1+d)\phi/N$. If we assume that $\frac{E_{out}}{E_{in}}$ is also moving along the circle of radius 1, it moves the same arc length there, but with radius 1. This implies that the angle relative to the origin is approximately

$$\theta \approx (1+d)\phi/N. \tag{2.42}$$

Hence the approximation in Equation 2.38. Note that the approximation grows worse as ϕ/N grows. This error can be kept in check in three ways: keeping d small, keeping ϕ small, and adding more stages (increasing N).

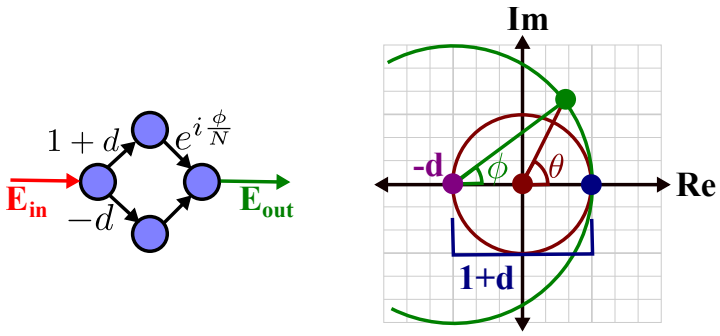

Figure 2.5 On the left, a sketch of the skew cycle stage. On the right, a plot of E_{out}/E_{in}. Notice that $\frac{d\theta}{d\phi} > 1$ when ϕ is small.

After designing the NSCI with the approximation from Equation 2.38 in mind, I found an alternative, more rigorous demonstration that the NSCI produces output phase $1 + d$ times faster near $\phi = 0$. The exact output for the NSCI is

$$E_{\text{out}} = \frac{1}{2} + \frac{1}{2}\left[(1+d)\left(e^{i\phi/N}\right) - d\right]^N. \tag{2.43}$$

Taking $\frac{d}{d\phi}$ of Equation 2.43, we have

$$\frac{dE_{\text{out}}}{d\phi} = \frac{1}{2}(1+d)i\left[(1+d)\left(e^{i\phi/N}\right) - d\right]^{N-1} e^{i\phi/N}. \tag{2.44}$$

Evaluating at $\phi = 0$, we have

$$\left.\frac{dE_{\text{out}}}{d\phi}\right|_{\phi=0} = \frac{1}{2}(1+d)i. \tag{2.45}$$

Compare Equation 2.45 to the result of making the same calculation with the archetypal interferometer (starting with Equation 2.2).

$$\left.\frac{dE_{\text{out}}}{d\phi}\right|_{\phi=0} = \frac{1}{2}i. \tag{2.46}$$

The complicated form of the NSCI and its construction based on an approximations warrants a demonstration of the NSCI producing the desired output. The following code produces this demonstration.

```
"""
Simulates and plots the output of an NSCI, to ensure that it behaves as
```

```
predicted.
"""

#Library imports
import numpy as np
import matplotlib.pyplot as plt

#Parameters
N = 10000
d = 9
phis = np.linspace(0.0, 2.0 * np.pi, num = 500)
source = 1.0

#Define a function for calculating the output of the NSCI
def NSCIout(phi):
    #The signal that passes through the skew cycles
    signal = 0.5 * source
    #Each skew cycle scales the signal by this much
    skewCycle = (-d) + ((1 + d) * np.exp(1.0j * phi / N))
    #Apply that scaling N times
    signal = signal * (skewCycle**N)

    #Recombine the direct signal to the observer and the skew cycle signal
    observer = np.abs(signal + (0.5 * source))

    #Return the magnitude of the signal seen at the output
    return(observer)



#Calculate the value for all values of phi
testObservers = NSCIout(phis)

#Create the plot
```

```
fig, ax = plt.subplots()

#Plot the NSCI output
ax.plot(phis, testObservers, label = 'NSCI')

#Plot archetypal interferometer as a control
archetype = source * np.abs( np.cos(phis / 2.0) )
ax.plot(phis,  archetype, label = 'archetypal')

#Label the axes and create a title
ax.set(
       xlabel = r'$\phi$',
       ylabel = r'$E_{out} / E_{in}$'
       )

#Add a legend
ax.legend()

#Save the figure
fig.savefig('NSCIsanity.pdf')
```

Figure 2.6 is the plot produced by this code. Notice that the NSCI does, indeed, oscillate faster than the archetypal interferometer, but the curve deforms as ϕ grows, since the approximation $(1+d)e^{i\phi} - d \approx e^{i(1+d)\phi}$ breaks down.

Interferometers are tools of precision measurement; it is natural to ask how precise an NSCI would be if constructed in the real world. I use the standard tools of error propagation [33] to approximate how a few sources of error would impact the precision of measurements by an NSCI, and use that to speculate about its usefulness as an actual device. I find that some hypothetical sources of error scale with $\sqrt{N}$, making the NSCI seem difficult to implement practically. I will examine the effect of introducing errors to edge-weight magnitude, total phase ϕ, and individual skew-cycle phase ϕ/N.

First, let us analyze the effect of introducing error to edge-weight magnitudes. If we are sure to include terms for all edges in the NSCI, the signal at the output is equal to

$$E_{\text{out}} = \frac{1}{2}(1)(1) + \frac{1}{2}\left[(1+d)\left(e^{i\phi/N}\right) - d\right]^N. \tag{2.47}$$

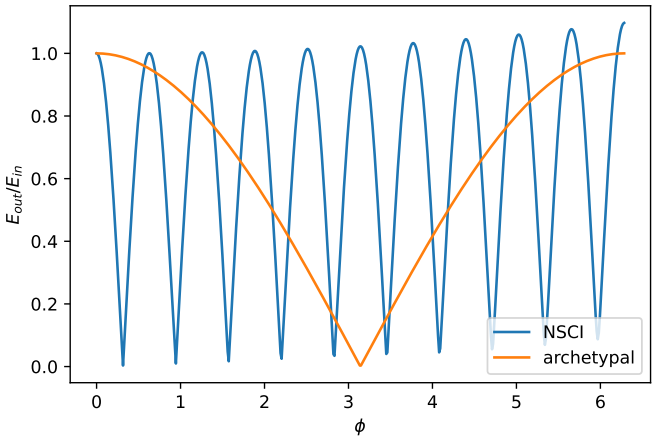

Figure 2.6 Demonstration of the NSCI, with parameters $d = 9$ and $N = 10,000$.

Let us introduce an error $\pm\delta w$. The error for the reference signal is

$$\delta\left[\frac{1}{2}(1)(1)\right] = \left[\frac{1}{2}(1)(1)\right]\sqrt{\left(\frac{\delta w}{1/2}\right)^2 + \left(\frac{\delta w}{1}\right)^2 + \left(\frac{\delta w}{1}\right)^2} \tag{2.48}$$

$$= \delta w\sqrt{\frac{3}{2}}. \tag{2.49}$$

The error for each skew cycle is

$$\delta\left[(1+d)\left(e^{i\phi/N}\right) - d\right] = \sqrt{\left[\delta\left((1+d)\left(e^{i\phi/N}\right)\right)\right]^2 + \left(\delta(d)\right)^2} \tag{2.50}$$

$$= \sqrt{\left((1+d)\sqrt{\left(\frac{\delta w}{1+d}\right)^2 + \left(\frac{\delta w}{1}\right)^2}\right)^2 + \delta w^2} \tag{2.51}$$

$$= \delta w\sqrt{2 + (1+d)^2}. \tag{2.52}$$

The error from all skew-cycle stages is

$$\delta\left(\frac{1}{2}\left[(1+d)\left(e^{i\phi/N}\right) - d\right]^N\right) =$$

$$\left|\frac{1}{2}\left[(1+d)\left(e^{i\phi/N}\right) - d\right)^N\right|\sqrt{\left(\frac{\delta w}{1/2}\right)^2 + N\left[\frac{\delta\left((1+d)\left(e^{i\phi/N}\right) - d\right)}{\left|(1+d)\left(e^{i\phi/N}\right) - d\right|}\right]^2}. \tag{2.53}$$

Expressing this as a relative error, we can manipulate the expression into a simpler radical form.

$$\frac{\delta\left(\frac{1}{2}\left[(1+d)\left(e^{i\phi/N}\right)-d\right]^N\right)}{\left|\frac{1}{2}\left[(1+d)\left(e^{i\phi/N}\right)-d\right]^N\right|} = \sqrt{4\delta w^2 + N\delta w^2\left[2+(1+d)^2\right]} \tag{2.54}$$

$$= \delta w\sqrt{4+2N+(1+d)^2N}. \tag{2.55}$$

For small ϕ, the intended regime of this interferometer, $\left|\frac{1}{2}\left[(1+d)\left(e^{i\phi/N}\right)-d\right]^N\right| \approx 1/2$. This implies

$$\delta\left[\frac{1}{2}\left((1+d)\left(e^{i\phi/N}\right)-d\right)^N\right] \approx \delta w\sqrt{1+\frac{N}{2}+\frac{(1+d)^2}{2}N}. \tag{2.56}$$

Finally, the estimated error for the output of the interferometer is

$$\delta E_{out} = \sqrt{\delta w^2\frac{3}{2}+\delta w^2\left[1+\frac{N}{2}+\frac{(1+d)^2}{2}N\right]} \tag{2.57}$$

$$= \frac{\delta w}{\sqrt{2}}\sqrt{5+N+(1+d)^2N}. \tag{2.58}$$

The error from edge weight magnitude grows like $\sqrt{N}$, and it grows linearly with respect to δw and $(1+d)$. This linear growth is to be expected; after all, the NSCI amplifies phase by $(1+d)$. The root growth with N introduces a tradeoff between managing this edge weight error and ensuring that the small angle approximation holds for the necessary range of ϕ. This tradeoff would make the NSCI difficult to implement in practice.

If we introduce an error to the total phase, $\delta\phi$, the error takes the form

$$\delta E_{out} = \left|\frac{\partial E_{out}}{\partial\phi}\right|\delta\phi. \tag{2.59}$$

Our next step is to calculate the partial derivative.

$$\frac{\partial E_{out}}{\partial\phi} = \frac{\partial}{\partial\phi}\frac{1}{2}\left\{1+\left[(1+d)\left(e^{i\phi/N}\right)-d\right]^N\right\} \tag{2.60}$$

$$= \frac{1}{2}i(1+d)\left[(1+d)\left(e^{i\phi/N}\right)-d\right]^{N-1}e^{i\phi/N}. \tag{2.61}$$

If we assume $(1+d)\left(e^{i\phi/N}\right)-d \approx 1$, which will hold for small ϕ/N (the intended regime of the NSCI), and take the magnitude, we can make the substitution

$$\left|\frac{\partial E_{out}}{\partial\phi}\right| \approx \frac{1}{2}(1+d). \tag{2.62}$$

Therefore,

$$\delta E_{out} \approx \frac{1+d}{2}\delta\phi. \tag{2.63}$$

This source of error grows linearly with $(1+d)$. This is unsurprising, since the NSCI essentially serves to amplify ϕ by $(1+d)$.

Now let us consider an error between stages of the interferometer, such that each skew cycle has an error $\delta\alpha$ added to its phase ϕ/N. Each stage has the term $\left[(1+d)\left(e^{i\phi/N}\right)-d\right]$. The error here will be

$$\delta\left((1+d)\left(e^{i\phi/N}\right)-d\right) = \frac{\partial\left|\left[(1+d)\left(e^{i\phi/N}\right)-d\right]\right|}{\partial(\phi/N)}\delta\alpha \tag{2.64}$$

$$= \left|i(1+d)e^{i\phi/N}\right|\delta\alpha \tag{2.65}$$

$$= (1+d)\delta\alpha. \tag{2.66}$$

After all of the stages are taken together, the output will have the relative error

$$\frac{\delta E_{out}}{|E_{out}|} = \sqrt{N\left[\frac{(1+d)\delta\alpha}{\left|(1+d)\left(e^{i\phi/N}\right)-d\right|}\right]^2} \tag{2.67}$$

$$\approx (1+d)(\delta\alpha)\sqrt{N}. \tag{2.68}$$

This source of error scales linearly with $(1+d)$ (like the others analyzed in this section) and like $\sqrt{N}$. Since we want N to be very large for this interferometer, this introduces a tension between keeping error in check and ensuring that the small angle approximation holds.

If building an NSCI involves sources of error like those in the three hypothetical situations I have explored here, controlling error would be a significant engineering challenge. Especially if real-world designs incurred error scaling with $\sqrt{N}$, a physical NSCI would be impractical. This does not mean that the NSCI is a pointless concept. It serves as a counterexample to general interferometer limitation. The NSCI shows that arbitrarily small changes to the total phase in a network can create arbitrarily large changes to the phase of vertex signals. If interferometer limitation had held for the general case, one could say that the change in signal phase is bounded above by the total edge weight phase change in a network. This is not the case. Phase can change quickly in large networks, obfuscating attempts to predict and bound those changes.

CHAPTER 3

GENERALIZED NETWORK MEASURES

This chapter introduces the interferometric adaptations of two standard complex network measures: path length and clustering. The path length measures the distance between two vertices by counting the number of edges required to travel from one vertex to another, while the clustering coefficient quantifies the tendency of a network to form triangles. The meanings of these concepts change for interferometer networks, so building up to the interferometric measures starts with generalizing these concepts in ways amenable to interferometers. For both of these measures, I discuss the meaning of the measure, I give the interferometric definition, and I perform an example calculation to show the measures at work and explain the decisions made defining them.

In the section on path strength, I argue that a path in an interferometer is best understood as the *path strength*: the product of edge weights, instead of a count of edges along the path (as is the case with path length). The most important individual path between a pair of vertices is the path with the strongest path strength. To capture interference, we introduce the apparent path strength (APS), which sums the signal transfer over all possible paths between two vertices. From the APS, we recover the apparent path length (APL), which allows us to create a path length measure that accounts for interference.

In the section on interferometric clustering, I develop a definition of clustering that focuses on triangles made of two paths: one with two edges, the other with a single edge. These two paths interfere, and my new definition of clustering quantifies this level of interference. I explain the process of starting with an existing definition of weighted clustering in the literature [34], tuning that measure to focus on the particular type of triangles that interfere, and adapting that measure to quantify interference.

3.1 Path Strength

As interferometer networks are meant to characterize the transfer of signals between vertices, a natural starting point for measuring an interferometer network is the measuring of paths in the network. The concept of path length on a network already exists to quantify travel along a network. The most basic definition of path length involves starting at one vertex and traveling edge-by-edge to other vertices, counting the steps along the way. When generalizing path length to real-number weighted networks, past works have proposed a sum of the form in Equation 3.1 [35], where α is a parameter that describes how much edge weight contributes to signal transfer or detracts from it.

$$l = \sum_{\text{edges in path}} (W_{ij})^{\alpha} \,. \tag{3.1}$$

Because of the sum in Equation 3.1, I call networks with this kind of path length *additive networks.* This is not the only way to think about travel.

Paths in interferometers are best characterized by the path strength, which is the product in Equation 3.2, since the edge weight in an interferometer network amplifies, attenuates, and phase-shifts the signal it carries.

$$p = \prod_{\text{edges in path}} |W_{ij}|. \tag{3.2}$$

We call networks of this kind *multiplicative networks* because of the product in Equation 3.2. An additive path length measure can be recovered by taking a logarithm of base w, where w is some characteristic edge weight (e.g., a maximum or mean edge weight magnitude), as shown in Equation 3.3.

$$l_p = \log_w(p). \tag{3.3}$$

For additive networks, a typical metric is the shortest path length between a pair of vertices. The diameter of a network is the maximum shortest path length between any pair of vertices [8]. The analogous measure for shortest path lengths in multiplicative networks is the strongest path strength. Note that the strongest path strength corresponds to computing the shortest path lengths if edges are weighted by $\log W_{ij}$ and Equation 3.1 is employed with $\alpha = 1$. I take advantage of this correspondence when computing averages. This is because the average strongest path strength and the average shortest path length do not correspond to one another, even though the individual path strengths and path lengths correspond. This effect occurs because an average of path strengths is biased towards strong paths, which correspond to short paths. However, an average of path lengths is biased towards long paths, which correspond to weak paths. For the sake of corresponding properly to previous work with additive networks, I chose to compute averages with path lengths, computed by taking logarithms of path strengths.

Strongest path strength works well enough for real-number weighted interferometers. However, this does not involve interference or complex numbers, which are the core features of interferometer networks. A measure focused on a single path cannot capture this; we need a path measure that captures multiple paths. The total signal between sent from vertex j to vertex i is the sum of the signals sent over all possible paths between them. In practice, for all but the simplest networks, this is impossible to compute directly. However, Equation 1.3 can be algebraically manipulated into Equation 3.4.

$$\vec{E} = (I - W)^{-1} \vec{S}. \tag{3.4}$$

The entries $\left[(I - W)^{-1}\right]_{ij}$ lend themselves to the straightforward interpretation of the apparent path strength (APS) between vertices i and j, P_{ij}.

$$P_{ij} = \left[(I - W)^{-1}\right]_{ij}. \tag{3.5}$$

An apparent path length (APL) can be computed by taking a logarithm.

$$(l_P)_{ij} = \log_w(|P_{ij}|) \tag{3.6}$$

Example 3.3 (Path measure calculations for a simple interferometer network)**.** I will show the calculations for the shortest path length, the strongest path strength, the APS, and the APL for the interferometer network depicted in Figure 3.1. This will elucidate the choices I made in defining our path measures.

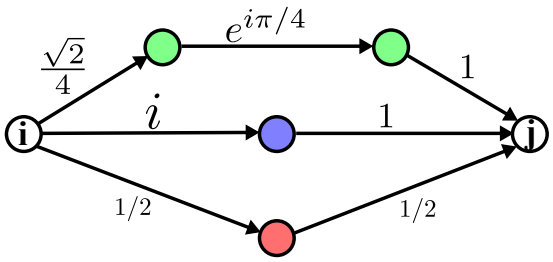

Figure 3.1 Example network for demonstrating the calculation of path measures.

I will start with the shortest path length. As a naive sum of edge weights, the path in gray is the shortest, with a length of 1. This demonstrates the inadequacy of the shortest path length measure for interferometers, since I know the least signal transfer will occur over this path; it is the least important path. If I had decided to depict an edge between i and j with weight 0, that would be the shortest path, but no signal transfers over it at all. I could try to rescue the additive path length measure by using $\alpha < 0$ in Equation 3.1, but this still would not quantify the amount of signal transfer as faithfully as a multiplicative measure.

The path strengths in this network are $\sqrt{2}/4$ for the path in green, 1 for the path in blue, and 0.25 for the path in gray. Thus, the strongest path strength from i to j is 1. This measure correctly identifies the most important path in the network. However, I have not yet accounted for the full effects of interference.

For the APS, I can start by expressing the network as a weighted adjacency matrix. Let us chose vertex 0 to be vertex i, 1 and 2 for the green vertexs, 3 for the blue vertex, 4 for the gray vertex, and 5 for vertex j. The adjacency matrix is

$$W = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \sqrt{2}/4 & 0 & 0 & 0 & 0 & 0 \\ 0 & e^{i\pi/4} & 0 & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0.5 & 0 \end{bmatrix}. \tag{3.7}$$

According to Equation 3.5, I have

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ \sqrt{2}/4 & 0 & 0 & 0 & 0 & 0 \\ \sqrt{2}/4e^{i\pi/4} & e^{i\pi/4} & 0 & 0 & 0 & 0 \\ i & 0 & 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0.5 + 1.25i & e^{i\pi/4} & 1 & 1 & 0.5 & 0 \end{bmatrix}. \tag{3.8}$$

The APS from i to j is $P_{0,5} = 0.5 + 1.25i$. The magnitude of the APS is $0.25\sqrt{29} \approx 1.35$. This is higher than the strongest path strength, but not quite as high as a mere sum of all path strengths, because the interference between the paths is not entirely constructive.

Now I can calculate the APL. We choose a characteristic edge weight of 0.5, though this is somewhat arbitrary. It only matters that we choose some characteristic edge weight so we can take a logarithm, allowing me to get an additive measure, and that we use the same characteristic edge weight for any other analysis we do on this network.

$$(l_P)_{ji} = \log_{0.5}\left(0.25\sqrt{29}\right) \approx -0.43. \tag{3.9}$$

A negative path length measure seems strange at first, but it is justified when considering that I chose a characteristic edge weight less than one. If I were to construct a path with a sequence of edges weighted 0.5, the signal would attenuate as path length increases. However, I have a net amplification of 1.35. If an attenuation corresponds to a positive path length, amplification is a negative path length.

3.2 Interferometric Clustering

Next, I build up to the interferometric clustering coefficient. Using the traditional clustering coefficient as a starting point, I develop an easily-computed local measure, which, in addition to the apparent path strength, I will use to characterize small-world interferometers. The clustering coefficient exists to measure the propensity of a network to form triangles. This property is significant in social networks, as it measures the probability that a friend of a friend is a friend, and it has applications in physical models, because such triangles can produce feedback or allow a vertex multiple paths to influence another vertex. Traditional, unweighted, undirected local clustering is defined as

$$C_i = \frac{\text{number of triangles connected to vertex } i}{\text{number of triples centered on vertex } i}, \tag{3.10}$$

where a "triple centered on vertex i" is a pair of vertices connected to i, which may or may not be connected to each other themselves [8]. This measure essentially means the number of actual triangles divided by the number of possible triangles.

Interferometer networks are directed, but the clustering coefficient is traditionally defined for undirected networks [8]. For directed networks, several types of triangles can form, and those triangles

serve different functions. The first choice I had to make for interferometric clustering was which triangles the measure would apply to. Fagiolo [36] divides these triangles into four classes: cycle, middleman, in, and out. A clustering measure can be defined with any of these triangle types (or combinations thereof), but middleman triangles lend themselves particularly well to an interferometric interpretation. As depicted in Figure 3.2, a middleman triangle forms two paths between a pair of vertices j and k: one direct, which we call the *shortcut*, and one indirect, passing through vertex i, which we call the *through-path*. The interferometric clustering at vertex i will compare these two paths. The interferometric clustering will be an average quantifying how much the shortcuts constructively interfere with the through-paths.

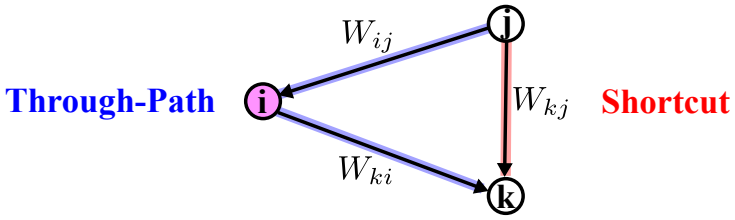

Figure 3.2 Schematic for the computation of the interferometric clustering at vertex i. The path through i is called the *through-path,* and is highlighted in blue. The path from j to k without i is called the *shortcut,* and is highlighted in red.

For weighted networks, there is a plethora of generalizations for the clustering coefficient [37], each with their own advantages and disadvantages. I have chosen to generalize the interferometric clustering coefficient from the weighted clustering coefficient presented in Zhang & Horvath [34], which takes the form

$$C_i = \frac{\sum_{j,k,j\neq k} w_{ki} w_{ij} w_{kj}}{\sum_{j,k,j\neq k} w_{ki} w_{ij}}. \tag{3.11}$$

This version lends itself to interpretation as a weighted average of the shortcut edge's edge weight, where the average is weighted by the path strength of the through-path. This is justified in this context, since interference is most important for signal transfer when it takes place between the strongest paths.

The definition of the *interferometric clustering* is given in Equation 3.12.

$$C_i = \frac{\sum_{j,k} \|W_{ki} W_{ij}\| \left(\|W_{kj} + W_{ki} W_{ij}\| - \|W_{ki} W_{ij}\|\right)}{\sum_{j,k,j\neq k} \|W_{ki} W_{ij}\|} \tag{3.12}$$

Norms are taken to deal with the phase in the complex numbers. The simple W_{kj} in Equation 3.11 is replaced by $(\|W_{kj} + W_{ki} W_{ij}\| - \|W_{ki} W_{ij}\|)$, which serves to measure how much the magnitude of the total signal from j to k increases when the shortcut is included. The function of this term is conceptually similar to the reverse triangle inequality. It is worth noting that this version of clustering can take a negative

value. This happens when the shortcut interferes destructively with the through-path, meaning that the signal from j to k is actually less than if there had been no shortcut at all.

I was faced with an interesting decision while defining interferometric clustering: I had to decide if it ought to include self-loops. This would happen by allowing the sum in Equation 3.12 to include cases where $j = k$. Traditionally, motifs like those depicted in Figure 3.3 are not counted, particularly because self edges are not a normal feature of networks. However, one can imagine an interferometric context where a beam is reflected back on itself, such as in Figure 3.4. These situations arise in other contexts too. For instance, one may want to apply network measures to density matrices in quantum mechanical problems. Density matrices usually have diagonal terms. Diagonal terms (of the form W_{ii}) correspond to self loops. I decided to make Equation 3.12 exclude $j = k$, conforming to tradition. Note that the results of this thesis do not depend on this decision. Appendix A contains the plots in Chapter 5, but run with self-loops allowed in the interferometric clustering coefficient. The differences between the plots in Appendix A and Chapter 5 are nearly imperceptible.

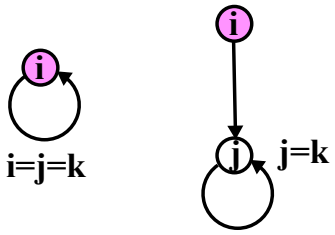

Figure 3.3 Examples of network motifs that are considered "triangles" by Equation 3.12, but are not traditionally counted in the clustering coefficient.

A simpler alternative to the $(\|W_{kj} + W_{ki}W_{ij}\| - \|W_{ki}W_{ij}\|)$ term was previously considered. Another potential definition of the interferometric clustering coefficient was

$$C_i = \frac{\sum_{j,k} W_{ki}W_{ij}W_{kj}^{\dagger}}{\sum_{j,k} \|W_{ki}W_{ij}\|}, \tag{3.13}$$

where $W_{kj}^{\dagger}$ is the notation for the complex conjugate of W_{kj}. This version of the clustering was complex-valued; the magnitude matched the weighted clustering coefficient of Equation 3.11, while the phase of C_i captured the interference. The way this would have worked is that the phase of $W_{ki}W_{ij}W_{kj}^{\dagger}$ equals the phase of the through-path minus the phase of the shortcut. While this definition resembled the original weighted clustering coefficient more closely, it did not capture the interference information in a way that would be easy to involve in the future calculation of the small-world coefficient. Instead, I opted for

Equation 3.12, which captures the extent of interference in the magnitude of the measure.

I conclude this section with an example of clustering calculations. After performing the calculations, I will use the results as a starting point for discussing our choices in defining the interferometric clustering.

Example 3.4. For the network depicted in Figure 3.5, I will demonstrate clustering calculations. First, I will take the magnitudes of all edges and use them in Equation 3.11 to get a weighted clustering. Then, I will try the rejected definition for a complex-valued interferometric clustering in Equation 3.13. Finally, I will compute the interferometric clustering (per Equation 3.12).

For the weighted clustering,

$$C_i = \frac{\sum_{j,k,j\neq k} w_{ki}w_{ij}w_{kj}}{\sum_{j,k,j\neq k} w_{ki}w_{ij}} \tag{3.14}$$

$$= \frac{1+0+1+0}{1+1+1+1} \tag{3.15}$$

$$= 0.5. \tag{3.16}$$

For the rejected definition of interferometric clustering in Equation 3.13,

$$C_i = \frac{\sum_{j,k} W_{ki}W_{ij}W_{kj}^{\dagger}}{\sum_{j,k} \|W_{ki}W_{ij}\|} \tag{3.17}$$

$$= \frac{i(-i)^{\dagger}+0+i(-i)^{\dagger}+0}{|i|+|i|+|i|+|i|} \tag{3.18}$$

$$= 0.5e^{i\pi}. \tag{3.19}$$

As expected, the result has the same magnitude as the weighted clustering and reports that the shortcuts are out of phase with the through-paths by π. For the interferometric clustering,

$$C_i = \frac{\sum_{j,k} \|W_{ki}W_{ij}\| \left(\|W_{kj}+W_{ki}W_{ij}\| - \|W_{ki}W_{ij}\|\right)}{\sum_{j,k} \|W_{ki}W_{ij}\|} \tag{3.20}$$

$$= \frac{2|i|\left(|i-i|-|i|\right)}{|i|+|i|+|i|+|i|} \tag{3.21}$$

$$= -0.5. \tag{3.22}$$

It happens that both proposed definitions of interferometric clustering give the same result, but for different reasons. This time, the interferometric clustering is negative because the shortcut interferes with the through-paths destructively. The interferometric clustering is real-valued, but it will return a negative value when the signal transfer with the shortcuts is less than the signal transfer would have been if there were no shortcuts at all.

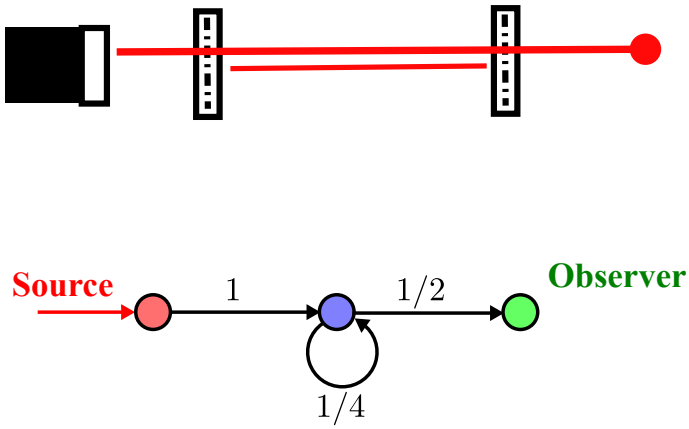

Figure 3.4 An example of a setup which could be expressed with a self loop. On the top, a diagram of the physical system is shown. From the left, a laser beam enters a cavity between two half-silvered mirrors. Half of the light from the cavity escapes to the observer; the other half is internally reflected. For the internally reflected light, half is reflected back into the cavity, and half escapes the system. On the bottom, a network diagram is drawn, where the inside of the cavity is treated as a single vertex. Of the light entering the cavity, half makes it to the observer, a quarter escapes the system (this is not depicted as an edge), and a quarter remains in the cavity. This creates a self loop.

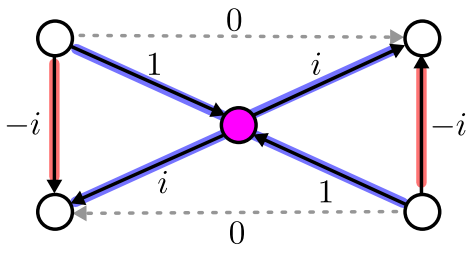

Figure 3.5 Example network for calculating interferometric clustering. The target vertex is highlighted in magenta. The through-paths are highlighted in blue, shortcuts are highlighted in red, and a couple nonexistant edges are indicated as dashed gray lines, to aid in the counting of triples.

CHAPTER 4

FEEDBACK IN INTERFEROMETERS

Interferometer networks have an apparent problem: feedback loops can create infinite signals from finite inputs, and this leads to apparent path lengths that go to negative infinity. Before I can move onward with developing models and drawing results, I need to be able to explain this behavior. I develop a simple example of this type of feedback, then I compare this example to the real-world example of the Fabry-Perot interferometer [38], demonstrating that this kind of feedback arises in edge cases of physical systems (and not from a problem with the definition of interferometer networks). Then I prove theorems about avoiding problematic feedback for future work. After setting these conditions for well-behaved interferometers, I can continue to explore interferometer networks with a guarantee that the matrix inverses in Equations 2.6 and 3.5 exist with bounded entries.

4.1 A Simple Example of Feedback, and its Correspondence to the Fabry-Perot Interferometer

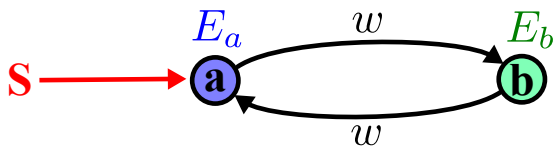

Figure 4.1 A diagram of a simple two vertex network that exhibits feedback. vertex a, with vertex signal E_a, has a fixed source input S. vertex b, with vertex signal E_b, is connected to vertex a with an incoming and an outgoing edge, both weighted with a complex number w. Equation 4.1 is the vertex signal equation for this network, and Equation 4.2 is the solution for the vertex signals.

To begin, let us examine a simple example of this runaway feedback. This phenomenon can arise in a simple 2-vertex network, as depicted in Figure 4.1. The vertex signal equation for this network takes the form

$$\begin{bmatrix} E_a \\ E_b \end{bmatrix} = \begin{bmatrix} 0 & w \\ w & 0 \end{bmatrix} \begin{bmatrix} E_a \\ E_b \end{bmatrix} + \begin{bmatrix} S \\ 0 \end{bmatrix}. \tag{4.1}$$

This has the solution

$$\begin{bmatrix} E_a \\ E_b \end{bmatrix} = \frac{S}{1-w^2} \begin{bmatrix} 1 \\ w \end{bmatrix}. \tag{4.2}$$

This solution diverges when $w = \pm 1$. At first glance, this seems like an absurd case for things to diverge. For runaway feedback, this case requires only requires each edge to carry 100% of its signal without dissipation. Perfect efficiency is certainly an idealization, but not one that typically causes models to give totally non-physical results.

Now that I have demonstrated that runaway feedback arises in otherwise innocuous-seeming situations, it is necessary to understand if this same situation arises in the physics of real-world interferometers. This simple two-vertex network corresponds with the Fabry-Perot interferometer [38], which is depicted in Figure 4.2. Going back and trying the vertex signal equation on it, it is shown that Equation 4.2 matches the results of the original paper.

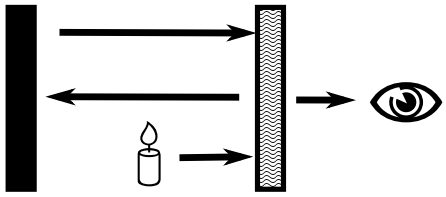

Figure 4.2 A simple Fabry-Perot interferometer. A light source (depicted as a candle) produces plane waves. The plane waves hit a partial mirror (a plate with a wavy pattern), then some of the light is transmitted to the observer (depicted as an eye), while the rest is reflected back. In the back, there is a perfect mirror (depicted as a black plate), which reflects back all light that hits it.

The beam starts with electric field strength E_0. It encounters the partial mirror, with reflectivity r, so rE_0 of the field reflects, while $(1-r)E_0$ passes through. The part that reflects travels to the rear mirror, then returns to the partial mirror, accumulating a phase of $e^{i2\phi}$ over that distance. Then, some reflects and some passes through the partial mirror. The process repeats with the reflected beam, splitting, reflecting, and phase shifting repeatedly. In total, the electric field at the output is

$$E = E_0(1-r)\sum_{n=0}^{\infty}(re^{i2\phi})^n. \tag{4.3}$$

As long as $0 < r < 1$, this infinite series converges to

$$E = E_0\frac{1-r}{1-re^{i2\phi}}. \tag{4.4}$$

Since intensity I is proportional to $|E|^2$,

$$I = I_0\frac{1}{1+4r(1-r)^{-2}\sin^2\phi}, \tag{4.5}$$

where I_0 is the intensity of the light at $\phi = 0$. Notice that $0 \leq I \leq I_0$, and that the infinite series for E diverges when $r = 1$.

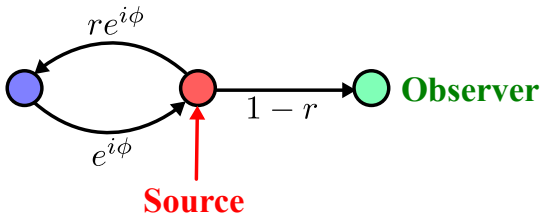

Figure 4.3 The Fabry-Perot Interferometer as an interferometer network. The reflectivity of the partial mirror r, the phase accumulated over the distance between the mirrors $e^{i\phi}$, and the transmitivity of the partial mirror $(1-r)$ create the edge weights.

As an interferometer network (depicted in Figure 4.3), the vertex signal equation is found by taking the sum of the signals entering each vertex.

$$E = (1-r)E_{\text{partial}}, \tag{4.6}$$
$$E_{\text{partial}} = e^{i\phi}E_{\text{perfect}} + E_0, \tag{4.7}$$
$$E_{\text{perfect}} = re^{i\phi}E_{\text{partial}}, \tag{4.8}$$

where E is the electric field strength at the observer, E_{partial} is the electric field strength at the partially-silvered mirror, E_{perfect} is the electric field strength at the perfectly-reflective mirror, and E_0 is the strength of the source. Solving this system of equations for E gives

$$E_{\text{partial}} = \frac{1}{1-re^{i2\phi}}E_0, \tag{4.9}$$
$$E_{\text{perfect}} = \frac{re^{i\phi}}{1-re^{i2\phi}}E_0, \tag{4.10}$$
$$E = \frac{1-r}{1-re^{i2\phi}}E_0, \tag{4.11}$$

Note that the expression for E matches Equation 4.4, that the expressions for E_{partial} and E_{perfect} only diverge in the same place as Equation 4.3, and that E is still always bounded by $0 \leq |E| \leq E_0$.

So, the only difference between the Fabry-Perot Interferometer and our simple test case is the presence of an observer, with which the feedback loop must share its output. Adding the observer with a nonzero edge limits the size of the output, and the output never exceeds the source signal.

The case of the Fabry-Perot Interferometer demonstrates that the vertex signal equation matches interferometry in the literature, even in this case with divergence. It also reveals a very satisfying hint at how to temper feedback: observers. Beyond simply requiring that edges in the network do not actively amplify signals–edge weights greater than one are an obvious source of runaway feedback–one must require that total output from a vertex must not exceed total input, and that the signal ought to reach a dead end

at an observer at some point. These ideas will be developed into theorems in the next section.

4.2 Conditions for Well-Behaved Interferometers

As seen earlier in the chapter, there are instances where runaway feedback produces unbounded signals, meaning unbounded apparent path strength (APS). APS is only defined when the inverse of $I - W$ exists, and even when APS exists, it may exhibit extremely large values unless the problem is well-conditioned. The goal of this chapter is to understand when the APS exists and how the APS can be bounded.

Before entering the theorems and proofs, it is beneficial to introduce the matrix and vector norms we use in this chapter, found in [39]. For vectors, length is measured with the ℓ_1 vector norm. For an arbitrary vector $\vec{x} \in \mathbb{C}^N$,

$$\|\vec{x}\|_1 = \sum_{i=1}^{N} |x_i|. \tag{4.12}$$

Notice that $\|\vec{x}\|_1 = 0$ if and only if $\vec{x} = \vec{0}$. Also note that this norm obeys the triangle inequality.

$$\|\vec{x} + \vec{y}\|_1 \leq \|\vec{x}\|_1 + \|\vec{y}\|_1. \tag{4.13}$$

Now, the norm of a matrix is defined based on how that matrix changes the norms of the vectors it acts on. In particular, the norm of a matrix A is defined as the most A can possible scale an arbitrary vector when acting on it.

$$\|A\|_1 = \sup_{\vec{x} \neq 0} \frac{\|A\vec{x}\|_1}{\|\vec{x}\|_1}. \tag{4.14}$$

This is the ℓ_1 matrix norm, induced by the ℓ_1 vector norm. It can be shown that, for the ℓ_1 matrix norm of $A \in \mathbb{C}^{m \times n}$,

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |A_{ij}|. \tag{4.15}$$

These norms give the tools to evaluate the scale of matrices and vectors, which I used to find conditions where $I - W$ is guaranteed to be invertible.

Theorem 4.3 (Existence of P)**.** *Let us have a network with N nodes, with each node having a maximum of k outbound connections, with all edge weights bounded above in magnitude by a number w, such that $kw < 1$. Let us call the adjacency matrix W, and the $N \times N$ identity matrix I. Then the matrix $I - W$ is invertible, and the apparent path strength matrix $P = (I - W)^{-1}$ exists.*

Proof. If it exists, P is the inverse of $(I - W)$. By the fundamental theorem of invertible matrices [32, 172], it will suffice to show that, for all $\vec{x} \neq \vec{0}$,

$$(I - W)\vec{x} \neq \vec{0}. \tag{4.16}$$

This will be true if

$$\|(I - W)\vec{x}\|_1 > 0, \tag{4.17}$$

where $\|\cdot\|_1$ is the ℓ_1 vector norm [39]. By the triangle inequality,

$$\|(I - W)\vec{x}\|_1 + \|W\vec{x}\|_1 \geq \|\vec{x}\|_1. \tag{4.18}$$

$$\Rightarrow \|(I - W)\vec{x}\|_1 \geq \|\vec{x}\|_1 - \|W\vec{x}\|_1. \tag{4.19}$$

We introduce the induced ℓ_1 matrix norm of W, $\|W\|_1$. This induced norm is consistent with the ℓ_1 vector norm [39]; by the definition of consistency, it has the property

$$\|W\vec{x}\|_1 \leq \|W\|_1\|\vec{x}\|_1. \tag{4.20}$$

Introducing the matrix norm into our inequality, we see

$$\|(I - W)\vec{x}\|_1 \geq (1 - \|W\|_1)\,\|\vec{x}\|_1. \tag{4.21}$$

Therefore, $(I - W)$ is invertible and P exists whenever $1 - \|W\|_1 > 0$.

Now, we must calculate the norms and show that P exists for our particular problem. The norm $\|W\|_1$ is calculated as

$$\|W\|_1 = \max_{1\leq j\leq N} \sum_{i=1}^{N} |W_{ij}|. \tag{4.22}$$

Given the structure of the network, we know that each column has, at most, k entries bounded above by w. This implies

$$\|W\|_1 \leq kw. \tag{4.23}$$

We have stipulated that $kw < 1$. This implies $\|W\|_1 < 1$. Therefore,

$$1 - \|W\|_1 > 0. \tag{4.24}$$

So $(I - W)$ is invertible and P exists. □

Corollary 4.4 (Bounding the entries of P)**.** *Furthermore, the entries of the P matrix in Theorem 4.3 are bounded. In particular, let us call $P_{\max} = \max_{i,j} |P_{ij}|$.*

$$P_{\max} \leq \frac{1}{1 - kw}.$$

Proof. By examining Equation 4.22, we can see that the ℓ_1 norm of P is an upper bound for $P_{\max}$. The ℓ_1 norm is defined [39] as

$$\|P\|_1 = \sup_{\vec{y}\neq 0} \frac{\|P\vec{y}\|_1}{\|\vec{y}\|_1}. \tag{4.25}$$

Let $\vec{x} = P\vec{y}$. This means $\|P\|_1$ can be equivalently expressed as

$$\|P\|_1 = \sup_{\vec{x}\neq 0} \frac{\|\vec{x}\|_1}{\|(I-W)\vec{x}\|_1}. \tag{4.26}$$

Now, as before, we use the triangle inequality and the consistency of the ℓ_1 matrix norm [39] to show

$$\|(I-W)\vec{x}\|_1 \geq (1-\|W\|_1)\,\|\vec{x}\|_1. \tag{4.27}$$

Therefore,

$$P_{\max} \leq \|P\|_1 \leq \frac{1}{1-\|W\|_1}. \tag{4.28}$$

By the same reasoning as Theorem 4.3, we know

$$\|W\|_1 \leq kw. \tag{4.29}$$

This implies

$$-kw \leq -\|W\|_1 \tag{4.30}$$

$$\Rightarrow 1-kw \leq 1-\|W\|_1 \tag{4.31}$$

$$\Rightarrow \frac{1}{1-\|W\|_1} \leq \frac{1}{1-kw}. \tag{4.32}$$

Therefore,

$$P_{\max} \leq \frac{1}{1-kw}. \tag{4.33}$$

□

These proofs inform the definitions of the adapted small-world model in the next chapter. By limiting the output of each node, I guarantee that these networks have well-defined and bounded APS. Furthermore, it suggests a motivation for generalizing the notion of network degree for interferometer networks. For unweighted networks, out-degree counts the number of edges coming out of a vertex. For real-weighted networks, out-strength quantifies the total output from a vertex. Similarly, the new generalized out-strength will quantify total node output. In particular, if we define the out-strength of a vertex to be

$$s_{\text{out},i} = \|\text{column } i \text{ of } W\|_1, \tag{4.34}$$

then the APS will exist and be bounded by

$$|P_{ij}| \leq \frac{1}{1-s_{\text{out,max}}}, \tag{4.35}$$

where $s_{\text{out,max}}$ is the maximum out-strength of any node, if $s_{\text{out},i} < 1$ for all nodes.

CHAPTER 5

THE SMALL-WORLD EFFECT FOR INTERFEROMETER NETWORKS

In this chapter, we analyze the small-world effect in an interferometric context, employing our newly defined measures to capture the changing behavior that arises due to interference. Small-world networks have two properties simultaneously: they have a high clustering coefficient, i.e. neighbors in a network tend to attach to their neighbor's neighbors. Also, they have short path lengths, i.e. getting from one node to another takes a small number of steps. Real-world networks with the small-world property often have other properties too, but I am concerned with clustering and path length here, because these are the measures needed to replicate the results in [31]. Random networks have short path lengths but low clustering. A ring lattice, where each node is connected to k nearest neighbors, has high clustering but long path lengths. Watts and Strogatz introduced a model that bridges this gap [31]. Starting with a ring lattice, they randomly rewired a few of the edges. This compromise creates both high clustering and short path lengths. This model captures an important network phenomenon and concerns itself with clustering and path length. However, the effects of phase on small-worldness have not been explored. In this chapter, I create an interferometric version of the Watts-Strogatz small-world model. Then, I show results from computational studies which apply my adapted network measures to demonstrate the effect of interference on the small world phenomenon and the importance of considering phase for complex-valued weights. Finally, I discuss some of the details of creating this code.

To test the generalized network measures, I need an interferometer network model to test them on. We use a generalized version of the Watts-Strogatz small-world model [31] As was the case with the Watts-Strogatz small-world model, I begin with a ring, then reshuffle edges according to a probability β. However, my model constructs the ring by drawing edges out from each node, and weighting the edges based on a total out-strength parameter s (to satisfy the conditions in Chapter 4), an out degree k and a phase ϕ. The model is depicted in Figure 5.1. The model's parameters are N (the number of nodes in the network), s (total output strength of nodes, which must be set ≤ 1 to control feedback, per Corollary 4.4), k (the number of edges coming out of each node), ϕ (the phase of edges in the model), and β (the probability than an edge's destination is randomly reshuffled).

At $\beta = 0$, the model produces a ring. At $0 < \beta < 1$, the model interpolates between order and disorder. At $\beta = 1$, the model produces a random network, later used as the baseline for the complex small-world coefficient.

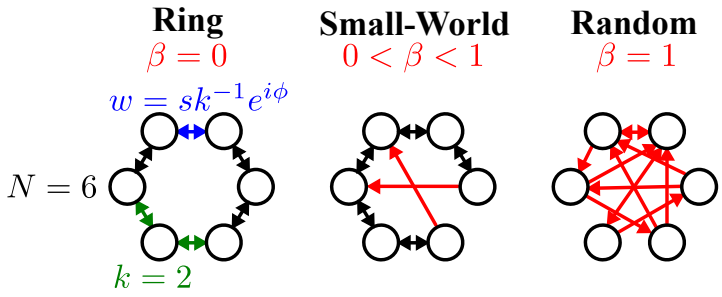

Figure 5.1 Depiction of the small-world interferometer model. At $\beta = 0$, a ring is formed with N nodes and degree k, with edges weighted with $w = sk^{-1}e^{i\phi}$. Then, a small-world network is formed by randomly rewiring edges according to a probability β. At $\beta = 1$, the small-world network becomes a random network.

A network is considered "small-world" if it has a high clustering coefficients and low vertex-to-vertex path lengths. Humphries & Gurney [40] defined the small-world coefficient to quantify this property, using a random network as a baseline. The small-world coefficient takes the form

$$S = \frac{\gamma}{\lambda}, \tag{5.1}$$

where

$$\gamma = \frac{C}{C_{\text{random}}}, \lambda = \frac{l}{l_{\text{random}}},$$

C is the mean clustering, l is the mean shortest path length between two nodes, C_{random} is the mean clustering of the random network baseline, and l_{random} is the mean shortest path length of the random baseline network. The idea is that if, compared to a random network, clustering is high but path lengths are short, the network is quantifiably "small-world."

To apply the small-world coefficient to interferometer networks, it must be able to handle both negative clustering coefficients (arising from destructive interference) and negative path lengths (arising from a net amplification in the apparent path length, explored in Example 3.3). Thus, the generalized small-world coefficient is

$$S_{\text{int}} = \frac{\gamma}{\lambda}, \tag{5.2}$$

$$\gamma = \frac{C + |C_{\text{random}}| - C_{\text{random}}}{|C_{\text{random}}|},$$

$$\lambda = \frac{l_A + |(l_A)_{\text{random}}| - (l_A)_{\text{random}}}{|(l_A)_{\text{random}}|}.$$

The adapted γ and λ definitions are constructed to have a few key properties. For γ, it reduces to the original definition of γ when all inputs are nonnegative numbers; the result is always nonnegative; if

$C = C_{\text{random}}$, then $\gamma = 1$; if $C > C_{\text{random}}$, then $\gamma > 1$; and if $C < C_{\text{random}}$, then $\gamma < 1$. Analogous properties hold for λ.

The interferometric small-world coefficient depends heavily on the phase of edge weights. To demonstrate this, I ran a suite of computational tests. First, as an illustrative case, I examined networks with $N = 500$, $k = 12$, $s = 0.95$. Note that $s = 0.95$ was chosen to satisfy the conditions from Chapter 4, while the choices of N and k were arbitrary, and other N and k configurations are tested later. For each selected configuration of β and ϕ, I ran at least 100 tests (more for sensitive values of β at $\phi = 0$), computed their strongest path strengths, apparent path strengths, real-weighted local clusterings, and interferometric clusterings, then averaged them for each set of model parameters. To illustrate the way phase impacts the small-world effect, for each tested ϕ value, I found the β value at which S_{int} was maximized. Figure 5.2 plots these measurements. Figure 5.3 plots S_{int} values over β for a few key values of ϕ, to convey what is happening behind the peak values of S_{int} in Figure 5.2.

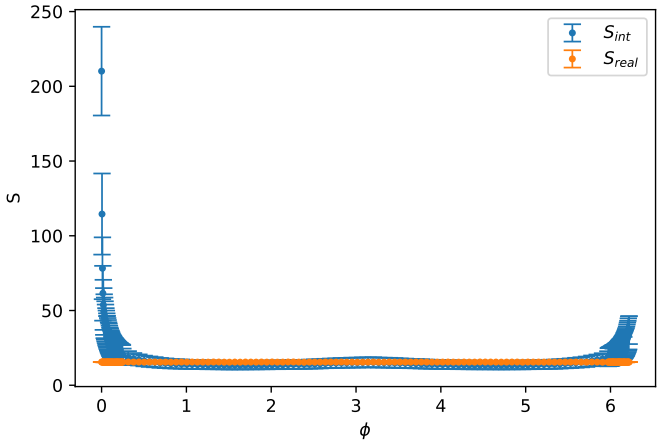

Figure 5.2 Peak S_{int} values for each configuration in ϕ. The traditional small world coefficient S (per Equation 5.2) is plotted for reference; it is constant because it does not depend on phase. This plot shows results for small-world interferometers with size $N = 500$ and $k = 12$.

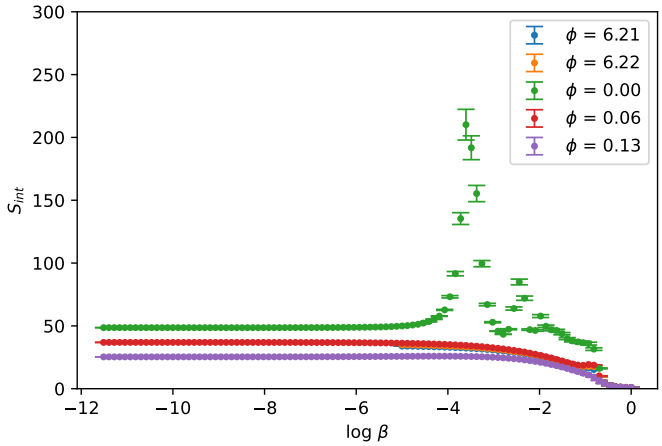

Figure 5.3 The interferometric small-world coefficient (Equation 5.2) plotted over β for a few select values of ϕ. This plot shows results for small-world interferometers with size $N = 500$ and $k = 12$.

While the example with $N = 500$, $k = 12$ serves to demonstrate that the small-world effect can change as ϕ varies, it is only a particular case. To show that this effect holds for many configurations, I ran a randomized suite of 100 tests with $100 \leq N \leq 1500$ and $8 \leq k \leq 20$. For each configuration of N and ϕ, we ran 50 trials and averaged their measures. In particular, I examined the interferometric small-world coefficient at $\phi = 0$ and $\phi = \pi$. Then, I computed the ratio of these two measurements. Figure 5.4 depicts a histogram of the base-10 logarithm of these ratios. Notice that, for all configurations, the logarithm is greater than zero, meaning that $S_{\text{int}}(\phi = 0) > S_{\text{int}}(\phi = \pi)$ for all trials. This means that some version of the effect in Figure 5.2 holds for all tested configurations.

All code used to generate the results of the Small-World Interferometer tests is posted on a public GitHub repository: `https://github.com/bkrawciw-mines/IntNets_Wendian`. This chapter is not intended to be an exhaustive discussion of all code, but I will give an explanation for the general structure and methodology of the programming for this project. I will discuss the algorithms for generating and measuring networks, the process of running the tests on HPC resources at the Colorado School of Mines, the process of generating visualizations of the results, and performing additional randomized trials.

The full code for the network algorithms is included in Appendix B. The module *nets.py* defines the function "ws", which produces the adjacency matrix of a small-world interferometer. This function has the child functions "makeRing", which forms the adjacency matrix of a ring network (no rewiring), and "rewire", which rewires a ring matrix according to a probability β. In addition to generating networks for the tests, *nets.py* also includes the implementations of interferometric clustering, apparent path length, weighted clustering [34], and strongest path length.

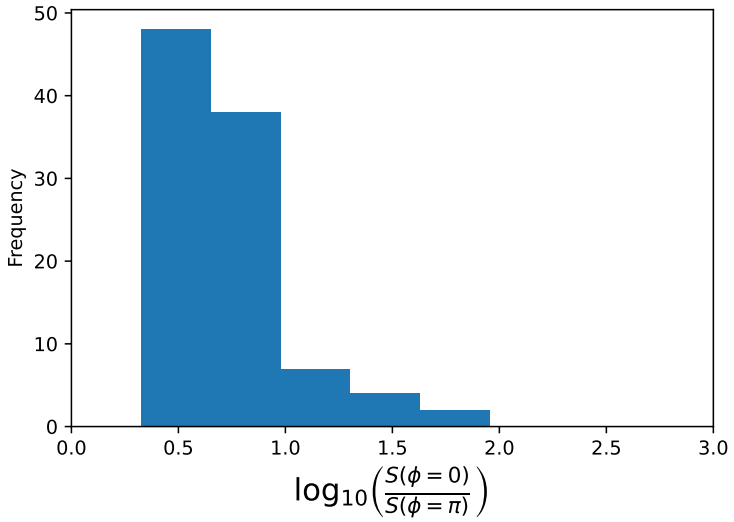

Figure 5.4 Histogram of the logarithms of the ratio $S_{int}(\phi = 0)/S_{int}(\phi = \pi)$. Notice thatbecause logarithms are always greater than zero, $S_{int}(\phi = 0) > S_{int}(\phi = \pi)$ for all tests.

The algorithms for computing the network measures warrant further discussion. In the clustering algorithm, I take advantage of efficient outer product calculations to compute the summation terms in Equation 3.12. For path length calculations, I switch between additive and multiplicative path lengths by taking logarithms or exponentials, depending on what is most efficient at the time. For instance, I convert to additive path lengths to take advantage of Dijkstra's algorithm in the function for strongest path lengths. It is also essential that paths are expressed additively (in log space) before averaging for the statistics. Otherwise, I found that the variation of the clustering and path length over β did not match the results from Watts and Strogatz [31]. This occurs because an average of additive path lengths is biased towards longer path lengths, which correspond to weaker multiplicative paths. On the other hand, an average of multiplicative paths is biased towards strong paths, with short path lengths. Thus, the two averages concern themselves with different sets of paths, and the results from [31] are concerned with the longer paths. The routines use sparse matrices when possible, which gives a considerable speed boost to the calculations of clustering (in the computations of outer products) and strongest path lengths (in Dijkstra's algorithm). However, sparse matrices are a hindrance in the computation of apparent path length. The matrix inverse of a sparse matrix is not necessarily sparse, and the sparse matrix format takes significantly more space for matrices that have many entries. Therefore, it was prudent to convert the adjacency matrix a dense matrix format in the apparent path length algorithm.

To collect data, I ran the code in Appendix C on the Wendian Cluster at the Colorado School of Mines. To run this program, I executed the following SLURM script, "runIntTests.sh":

```
#!/bin/bash
#SBATCH --partition lcarr
#SBATCH --ntasks=200
#SBATCH --cpus-per-task=1
#SBATCH --time=02:30:00 # time in HH:MM:SS
#SBATCH --mem=40GB
#SBATCH -o OutputIntTests # standard print output labeled with SLURM job id %j
#SBATCH -e ErrorIntTests  # standard print error  labeled with SLURM job id %j

# load python module
module load apps/python3/2020.02
module load compilers/gcc/9.3.1 mpi/openmpi/gcc-cuda/4.1.2

# activate conda environment required to run code
source activate IntNets

echo "Beginning interferometer tests"
cd bins
srun python -u -m mpi4py.futures IntTests.py
echo "Tests complete"
```

This script executes 300 parallel tasks, allowing the interferometer tests to execute within the externally-imposed time limit. HPC policy requires a time limit, so 2 hours, 30 minutes was selected by trial and error.

In "IntTests.py" itself, the following steps are taken: First, the parameter space, which enumerates all desired combinations of N, k, β, ϕ, and s (per the small-world interferometer model) is defined. I include redundancy when generating networks over these parameter combinations. During the rewiring stage of generating small-world interferometers, randomness is introduced, so it is necessary to perform multiple trials in order to capture the average behavior. This is especially important for small networks or networks with few nearest neighbor connections, where a small number of rewirings can drastically change the behavior of a single trial. Then, a function for running each individual test is defined. In the function, a small-world interferometer's adjacency matrix is generated, then the network measures are computed for that adjacency matrix. These tests are run in parallel using the MPIPoolExecutor. Finally, the results are

recorded in a comma-separated-values (csv) file.

To produce the reported results, the data from the interferometer test csv file was run through the code in Appendix D. This code cleans the data (removing any infinite values, for example), averages redundant tests, computes the generalized small-world coefficient, and creates plots.

To confirm that the effects seen in Figure 5.3 and Figure 5.2 are not particular to the specific parameters used in those trials, I re-ran the tests over a random suite of parameters. To generate the data in Figure 5.4, I ran a very similar program to Appendix C, but over the random N and k values depicted in Figure 5.5, a fine grid in β, and only $\phi = 0$ and $\phi = \pi$, which empirically seem to be near the maxima/minima of the small world coefficient. I chose to use a randomized suite of N and k values instead of a grid for two reasons: first, a grid would have required more total parameter sets to cover the same number of N and k values. Second, I was concerned that a grid would bias the results to networks where N and k were multiples of the same number.

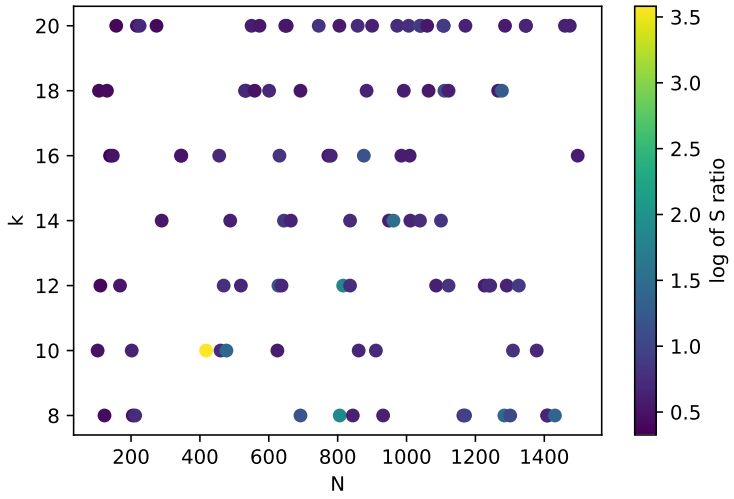

Figure 5.5 A scatter plot highlighting the randomized N and k values used to produce the data in Figure 5.4. The results of the trials (the ratio of small-world coefficients) is indicated by a color map.

The way I wrote the code for these results ensures that the code runs efficiently enough for large sample sizes, which in turn assures us that my results arise from the genuine behavior of interferometer networks and not statistical anomalies. In addition, the code was written to be easily understood and replicated by third parties, as much as possible with high-performance computing systems, which can differ from each other greatly. This unfortunately increases the difficulty of replication, but I hope that this difficulty is mostly relegated to the SLURM scripts, and that the python code itself is easily reused.

The notable results from these tests were that S_{int} changes with respect to ϕ, S_{int} is greater than the traditional S at $\phi = 0$, and S_{int} can dip below traditional S around $\phi = \pi$. These results indicate that destructive interference destroys long-range signal transfer near $\phi = \pi$, while constructive interference enhances long-range signal transfer near $\phi = 0$. The constructive interference produces much more signal transfer than the mere strongest path strength measure would have anticipated. In both cases, constructive or destructive interference, strongest path strength fails to predict the behavior of signals on the network, since it only accounts for ine path between each pair of nodes. In contrast, apparent path strength does capture the interference behavior, because its form includes signals from all possible paths between each pair of nodes, since it is defined with the matrix inverse form of Equation 3.5.

CHAPTER 6

THE APPLICATION OF GENERATING FUNCTIONS TO MATRICES

I have fused the concepts of adjacency matrices and generating functions to robustly study the effects of large networks. Large networks are the domain of complex network theory. Much fruitful work has been done on the statistics of large networks, including modeling the logarithmic growth of shortest path lengths for random networks [41], explaining the small-world effect with a model that mixes ring lattices with random rewiring [31], and modeling power law degree distributions in networks with preferential attachment [42]. This work was done with the traditional tools of network science, often combinatorics or adjacency matrices. Applying generating functions here will expand the tools available to network scientists.

Adjacency matrices will be my starting point. They store edges between vertices as entries in a matrix. The column corresponds to the vertex the edge is leaving, the row corresponds to the vertex the edge enters, and the value stores the edge's weight. Generating functions are a powerful tool for analytically treating sequences when direct computation is infeasible [23]. Combined, we have the possibility of storing and manipulating networks of immense size, sizes that would normally preclude exact numerical treatment by human or machine.

I have created two tools for this study: The *array generating function* (AGF) and the *transformation operator* (TO). The AGF represents the layout of terms in a matrix, treating the matrix as a two-dimensional list of numbers. The AGF is useful for constructing matrices and performing network measure calculations on a matrix. Constructing networks from recursive patterns will become as simple as addition, multiplication, and function composition. The generating function toolbox makes the evaluation of sums simpler, allowing the sums and averages of network measures to be computable where they otherwise might not be. The TO captures the behavior of the adjacency matrix during matrix multiplication. This can be useful for transforming AGFs as a step in network construction. For instance, a spread TO (defined later) can space out the indices of a network, allowing new nodes to be introduced. The TO can also be used for transforming state vectors on a network in lieu of an adjacency matrix. For instance, this technique could be used with interferometer networks to create a generating function version of the node-signal equation (1.3).

In this chapter, I begin with basic definitions and notations for working with matrices and generating functions. I continue by introducing the AGF and the TO. I conclude this chapter by giving methods for converting between adjacency matrices, AGFs, and TOs.

In this chapter, I lay out our basic matrix, vector, and generating function notation. Table 6.1 lays out the notation used for matrices and vectors. The basis matrix $\Delta^{k,l}$ is defined according to

$$[\Delta^{k,l}]_{i,j} = \begin{cases} 1, & i = k, j = l \\ 0, & \text{otherwise} \end{cases}. \tag{6.1}$$

This will be used later to decompose matrices into a sum of basis matrices, then to reconstruct them as transformation operators (which will be defined later).

Example 6.5 (The basis matrices for $\mathbb{R}^{2\times 2}$)**.** There are four distinct basis matrices for 2×2 matrices, corresponding to the four entries of a 2×2 matrix.

$$\begin{aligned} \Delta^{0,0} &= \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \Delta^{0,1} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \\ \Delta^{1,0} &= \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, \Delta^{1,1} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}. \end{aligned} \tag{6.2}$$

Table 6.1 Vector and matrix notations

Example	Definition
A	Matrices
$\vec{b}$	Vectors
$A_{i,j}$	The entry of A at i, j
$[AB]_{i,j}$	The entry of AB at i, j
$\Delta^{k,l}$	Basis matrix (Equation 6.1)

I lay out the notation used for ordinary power series generating functions in Table 6.2.

Table 6.2 Sequence and generating function notations

Example	Definition
a_n	The nth entry of sequence a
$\{a_n\} \stackrel{OPS}{\longleftrightarrow} f(x)$	$f(x) = \sum_{n=0}^{\infty} a_n x^n$ is the ordinary power series (OPS) generating function of the sequence $\{a_n\}$.
$\hat{D}f(x)$	The formal power series derivative of $f(x)$. If $\{a_n\} \stackrel{OPS}{\longleftrightarrow} f(x)$, then $\hat{D}f(x) = \sum_{n=1}^{\infty} n a_n x^{n-1}$.
$\hat{D}^m f(x)$	The mth formal power series derivative of $f(x)$.
$\hat{I}f(x)$	The formal power series antiderivative of $f(x)$. If $\{a_n\} \stackrel{OPS}{\longleftrightarrow} f(x)$, then $\hat{I}f(x) = \sum_{n=0}^{\infty} \frac{a_n}{n+1} x^{n+1}$.
$\hat{I}^m f(x)$	The mth formal power series antiderivative of $f(x)$.
$[x^m]f(x)$	If $\{a_n\} \stackrel{OPS}{\longleftrightarrow} f(x)$, then $[x^m]f(x) = a_m$.

Matrices can be thought of in two ways: s transformation, either on vectors or other matrices, or an object to be transformed (usually by other matrices). The array generating function exists to be the second

kind of object. Array generating functions are a way of encoding the entries of a matrix so they can later be summed, averaged, transformed, or otherwise manipulated. In this section, we will start by defining the array generating function and giving the basic notation. Then we will explore the properties of array generating functions to better understand how they can be acted upon.

The array generating function (AGF) is given by the following definition:

Definition 6.1 (Array Generating Function)**.** Let A be a matrix in $\mathbb{R}^{m\times n}$. The array generating function (AGF) of A, denoted as $A \xleftrightarrow{AGF} A(x,y)$, is defined as

$$A(x,y) = \sum_{i=0}^{m-1}\sum_{i=0}^{n-1} A_{i,j}x^i y^j. \tag{6.3}$$

Next, I examine some of the basic properties of the array generating function (AGF). Operations like left and right matrix multiplication act on the columns and rows of matrices. This means it will be valuable to be able to extract individual rows and columns from the AGF. Then, we will look at the multiplicative properties of the AGF. The AGF is not made to multiply like a matrix, so problems arise in replicating matrix multiplication. AGFs exist as objects to be transformed, but do not adequately serve as transformations themselves. Finally, we will see how AGFs lend themselves to the construction of arbitrarily large matrices via shifting, adding, and using infinite series tools to make larger AGFs out of smaller AGFs.

The columns and rows of a matrix can be readily extracted from the AGF. Let $A \xleftrightarrow{AGF} A(x,y)$. Then $[x^i]A(x,y)$ is the ordinary power series of the ith row of A with respect to y, while $[y^i]A(x,y)$ is the ordinary power series of the ith column of A with respect to x.

The AGF does not have the functionality of matrix multiplication. To demonstrate this, let us examine the product of two AGFs. Let $A \xleftrightarrow{AGF} A(x,y)$ and $B \xleftrightarrow{AGF} B(x,y)$. If we treat them as acting on separate pairs of variables x, y and w, z, then

$$\begin{aligned} A(x,y)B(w,z) &= \left(\sum_{i,j} A_{i,j}x^i y^j\right)\left(\sum_{k,l} B_{k,l}w^k z^l\right) \\ &= \sum_{i,j,k,l} A_{i,j}B_{k,l}x^i y^j w^k z^l. \end{aligned} \tag{6.4}$$

This is analogous to taking the tensor product.

$$[A\otimes B]_{ijkl} = A_{ij}B_{kl}. \tag{6.5}$$

If we try to take the product with each AGF using the same pair of variables, we apply the Cauchy product rule to get

$$\begin{aligned} A(x,y)B(x,y) &= \left(\sum_{i,j} A_{i,j}x^i y^j\right)\left(\sum_{k,l} B_{k,l}x^k y^l\right) \\ = \sum_{i,j,k,l} A_{i,j}B_{k,l}x^i y^j x^k y^l &= \sum_{m=0}^{\infty}\sum_{n=0}^{\infty} x^i y^j \sum_{r=0}^{m-1}\sum_{s=0}^{n-1} A_{r,s}B_{m-r,n-s}. \end{aligned} \tag{6.6}$$

Mimicking matrix multiplication is a desirable property, but not one that the AGF has. This necessitates the TO.

Through a few examples, I will show the process of constructing AGFs. Adding terms to an AGF is as easy as adding higher order terms to a polynomial. Additionally, once a simple AGF has been built, it can be multiplied by powers of x and y to shift it right or down, then added together to make more complex AGFs. Furthermore, AGFs can be recursively constructed by the same methods as ordinary power series generating functions. AGFs using infinite power series can be used to describe infinite-dimensional matrices (see Example 6.6). Then, they can either be converted into TOs, acted upon by TOs, or evaluated at specific values of x and y to compute network measures. These features make AGFs a useful tool for constructing complicated networks.

Example 6.6 (AGF of the Identity Matrix)**.** The identity matrix consists of ones along the main diagonal and zeros elsewhere. Let us call the $n \times n$ identity matrix E_n. Then $E_n \overset{AGF}{\longleftrightarrow} E_n(x,y)$ is found by taking the sum

$$E_n(x,y) = \sum_{i=0}^{n-1} x^i y^i. \tag{6.7}$$

This is a finite geometric series, which allows simplification into the form

$$E_n(x,y) = \frac{(xy)^n - 1}{xy - 1}. \tag{6.8}$$

Alternatively, for the infinite-dimensional identity matrix E, $E \overset{AGF}{\longleftrightarrow} E(x,y)$ takes the form

$$E(x,y) = \sum_{i=0}^{\infty} x^i y^i. \tag{6.9}$$

This infinite geometric series has the simplified form

$$E(x,y) = \frac{1}{1-xy}. \tag{6.10}$$

Example 6.7 (AGF of the Breadth-First Binary Tree)**.** The binary tree, labeled breadth-first, is depicted as a network in Figure 6.1. Its adjacency matrix will be

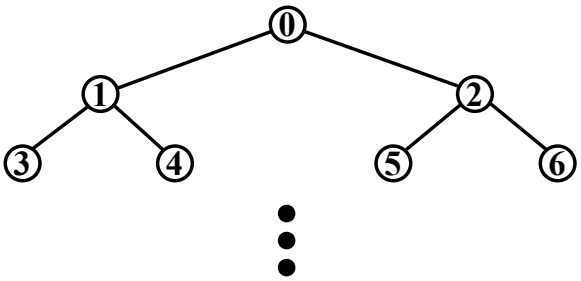

Figure 6.1 The binary tree, labeled breadth-first, drawn out to two layers below the root node.

$$A_{\text{Binary}} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & \\ & & & & & & & \ddots \end{bmatrix} \tag{6.11}$$

Notice that the AGF of the part above the diagonal will take the form

$$A_{\text{half}}(x,y) = \sum_{n=0}^{\infty} x^n y^{2n} \left(y + y^2\right). \tag{6.12}$$

Meanwhile, the part below the diagonal is symmetric, meaning its AGF will be the reflection of the upper half, $A_{\text{half}}(y,x)$. Therefore,

$$\begin{aligned} A_{\text{binary}}(x,y) &= A_{\text{half}}(x,y) + A_{\text{half}}(y,x) \\ &= \sum_{n=0}^{\infty} x^n y^{2n} \left(y + y^2\right) + y^n x^{2n} \left(x + x^2\right). \end{aligned} \tag{6.13}$$

This has a natural extension beyond the binary tree to the k-ary tree:

$$A_{\text{k-ary}}(x,y) = \sum_{n=0}^{\infty} x^n y^{kn} \left(\sum_{i=1}^{k} y^i\right) + x^{kn} y^n \left(\sum_{i=1}^{k} x^i\right). \tag{6.14}$$

Now, let us introduce the transformation operator (TO). I give the necessary definitions for TOs, starting with their analogous basis ($\hat{\Delta}^{i,j}$), then the definition of a TO, and finally defining operator equivalence. Afterwards, I will demonstrate the properties of TOs, which will elucidate some of the choices made in defining the TO.

Definition 6.2 (The Basis Transformation Operator)**.** The basis TO, $\hat{\Delta}_{i,j}$, when applied to a formal power series $f(x)$ with respect to x, is defined as

$$\hat{\Delta}_x^{i,j} f(x) = x^{i-j} \hat{I}^j \left(1 - \hat{I}\hat{D}\right) \hat{D}^j f(x). \tag{6.15}$$

Definition 6.3 (Transformation Operator (TO))**.** Let A be a matrix in $\mathbb{R}^{m \times n}$. The TO of A, denoted as $A \overset{TO}{\longleftrightarrow} \hat{A}$, is defined as

$$\hat{A} = \sum_{i=0}^{m} \sum_{i=0}^{n} A_{i,j} \hat{\Delta}^{i,j}. \tag{6.16}$$

Definition 6.4 (TO with Respect to a Particular Variable)**.** If we need to specify the variable the TO is taken with respect to, we write $\hat{A}$ with respect to x as $\hat{A}_x$. We may need to do this if, for instance, we perform $\hat{A}_x f(x, y)$.

Definition 6.5 (Operator Equivalence)**.** Two operators, $\hat{A}$ and $\hat{B}$, which act on formal power series, are considered equivalent if, for any formal power series $f(x)$,

$$\hat{A}_x f(x) = \hat{B}_x f(x).$$

We write $\hat{A} = \hat{B}$.

Since the definition of TOs involves the large (possibly infinite) summation of basis TOs in equation 6.16, it may be difficult to evaluate in practice. However, using equivalence per Definition 6.5, it is often possible to find an equivalent operator that is much easier to evaluate.

The way TOs are defined in Definition 6.3 does not make it immediately clear how they act like the transformations of the matrices that generate them. In this subsection, we start by examining the properties of the $\hat{I}$ and $\hat{D}$ operators, then we build up to a property of TOs called "analogous representation." This will make it clear that TOs really do act on functions like their related matrices act on vectors. We begin by combining the $\hat{I}$ and $\hat{D}$ operators. This combination will allow us to remove the first terms from a formal power series, as demonstrated in the ID Lemma below. Once we have shown this, we will use it as a building block for the basis TOs, which are in turn used to construct all other TOs.

Lemma 6.5 (ID Lemma)**.** *Let $f(x)$ be an arbitrary formal power series, and let $\{a_i\}$ be the sequence it generates. Then*

$$\hat{I}\hat{D} f(x) = \sum_{i=1}^{\infty} a_i x^i = f(x) - a_0. \tag{6.17}$$

Furthermore,

$$\hat{I}^n \hat{D}^n f(x) = \sum_{i=n}^{\infty} a_i x^i. \tag{6.18}$$

Proof. This will be a proof by induction. First, we prove the base case, Equation 6.17. We prove this by applying the definitions of $\hat{I}$ and $\hat{D}$ in Table Table 6.2 and performing algebraic manipulation. Then, we will prove the inductive hypothesis, which proves that the ID lemma holds for n if it holds for $n-1$. Since we will have proven the case $n=1$, the inductive hypothesis will subsequently prove the ID lemma for $n = 2, 3, 4, ...$

$$\hat{I}\hat{D}f(x) = \hat{I}\sum_{i=1}^{\infty} i a_i x^{i-1} \tag{6.19}$$

Let $b_i = (i+1)a_{i+1}$. Then

$$\begin{aligned} \hat{D}f(x) &= \sum_{i=1}^{\infty} b_{i-1} x^{i-1} \\ &= \sum_{i=0}^{\infty} b_i x^i. \end{aligned} \tag{6.20}$$

So

$$\begin{aligned} \hat{I}\hat{D}f(x) &= \hat{I}\sum_{i=0}^{\infty} b_i x^i \\ = \sum_{i=0}^{\infty} \frac{b_i}{i+1} x^{i+1} &= \sum_{i=0}^{\infty} \frac{(i+1)a_{i+1}}{i+1} x^{i+1} \\ = \sum_{i=0}^{\infty} a_{i+1} x^{i+1} &= \sum_{i=1}^{\infty} a_i x^i. \end{aligned} \tag{6.21}$$

If we apply $\hat{I}$ and $\hat{D}$ twice, we have

$$\hat{I}^2\hat{D}^2 f(x) = \hat{I}^2\hat{D}\sum_{i=0}^{\infty} b_i x^i. \tag{6.22}$$

Let $c_i = (i+1)b_{i+1}$. Then

$$\begin{aligned} \hat{I}^2\hat{D}^2 f(x) &= \hat{I}^2\sum_{i=0}^{\infty} c_i x^i \\ = \hat{I}\sum_{i=0}^{\infty} \frac{c_i}{i+1} x^{i+1} &= \hat{i}\sum_{i=1}^{\infty} b_i x^i \\ = \sum_{i=1}^{\infty} \frac{b_i}{i+1} x^{i+1} &= \sum_{i=2}^{\infty} a_i x^i. \end{aligned} \tag{6.23}$$

For each higher power of $\hat{I}$ and $\hat{D}$, we start with an expression of the form

$$\hat{I}^{n-1}\hat{D}^{n-1}f(x) = \sum_{i=n-1}^{\infty} a_i x^i. \tag{6.24}$$

Thus, it can be shown that for any $n \in \mathbb{N}$

$$\begin{aligned} \hat{I}^n\hat{D}^n f(x) &= \hat{I}\left(\hat{I}^{n-1}\hat{D}^{n-1}\right)\hat{D}f(x) \\ = \hat{I}\left(\hat{I}^{n-1}\hat{D}^{n-1}\right)\sum_{i=0}^{\infty} b_i x^i &= \hat{I}\sum_{i=n-1}^{\infty} b_i x^i \\ = \sum_{i=n-1}^{\infty} \frac{b_i}{i+1}x^{i+1} &= \sum_{i=n-1}^{\infty} \frac{(i+1)a_{i+1}}{i+1}x^{i+1} \\ = \sum_{i=n-1}^{\infty} a_{i+1}x^{i+1} &= \sum_{i=n}^{\infty} a_i x^i. \end{aligned} \tag{6.25}$$

□

Next, we can prove that the TO basis represents the basis matrices $\Delta^{i,j}$. In other words, just as $\Delta^{i,j}$ selects entry j from the input vector and adds it to entry i of the output vector, $\hat{\Delta}^{i,j}$ selects the coefficient of x^j in the input function and adds it to the coefficient of x^i in the output function.

Theorem 6.6. *Let $b(x)$ be an arbitrary formal power series, and let $\{b_j\}$ be the sequence it generates. Then*

$$\hat{\Delta}^{i,j}b(x) = x^i b_j. \tag{6.26}$$

Proof. From Definition 6.2 we have

$$\begin{aligned} \hat{\Delta}^{i,j}b(x) &= \left[x^{i-j}\hat{I}^j\left(1-\hat{I}\hat{D}\right)\hat{D}^j\right]b(x) \\ &= x^{i-j}\left(\hat{I}^j\hat{D}^j - \hat{I}^{j+1}\hat{D}^{j+1}\right)b(x) \end{aligned} \tag{6.27}$$

Employing Lemma 6.5 yields

$$\begin{aligned} x^{i-j}\left(\hat{I}^j\hat{D}^j - \hat{I}^{j+1}\hat{D}^{j+1}\right)b(x) &= x^{i-j}\left(\sum_{j}^{\infty} b_j x^j - \sum_{j+1}^{\infty} b_j x^j\right) \\ &= x^{i-j}x^j b_j. \end{aligned} \tag{6.28}$$

Therefore,

$$\hat{\Delta}^{i,j}b(x) = x^i b_j. \tag{6.29}$$

□

The following result illustrates the connection between TOs and matrix representations.

Theorem 6.7 (Analogous Representation)**.** *Let A be a matrix in $\mathbb{R}^{n\times m}$ and $\hat{A}$ be its TO. Then, for any $\vec{b}\in\mathbb{R}^m$, the matrix multiplication $\vec{y}=A\vec{b}$ is analogous to the TO operation*

$$y(x)=\hat{A}b(x). \tag{6.30}$$

with $\{b_i\}\overset{OPS}{\longleftrightarrow}b(x)$ and $\{y_n\}\overset{OPS}{\longleftrightarrow}y(x)$.

Proof. We will show by direct computation that the function produced by taking $\hat{A}b(x)$ is equal to the generating function for the vector $\vec{y}=A\vec{b}$. Per Definition 6.3,

$$\begin{aligned}\hat{A}b(x)&=\left(\sum_{i,j}A_{i,j}\hat{\Delta}^{i,j}\right)b(x)\\&=\sum_{i,j}A_{i,j}\hat{\Delta}^{i,j}b(x)\end{aligned} \tag{6.31}$$

Applying Theorem 6.6 yields

$$\begin{aligned}\hat{A}b(x)&=\sum_{i,j}A_{i,j}x^ib_j\\=\sum_i x^i\sum_j A_{i,j}b_j&=\sum_i x^iy_j\\&=y(x).\end{aligned} \tag{6.32}$$

□

TOs can be multiplied and added together to form other TOs, paralleling the matrices they represent. The properties of TO addition are obvious and trivially provable given the way they are defined in Definition 6.3. However, the multiplicative properties of TOs are not immediately clear, so they deserve some attention. In this section, we will demonstrate that TOs are linear operators, then we will show that TO multiplication is associative but non-commutative, then we will finally cap off this section by proving that the product of TOs properly represents the respective product of matrices.

Theorem 6.8 (Linearity of TOs)**.** *Let $\hat{A}$ be an arbitrary TO, let $f(x)$ and $g(x)$ be arbitrary formal power series, and let α, β be arbitrary real numbers. Then*

$$\hat{A}\left(\alpha f(x)+\beta g(x)\right)=\alpha\hat{A}f(x)+\beta\hat{A}g(x). \tag{6.33}$$

Proof.

$$\begin{aligned}
\hat{A}\left(\alpha f(x)+\beta g(x)\right) &= \left(\sum_{i,j} A_{i,j}\hat{\Delta}^{i,j}\right)\left(\alpha f(x)+\beta g(x)\right)\\
&= \sum_{i,j} A_{i,j}\hat{\Delta}^{i,j}\left(\alpha f(x)+\beta g(x)\right)\\
&= \sum_{i,j} A_{i,j}x^i[x^j]\left(\alpha f(x)+\beta g(x)\right)\\
&= \sum_{i,j} A_{i,j}x^i\left(\alpha[x^j]f(x)+\beta[x^j]g(x)\right)\\
&= \alpha\sum_{i,j} A_{i,j}x^i[x^j]f(x)+\beta\sum_{i,j} A_{i,j}x^i[x^j]g(x)\\
&= \alpha\sum_{i,j} A_{i,j}\hat{\Delta}^{i,j}f(x)+\beta\sum_{i,j} A_{i,j}\hat{\Delta}^{i,j}g(x)\\
&= \alpha\hat{A}f(x)+\beta\hat{A}g(x).
\end{aligned} \tag{6.34}$$

□

Theorem 6.9 (Associativity of TOs)**.** *Let $\hat{A}$, $\hat{B}$, and $\hat{C}$ be arbitrary TOs, and let $f(x)$ be an arbitrary formal power series. Then,*

$$\hat{A}\left(\hat{B}f(x)\right) = \left(\hat{A}\hat{B}\right)f(x), \tag{6.35}$$

and, furthermore,

$$\left(\hat{A}\hat{B}\right)\hat{C} = \hat{A}\left(\hat{B}\hat{C}\right) \tag{6.36}$$

Proof. We start by proving equation (6.35).

$$\begin{aligned}
\hat{A}\left(\hat{B}f(x)\right) &= \hat{A}\sum_{k,l} B_{k,l}x^k[x^l]f(x)\\
&= \sum_{i,j} A_{i,j}x^i[x^j]\left(\sum_{k,l} B_{k,l}x^k[x^l]f(x)\right)\\
&= \sum_{i,j,l} A_{i,j}x^iB_{j,l}[x^l]f(x)\\
&= \sum_{i,j,l} x^iA_{i,j}B_{j,l}[x^l]f(x).
\end{aligned} \tag{6.37}$$

$$\begin{aligned}\left(\hat{A}\hat{B}\right) f(x) &= \left[\left(\sum_{i,j} A_{i,j}\hat{\Delta}^{i,j}\right)\left(\sum_{k,l} B_{k,l}\hat{\Delta}^{k,l}\right)\right] f(x) \\ &= \left(\sum_{i,j,k,l} A_{i,j}B_{k,l}\hat{\Delta}^{i,j}\hat{\Delta}^{k,l}\right) f(x) \\ &= \left(\sum_{i,j,l} A_{i,j}B_{j,l}\hat{\Delta}^{i,l}\right) f(x) \\ &= \sum_{i,j,l} x^i A_{i,j}B_{j,l}[x^l]f(x).\end{aligned} \tag{6.38}$$

Therefore,

$$\hat{A}\left(\hat{B}f(x)\right) = \left(\hat{A}\hat{B}\right) f(x), \tag{6.39}$$

Now we prove equation (6.36).

$$\begin{aligned}\left(\hat{A}\hat{B}\right)\hat{C} &= \left(\sum_{i,j,l} A_{i,j}B_{j,l}\hat{\Delta}^{i,l}\right)\hat{C} \\ &= \left(\sum_{i,j,l} A_{i,j}B_{j,l}\hat{\Delta}^{i,l}\right)\sum_{m,n} C_{m,n}\hat{\Delta}^{m,n} \\ &= \sum_{i,j,l,m,n} A_{i,j}B_{j,l}C_{m,n}\hat{\Delta}^{i,l}\hat{\Delta}^{m,n} \\ &= \sum_{i,j,l,n} A_{i,j}B_{j,l}C_{l,n}\hat{\Delta}^{i,n}.\end{aligned} \tag{6.40}$$

$$\begin{aligned}\hat{A}\left(\hat{B}\hat{C}\right) &= \left(\sum_{i,j} A_{i,j}\hat{\Delta}^{i,j}\right)\left(\sum_{k,l,n} B_{k,l}C_{l,n}\hat{\Delta}^{k,n}\right) \\ &= \sum_{i,j,k,l,n} A_{i,j}B_{k,l}C_{l,n}\hat{\Delta}^{i,j}\hat{\Delta}^{k,n} \\ &= \sum_{i,j,l,n} A_{i,j}B_{j,l}C_{l,n}\hat{\Delta}^{i,n}.\end{aligned} \tag{6.41}$$

Therefore,

$$\left(\hat{A}\hat{B}\right)\hat{C} = \hat{A}\left(\hat{B}\hat{C}\right). \tag{6.42}$$

□

Theorem 6.10 (Non-Commutativity of TOs)**.** *TO multiplication is not commutative. It suffices to show that there exist TOs $\hat{A}$ and $\hat{B}$ such that TO multiplication is non-commutative. That is to say*

$$\hat{A}\hat{B} \neq \hat{B}\hat{A}. \tag{6.43}$$

Proof. This is a proof by counterexample. Notice that

$$\hat{\Delta}^{0,1}\hat{\Delta}^{1,1} = \hat{\Delta}^{0,1}, \tag{6.44}$$

but

$$\hat{\Delta}^{1,1}\hat{\Delta}^{0,1} = 0. \tag{6.45}$$

Therefore,

$$\hat{\Delta}^{0,1}\hat{\Delta}^{1,1} \neq \hat{\Delta}^{1,1}\hat{\Delta}^{0,1}. \tag{6.46}$$

□

Theorem 6.11 (Analogous Representation of TO Products)**.** *Let A be a matrix in $\mathbb{R}^{n\times m}$ and B be a matrix in $\mathbb{R}^{m\times l}$, with their respective TOs $\hat{A}$ and $\hat{B}$. Also let $\vec{b}$ be an arbitrary vector in $\mathbb{R}^l$, with $\{b_i\} \overset{OPS}{\longleftrightarrow} b(x)$, $\vec{y} = AB\vec{b}$, and $\{y_i\} \overset{OPS}{\longleftrightarrow} y(x)$. Then*

$$\hat{A}\hat{B}b(x) = y(x). \tag{6.47}$$

Proof.

$$\begin{aligned}
\hat{A}\hat{B}b(x) &= \sum_{i,j,k} x^i A_{i,j} B_{j,k} b_k \\
&= \sum_i x^i \sum j,k A_{i,j} B_{j,k} b_k \\
&= \sum_i x^i y_i \\
&= y(x)
\end{aligned} \tag{6.48}$$

□

These proofs about proper representation all assume that, for a TO of a matrix in $\mathbb{R}^{m\times n}$, we start with a vector in $\mathbb{R}^n$. Suppose we start with a vector of the wrong size, residing in $\mathbb{R}^l : l \neq n$ instead. In this case, the matrix multiplication is not defined, but the TO operation is. If the vector is too small, the nonexistent terms are treated as zeros. If the vector is too large, the extra terms are mapped to zero. One consequence of this is that infinite-dimensional TOs do not have to be applied to infinite-dimensional power series; they can be used just as well on finite-dimensional power series. The simplest example of this is the identity TO, 1. This is infinite dimensional, but multiplying it by any finite-dimensional power series returns a copy of that power series.

Example 6.8 (Example of the flexibility of TO size)**.** As matrices, the multiplication

$$\begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 5 \end{bmatrix} \tag{6.49}$$

is not defined. However, if we convert the matrix to a TO and the vector to a formal power series, we can perform the operation

$$\left(\hat{\Delta}^{0,0} + 2\hat{\Delta}^{0,1} + 3\hat{\Delta}^{1,1}\right)\left(2x + 5x^2\right) = 4 + 6x. \tag{6.50}$$

Example 6.9 (Infinite-Dimensional Identity TO)**.** A TO is meant to treat a generating function the same way its matrix treats a vector. The simplest example is the identity matrix E. It turns out that the infinite dimensional case is easiest to start with, since a finite-dimensional matrix would essentially truncate the input generating function first. Now, the identity matrix returns the vector it takes in. Multiplication by one does the same thing.

$$E = \begin{bmatrix} 1 & 0 & 0 & 0 & \\ 0 & 1 & 0 & 0 & \\ 0 & 0 & 1 & 0 & \\ 0 & 0 & 0 & 1 & \\ & & & & \ddots \end{bmatrix} \xleftrightarrow{TO} 1. \tag{6.51}$$

If we apply this TO to any power series $f(x)$, it always returns itself, since $1f(x) = f(x)$.

In contrast, the finite-dimensional identity TO, $1 - \hat{I}^n\hat{D}^n$, only acts as an identity on dimensions n and fewer.

Example 6.10 (3D Identity TO)**.** Let us apply the 3D Identity TO, $1 - \hat{I}^3\hat{D}^3$, to two different formal power series: $f(x) = 2 + x + 6x^2$ and $g(x) = 1 + x^2 + 5x^5 + 6x^7$ to see how a finite-dimensional identity TO acts on power series of different dimensions.

$$\begin{aligned} \left(1 - \hat{I}^3\hat{D}^3\right) f(x) &= (2 + x + 6x^2) - \hat{I}^3\hat{D}^3(2 + x + 6x^2) \\ &= (2 + x + 6x^2) - \hat{I}^3(0) \\ &= (2 + x + 6x^2) \\ &= f(x). \end{aligned} \tag{6.52}$$

$$\begin{aligned} \left(1 - \hat{I}^3\hat{D}^3\right) g(x) &= (1 + x^2 + 5x^5 + 6x^7) - \hat{I}^3\hat{D}^3(1 + x^2 + 5x^5 + 6x^7) \\ &= (1 + x^2 + 5x^5 + 6x^7) - \hat{I}^3(300x^2 + 1260x^4) \\ &= (1 + x^2 + 5x^5 + 6x^7) - (5x^5 + 6x^7) \\ &= 1 + x^2 \\ &\neq g(x). \end{aligned} \tag{6.53}$$

Another consequence of this size flexibility is that TOs can be expanded. The tools for this include infinite-dimensional TOs, the spread TO (Example 6.14), and array generating functions.

The definitions in Section 6.3 of the TO and basis TO likely seem rather arcane when taken at face value. The goal of this subsection is to explain the motivation behind those definitions through a series of

examples.

Example 6.11 (Lower diagonal)**.** If multiplication by one is the identity, the next natural thing to try is multiplication by x, since we are dealing with power series. Doing this shifts the power series down by one entry. This corresponds to a matrix with ones in the lower diagonal.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & \\ 1 & 0 & 0 & 0 & \\ 0 & 1 & 0 & 0 & \\ 0 & 0 & 1 & 0 & \\ & & & & \ddots \end{bmatrix} \overset{TO}{\longleftrightarrow} x. \tag{6.54}$$

Furthermore, we can create even lower diagonals by using higher powers of x.

Example 6.12 (Upper diagonals)**.** If multiplying by x creates a lower diagonal, does dividing by x create an upper diagonal? Not quite. For example, if we start with $f(x) = 1 + x + x^2$, $x^{-1}f(x) = \frac{1}{x} + 1 + x$. That first term is a problem; it takes us out of the realm of power series. If we were to remove it first, everything would work out fine. Enter $\hat{I}\hat{D}$, which does just that (per Lemma 6.5).

$$\begin{bmatrix} 0 & 1 & 0 & 0 & \\ 0 & 0 & 1 & 0 & \\ 0 & 0 & 0 & 1 & \\ 0 & 0 & 0 & 0 & \\ & & & & \ddots \end{bmatrix} \overset{TO}{\longleftrightarrow} \frac{1}{x}\hat{I}\hat{D}. \tag{6.55}$$

Example 6.13 ($\hat{\Delta}^{0,0}$)**.** Infinite matrices are fine and dandy, but most of the existing matrices we would want to convert to TOs are finite, and we might want to modify TOs term-by-term in infinite matrices too. We need a way to isolate a single entry. This will involve another application of $\hat{I}\hat{D}$.

$$\Delta^{0,0} = \begin{bmatrix} 1 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & \\ & & & & \ddots \end{bmatrix} \overset{TO}{\longleftrightarrow} 1 - \hat{I}\hat{D} = \hat{\Delta}^{0,0}. \tag{6.56}$$

Furthermore, we can construct single-entry TOs further down the diagonal with more consecutive uses of $\hat{I}$ and $\hat{D}$.

$$\Delta^{j,j} \overset{TO}{\longleftrightarrow} \hat{I}^j\hat{D}^j - \hat{I}^{j+1}\hat{D}^{j+1} = \hat{I}^j\left(1 - \hat{I}\hat{D}\right)\hat{D}^j = \hat{\Delta}^{j,j}. \tag{6.57}$$

Finally, all we need to do to get the basis TOs, per Definition 6.3, is to shift off the diagonal by multiplying by x^{i-j}. We do not even have to worry about using $\hat{I}$ and $\hat{D}$ to remove more terms first, because $|i - j| \leq j$, and j terms have already been removed by $\hat{\Delta}^{j,j}$.

Example 6.14 (The Spread TO)**.** In the recursive construction of large matrices, it is often useful to spread out the entries of that matrix. This is done by creating rows and columns of zeros and splicing them

in between the rows and columns of an existing matrix. When applying the spread operator to an AGF, this amounts to taking $A(x^2, y^2)$ instead of $A(x, y)$. It turns out that the operator that does this is specifically a TO, which can be shown by writing out its analogous matrix. Let us begin by thinking about how the spread TO ought to act upon a vector's generating function. For $\{a_n\} \xleftrightarrow{OPSGF} f(x)$, the spread TO $\hat{S}$ ought to give us

$$\hat{S}f(x) = f(x^2). \tag{6.58}$$

This amounts to taking each $[x^i]f(x)$ and mapping it to x^{2i}. Therefore,

$$\hat{S} = \sum_{i=0}^{\infty} \hat{\Delta}^{2i,i}. \tag{6.59}$$

This operator can be constructed from a sum of basis TOs, which are analogous to basis matrices. To apply the spread TO to a matrix, we must spread both the rows and columns. For a matrix A, with TO $A \xleftrightarrow{AGF} A(x, y)$, we apply the spread TO to both x and y.

$$A(x^2, y^2) = \hat{S}_x \hat{S}_y A(x, y). \tag{6.60}$$

Note that, since $\hat{S}_x$ and $\hat{S}_y$ act on different variables, they commute with each other.

Example 6.15 (Binary Tree by Recursion)**.** This example demonstrates combining TOs and AGFs for recursively constructing large matrices. We will start with the AGF of a primitive depth-1 binary tree, then recursively grow it to arbitrary size using TOs. The AGF of a depth-1 binary tree is

$$A_1(x, y) = y + y^2 + x + x^2. \tag{6.61}$$

The recursive step for each depth level conceptually does this: it applies the spread TO to make room for two copies of the AGF, shifts them to fit between each other, adds them together into one AGF, and connects both copies to a new root node. The process is depicted graphically in Figure Figure 6.2. As an equation, the inductive step takes the form

$$A_{n+1}(x, y) = y + y^2 + x + x^2 + \left(xy + x^2y^2\right)\left(\hat{S}_x \hat{S}_y A_n(x, y)\right). \tag{6.62}$$

In practice, neither the AGF nor the TO will be useful for all situations; they are both necessary. It will be useful to be able to convert between them. The definitions of AGFs and of TOs allow us to take either a TO or AGF, turn it back into the matrix that generated it, then construct the AGF or TO we want to turn it into. However, this method is cumbersome, and we are in need of more elegant solutions.

To convert a TO to an AGF, the idea is that we have the TO act on the AGF of the identity matrix. TOs happen to transform AGFs the same way they transform power series generating functions, only an AGF is a list of column vectors, instead of only one.

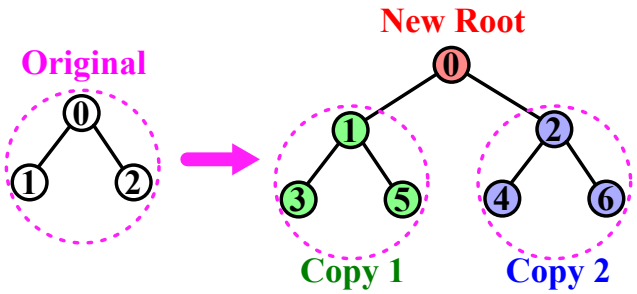

Figure 6.2 Demonstration of the recursive step for generating a binary tree. In particular, this figure goes between depth 1 and depth 2.

Theorem 6.12 (Converting TO to AGF)**.** *Let A be a matrix in $\mathbb{R}^{m\times n}$, with $A \overset{TO}{\longleftrightarrow} \hat{A}$. Then*

$$A \overset{AGF}{\longleftrightarrow} \hat{A}\frac{1}{1-xy}. \tag{6.63}$$

Proof.

$$\begin{aligned}\hat{A}\frac{1}{1-xy} &= \left(\sum_{i,j} A_{i,j}\hat{\Delta}_x^{i,j}\right)\left(\sum_{n=0}^{\infty}(xy)^n\right)\\ &= \sum_{i,j} A_{i,j}x^i[x^j]\sum_{n=0}^{\infty}(xy)^n\\ &= \sum_{i,j} A_{i,j}x^iy^j.\end{aligned} \tag{6.64}$$

Therefore,

$$A \overset{AGF}{\longleftrightarrow} \hat{A}\frac{1}{1-xy}. \tag{6.65}$$

□

Theorem 6.13 (Converting AGF to TO)**.** *Let A be a matrix in $\mathbb{R}^{m\times n}$, with $A \overset{AGF}{\longleftrightarrow} A(x,y)$. Then*

$$A \overset{TO}{\longleftrightarrow} A\left(x, \frac{1}{x}\hat{I}\hat{D}\right) - A\left(x, \frac{1}{x}\hat{I}^2\hat{D}^2\right). \tag{6.66}$$

Proof. First, we will prepare the AGF for the computation in Equation 6.66 by carefully expressing it in terms of an x and y that do not commute. Then, we will substitute x and y with the operators in Equation 6.66. Finally, we will show that the operator that springs from this substitution is the TO.

Per Definition 6.1,

$$A(x,y) = \sum_{i,j} A_{i,j}x^iy^j. \tag{6.67}$$

Note that it is essential to decompose $A(x,y)$ into this precise form before continuing, because from this point forward x and y are substituted for operators which do not commute. Thus powers of x must always

precede powers of y in each and every term.

$$\begin{aligned} A\left(x, \frac{1}{x}\hat{I}\hat{D}\right) &= \sum_{i,j} A_{i,j} x^i x^{-j} \hat{I}^j \hat{D}^j \\ &= \sum_{i,j} A_{i,j} x^{i-j} \hat{I}^j \hat{D}^j. \end{aligned} \tag{6.68}$$

$$\begin{aligned} A\left(x, \frac{1}{x}\hat{I}^2\hat{D}^2\right) &= \sum_{i,j} A_{i,j} x^i x^{-j} \hat{I}^{j+1} \hat{D}^{j+1} \\ &= \sum_{i,j} A_{i,j} x^{i-j} \hat{I}^{j+1} \hat{D}^{j+1}. \end{aligned} \tag{6.69}$$

This implies

$$\begin{aligned} A\left(x, \frac{1}{x}\hat{I}\hat{D}\right) - A\left(x, \frac{1}{x}\hat{I}^2\hat{D}^2\right) &= \sum_{i,j} A_{i,j} x^{i-j} \hat{I}^j \hat{D}^j - \sum_{i,j} A_{i,j} x^{i-j} \hat{I}^{j+1} \hat{D}^{j+1} \\ &= \sum_{i,j} A_{i,j} x^{i-j} \hat{I}^j \left(1 - \hat{I}\hat{D}\right) \hat{D}^j \\ &= \sum_{i,j} A_{i,j} \hat{\Delta}^{i,j}. \end{aligned} \tag{6.70}$$

By Definition 6.3, this implies

$$A \overset{TO}{\longleftrightarrow} A\left(x, \frac{1}{x}\hat{I}\hat{D}\right) - A\left(x, \frac{1}{x}\hat{I}^2\hat{D}^2\right). \tag{6.71}$$

□

CHAPTER 7

DISCUSSION AND CONCLUSION

In this thesis, I defined *interferometer networks*, which serve as an archetype for network problems exhibiting phase. I explored the ways signals can change in an interferometer effect as edge-weight phases are varied. Then, I defined interferometric versions of the clustering coefficient and path length. Since the definition of *apparent path strength* involves taking a matrix inverse, which may or may not exist, especially in the case of runaway feedback, I developed a deeper understanding of feedback (and how to avoid it). After building up the foundation for analyzing interferometer networks, I applied what I had learned to demonstrate that interferometric measures are necessary for understanding the behavior of interferometer networks for the case of the small-world interferometer, and I introduced the intermingling of matrices and generating functions. Now, I will draw conclusions from these results and discuss their implications.

Though I developed interferometer networks with optics in mind, their mathematical form is very general. Any system in which signals are multiplied by edge weights and added together at vertices will follow the vertex-signal equation exactly (Equation 1.3). For systems that deviate slightly from this paradigm, the vertex signal equation can be modified to accomodate the new problem, but the essence of complex numbers being added together (and thus interfering) will still be preserved.

Since interferometers were our physical inspiration for this investigation, my first foray into the network theory of interferometers was towards the goal of using more complex interferometer networks to improve the precision of interferometric measurements. My first intuition was that the archetypal interferometer (Figure 2.1) was the best interferometer, a kind of platonic ideal, and that more complex interferometer networks would only detract from this simple structure. This is why I tried proving interferometer limitation. The creation of the N-stage skew cycle interferometer, which does outperform the archetypal interferometer (by producing faster change at the output than the total change applied to inputs), came as a surprise to me. The sensitivity of the N-stage skew cycle interferometer to a few hypothetical sources of error indicated that the physical implementation of this device would likely be difficult, but it introduced the possibility that a real-world network could be found where small changes to inputs could create rapid changes to outputs. These systems would be extremely sensitive, and thus fascinating objects of study. One avenue for future research would be the creation of models that exhibit this property, examining those models to find the network features that create this sensitivity, and understanding why those properties would (or would not) arise in real systems.

My next investigation was into the application of network measures to interferometer networks. In defining our measures (*apparent path length* in Equation 3.5, and *interferometric clustering* in Equation 3.12), I wanted to ensure that their definitions captured the interferometric essence, with complex numbers being added together and interfering. This is not the only way to generalize network measures to complex-valued edge weights. Interestingly, while this thesis was being written, other researchers conducted a similar investigation and posted a preprint of their findings to the arXiv [43]. While my work generalized to complex-valued networks with a specific model of signal transfer in mind, Bottcher and Porter did not base their generalizations on a particular problem. As a result, my measures capture the behavior of interfering signals, while their measures remain more general. Both generalizations have their place; using one or the other is context dependent. Their version of the local complex-weighted clustering coefficient was defined as

$$c_i^w = \frac{1}{k_i(k_i-1)} \sum_{j,k} \tilde{w}_{ij}\tilde{w}_{jk}\tilde{w}_{ki}, \tag{7.1}$$

where k_i is the unweighted degree at vertex i and the edge weights $\tilde{w}_{ij}$ are normalized by the maximum edge weight in the network. Equation 7.1 differs from my definition in a few ways. First, it is generalized from a different definition of weighted clustering [44], which, among other things, differs from my source [34] by employing a geometric mean instead of an arithmetic mean. Equation 7.1 also produces a complex number, while interferometric clustering does not. Finally, the most substantial way Equation 7.1 differs is that it does not capture the interference between competing paths like the interferometric clustering does. If we interpret Equation 7.1 as being defined for directed networks, then it counts cycle triangles (instead of middleman triangles, like our measure [36]). Through this lens, Equation 7.1 does account for interference amongst different cycles via the sum, but it cannot account for interference within triangles themselves. For path length, the preprint does not redefine it per se, but their definitions of quantum-random walk centrality, closeness centrality, and betweenness centrality echo the apparent path strength idea of summing paths to or from a vertex, and their discussion of matrix powers and walks involves interfering paths. Interestingly, the apparent path strength happens to be the sum of all powers of the complex-weighted adjacency matrix. To prove this, we only need the uniqueness of the matrix inverse [32], and we show that the apparent path strength P (Equation 3.5) and the sum of all matrix powers both invert $(I - W)$

$$(I-W)P = (I-W)(I-W)^{-1} = I. \tag{7.2}$$

$$(I-W)\sum_{n=0}^{\infty} W^n = \left(\sum_{n=0}^{\infty} W^n\right) - \left(\sum_{n=1}^{\infty} W^n\right) \tag{7.3}$$

$$= W^0 = I. \tag{7.4}$$

Note that we had to assume that the infinite series in these equations converge. In this thesis, I only spend significant time developing interferometric versions of path length and clustering. Future explorations could involve creating interferometric counterparts to the plethora of measures in [43] and performing a more thorough comparison of our measures.

After defining our new network measures, I noticed the potential problems that arise with defining apparent path strength as a matrix inverse. Physically, these problems manifest as runaway feedback loops. I demonstrated through analysis of the Fabry-Perot interferometer [38] that these problems are physical, and not merely a quirk of our definition of the interferometer network, but Theorems 4.3 and 4.4 only presented one way of preventing this: ensuring that dissipation occurs at every vertex. In the small-world interferometer network model, I introduced a 5% dissipation at every vertex. However, other ways of preventing feedback could arise from the structure of networks. Identifying these means of preventing runaway feedback and finding those mechanics in real-world systems could be a fruitful avenue of future exploration.

The purpose of the work in Chapter 5 was to demonstrate where interferometric measures are necessary. I took the well-known phenomenon of the small-world effect, and recreated it on interferometers. I found that the rich interference behavior is lost when carelessly applying real-valued measures to the interferometers, but interferometric measures do capture this interference. Opportunities for future work include analytically modeling $S(\beta, \phi, N, k)$, to grant a better understanding of how interference affects signal transfer at all-possible regions of the parameter space; application of these measures to quantum problems, including quantum random walks and condensed matter models; application of these measures to neural networks as they act on models that exhibit phase transitions, to see if interferometric measures coincide with phase transitions; introducing variability to edge weights, to see how interferometric measures would change for interference problems in the real-world; and testing interferometric measures on real-world data sets.

Array generating functions and *transformation operators* take on both the benefits and the challenges of generating functions. The primary benefit of generating functions is that they can compress large, recursive sequences down into short expressions. This is especially promising for modeling networks with fractal geometry [45, 46], since the self-similarity of fractals is recursive. The primary challenge is that, for sequences that deviate from recursive structure, the recursion benefit of generating functions is lost, and the generating functions becomes tedious and cumbersome instead of helpful. It is clear that the generating function is useful for modeling ordered networks like lattices, and that the method is not useful for random networks. However, it is an open question when array generating functions are appropriate for networks between the two extremes. For instance, the Watts-Strogatz model [31] takes the form of a ring lattice with

a small number of random rewirings.

In addition to this open question, the possibilities for future exploration of array generating functions and transformation operators are plentiful. One possible exploration is converting common networks to functions and common functions to networks. For example, it could be useful to find the counterparts to lattices, Erdős-Rényi random networks, Watts-Strogatz small-world networks, Barabási-Albert scale-free networks, Leguerre polynomials, Chebyshev polynomials, exponentials, gaussians, trigonometric functions, and hyperbolic trigonometric functions. Furthermore, this could develop into creating different bases for networks. It is possible that new network measures could arise from decomposing array generating functions into a Fourier basis. Another sphere of function-related behaviors to explore is asymptotics. We may learn about the common network structure of two networks that have asymptotically equivalent array generating functions or transformation operators. It is possible that this could give rise to new ways to approximate the structure of complex networks.

Another avenue of potential development is the use of transformation operators to analyze paths on networks. The combinatorics of paths on networks can be examined by constructing a power series where each coefficient is a power of the transformation operator, analogously to how apparent path strength is related to a power series of the weighted adjacency matrix (Equations 7.2 and 7.4). This path generating function could track all possible walks on a network, with each term corresponding to the multiplicities of a particular path length. This could be further used to number the distinct walks of any path length by evaluating the array generating function terms at $x = 1$, $y = 1$ (where x and y are the two variables in the array generating function's 2D power series), disposing of vertex indices. This becomes a graph invariant, a network property that does not depend on a particular indexing scheme. Graph invariants are important for both quantifying the properties of networks and for posing necessary (but not sufficient) criteria for the graph isomorphism problem [47–49]. Current algorithms for solving graph isomorphism, the problem of demonstrating that two networks are identical up to the order of vertex indices, rely on a combination of graph invariants and brute force computation [50]. Further development of graph invariants based on generating function objects that remove vertex index information may be key in constructing a sufficient set of conditions for graph isomorphism. This will be especially necessary for solving graph isomorphism for networks of infinite size, like those possible with array generating functions, since traditional algorithms can only work on finite networks.

There is also the possibility of applying this generating function methodology to the mainstays of complex network theory. Array generating functions and transformation operators could potentially be used to compute network measures like degree, clustering, or assortativity. Since array generating functions and transformation operators can easily handle networks of infinite size, they may provide useful

techniques for proving that network properties hold in the large size limit. Since transformation operators capture the behavior of the adjacency matrix under matrix multiplication, they are a possible tool for computing spectral measures of networks. These techniques could contribute to random matrix theory.

Potential subfields arise from bridging functional analysis and network theory. The generating function methods provide an alternative to adjacency matrices, with the potential to represent unlabeled networks. AGFs and TOs are potential tools for the more challenging problems of complex network theory. The places generating functions and matrices can be taken together are myriad, and afford exciting opportunities for future explorations.

The creation of interferometer networks allows the modeling of interfering physical systems. I have explored how these networks create phase change, can be measured by path lengths and clustering, how feedback loops behave, and where interferometric measures are necessary to characterize them. The introduction of array generating functions and transformation operators allow the capture of large, recursively structured networks, and create many possibilities for the mathematical analysis of networks. This suite of tools extends the reach of network science.

REFERENCES

[1] Anatol Rapoport; William J. Horvath. A study of a large sociogram. *Banks in Insurance Report*, 6, Jan 2007. doi: 10.1002/bs.3830060402.

[2] John F. Padgett and Christopher K. Ansell. Robust action and the rise of the medici, 1400-1434. *The American journal of sociology*, 98(6):1259–1319, 1993. ISSN 0002-9602.

[3] Duncan J. Watts. *Small Worlds: The Dynamics of Networks between Order and Randomness (Princeton Studies in Complexity)*. Princeton University Press, 2003. ISBN 0691117047; 9780691117041.

[4] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM'99 conference: applications, technologies, architectures, and protocols for computer communication*, volume 29 of *SIGCOMM '99*, pages 251–262, NEW YORK, 1999. ACM. ISBN 1581131356.

[5] Andre Broido and kc claffy. Internet topology: connectivity of IP graphs. In Sonia Fahmy and Kihong Park, editors, *Scalability and Traffic Control in IP Networks*, volume 4526, pages 172 – 187. International Society for Optics and Photonics, SPIE, 2001. doi: 10.1117/12.434393.

[6] Ed Bullmore and Olaf Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature reviews. Neuroscience*, 10(3):186–198, 2009. ISSN 1471-003X.

[7] Andrea Avena-Koenigsberger, Bratislav Misic, and Olaf Sporns. Communication dynamics in complex brain networks. *Nature reviews. Neuroscience*, 19(1):17–33, 2018. ISSN 1471-003X.

[8] M.E.J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, May 2003.

[9] Bhuvanesh Sundar, Marc Andrew Valdez, Lincoln D. Carr, and Kaden R. A. Hazzard. Complex-network description of thermal quantum states in the ising spin chain. *Physical review. A*, 97 (5), 2018. ISSN 2469-9926.

[10] Shehtab Zaman and Wei-Cheng Lee. Real-space visualization of quantum phase transitions by network topology. *Phys. Rev. E*, 100:012304, Jul 2019. doi: 10.1103/PhysRevE.100.012304.

[11] A.A Bagrov, M Danilov, S Brener, M Harland, A I Lichtenstein, and M.I Katsnelson. Detecting quantum critical points in the tt fermi-hubbard model via complex network theory. *Scientific reports*, 10(1):1–9, 2020. ISSN 2045-2322.

[12] Y. Aharonov and D. Bohm. Significance of electromagnetic potentials in the quantum theory. *Phys. Rev.*, 115:485–491, Aug 1959. doi: 10.1103/PhysRev.115.485.

[13] Charles A Stafford, David M Cardamone, and Sumit Mazumdar. The quantum interference effect transistor. *Nanotechnology*, 18(42):424014, Sep 2007. doi: 10.1088/0957-4484/18/42/424014.

[14] A. A. Michelson and E. W. Morley. On the relative motion of the earth and the luminiferous ether. *American Journal of Science*, s3-34(203):333–345, 11 1887. doi: 10.2475/ajs.s3-34.203.333.

[15] Gianni Pascoli. The sagnac effect and its interpretation by paul langevin. *Comptes Rendus Physique*, 18(9):563–569, 2017. ISSN 1631-0705. doi: https://doi.org/10.1016/j.crhy.2017.10.010. Science in the making: The Comptes rendus de l'Académie des sciences throughout history.

[16] Mauro Faccin, Tomi Johnson, Jacob Biamonte, Sabre Kais, and Piotr Migdał. Degree distribution in quantum walks on complex networks. *Phys. Rev. X*, 3:041007, Oct 2013. doi: 10.1103/PhysRevX.3.041007.

[17] Dawei Lu, Jacob D. Biamonte, Jun Li, Hang Li, Tomi H. Johnson, Ville Bergholm, Mauro Faccin, Zoltán Zimborás, Raymond Laflamme, Jonathan Baugh, and Seth Lloyd. Chiral quantum walks. *Physical review. A*, 93(4), 2016. ISSN 2469-9926.

[18] Mark Goldsmith, Guillermo García-Pérez, Joonas Malmi, Matteo A C Rossi, Harto Saarinen, and Sabrina Maniscalco. Link prediction with continuous-time classical and quantum walks. *arXiv.org*, 2022. ISSN 2331-8422.

[19] Yang-Yu Liu, Jean-Jacques Slotine, and Albert-László Barabási. Observability of complex systems. *Proceedings of the National Academy of Sciences - PNAS*, 110(7):2460–2465, 2013. ISSN 0027-8424.

[20] Arthur N Montanari, Chao Duan, Luis A Aguirre, and Adilson E Motter. Functional observability and target state estimation in large-scale networks. *Proceedings of the National Academy of Sciences - PNAS*, 119(1):1–, 2022. ISSN 0027-8424.

[21] Paul Scherz and Dr. Simon Monk. *Theory*, chapter 2, pages 2–268. Practical Electronics for Inventors, Third Edition. McGraw-Hill Education, New York, 3rd edition edition, 2013. ISBN 9780071771337.

[22] N.N. HANCOCK. Chapter 3 - application of matrix algebra to static electrical networks. In N.N. HANCOCK, editor, *Matrix Analysis of Electrical Machinery (Second Edition)*, pages 21–36. Pergamon, second edition edition, 1974. ISBN 978-0-08-017899-8. doi: https://doi.org/10.1016/B978-0-08-017899-8.50007-X.

[23] Herbert S Wilf. *Generatingfunctionology*. Academic Press, 2014. ISBN 9780127519555.

[24] Joy Morris. *Generating functions*, pages 61–68. Joy Morris, Lethbridge, Alberta, version 2.1 edition, 2022.

[25] Charles M. Grinstead. *Generating functions*, pages 365–404. American Mathematical Society, 1997. ISBN 1-306-36906-1.

[26] Mireille Bousquet-Mélou and Marko Petkovšek. Linear recurrences with constant coefficients: the multivariate case. *Discrete Mathematics*, 225(1):51–75, 2000. ISSN 0012-365X. doi: https://doi.org/10.1016/S0012-365X(00)00147-3. FPSAC'98.

[27] Robin Pemantle and Mark C. Wilson. Twenty combinatorial examples of asymptotics derived from multivariate generating functions. *SIAM Review*, 50(2):199–272, 2008. doi: 10.1137/050643866.

[28] N L Zamarashkin and E E Tyrtyshnikov. Distribution of eigenvalues and singular values of toeplitz matrices under weakened conditions on the generating function. *Sbornik. Mathematics*, 188(7-8): 1191–1201, 1997. ISSN 1064-5616.

[29] N. L. Zamarashkin, E. E. Tyrtyshnikov, and V. N. Chugunov. Functions generating normal toeplitz matrices. *Mathematical Notes*, 89(3-4):480–483, 2011. ISSN 0001-4346.

[30] Richard P. Stanley. *The Transfer-matrix method*, pages 241–262. Enumerative Combinatorics. Wadsworth & Brooks/Cole Advanced Books & Software, Belmont, CA, 1st edition edition, 1986. ISBN 978-1-4615-9765-0.

[31] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393: 440–442, June 1998. doi: https://doi.org/10.1038/30918.

[32] David Poole. *Linear Algebra: A Modern Introduction*. Cengage, Mason, OH, 2014. ISBN 1285463242.

[33] John R. Taylor. *Chapter 3: Propagation of Uncertainties*, page 45–91. University Science Books, 2nd edition, 1997. ISBN 9780935702750.

[34] Bin Zhang and Steve Horvath. A general framework for weighted gene co-expression network analysis. *Statistical Applications in Genetics and Molecular Biology*, 4, 2005. doi: 10.2202/1544-6115.1128.

[35] Tore Opsahl, Filip Agneessens, and John Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, 32(3):245–251, 2010. ISSN 0378-8733. doi: https://doi.org/10.1016/j.socnet.2010.03.006.

[36] Giorgio Fagiolo. Clustering in complex directed networks. *Physical Review E*, 76:026107, August 2007. doi: https://doi.org/10.1103/PhysRevE.76.026107.

[37] Jari Saramaki, Mikko Kivela, Jukka-Pekka Onnela, Kimmo Kaski, and Janos Kertesz. Generalizations of the clustering coefficient to weighted complex networks. *Physical Review E*, 75:1–4, February 2007. doi: https://doi.org/10.1103/PhysRevE.75.027105.

[38] A. Perot and C. Fabry. On the application of the interference phenomena to the solution of various problems of spectroscopy and metrology. *Bulletin Astronomique*, 16:87–115, January 1899. doi: 10.1086/140557.

[39] Larisa Beilina, Evgenii Karchevskii, and Mikhail Karchevskii. *Chapter 6: Vector and Matrix Norms*, pages 209–229. Springer International Publishing, Cham, 2017. ISBN 3319573020.

[40] Mark D. Humphries and Kevin Gurney. Network 'small-world-ness': A quantitative method for determining canonical network equivalence. *PLoS ONE*, 3, April 2008. doi: https://doi.org/10.1371/journal.pone.0002051.

[41] Béla Bollobás. The diameter of random graphs. *Transactions of the American Mathematical Society*, 267(1):41–52, 1981. doi: 10.1090/s0002-9947-1981-0621971-7.

[42] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science (American Association for the Advancement of Science)*, 286(5439):509–512, 1999. ISSN 0036-8075.

[43] Lucas Bottcher and Mason Porter. Complex networks with complex weights. *arXiv*, December 2022. URL https://arxiv.org/abs/2212.06257.

[44] Jukka-Pekka Onnela, Jari Saramäki, János Kertész, and Kimmo Kaski. Intensity and coherence of motifs in weighted complex networks. *Physical Review E 2005-jun 13 vol. 71 iss. 6*, 71, jun 2005. doi: 10.1103/PhysRevE.71.065103.

[45] Tao Wen and Kang Hao Cheong. The fractal dimension of complex networks: A review. *Information Fusion*, 73:87–102, 2021. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2021.02.001.

[46] K.-I. Goh, G. Salvi, B. Kahng, and D. Kim. Skeleton and fractal scaling in complex networks. *Phys. Rev. Lett.*, 96:018701, Jan 2006. doi: 10.1103/PhysRevLett.96.018701.

[47] Bogdan Nica. *A Brief Introduction to Spectral Graph Theory*. EMS Press, may 2018. doi: 10.4171/188.

[48] M Randic, Xiaofeng Guo, T Oxley, and H Krishnapriyan. Wiener matrix: Source of novel graph invariants. *Journal of chemical information and computer sciences*, 33(5):709–716, 1993. ISSN 0095-2338.

[49] Terry Rudolph. Constructing physically intuitive graph invariants. *arXiv.org*, 2002. ISSN 2331-8422.

[50] Martin Grohe and Pascal Schweitzer. The graph isomorphism problem. *Commun. ACM*, 63(11): 128–134, oct 2020. ISSN 0001-0782. doi: 10.1145/3372123.

APPENDIX A

SMALL-WORLD EFFECT PLOTS ACCOUNTING FOR SELF LOOPS

As discussed in Chapter 3, the interferometric clustering can be slightly modified to account for self-loops. There are reasons to make this inclusion, but it breaks with tradition, so I decided to exclude self loops in the main results. However, as I will show with these plots, the results hardly change at all from those in Chapter 5 when I make this inclusion. Figure A.1 recreates Figure 5.2, Figure A.2 recreates Figure 5.3, Figure A.3 recreates Figure 5.4, and Figure A.4 is the counterpart to Figure 5.5.

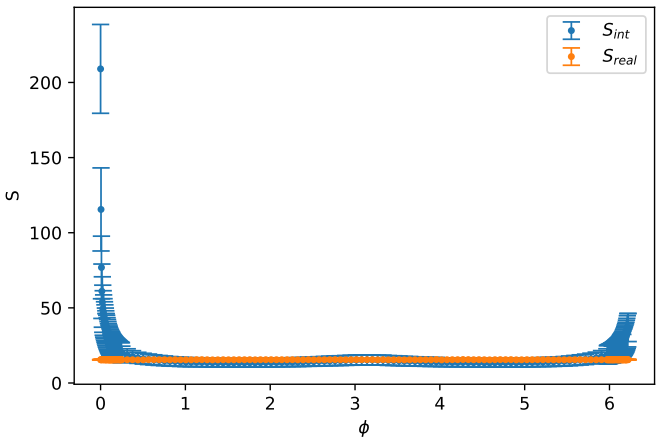

Figure A.1 Peak S_{int} values for each configuration in ϕ. The traditional small world coefficient S (per Equation 5.2) is plotted for reference; it is constant because it does not depend on phase. This plot shows results for small-world interferometers with size $N = 500$ and $k = 12$. This version of the plot was created with a version of the clustering coefficient that accounts for possible self loops.

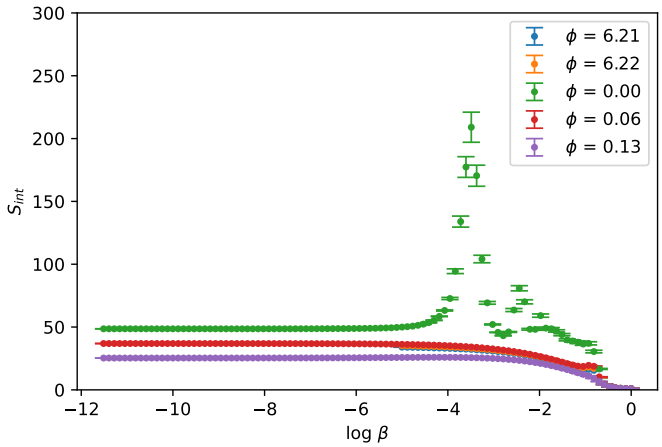

Figure A.2 The interferometric small-world coefficient plotted over β for a few select values of ϕ. This plot is for small-world interferometers with size $N = 500$ and $k = 12$. This version of the plot was created with a version of the clustering coefficient that accounts for possible self loops.

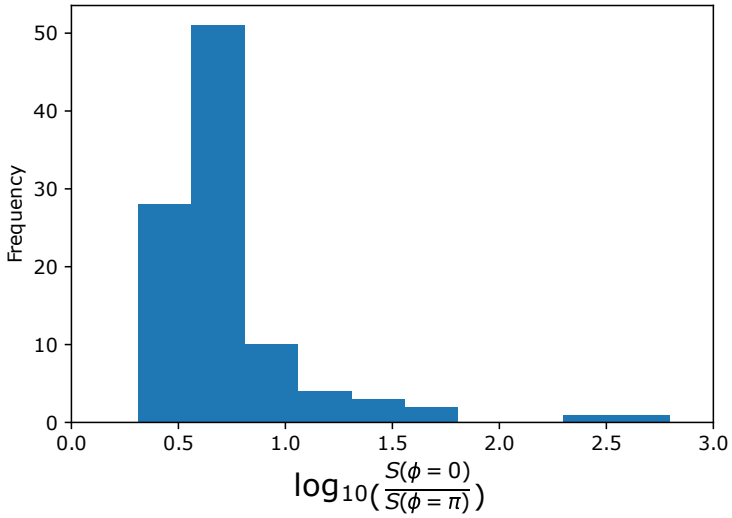

Figure A.3 Histogram of the logarithms of the ratio $S_{int}(\phi = 0)/S_{int}(\phi = \pi)$. Notice thatbecause logarithms are always greater than zero, $S_{int}(\phi = 0) > S_{int}(\phi = \pi)$ for all tests. This version of the plot was created with a version of the clustering coefficient that accounts for possible self loops.

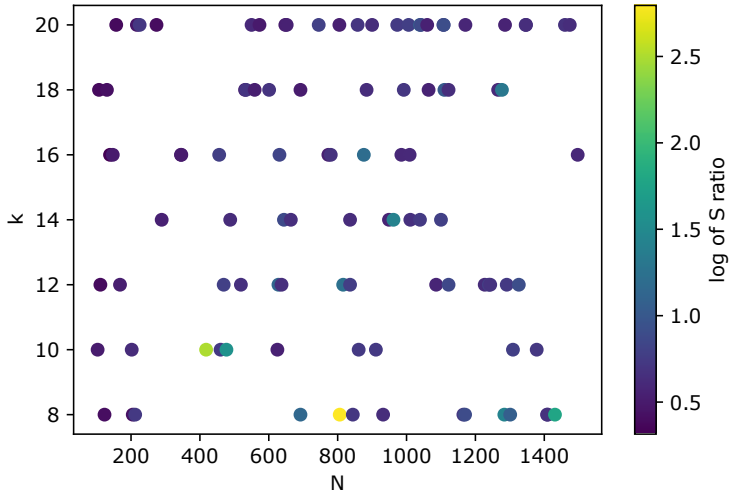

Figure A.4 A scatter plot highlighting the randomized N and k values used to produce the data in Figure A.3. The results of the trials (the ratio of small-world coefficients) is indicated by a color map.

APPENDIX B

NETWORK ALGORITHMS MODULE

This program includes the basic network methods for the small-world interferometer work. It generates small-world interferometers and defines functions for computing path length and clustering measures. This program makes calculations efficient through two primary methods: vectorization and sparse matrices. The code uses vectorized calculations on adjacency matrices through the numpy library, which runs repeated calculations through efficient, low-level algorithms in the c language. In addition, scipy sparse matrices are used wherever matrices are anticipated to have few entries. This speeds up calculations, because sparse matrices simply ignore empty entries instead of explicitly storing and calculating on zeroes. The exception to this is the apparent path strength, due to reasons discussed in Chapter 5.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Nets
Benjamin Krawciw
9/29/22

The necessary network theory functions for performing the Interferometer Small-
World Tests
"""

#Library imports
import numpy as np
import numpy.ma as ma
import scipy.linalg as sp
import scipy.sparse as sparse
from time import time

#Create a random number generator
rng = np.random.default_rng(int(time()))
```

```
#Tolerance for numerical things
TOL = 1e-8

#Creates a ring network
def makeRing(N, khalf):
    #Ensure that khalf is a valid input
    if khalf <= (N // 2):
        khalf_checked = khalf // 1
    else:
        khalf_checked = N // 2

    #Function that determines if entry (i, j) is included in ring
    def ringLogic(i, j):
        dist = np.minimum((j - i) % N, (i - j) % N)
        far_enough = np.greater(dist, 0)
        close_enough = np.less_equal(dist, khalf_checked)
        return(np.logical_and(far_enough, close_enough))

    #Unweighted adjacency matrix
    A = np.fromfunction(ringLogic, (N, N))

    return(A)

#Function that randomly rewires ring according to beta
def rewire(A, beta):
    N = (A.shape)[0]

    #Select edges to rewire
    B = (np.less_equal(rng.random((N, N)), beta)) * A

    #Do not rewire self edges
    B = (np.ones((N, N)) - np.eye(N)) * B
```

```
    #Remove self-edges from rewire pool
    candidates = ma.masked_array(B, mask = np.eye(N))

    #Shuffle candidates
    choices = rng.permuted(candidates[~candidates.mask], axis = 0)
    Shuff = np.zeros_like(B)
    Shuff[~candidates.mask] = choices

    #Generate the rewired matrix
    return(A.astype(int) - B.astype(int) + Shuff.astype(int))



#Returns the adjacency matrix of a small-world network with the given parameters
def ws(params):
    #Unpack parameters
    N, khalf, beta, phi, weighting = (params[0], params[1], params[2],
                                      params[3], params[4])

    #Generate ring
    ring = makeRing(N, khalf)

    #Rewire ring
    rewired = rewire(ring, beta)

    #Add phase
    w = (weighting * (1.0 /  khalf)) * np.exp(1.0j * phi)
    W = rewired.astype(complex) * w

    #Return adjacency matrix as a sparse matrix
    return(sparse.csr_matrix(W))

#General clutering formula
```

```
def __clustHelper(W):
    #These functions only work for dense matrices
    Wsparse = W
    if not(sparse.issparse(W)):
        Wsparse = sparse.csr_matrix(W)

    #Record the network size
    N = np.min(W.shape)

    #Compute the numerator
    num = np.zeros(N)
    with np.nditer(num, op_flags = ['writeonly'], flags = ['f_index']) as iterator:
        for entry in iterator:
            #Record index
            i = iterator.index
            #Compare paths through node i to shortcuts
            throughPaths = Wsparse[:, i] * Wsparse[i, :]
            adjShortcuts = np.abs(throughPaths + Wsparse) - np.abs(throughPaths)
            mat = (abs(throughPaths)).multiply(adjShortcuts)
            #Sum all triads on node i
            norm = mat.sum()
            entry[...] = norm

    #Compute the denominator
    den = np.zeros(N)
    with np.nditer(den, op_flags = ['writeonly'], flags = ['f_index']) as iterator:
        for entry in iterator:
            #Record index
            i = iterator.index
            #Compare paths through node i to path of 1
            throughPaths = Wsparse[:, i] * Wsparse[i, :]
            adjShortcuts = np.ones(N)
            adjShortcuts[i] = 0 #Do not anticipate self-loops
```

```
            mat = np.abs(throughPaths)
            #Sum and record
            norm = mat.sum()
            entry[...] = norm

    #Meshing vector, scaled and shifted to match unweighted clustering
    Mvec = np.divide(num, den + TOL, out=np.zeros_like(num), where = den != 0.0)
    avg = np.mean(Mvec)
    return avg

#Returns the traditional, real-valued clustering coefficient
def Creal(W):
    return __clustHelper(np.abs(W))

#Returns the magnitude of the interferometric meshing
def Mesh(W):
    return __clustHelper(W)

#Returns the real-valued strongest path strength
def SPL(W):
    #Convert multiplicative path lengths to additive ones by taking a logarithm
    Wlog = sparse.csr_matrix((-np.log(np.abs(W.data)),
                              W.indices,
                              W.indptr))

    #Ensure the matrix is square
    wlogShape = Wlog.get_shape()
    newSize = min(wlogShape[0], wlogShape[1])
    #print(Wlog.get_shape())
    Wlog.resize(newSize, newSize)

    #Use Dijkstra's algorithm to solve for shortest paths in log space
    logShorts = sparse.csgraph.shortest_path(csgraph = Wlog)
```

```
    #Return the average
    SPL = np.mean(logShorts, (0, 1), where = (np.abs(logShorts) != np.inf))
    return(SPL)

#Returns the magnitude of the apparent path strength
def APL(W):
    #This measure comes from treating the network as an interferometer
    #This involves solving for the node-signal-value vector E
    #A constant-source-to-node vector S is also supplied
    #E is the solution to WE + S = E
    #So, the apparent paths are the entries of the MP inverse of (I-W)
    N = (W.shape)[0]
    WI = sparse.eye(N, format = "csr") - W

    #Ensure the matrix is square
    wShape = WI.get_shape()
    newSize = min(wShape[0], wShape[1])
    WI.resize((newSize, newSize))

    #Compute matrix inverse
    P = sp.inv(np.nan_to_num(WI.todense()))
    #print(P @ WI)
    APL = np.mean(-np.log(np.abs(P) + TOL), (0, 1))
    return(APL)

'''
#Testing the functions
ring = makeRing(100, 18)
#print(ring)
rewired = rewire(ring, 0.1)
#print(rewired)
W = ws([100, 8, 1.0, np.pi / 4])
#print(W.diagonal())
```

```
#print(W)
Cr = Creal(W)
print(Cr)
M = Mesh(W)
print(M)
ap = APL(W)
#print(ap)
sps = SPL(W)
#print(sps)
'''
```

APPENDIX C

CODE THAT RUNS INTERFEROMETER TESTS ON HPC

This program creates the list of tests to run (called the parameter space, or pspace for short), defines a function for running a network trial, runs the trials in parallel through MPI, and records the results in a comma-separated-values file. If one wished to implement this program in serial on a regular computer, they would only need to replace the MPIPoolExecutor with a for loop. However, running this program in serial would take a very long time.

Similar programs to this one run the supplemental tests for interferometers near $\phi = 0$ and $\beta = 1$, and also the randomized trials depicted in Figure 5.4 and Figure 5.5. Because they are so similar, they are not included in this thesis. However, all code is publicly available on https://github.com/bkrawciw-mines/IntNets_Wendian.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Interferometer Small-World Tests
Benjamin Krawciw
9/30/2022

Creates a suite of watts-strogatz interferometers, computes their network
measures, and saves them using an HPC cluster.
"""

#Library imports
import numpy as np
import nets
import csv
import itertools
from mpi4py.futures import MPIPoolExecutor
from time import time
```

```
#Name for the output file
outFileName = 'full500.csv'

#Create the parameter space for testing
Nrange = [500]
#Redundancy for tighter statistics
redundancy = 100
Ns = []
#Small networks need more trials
for Nval in Nrange:
    reps = redundancy * ((Nrange[-1]) // (Nval))
    [Ns.append(Nval) for i in range(reps)]
Ns = np.array(Ns)
kRange = [6]
#Create a beta space
betaRange = np.logspace(-5.0, 0.0, num = 100, base = 10, endpoint = True)
#Create a phi space
phiRange = np.linspace(0.0, 2.0 * np.pi, num = 100, endpoint = False)
weighting = [0.95]
#Total parameter space for tests
pSpace = itertools.product(Ns, kRange, betaRange, phiRange, weighting)

#Function for each independent test
def Stest(params):
    #print('sTest', params)
    #Create a WS interferometer
    W = nets.ws(params)

    #Calculate small-world coefficients
    C, M = nets.Creal(W), nets.Mesh(W)
    SPL, APL = nets.SPL(W), nets.APL(W)

    #print(psutil.virtual_memory().used)
```

```
    return(params + (C, M, SPL, APL))


#Run all tests
if __name__ == '__main__':
    #Start a timer
    tStart = time()

    #Run an MPI executor on the available nodes
    with MPIPoolExecutor() as executor:
        #Use the executor to run the Stest process on parameters in pSpace
        results = executor.map(Stest, pSpace)

        #Creating the file to log data
        with open(outFileName, 'w', newline= '' ) as csvFile:
            writer = csv.writer(csvFile)
            writer.writerow(['N', 'kHalf', 'beta', 'phi', 'weighting', 'C', 'M', 'SPL', 'APL'])
            writer.writerows(results)

        executor.shutdown(wait = False)

    #Report the execution time
    print("Tests completed. Time: %f s" % (time() - tStart))
```

APPENDIX D

DATA VISUALIZATION CODE

This program takes in the data generated by the code in Appendix C, calculates the small-world coefficients, and creates the plots in Figure 5.3 and Figure 5.2. Similar code plots the results of the randomized-parameter trials, which is not included here, but is included at https://github.com/bkrawciw-mines/IntNets_Wendian.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Data Visualization
Benjamin Krawciw
10/3/2022

This program plots the results for the small-world interferometer tests
"""

#Library imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats #For filtering outliers

#Numerical tolerance
TOL = 1e-8

#Reading in the CSV data
inFileName = 'full500.csv'
dataFrame = pd.read_csv(inFileName, delimiter = ',')
#Including data patches (supplemental data in sensitive areas)
betaPatch = 'betaPatch_redundancy.csv'
```

```
betaPatchFrame = pd.read_csv(betaPatch, delimiter = ',')
dataFrame = pd.concat((dataFrame, betaPatchFrame))
phiPatch = 'phiPatch.csv'
phiPatchFrame = pd.read_csv(phiPatch, delimiter = ',')
dataFrame = pd.concat((dataFrame, phiPatchFrame))
#Treat infinite values as invalid
dataFrame.replace([np.inf, -np.inf], np.nan, inplace = True)
#Find outliers in APL and M
MzScore = np.abs(stats.zscore(dataFrame['M']))
APLzScore = np.abs(stats.zscore(dataFrame['APL'], nan_policy='omit'))
dataFrame = dataFrame.loc[(APLzScore < 3) & (MzScore < 3)]

#Group trials by input parameters, then compute means
sortNets = dataFrame.groupby(['N', 'kHalf', 'beta', 'phi', 'weighting'],
                             as_index = False)
#Using a more stable version of the average to handle volatility in APL
measures = sortNets.mean()
measuresErrs = sortNets.std()
numTests = sortNets.count()

#Case study: How do N and kHalf change things?
#Select a particular beta and phi value
NkData = measures.loc[(measures['beta'] == max(measures['beta']))
                      & (measures['phi'] == 0)]

NkN, Nkk = NkData['N'].to_numpy(), NkData['kHalf'].to_numpy()
NkM, NkSPL = NkData['M'].to_numpy(), NkData['SPL'].to_numpy()
NkAPL = NkData['APL'].to_numpy()

fig, ax = plt.subplots()
scat = ax.scatter(NkN, Nkk, c = NkM, s = 100)
ax.set(
        title = "Clustering over kHalf and N",
```

```
        xlabel = 'N',
        ylabel = 'kHalf'
        )
fig.colorbar(scat, label = 'Clustering')
fig.show()

fig, ax = plt.subplots()
scat = ax.scatter(NkN, Nkk, c = NkSPL, s = 100)
ax.set(
        title = "SPL over kHalf and N",
        xlabel = 'N',
        ylabel = 'kHalf'
        )
fig.colorbar(scat, label = 'SPL')
fig.show()

fig, ax = plt.subplots()
scat = ax.scatter(NkN, Nkk, c = NkAPL, s = 100)
ax.set(
        title = "APL over kHalf and N",
        xlabel = 'N',
        ylabel = 'kHalf'
        )
fig.colorbar(scat, label = 'SPL')
fig.show()



#Case study: How do beta and phi change things?
bpData = measures.loc[(measures['N'] == max(measures['N']))
                      & (measures['kHalf'] == 6)]
bpb, bpp = bpData['beta'].to_numpy(), bpData['phi'].to_numpy()
bpM, bpSPL = bpData['M'].to_numpy(), bpData['SPL'].to_numpy()
bpAPL = bpData['APL'].to_numpy()
```

```
fig, ax = plt.subplots()
scat = ax.scatter(np.log(bpb), bpp, c = bpM, s = 100)
ax.set(
        title = "Clustering over beta and phi",
        xlabel = 'log(beta)',
        ylabel = 'phi'
        )
fig.colorbar(scat, label = 'Clustering')
fig.show()

fig, ax = plt.subplots()
scat = ax.scatter(np.log(bpb), bpp, c = bpSPL, s = 100)
ax.set(
        title = "SPL over beta and phi",
        xlabel = 'log(beta)',
        ylabel = 'phi'
        )
fig.colorbar(scat, label = 'SPL')
fig.show()

fig, ax = plt.subplots()
scat = ax.scatter(np.log(bpb), bpp, c = bpAPL, s = 100)
ax.set(
        title = "APL over beta and phi",
        xlabel = 'log(beta)',
        ylabel = 'phi'
        )
fig.colorbar(scat, label = 'APL')

#Recreating previous results
SmaxDat = measures.loc[(measures['N'] == max(measures['N']))
                       & (measures['kHalf'] == 6)]
SmaxErrs = measuresErrs.loc[(measuresErrs['N'] == max(measures['N']))
```

```
                    & (measuresErrs['kHalf'] == 6)]
SmaxCounts = numTests.loc[(measuresErrs['N'] == max(measures['N']))
                    & (measuresErrs['kHalf'] == 6)]
M = SmaxDat['M']
SPL = SmaxDat['SPL']
APL = SmaxDat['APL']

#Create set of random baselines
randDat = SmaxDat.loc[SmaxDat['beta'] == 1.0]
randM = M.loc[(SmaxDat['beta'] == 1.0)]
randSPL = SPL.loc[(SmaxDat['beta'] == 1.0)]
randAPL = APL.loc[(SmaxDat['beta'] == 1.0)]

#Function to compute gamma and lambda for a single config (real and complex)
def GammaLambda(row):

    #Unpack row
    N, kHalf, beta, phi, weighting, C, M, SPL, APL = row



    #Reference row
    ref = measures.loc[
        (measures['N'] == N) &
        (measures['kHalf'] == kHalf) &
        (measures['beta'] == 1.0) &
        (measures['phi'] == phi)
        ].to_numpy()[0]
    refN, refK, refB, refPhi, refWeight, refC, refM, refSPL, refAPL = ref

    #Create output row
    out = np.array([
        N,
        kHalf,
```

```
        beta,
        phi,
        weighting,
        C / refC,
        (M + np.abs(refM) - refM) / (np.abs(refM) + TOL),
        SPL / refSPL,
        (APL + np.abs(refAPL) - refAPL) / (np.abs(refAPL) + TOL)
        ])

    return(out)

#Compute gammas and lambdas
GamLam = SmaxDat.apply(GammaLambda, axis = 'columns', raw = True,
                       result_type = 'expand')

#Compute small-world coefficients for a single config
def Scomp(gammaLamRow):
    #Unpack row
    N, kHalf, beta, phi, weighting, GamReal, GamComp, LambReal, LambComp = gammaLamRow
    #Create output row
    out = pd.Series([N, kHalf, beta, phi, weighting,
                     GamReal / (LambReal + TOL),
                     GamComp / (LambComp + TOL)])

    return(out)

#Computing small-world coefficients
Svals = GamLam.agg(Scomp, axis = 'columns')

#Relabeling svals
Svals = pd.DataFrame(Svals.values,
                     columns = ['N', 'kHalf', 'beta', 'phi', 'weighting',
                                'Sreal', 'Scomp'])
```

```
#Find maximum over beta for each phi
Smax = Svals.groupby('phi', as_index = False).max()

#Characterize S over beta
SmaxBeta = Svals.groupby(['N', 'kHalf', 'phi'], as_index = False).max()
SmaxBetaErr = Svals.groupby(['N', 'kHalf', 'phi'], as_index = False).std()

#Plot S over beta for a few interesting places
plt.figure()
#Identify all unique phi values
phis = Svals['phi'].unique()
for i in [-2, -1, 0, 1, 2]:
    index = i
    phi = phis[index]
    pDat = Svals.loc[Svals['phi'] == phi]
    #Estimate swcoeff error as the product of relative C and P errors times S.
    Mdat = M.loc[Svals['phi'] == phi]
    Pdat = APL.loc[Svals['phi'] == phi]
    Cerr = (SmaxErrs.loc[SmaxErrs['phi'] == phi])['M'] / Mdat
    Perr = (SmaxErrs.loc[SmaxErrs['phi'] == phi])['APL'] / Pdat
    pDatErr = pDat['Scomp'] * np.sqrt(Cerr**2 + Perr**2)
    #Divide by sqrt(numTests) to get standard dev of the mean
    nTests = min(SmaxCounts.loc[SmaxCounts['phi']==phi]['M'])
    pDatErr = pDatErr / np.sqrt(nTests)
    betax = np.log(pDat['beta'].to_numpy())
    plt.errorbar(betax, pDat['Scomp'], pDatErr, label = (r"$\phi$ = %0.2f" %phi),
                 elinewidth=1,
                 capsize=5,
                 marker='.',
                 lw = 0)
plt.legend()
plt.xlabel(r"log $\beta$")
```

```
plt.ylabel(r'$S_{{int}}$')
plt.ylim(0, 300)
plt.savefig("sw_overbeta.pdf")

#Plot Smax over phi
plt.figure()
plt.errorbar(SmaxBeta['phi'], SmaxBeta['Scomp'], SmaxBetaErr['Scomp'], label = r'$S_{int}$',
             elinewidth=1,
             capsize=5,
             marker='.',
             lw = 0)
plt.errorbar(SmaxBeta['phi'], SmaxBeta['Sreal'], 0, label = r'$S_{real}$',
             elinewidth=1,
             capsize=5,
             marker='.',
             lw = 0)
plt.xlabel(r'$\phi$')
plt.ylabel('S')
plt.legend()
plt.savefig("sw_overphi.pdf")
```