UNIVERSITY OF CALIFORNIA,
IRVINE

Diffusion Distance: Efficient Computation and Applications

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Cory Braker Scott

Dissertation Committee:
Prof. Eric Mjolsness, Chair
Prof. Alexander Ihler
Prof. Diane Adele Oyen[1]
Prof. Padhraic Smyth

2021

[1]Los Alamos National Laboratory

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACKNOWLEDGMENTS

Writing this dissertation was a monumental effort, and one which would not have been possible without the support of many of my friends, collaborators, and loved ones. Since this acknowledgements page is finite, I can only thank a few of them:

Thanks to my advisor, Eric Mjolsness, for taking a chance on a grad student with no papers to his name; and thanks to Diane Oyen, Steven Janke, Matthew Whitehead, Padhraic Smyth, and Alexander Ihler for all of their advice and academic encouragement.

Thanks to my parents, Thomas Scott and Elizabeth Braker, who have always encouraged me to forge my own path and to approach the world with an inquisitive and compassionate mindset. Thanks as well to my brother, Ben Scott, who I am super proud to call a fellow scientist. Your support means the world to me.

Thanks to my partner, Julia Boese, who has been a bedrock of support at times when my work was trying and difficult.

Finally, thanks to my other friends, partners, and family, who have all encouraged me to grow in ways I couldn't have anticipated over the last few years. This success couldn't have happened without you all.

CURRICULUM VITAE

Cory Braker Scott

EDUCATION

Doctor of Philosophy in Computer Science — **2021**
University of California, Irvine — *Irvine, CA*

Master of Science in Computer Science — **2017**
University of California, Irvine — *Irvine, CA*

Bachelor of Arts in Computer Science — **2013**
Colorado College — *Colorado Springs, CO*

Bachelor of Arts in Mathematics — **2013**
Colorado College — *Colorado Springs, CO*

RESEARCH EXPERIENCE

Graduate Research Assistant — **2015–2021**
University of California, Irvine — *Irvine, California*

Graduate Research Assistant — **2018–2021**
Los Alamos National Labs — *Los Alamos, NM*

Machine Learning Intern — **2017, 2020-2021**
TAE Technologies, Inc — *Rancho Santa Margarita, CA*

Software Engineering Intern — **2016**
Charles River Analytics, Inc — *Cambridge, MA*

REU Participant — **2012**
Boise State University — *Boise, ID*

TEACHING EXPERIENCE

Teaching Assistant — **2015–2021**
University of California, Irvine — *Irvine, CA*
Courses: Introduction to Programming, Introduction to Optimization

Paraprofessional — **2013–2014**
Colorado College — *Colorado Springs, CO*

Teaching Assistant — **2011–2013**
Colorado College — *Colorado Springs, CO*

REFEREED JOURNAL PUBLICATIONS

Graph Diffusion Distance: Properties and Efficient Computation **2021**
PLOS ONE

StressNet - Deep learning to predict stress with fracture propagation in brittle materials **2021**
Nature: Materials Degradation

Graph prolongation convolutional networks: explicitly multiscale machine learning on graphs with applications to modeling of cytoskeleton **2020**
IOP Machine Learning: Science and Technology

Detection and prediction of a beam-driven mode in Field-Reversed Configuration plasma with Recurrent Neural Networks **2020**
Nuclear Fusion

Multilevel Artificial Neural Network Training for Spatially Correlated Learning **2019**
SIAM Journal on Scientific Computing

Algebraic properties of generalized Rijndael-like ciphers. **2014**
Groups Complexity Cryptology

REFEREED CONFERENCE PUBLICATIONS

Physics-Informed Spatiotemporal Deep Learning for Emulating Coupled Dynamical Systems **March 2020**
AAAI Spring Symposium

SOFTWARE

DiffusionDistance `https://github.com/scottcb/DiffusionDistance`
Collection of codes to calculate diffusion distance between graph Laplacians.

MsANN `https://github.com/scottcb/MsANN`
A machine learning model which learns at multiple spatial scales.

ABSTRACT OF THE DISSERTATION

Diffusion Distance: Efficient Computation and Applications

By

Cory Braker Scott

Doctor of Philosophy in Computer Science

University of California, Irvine, 2021

Prof. Eric Mjolsness, Chair

How is the shape of a graph captured by the way heat diffuses between its nodes? The Laplacian Exponential Kernel of a graph is a matrix whose eigenvalues and eigenvectors describe this heat (or more generally, probability) diffusion process as a function of time. Previous work has shown that the Laplacian can be gainfully used for comparing graphs, but these methods are limited to graphs of the same size. This work focuses on generalizing one such measure, Graph Diffusion Distance (GDD), making it capable of comparing graphs of varying size. Calculating these distances involves solving a complicated multivariate optimization problem, and we will detail a novel optimization algorithm for doing so. This procedure outperforms naive univariate optimization by a speedup of as much as 1000x. One key feature of this procedure is that it produces a coarsening operator which attempts to align the two heat kernels to agree with each other as much as possible. These operators can be used as the coarsening step in a convolutional neural network, resulting in a 10x increase in training efficiency. We will show how these "Graph Prolongation Convolutional Networks" can be used to accelerate molecular dynamics simulations of proteins. Finally, we will also discuss some applications of the GDD, including 2D and 3D shape analysis and characterization of plant cell growth.

Chapter 1

Introduction

1.1 Introduction

Structure comparison, as well as structure summarization, is a ubiquitous problem, appearing across multiple scientific disciplines. In particular, many scientific problems (e.g. inference of molecular properties from structure, pattern matching in data point clouds and scientific images) may be reduced to the problem of inexact graph matching: given two graphs, compute a measure of similarity that gainfully captures structural correspondence between the two. Similarly, many algorithms for addressing multiple scales of dynamical behavior rely on methods for automatically coarsening the computational graph associated with some model architecture.

In this work we present a distance metric for undirected graphs, based on the Laplacian exponential kernel. This measure generalizes the work of Hammond et al. [48] on graph diffusion distance for graphs of equal size; crucially, our distance measure allows for graphs of inequal size. We formulate the distance measure as the solution to an optimization problem dependent on a comparison of the two graph Laplacians. This problem is a nested opti-

mization problem, with the innermost layer consisting of multivariate optimization subject to matrix constraints (e.g. orthogonality). To compute this dissimilarity score efficiently, we also develop and demonstrate the lower computational cost of an algorithm which calculates upper bounds on the distance. This algorithm finds a prolongation/restriction operator, P, which produces an optimally coarsened version of the Laplacian matrix of a graph. Prolongation/restriction operators produced via the method in this paper can be used to accelerate the training of neural networks (both flat ANNs, as we will see in Chapter 6, and graph neural networks, as we will see in Chapter 7).

1.2 Prior Work

Quantitative measures of similarity or dissimilarity between graphs have been studied for decades owing to their relevance for problems in pattern recognition including structure-based recognition of extended and compound objects in computer vision, prediction of chemical similarity based on shared molecular structure, and many other domains. Related problems arise in quantitative modeling, for example in meshed discretizations of partial differential equations and more recently in trainable statistical models of data that feature graph-like models of connectivity such as Bayes Networks, Markov Random Fields, and artificial neural networks. A core problem is to define and compute how "similar" two graphs are in a way that is invariant to a permutation of the vertices of either graph, so that the answer doesn't depend on an arbitrary numbering of the vertices. On the other hand unlike an arbitrary numbering, problem-derived semantic *labels* on graph vertices may express real aspects of a problem domain and may be fair game for detecting graph similarity (we explore the use of edge information in Section 8.2. The most difficult case occurs when such labels are absent, for example in an unstructured mesh, as we shall assume. Here we detail several measures of graph dissimilarity, chosen by historical significance and similarity to our

measure.

We mention just a few prior works to give an overview of the development of graph distance measures over time, paying special attention to those which share theoretical or algorithmic characteristics with the measure we introduce. Our mathematical distinctions concern the following properties:

- Does the distance measure require an inner optimization loop? If so is it mainly a discrete or continuous optimization formulation?
- Does the distance measure calculation naturally yield some kind of explicit *map* from real-valued functions on vertices of one graph to functions on vertices of the other? (A map from vertices to vertices would be a special case.) If we use the term "graph signal" to mean a function $f : V(G_1) \to S$ which identifies each vertex of a graph G_1 with some state $s \in S$, then a map-explicit graph distance is one which as part of its output provides a new function $f' : V(G_2) \to S$ which approximates the behavior of f. 'Approximates' and 'behavior' are here left undefined as these would need to be problem-specific.
- Is the distance metric definable on the spectrum of the graph alone, without regard to other data from the same graph? The "spectrum" of a graph is a graph invariant calculated as the eigenvalues of a matrix related to the adjacency matrix of the graph. Depending on context, the spectrum can refer to eigenvalues of the adjacency matrix, graph Laplacian, or normalized graph Laplacian of a graph. We will usually take the underlying matrix to be the graph Laplacian, defined in detail in Section 1.4.2. Alternatively, does it take into account more detailed "structural" aspects of the graph? This categorization (structural vs. spectral) is similar to that introduced in [28].

For each of the graph distance variants discussed here, we label them according to the

above taxonomy. For example, the two prior works by Eschera et. al. and Hammond et al (discussed in Sections 1.2.4 and 1.2.5) would be labelled as (structural, explicit, disc-opt) and (spectral, implicit, non-opt), respectively. Our distance measure[1] defined in detail in Chapter 2 would be labelled (spectral, explicit, cont-opt).

1.2.1 Quadratic Matching of Points and Graphs (structural, explicit, cont-opt)

As a first example, some graph comparison methods focus on the construction of a point-to-point correspondence between the vertices of two graphs. Gold et. al. [41] define the dissimilarity between two unlabelled weighted graphs (with adjacency matrices $A^{(1)}$ and $A^{(2)}$ and n_1 and n_2 vertices, respectively) as the solution to the following optimization problem (for real-valued $M = [m_{ij}]$:

$$\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{n_2}\sum_{k=1}^{n_1}\left(\sum_{l=1}^{n_2} A^{(1)}_{jl} m_{lk} - \sum_{p=1}^{n_1} m_{jp} A^{(2)}_{pk}\right)^2 = \left|\left|A^{(1)}M - MA^{(2)}\right|\right|_F^2 \\
\text{subject to} \quad & \sum_{i=1}^{n_2} m_{ij} = 1, && j = 1 \dots n_1 \\
& \sum_{j=1}^{n_1} m_{ij} = 1, && i = 1 \dots n_2 \\
& m_{ij} \geq 0 && i = 1 \dots n_2 \\
& && j = 1 \dots n_1
\end{aligned} \tag{1.1}$$

where $||\cdot||_F^2$ is the squared Frobenius norm. This problem is similar in structure to the optimization considered in Section 2.4 and Chapter 4: a key difference being that Gold et al. consider optimization over real-valued matchings between graph vertices, whereas we

[1] with the exception of the sparsity-seeking variants, which are not spectral.

consider 0-1 valued matchings between the eigenvalues of the graph Laplacians. In [42] and [83] the authors present computational methods for computing the optimum of 1.1, and demonstrate applications of this distance measure to various machine learning tasks such as 2D and 3D point matching, as well as graph clustering. Gold et al. also introduce *softassign*, a method for performing combinatorial optimization with both row and column constraints, similar to those we consider.

1.2.2 Cut-Distance of Graphs (structural, implicit, disc-opt)

Lovász [68] defines the *cut-distance* of a pair of graphs as follows: Let the $\square$-norm of a matrix B be given by:

$$||B||_{\square} = \frac{1}{n^2} \max_{S,T \subseteq 1 \ldots n} \left| \sum_{i \in S, j \in T} B_{ij} \right| \tag{1.2}$$

Given two labelled graphs G_1, G_2, on the same set of vertices, and their adjacency matrices A_1 and A_2, the cut-distance $d_{\text{cut}}(G_1, G_2)$ is then given by

$$D_{\text{cut}}(G_1, G_2) = ||A_1 - A_2||_{\square} \tag{1.3}$$

(for more details, see [68]). Computing this distance requires combinatorial optimization (over all vertex subsets of G_1, G_2) but this optimization does not result in an explicit map between G_1 and G_2. This distance metric is grounded in the theory of *graphons*, mathematical objects which are a natural infinite-sized generalization of dense graphs. However, all sparse graphs are similar in cut-distance to the zero graphon (see [68]), making cut-distance less useful for real-world problems.

1.2.3 Wasserstein Earth Mover Distance (spectral, implicit, disc-opt)

One common metric between graph spectra is the Wasserstein Earth Mover Distance. Most generally, this distance measures the cost of transforming one probability density function into another by moving mass under the curve. If we consider the eigenvalues of a (possibly weighted) graph as point masses, then the EMD measures the distance between the two spectra as the solution to a transport problem (transporting one set of points to the other, subject to constraints e.g. a limit on total distance travelled or a limit on the number of 'agents' moving points). The EMD has been used in the past in various graph clustering and pattern recognition contexts; see [44]. In the above categorization, this is an optimization-based spectral distance measure, but is implicit, since it does not produce a map from vertices of G_1 to those of G_2 (informally, this is because the EMD is not translating one set of eigenvalues into the other, but instead transforming their respective histograms). Recent work applying the EMD to graph classification includes [27] and [71]. Some similar recent works [69, 21] have used optimal transport theory to compare graphs. In this framework, signals on each graph are smoothed, and considered as draws from probability distribution(s) over the set of all graph signals. An optimal transport algorithm is used to find the optimal mapping between the two probability distributions, thereby comparing the two underlying graphs.

1.2.4 Graph-Edit Distance

The graph edit distance measures the total cost of converting one graph into another with a sequence of local edit moves, with each type of move (for example, vertex deletion or addition, edge deletion or addition, edge division or contraction) incurring a specified cost. Costs are chosen to suit the graph analysis problem at hand; determining a cost assignment

which makes the edit distance most instructive for a certain set of graphs is both problem-dependent and an active area of research. The distance measure is then the sum of these costs over an optimal sequence of edits, which must be found using some optimization algorithm i.e. a shortest-path algorithm (the best choice of algorithm may vary, depending on how the costs are chosen). The sequence of edits may or may not (depending on the exact set of allowable edit moves) be adaptable into an explicit map between vertex-sets. Classic pattern recognition literature includes: [31] [32] [37] [88] .

1.2.5 Diffusion Distance due to Hammond et al. [48]

We discuss this recent distance metric more thoroughly below. This distance measures the difference between two graphs as the maximum discrepancy between probability distributions which represent single-particle diffusion beginning from each of the nodes of G_1 and G_2. This distance is computed by comparing the eigenvalues of the heat kernels of the two graphs. The optimization involved in calculating this distance is a simple unimodal optimization over a single scalar, t, representing the passage of time for the diffusion process on the two graphs; hence we do not count this among the "optimization based" methods we consider.

1.2.6 Novel Diffusion-Derived Measures

In this work, we introduce a family of related graph distance measures. These measures compare two graphs in terms of similarity of a set of probability distributions describing single-particle diffusion on each graph. For two graphs G_1 and G_2 with respective Laplacians $L(G_1)$ and $L(G_2)$, the matrices $e^{tL(G_1)}$ and $e^{tL(G_2)}$ are called the *Laplacian Exponential Kernels* of G_1 and G_2 (t is a scalar representing the passage of time). The column vectors of these matrices describe the probability distribution of a single-particle diffusion process

starting from each vertex, after t time has passed. The norm of the difference of these two kernels thus describes how different these two graphs are, from the perspective of single-particle diffusion, at time t. Since these distributions are identical at very-early and very late times t (we formalize this notion in Section 2.1), a natural way to define a graph distance is to take the supremum over all t[2]. When the two graphs are the same size, so are the two kernels, which may therefore be directly compared with a matrix norm. This case is the case considered by Hammond et al. [48]. However, to compare two graphs of different sizes, we need a mapping between the column vectors of $e^{tL(G_1)}$ and $e^{tL(G_2)}$.

One such mapping is optimization over a suitably constrained prolongation/restriction operator between the graph Laplacians of the two graphs. This operator is a permutation-invariant way to compare the behavior of a diffusion process on each. The prolongation map P thus calculated may then be used to map signals (by which we mean values associated with vertices or edges of a graph) on G_1 to the space of signals on G_2 (and vice versa). In Chapters 6 and 7 we implicitly consider the weights of an artificial neural network model to be graph signals, and use these operators to train a hierarchy of linked neural network models.

We also, in sections 3.3 and 3.4 consider a time conversion factor between diffusion on graphs of unequal size, and consider the effect of limiting this optimization to sparse maps between the two graphs (again, our case reduces to Hammond when the graphs in question are the same size, dense P and R matrices are allowed, and our time-scaling parameter is set to 1).

In this work, we present an algorithm for computing the type of nested optimization given in our definition of distance (Equations 2.2 and 2.3). The innermost loop of our distance measure optimization consists of a Linear Assignment Problem (LAP, defined below) where the entries of the cost matrix have a nonlinear dependence on some external variable. Our algorithm greatly reduces both the count and size of calls to the external LAP solver. We

[2]We will assume that the two graphs are undirected and each consist of only one component, as otherwise this supremum is not guaranteed to be finite and therefore informative.

use this algorithm to compute an upper bound on our distance measure, but it could also be useful in other similar nested optimization contexts: specifically, nested optimization where the inner loop consists of a linear assignment problem whose costs depend quadratically on the parameter in the outermost loop.

1.3 Outline

The goal of this manuscript is to develop the theory and practice of comparing graphs using Graph Diffusion Distance (GDD). The remainder of this chapter (Chapter 1) defines basic mathematical terminology and framework necessary for the remainder of the work. Chapter 2 defines Graph Diffusion Distance and the variants thereof considered. Efficiently computing these distance metrics requires a novel algorithm, which we motivate and explain in Chapter 4. Chapters 3 and 5 explore theoretical and numeric properties of GDD, respectively. Chapters 6, 7, and 8 showcase several applications of GDD to various scientific tasks. Chapters 6 and 7 in particular are structured as self-contained investigations and may be read without material from Chapters 2-5, although material from Section 1.4 may be necessary for understanding notation.

1.4 Mathematical Background

In this section we briefly define terminology and notation which will be useful in the exposition and proofs to follow.

1.4.1 Desirable Characteristics for Distance Metrics

The ideal for a quantitative measure of similarity or distance on some set S is usually taken to be a distance *metric* $d : S \times S \mapsto \mathbb{R}$ satisfying for all $x, y, z \in S$:

- Non-negativity: $d(x, y) \geq 0$
- Identity: $d(x, y) = 0 \iff x = y$
- Symmetry: $d(x, y) = d(y, x)$
- Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$

Then (S, d) is a *metric space*. Euclidean distance on $\mathbb{R}^d$ and geodesic distance on manifolds satisfy these axioms. They can be used to define algorithms that generalize from $\mathbb{R}^d$ to other spaces. A variety of weakenings of these axioms are required in many applications, by dropping some axioms and/or weakening others. For example if S is a set of nonempty sets of a metric space S_0, one can define the "Hausdorff distance" on S which is an *extended pseudometric* that obeys the triangle inequality but not the Identity axiom and that can take values including $+\infty$. As another example, any measure measure of distance on graphs which is purely spectral (in the taxonomy of Section 1.2) cannot distinguish between graphs which have identical spectra. We discuss this in more detail in Section 2.3.

Additional properties of distance metrics that generalize Euclidean distance may pertain to metric spaces related by Cartesian product, for example, by summing the squares of the distance metrics on the factor spaces. We will consider an analog of this property in Section 3.6.

1.4.2 Definitions

Graph Laplacian: For an undirected graph G with adjacency matrix A and vertex degrees $d_1, d_2 \dots d_n$, we define the Laplacian of the graph as

$$\begin{aligned} L(G) &= A - diag(\{d_1, d_2 \dots d_n\}) \\ &= A - diag(\mathbf{1} \cdot A) \\ &= A(G) - D(G). \end{aligned} \tag{1.4}$$

The eigenvalues of this matrix are referred to as the spectrum of G. See [9, 26] for more details on graph Laplacians and spectral graph theory. $L(G)$ is sometimes instead defined as $D(G) - A(G)$; we take this sign convention for $L(G)$ because it agrees with the standard continuum Laplacian operator, Δ, of a multivariate function f: $\Delta f = \sum_{i=1}^{n} \frac{\delta^2 f}{\delta x_i^2}$.

Frobenius Norm: The squared Frobenius norm, $||A||_F^2$ of a matrix A is given by the sum of squares of matrix entries. This can equivalently be written as $\mathrm{Tr}[A^T A]$.

Linear Assignment Problem (LAP): We take the usual definition of the Linear Assignment Problem (see [18], [19]): we have two lists of items S and R (sometimes referred to as "workers" and "jobs"), and a cost function $c : S \times R \to \mathbb{R}$ which maps pairs of elements from S and R to an associated cost value. This can be written as a linear program for real-valued

x_{ij} as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c(s_i, r_j)x_{ij} \\
\text{subject to} \quad & \sum_{i=1}^{m} x_{ij} \leq 1, \qquad j = 1 \ldots n \\
& \sum_{j=1}^{n} x_{ij} \leq 1, \qquad i = 1 \ldots m \\
& x_{ij} \geq 0 \qquad\quad i = 1 \ldots m, j = 1 \ldots n
\end{aligned} \tag{1.5}$$

Generally, "Linear Assignment Problem" refers to the square version of the problem where $|S| = |R| = n$, and the objective is to allocate the n jobs to n workers such that each worker has exactly one job and vice versa. The case where there are more workers than jobs, or vice versa, is referred to as a Rectangular LAP or RLAP. In practice, the conceptually simplest method for solving an RLAP is to convert it to a LAP by *augmenting* the cost matrix with several columns (rows) of zeros. In this case, solving the RLAP is equivalent to solving a LAP with size $max(n, m)$. Other computational shortcuts exist; see [12] for details. Since the code we use to solve RLAPs takes the augmented cost matrix approach, we do not consider other methods in this paper.

Matching: we refer to a 0-1 matrix M which is the solution of a particular LAP as a "matching". We may refer to the "pairs" or "points" of a matching, by which we mean the pairs of indices (i, j) with $M_{ij} = 1$. We may also say in this case that M "assigns" i to j. Given two matrices A_1 and A_2, and lists of their eigenvalues $\{\lambda_1^{(1)}, \lambda_2^{(1)}, \ldots, \lambda_{n_1}^{(1)}\}$ and $\{\lambda_1^{(2)}, \lambda_2^{(2)}, \ldots, \lambda_{n_2}^{(2)}\}$, with $n_2 \geq n_1$, we define the *minimal eigenvalue matching* $m^*(A_1, A_2)$

as the matrix which is the solution of the following constrained optimization problem:

$$m^*(A_1, A_2) = \arg\inf_M \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} M_{i,j}(\lambda_j^{(1)} - \lambda_i^{(2)})^2 \tag{1.6}$$
$$\text{subject to} \quad \left(M \in \{0,1\}^{n_2 \times n_1}\right) \wedge \left(\sum_{i=1}^{n_2} M_{i,j} = 1\right) \wedge \left(\sum_{j=1}^{n_1} M_{i,j} \leq 1\right)$$

In the case of eigenvalues with multiplicity > 1, there may not be one unique such matrix, in which case we distinguish matrices with identical cost by the lexicographical ordering of their occupied indices and take $m^*(A_1, A_2)$ as the first of those with minimal cost. This matching problem is well-studied and efficient algorithms for solving it exist; we use a Python language implementation [22] of a 1957 algorithm due to Munkres [74]. Additionally, given a way to enumerate the minimal-cost matchings found as solutions to this eigenvalue matching problem, we can perform combinatorial optimization with respect to some other objective function g, in order to find optima of $g(P)$ subject to the constraint that P is a minimal matching.

Hierarchical Graph Sequences: A Hierarchical Graph Sequence (HGS) is a sequence of graphs, indexed by $l \in \mathbb{N} = 0, 1, 2, 3 \ldots$, satisfying the following:

- G_0 is the graph with one vertex and one self-loop, and;
- Successive members of the lineage grow roughly exponentially - that is, there exists some base b such that the growth rate (of nodes) as a function of level number l is $O(b^{l^{1+\epsilon}})$, for all $\epsilon > 0$.

Graded Graph: A graded graph is a graph along with a vertex labelling, where vertices are labelled with non-negative integers such that Δl, the difference in label over any edge, is in $\{-1, 0, 1\}$. We will refer to the $\Delta l = 0$ edges as "within-level" and the $l = \pm 1$ edges as "between-level".

Graph Lineages: A graph lineage is a graded graph with two extra conditions:

- The vertices and edges with $\Delta l = 0$ form a HGS; and
- the vertices and edges with $\Delta l = \pm 1$ form a HGS of bipartite graphs.

More plainly, a graph lineage is an exponentially growing sequence of graphs along with ancestry relationships between nodes. We will also use the term graph lineage to refer to the HGS in the first part of the definition (it will be clear from context which sense we are using). Some intuitive examples of graph lineages in this latter sense are the following:

- Path graphs or cycle graphs of size b^n for any integer b.
- More generally, grid graphs of any dimension d, of side length b, yielding a lineage which grows with size b^{d^n} (with periodic or nonperiodic boundary conditions).
- For any probability distribution $p(x, y)$ whose support is points in the unit square, we can construct a graph by discretizing the map of p as a function of x and y, and interpreting the resulting matrix as the adjacency matrix of a graph. For a specific probability distribution p, the graphs derived this way with discretizations of exponentially increasing bin count form a graph lineage.
- The *triangulated mesh* is a common object in computer graphics [81, 73, 96], representing a discretization of a 2-manifold embedded in R^3. Finer and finer subdivisions of such a mesh constitute a graph lineage.

Several examples of graph lineages are used in the discussion of the numerical properties of Graph Diffusion Distance in Section 5.1. Additional examples (a path graph and a triangulated mesh) can be found in Figures 1.1 and 1.2.

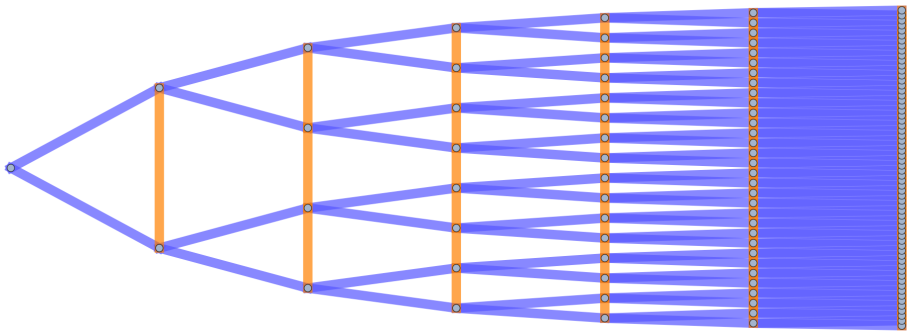

Figure 1.1: The first seven levels of the graph lineage of path graphs, with ancestry relationships. $\Delta l = 0$ edges are colored in orange, $\Delta l = \pm 1$ edges are colored in blue. Self-loops are not illustrated.

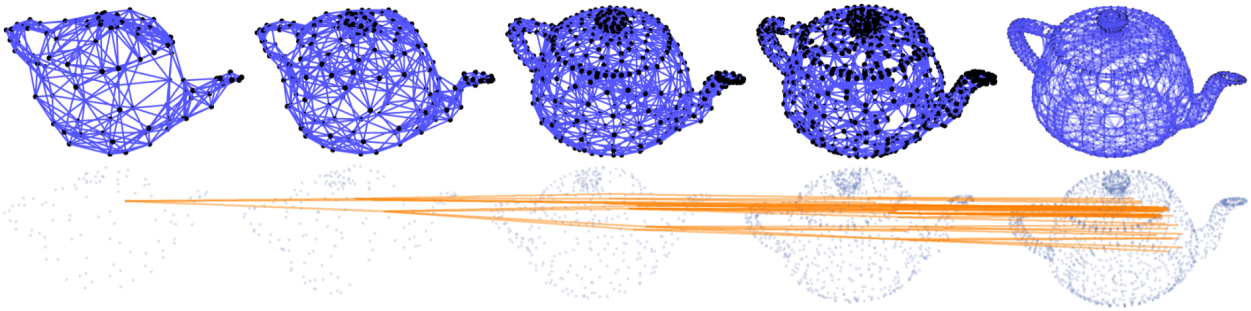

Figure 1.2: Top: subsamples of a mesh of the Utah teapot, of increasing density (each node is connected to its 8 nearest neighbors by the $\Delta l = \pm 0$ edges, rendered in blue). These samples form a graph lineage ($\Delta l = \pm 1$ edges are not illustrated). Bottom: the same set of nodes, with only $\Delta l = \pm 1$ edges plotted (in orange) for one node from the coarsest level and its descendants.

Kronecker Product and Sum of matrices: Given a $(k \times l)$ matrix M, and some other matrix N, the Kronecker product is the block matrix

$$M \otimes N = \begin{bmatrix} m_{11}N & \cdots & m_{1l}N \\ \vdots & \ddots & \vdots \\ m_{k1}N & \cdots & m_{kl}N \end{bmatrix}.$$

See [52], Section 11.4, for more details about the Kronecker Product. If M and N are square,

their Kronecker Sum is defined, and is given by

$$M \oplus N = M \otimes I_N + I_M \otimes N$$

where we write I_A to denote an identity matrix of the same size as A.

Box Product ($\square$) of graphs: For G_1 with vertex set $U = \{u_1, u_2 \ldots\}$ and G_2 with vertex set $V = \{v_1, v_2 \ldots\}$, $G_1 \square G_2$ is the graph with vertex set $U \times V$ and an edge between (u_{i_1}, v_{j_1}) and (u_{i_2}, v_{j_2}) when either of the following is true:

- $i_1 = i_2$ and v_{j_1} and v_{j_2} are adjacent in G_2, or
- $j_1 = j_2$ and u_{i_1} and u_{i_2} are adjacent in G_1.

This may be rephrased in terms of the Kronecker Sum $\oplus$ of the two matrices:

$$A(G_1 \square G_2) = A(G_1) \oplus A(G_2) = A(G_1) \otimes I_{|G_2|} + I_{|G_1|} \otimes A(G_2) \tag{1.7}$$

Cross Product ($\times$) of graphs: For G_1 with vertex set $U = \{u_1, u_2 \ldots\}$ and G_2 with vertex set $V = \{v_1, v_2 \ldots\}$, $G_1 \times G_2$ is the graph with vertex set $U \times V$ and an edge between (u_{i_1}, v_{j_1}) and (u_{i_2}, v_{j_2}) when both of the following are true:

- u_{i_1} and u_{i_2} are adjacent in G_1, and
- v_{j_1} and v_{j_2} are adjacent in G_2.

We include the standard pictorial illustration of the difference between these two graph products in Figure 1.3.

Grid Graph: a grid graph (called a lattice graph or Hamming Graph in some texts [16]) is the distance-regular graph given by the box product of path graphs $P_{a_1}, P_{a_1}, \ldots P_{a_k}$ (yielding

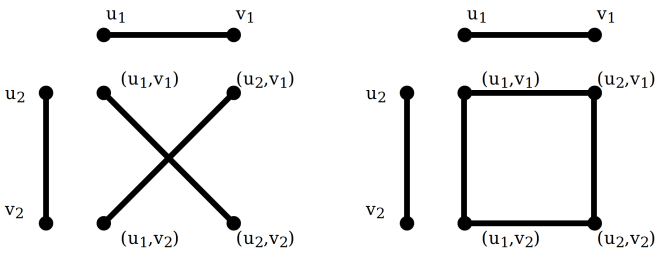

Figure 1.3: Two types of graph product: the Cross product ($G_1 \times G_2$, left) and Box product ($G_1 \square G_2$, right). For two edges $v_1 \sim u_1 \in G_1$ and $v_2 \sim u_2 \in G_2$, we illustrate the resultant edges in the set of vertices $\{(u_1, v_1), (u_2, v_1), (u_1, v_2), (u_2, v_2)\}$ in the graph product.

a grid with aperiodic boundary conditions) or by a similar list of cycle graphs (yielding a grid with periodic boundary conditions).

Prolongation map: A prolongation map between two graphs G_1 and G_2 of sizes n_1 and n_2, with $n_2 \geq n_1$, is an $n_2 \times n_1$ matrix of real numbers which is an optimum of the objective function of equation 6.1 below (possibly subject to some set of constraints $C(P)$).

Chapter 2

Diffusion Distance

In this Chapter we provide the definition of Graph Diffusion Distance, as well as providing motivation for why the optimiztion over t is an essential component of the GDD calculation. We also briefly introduce some variants of GDD which will be covered in more detail in Chapter 3. The diffusion distance calculations presented throughout this thesis depend on an upper bound of the innermost optimization over P and α; in Section 2.4 we define a lower bound on the same optimization. This lower bound will be useful in some of the GDD property proofs in Chapter 3.

2.1 Diffusion Distance Definition

We generalize the diffusion distance defined by Hammond et al. [48]. This distortion measure between two undirected graphs G_1 and G_2, of the same size, was defined as:

$$D_{\text{Hammond}}(G_1, G_2) = \sup_t \left|\left|e^{tL_1} - e^{tL_2}\right|\right|_F^2 \tag{2.1}$$

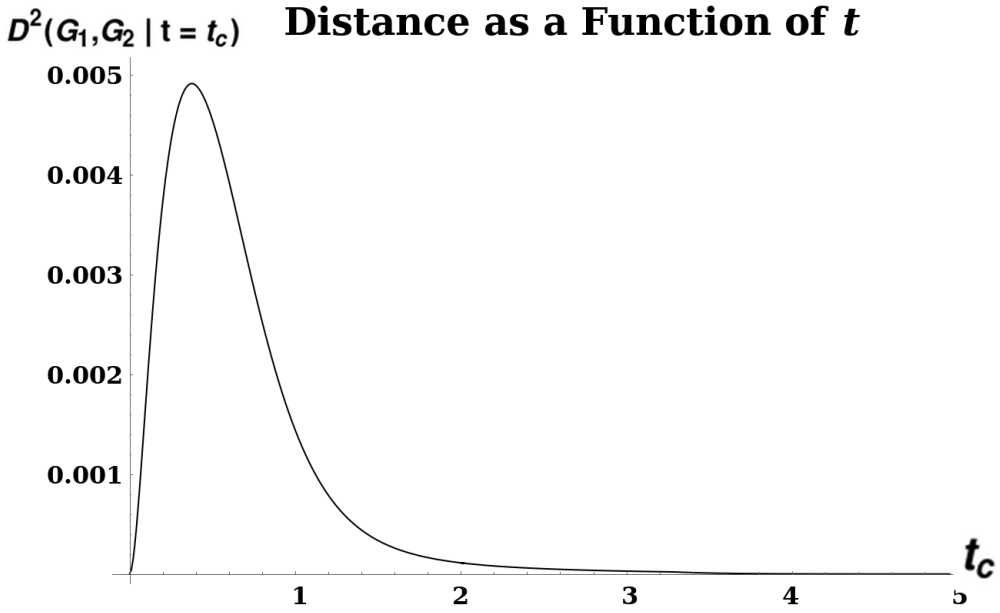

Figure 2.1: A plot illustrating unimodality of diffusion distance. D^2 was calculated between two grid graphs Sq_7 and Sq_8 of size 7×7 and 8×8, respectively. The distance is given by the formula $D^2\left(\mathrm{Sq}_7, \mathrm{Sq}_8 \middle| t\right) = \inf_{\alpha>0} \inf_{P|\mathcal{C}(P)} \left|\left| P e^{\frac{t}{a} L(\mathrm{Sq}_7)} - e^{t\alpha L(\mathrm{Sq}_8)} P \right|\right|_F^2$ as a function of t. The peak, at $t \approx .318$, yields the distance $D^2\left(\mathrm{Sq}_7, \mathrm{Sq}_8\right)$.

where L_i represents the graph Laplacian of G_i.

This may be interpreted as measuring the maximum divergence, as a function of t, between diffusion processes starting from each vertex of each graph, as measured by the squared Euclidean distance between the column vectors of e^{tL_i}. Each column v_j of e^{tL_i} (which is called the Laplacian Exponential Kernel) describes a probability distribution of visits (by a random walk of duration t, with node transition probabilities given by the columns of e^L) to the vertices of G_i, starting at vertex j. This distance metric is then measuring the difference between the two graphs by comparing these probability distributions; the motivation between taking the supremum over all t is that the value of the objective function at the maximum is the most these two distributions can diverge. See Figure 2.1 for an example of a distance calculation, with a characteristic peak.

For further intuition about why the peak is the most natural place to take as the distance,

rather than some other arbitrary time, note that at very early times and very late times, the probability distribution of vertex visits is agnostic to graph structure: at early times no diffusion has had a chance to take place, while at very late times the distribution of vertex-visits converges to the stationary state[1] for each connected component of the graph. Hence we are most interested in a regime of t-values in between these extremes, where differences in G_1 and G_2 are apparent in their differing probability distributions.

Our contribution generalizes this measure to allow for graphs of differing size. We add two variables to this optimization: a *prolongation* operator, P (represented as a rectangular matrix), and a time-scaling factor, α. The dissimilarity between two graphs G_1 and G_2 (with Laplacians $L_i = L(G_i)$) is then defined as:

$$D^2(G_1, G_2) = \sup_{t>0} \inf_{\alpha>0} \inf_{P|\mathcal{C}(P)} \left|\left| Pe^{\frac{t}{\alpha}L_1} - e^{\alpha t L_2} P \right|\right|_F^2 \tag{2.2}$$

where $\mathcal{C}(P)$ represents some set of constraints on the matrix P. For the remainder of this work we use $D(G_1, G_2)$ to refer to the distance and $D^2(G_1, G_2)$ to refer to the squared distance - this notation is chosen to simplify the exposition of some proofs. It will be convenient for later calculations to introduce and assume the concept of *transitive constraints* - by which we mean that for any constraint $\mathcal{C}$, satisfaction of $\mathcal{C}$ by P_1 and P_2 implies satisfaction of $\mathcal{C}$ by their product P_1P_2 (when such a product is defined). Some (non-exclusive) examples of transitive constraints include orthogonality, particular forms of sparsity, and their conjunctions.

The simplest transitive constraint we will consider is that P should be orthogonal. Intuitively, an orthogonal P represents a norm-preserving map between nodes of G_1 and nodes of G_2, so we are measuring how well diffusion on G_1 approximates diffusion on G_2, as projected by P. Note that since in general P is a rectangular matrix it is not necessarily true that $PP^T = I$. We assume that $|G_1| = n_1 \leq n_2 = |G_2|$; if not, the order of the operands is

[1] Because the graphs are undirected, a stationary state is guaranteed to exist.

switched, so that P is always at least as tall as it is wide. We also briefly consider a sparsity constraint in Section 3.4 below. Since sparsity is more difficult to treat numerically, our default constraint will be orthogonality alone. Other constraints could include bandedness and other structural constraints. We also note that because L is finite-dimensional, the exponential map is continuous and therefore we can swap the order of optimization over t and α. The optimization procedure outlined in this thesis optimizes these variables in the order presented above (namely: an outermost loop of maximization over t, a middle loop of minimization over α given t, and an innermost loop of minimization over P given t and α).

The other additional parameter, α, controls dilation between the passage of time in the two graphs, to account for different scales. Again, the intuition is that we are interested in the difference between structural properties of the graph (from the point of view of single-particle diffusion) independent of the absolute number of nodes in the graph. As an example, diffusion on an $n \times n$ grid is a reasonably accurate approximation of more rapid diffusion on a $2n \times 2n$ grid, especially when n is very large. In our discussion of variants of this dissimilarity score, we will use the notation $D^2(G_1, G_2|x = c)$ to mean restrictions of any of our distortion measure equations where variable x is held to a constant value c; In cases where it is clear from context which variable is held to a fixed value c, we will write $D^2(G_1, G_2|c)$

At very early times the second and higher-order terms of the Taylor Series expansion of the matrix exponential function vanish, and so $e^{tL} \approx I + tL$. This motivates the *early-time* or "linear" version of this distance, $\tilde{D}$:

$$\tilde{D}^2(G_1, G_2) = \inf_{\alpha>0} \inf_{P|\mathcal{C}(P)} \left|\left| \frac{1}{\alpha} PL_1 - \alpha L_2 P \right|\right|_F^2 \tag{2.3}$$

$$\approx \frac{1}{t^2} \left(\inf_{\alpha>0} \inf_{P|\mathcal{C}(P)} \left|\left| Pe^{\frac{t}{\alpha}L_1} - e^{\alpha t L_2} P \right|\right|_F^2 \right) \tag{2.4}$$

(Note that the identity matrices cancel). The outermost optimization (maximization over t)

is removed for this version of the distance, as t can be factored out:

$$\left|\left|\frac{t}{\alpha}PL_1 - \alpha t L_2 P\right|\right|_F^2 = t^2\left|\left|\frac{1}{\alpha}PL_1 - \alpha L_2 P\right|\right|_F^2 \tag{2.5}$$

This means that for the linear version, we only optimize α and P. For the exponential version of the dissimilarity score, we note briefly that the supremum over t of our objective function must exist, since for any G_1, G_2:

$$D^2(G_1, G_2) \leq D^2\left(G_1, G_2 \middle| \alpha = 1, P = \begin{bmatrix} I \\ 0 \end{bmatrix}\right) \tag{2.6}$$

In other words, the infimum over all P and α is bounded above by any particular choice of values for these variables. Since

$$D^2\left(G_1, G_2 \middle| t = 0, \alpha = 1, P = \begin{bmatrix} I \\ 0 \end{bmatrix}\right) = 0, \qquad \text{and} \tag{2.7}$$

$$\lim_{t_c \to \infty} D^2\left(G_1, G_2 \middle| t_c, \alpha = 1, P = \begin{bmatrix} I \\ 0 \end{bmatrix}\right) = 0 \tag{2.8}$$

this upper bound must have a supremum (possibly 0) at some $t^* \in [0, \infty)$. Then

$$D^2\left(G_1, G_2 \middle| t^*, \alpha = 1, P = \begin{bmatrix} I \\ 0 \end{bmatrix}\right) \tag{2.9}$$

must be finite and therefore so must the objective function.

2.2 Directedness of Distance and Constraints

We note that this distance measure, as defined so far, is *directed*: the operands G_1 and G_2 serve differing roles in the objective function. This additionally makes the constraint predicate $\mathcal{C}(P)$ ambiguous: when we state that $\mathcal{C}$ represents orthogonality, it is not clear whether we are referring to $P^TP = I$ or $PP^T = I$ (only one of which can be true for a non-square matrix P). To remove this ambiguity, we will, for the computations in the rest of this manuscript, define the distance metric to be symmetric: the distance between G_1 and G_2 with $|G_1| \leq |G_2|$ is always $D(G_1, G_2)$. P is then always at least as tall as it is wide, so of the two choices of orthogonality constraint we select $P^TP = I$.

2.3 Variants of Distance Measure

Thus far we have avoided referring to this graph dissimilarity function as a "distance metric". As we shall see later, full optimization of Equations 2.2 and 2.3 over α and P is too loose, in the sense that the distances $D(G_1, G_2)$, $D(G_2, G_3)$, and $D(G_1, G_3)$ do not necessarily satisfy the triangle inequality. The same is true for $\tilde{D}$. See Section 5.3 for numerical experiments suggesting a particular parameter regime where the triangle inequality is satisfied. We thus define several restricted/augmented versions of both D and $\tilde{D}$ which are guaranteed to satisfy the triangle inequality. These different versions are summarized in Table 2.1. These variously satisfy some of the conditions necessary for generalized versions of distance metrics, including:

- Premetric: a function $d(x, y)$ for which $d(x, y) \geq 0$ and $d(x, y) = d(y, x)$ for all x, y.
- Pseudometric: As a premetric, but additionally $d(x, z) \leq d(x, y) + d(y, z)$ for all x, y, z.
- ρ-inframetric: As a premetric, but additionally $d(x, z) \leq \rho\,(d(x, y) + d(y, z))$ and

$d(x, y) = 0$ if and only if $x = y$, for all x, y, z.

Additionally, we note here that a distance measure on graphs using Laplacian spectra can at best be a pseudometric, since isospectral, non-isomorphic graphs are well-known to exist [40][107]. Characterizing the conditions under which two graphs are isospectral but not isomorphic is an open problem in spectral graph theory. However, previous computational work has led to the conjecture that "almost all" graphs are uniquely defined by their spectra [15][17][108], in the sense that the probability of two graphs of size n being isospectral but not isomorphic goes to 0 as $n \to \infty$. Furthermore, our numerical experiments seem to show empirically that the violation of the triangle inequality is bounded, in the sense that $D(G_1, G_3) \leq \rho * (D(G_1, G_2) + D(G_2, G_3))$ for $\rho \approx 2.1$. This means that even in the least restricted case our similarity measure may be a 2.1-infra-pseudometric on graphs (and, since such isospectral, non-isomorphic graphs are relatively rare, it behaves like a $2.1-$inframetric). As we will see in Chapter 3, in some more restricted cases we can prove triangle inequalities, making our measure a pseudometric. In Section 4.1, we discuss an algorithm for computing the optima in Equations (2.2) and (2.3). First, we discuss some theoretical properties of this dissimilarity measure.

2.4 Spectral Lower Bound

In Theorem 4.4.1 of Chapter 4 we will derive and make use of an upper bound on the graph distance $\tilde{D}(G_1, G_2)$. This upper bound is calculated by constraining the variable P to be not only orthogonal, but also $P = U_2 M U_1^T$ where M is the solution (i.e. "matching", in the terminology of that section) to a Linear Assignment problem with costs given by a function of the eigenvalues of $L(G_1)$ and $L(G_2)$. In this section we derive a similar lower bound on the distance.

Let G_1 and G_2 be undirected graphs with Laplacians $L_1 = L(G_1)$ and $L_2 = L(G_2)$, and let $\alpha > 0$ be constant. By Equation (4.5), we have

$$\tilde{D}^2(G_1, G_2) = \inf_{\alpha>0} \inf_{P^T P = I} \left(\sum_{i=1}^{n_2} \sum_{j=1}^{n_1} p_{ij}^2 \left(\frac{1}{\alpha} \lambda_j^{(1)} - \alpha \lambda_i^{(2)} \right)^2 \right). \tag{2.10}$$

The following upper bound on $\tilde{D}$ is achieved by constraining P to be not only orthogonal, but related to a constrained matching problem between the two lists of eigenvalues:

$$\begin{array}{lll} \tilde{D}^2(G_1, G_2) \leq \inf_{\alpha>0} \inf_M & \left|\left| \frac{1}{\alpha} M\Lambda_1 - \alpha \Lambda_2 M \right|\right|_F^2 & \\ \text{subject to} & \sum_{i=1}^{n_2} m_{ij} \leq 1, & j = 1 \ldots n_1 \\ & \sum_{j=1}^{n_1} m_{ij} \leq 1, & i = 1 \ldots n_2 \\ & m_{ij} \geq 0 & i = 1 \ldots n_2, j = 1 \ldots n_1, \end{array} \tag{2.11}$$

where Λ_1 and Λ_2 are diagonal matrices of the eigenvalues of L_1 and L_2 respectively. Here we used the explicit map $\tilde{P} = U_2^T P U_1$ as a change of basis; we then converted the constraints on P into equivalent constraints on $\tilde{P}$, and imposed additional constraints so that the resulting optimization (a linear assignment problem) is an upper bound. See the proof of Theorem 4.4.1 for the details of this derivation. We show in this section that a less constrained assignment problem is a lower bound on $\tilde{D}^2$. We do this by computing the same mapping $\tilde{P} = U_2^T P U_1$ and then dropping some of the constraints on $\tilde{P}$ (which is equivalent to dropping constraints on P, yielding a lower bound). The constraint $P^T P = I$ is the conjunction of n_1^2 constraints on the column vectors of P: if $\mathbf{p}_i$ is the ith column of P, then $P^T P = I$ is

equivalent to:

$$\mathbf{p}_i \cdot \mathbf{p}_i = 1 \qquad \forall i = 1 \dots n_1 \tag{2.12}$$

$$\mathbf{p}_i \cdot \mathbf{p}_i = 0 \qquad \forall i = 1 \dots n_1, j = 1 \dots i-1, i+1 \dots n_1, \tag{2.13}$$

If we discard the constraints in Equation (2.13), we are left with the constraint that every column of p must have unit norm.

Construct the "spectral lower bound matching" matrix $P^{(\mathrm{SLB})}$ as follows:

$$P_{i,j}^{(\mathrm{SLB})} = \begin{cases} 1 & \text{if i} = \arg\min_k \left(\frac{1}{\alpha}\lambda_j^{(1)} - \alpha\lambda_k^{(k)}\right)^2 \\ 0 & \text{otherwise.} \end{cases} \tag{2.14}$$

For any α, this matrix is the solution to a matching problem (less constrained than the original optimization over all P) where each $\lambda_j^{(1)}$ is assigned to the closest $\lambda_i^{(2)}$, allowing collisions. It clearly satisfies the constraints in Equation (2.12), but may violate those in Equation (2.13). Thus, we have

$$\begin{aligned} \tilde{D}^2(G_1, G_2) &= \inf_{\alpha>0} \inf_{P^T P = I} \left(\sum_{i=1}^{n_2} \sum_{j=1}^{n_1} p_{ij}^2 \left(\frac{1}{\alpha}\lambda_j^{(1)} - \alpha\lambda_i^{(2)} \right)^2 \right). \\ &\geq \tilde{D}^2 \left(G_1, G_2 \,\middle|\, P^{(\mathrm{SLB})} \right) \end{aligned} \tag{2.15}$$

Various algorithms exist to rapidly find the member of a set of points which is closest to some reference point (for example, KD-Trees [11]). For any α, the spectral lower bound can be calculated by an outer loop over alpha and an inner loop which applies one of these methods. We do not consider joint optimization of the lower bound over P and α in this

work.

2.5 Summary of Distance Metric Versions

Table 2.1 summarizes the variants of our distance metric.

t	α	s	Classification	Treatment in this manuscript
Fixed at $t_c < \epsilon$	Fixed at $\alpha_c = 1$	$s = 0$	Pseudometric	Defined in Equation 3.5. Optimized by one pass of LAP solver. Triangle inequality proven in Theorem 3.2.2.
Fixed at $t_c < \epsilon$	Fixed at $\alpha_c = (\frac{n_1}{n_2})^r$	$s = 0$	Pseudometric	Defined in Equation (3.11). Optimized by one pass of LAP solver. Triangle inequality proven in Theorem 3.3.1.
Fixed at $t_c < \epsilon$	Optimized	$s = 0$	Premetric	Defined in Equation 2.3. Optimized by Algorithm 1. Triangle inequality violations examined experimentally in Section 5.3.
Optimized	Fixed at $\alpha_c = 1$	$s = 0$	Metric	When $\|G_1\| = \|G_2\|$, this is Hammond et. al's version of graph distance.
Optimized	Optimized	$s = 0$	Premetric	Defined in Equation 2.2. Optimized by Algorithm 2. Graph Product upper bound proven in Theorem 3.6.1. Triangle inequality violations examined experimentally in Section 5.3. Used to calculate graph distances in Sections 5.4 and 5.5.
Fixed at $t_c < \epsilon$	Fixed at $\alpha_c = 1$	$s > 0$	Pseudometric	Triangle inequality proven in Theorem 3.2.2.
Fixed at $t_c < \epsilon$	Fixed at $\alpha_c = (\frac{n_1}{n_2})^r$	$s > 0$	Pseudometric	Triangle inequality proven in Theorem 3.3.1.
Optimized	Optimized	$s > 0$		Discussed in Section 3.4.

Table 2.1: Summary of this thesis's investigation of different forms of our graph dissimilarity measure. In this work, we systematically explore properties of this measure given sparsity parameter $s = 0$, and various regimes of t (fixed at some early time, or maximized over all t) and α (fixed at $\alpha = 1$, fixed at a constant power r of the ratio of graph sizes, or minimized over all α. We leave exploration of nonzero values of the sparsity parameter to future work. Variants not explicitly called out are not considered. In the case where α and t are both optimized and $s > 0$, it is unclear which of the metric conditions GDD satisfies, hence the corresponding classification is left blank.

Chapter 3

Theoretical Properties of GDD

Having introduced Graph Diffusion Distance in Chapter 2, we proceed to prove several of properties of various instances of our graph dissimilarity score, including triangle inequalities for some specific versions and an upper bound on the distance between two graph products. We will here rely heavily on various properties of the Kronecker sum and product of matrices which may be found in [52], Section 11.4.

3.1 Optimization over P is equivalent to an eigenvalue matching problem

For the purpose of the calculations in this section, we restrict ourselves to the "diffusion" term of our objective function 6.1 (the term which coerces two diffusion processes to agree), which we will write as

$$D_{P,\alpha}(G_1, G_2) = \left|\left| \frac{1}{\sqrt{\alpha}} P L_1 - \sqrt{\alpha} L_2 P \right|\right|_F . \tag{3.1}$$

Because L_1 and L_2 are each real and symmetric, they may both be diagonalized as $L_i = U_i \Lambda_i U_i^T$ where U_i is a rotation matrix and Λ_i is a diagonal matrix with the eigenvalues of L_i on the diagonal. Substituting into 3.1, and letting $\tilde{P} = U_2^T P U_1$, we have

$$\begin{aligned} D_{P,\alpha}(G_1, G_2) &= \left|\left| \frac{1}{\sqrt{\alpha}} P L_1 - \sqrt{\alpha} L_2 P \right|\right|_F \\ &= \left|\left| \frac{1}{\sqrt{\alpha}} P U_1 \Lambda_1 U_1^T - \sqrt{\alpha} U_2 \Lambda_2 U_2^T P \right|\right|_F \\ &= \left|\left| \frac{1}{\sqrt{\alpha}} \left(U_2^T P U_1\right) \Lambda_1 - \sqrt{\alpha} \Lambda_2 \left(U_2^T P U_1\right) \right|\right|_F \\ &= \left|\left| \frac{1}{\sqrt{\alpha}} \tilde{P} \Lambda_1 - \sqrt{\alpha} \Lambda_2 \tilde{P} \right|\right|_F \end{aligned} \tag{3.2}$$

where $\tilde{P}$ is an orthogonal matrix $\tilde{P}^T \tilde{P} = I$ if and only if P is as well. Since the Frobenius norm is invariant under multiplication by rotation matrices, 3.2 is a re-formulation of our original Laplacian matrix objective function in terms of the spectra of the two graphs. Optimization of this modified form of the objective function subject to orthogonality constraints on P is upper-bounded by optimization over matchings of eigenvalues: for any fixed α the eigenvalue-matching problem has the same objective function, but our optimization is over all real valued orthogonal P. The orthogonality constraint is a relaxed version of the constraints on matching problems (Equation 1.6) discussed in subsection 1.4.2, since matching matrices M are also orthogonal ($M^T M = I$). Many algorithms exist for solving the inner partial and 0-1 constrained minimum-cost assignment problems, such as the Munkres algorithm [74] (also in subsection 1.4.2).

We note three corollaries of the above argument. Namely, because the Frobenius norm is invariant under the mapping to and from eigenspace:

1. Optimal or near-optimal $\tilde{P}$ in eigenvalue-space maintain their optimality through the mapping $U_2 \cdot U_1^T$ back to graph-space.

2. Solutions which are within ϵ of the optimum in $\tilde{P}$-space are also within ϵ of the optimum in P-space; and

3. More precisely, if they exist, zero-cost eigenvalue matchings correspond exactly with zero-cost P.

A natural next question would be why it might be worthwhile to work in the original graph-space, rather than always optimizing this simpler eigenvalue-matching problem instead. In many cases (path graphs, cycle graphs) the spectrum of a member G_l of a graph lineage is a subset of that of G_{l+1}, guaranteeing that zero-cost eigenvalue matchings (and thus, by the argument above, prolongations with zero diffusion cost) exist. However, when this is not the case, the above argument only upper bounds the true distance, since the matching problem constraints are more strict. Thus, numerical optimization over P, with orthogonality constraints only, may find a better bound on $D^{P,\alpha}(G_l, G_{l+1})$.

3.2 Triangle Inequality for $\alpha = 1$

In this section, we show that both the linear and exponential versions of diffusion distance satisfy the triangle inequality when $\alpha = 1$.

Lemma 3.2.1. *For any matrices M and P, with P satisfying $P^TP = I$,*
$||PM||_F^2 \leq ||M||_F^2$ *and* $||MP||_F^2 \leq ||M||_F^2$.

Proof. Suppose without loss of generality that $P^TP = I$. Then:

1. $||PM||_F^2 = \mathrm{Tr}[M^TP^TPM] = \mathrm{Tr}[M^TM] = ||M||_F^2$

2. If $P^TP = I$, then letting $PP^T = \Pi$, Π is a projection operator satisfying $\Pi^T = \Pi = \Pi^2$. Then,

$$
\begin{aligned}
||M||_F^2 = \mathrm{Tr}[M^T M] &= \mathrm{Tr}[M^T M(\Pi + (I - \Pi))] \\
&= \mathrm{Tr}[M^T M \Pi] + \mathrm{Tr}[M^T M(I - \Pi)] \\
&= \mathrm{Tr}[M^T M P P^T] + \mathrm{Tr}[M^T M(I - \Pi)^2] \\
&= ||MP||_F^2 + ||M(I - \Pi)||_F^2 \\
&\geq ||MP||_F^2
\end{aligned} \tag{3.3}
$$

□

Theorem 3.2.2. *$\tilde{D}^2$ satisfies the triangle inequality for $\alpha = 1$.*

Proof. Let G_1, G_2, G_3 be simple graphs, with Laplacians L_1, L_2, L_3. Let

$$
P_{31} = \arg \inf_{P|\mathcal{C}(P)} ||PL_1 - L_3 P||_F^2. \tag{3.4}
$$

P_{31} is guaranteed to exist for constraints $\mathcal{C}$ which form a compact space of matrices; orthogonality constraints are an example, since the space of orthogonal matrices is closed and bounded. Then

$$
\begin{aligned}
\tilde{D}^2(G_1, G_3 \,|\alpha = 1) = ||P_{31}L_1 - L_3 P_{31}||_F^2 &= \inf_{P|\mathcal{C}(P)} ||PL_1 - L_3 P||_F^2 \\
&\leq \inf_{P_{32}, P_{21}|\mathcal{C}(P_{32}P_{21})} ||P_{32}P_{21}L_1 - L_3 P_{32}P_{21}||_F^2
\end{aligned} \tag{3.5}
$$

where we write $\mathcal{C}(P_{32}P_{21})$ to signify that the product $P_{32}P_{21}$ satisfies the original transitive constraints on P, e.g. orthogonality and/or sparsity. Since the constraint predicate $\mathcal{C}(P)$

satisfies Equation (3.12), then so does their product, so we may write

$$\begin{aligned}
\tilde{D}(G_1, G_3 \,|\alpha = 1) &\leq \inf_{P_{32}|\mathcal{C}(P_{32})} \inf_{P_{21}|\mathcal{C}(P_{21})} ||P_{32}P_{21}L_1 - L_3P_{32}P_{21}||_F \\
&= \inf_{P_{32}|\mathcal{C}(P_{32})} \inf_{P_{21}|\mathcal{C}(P_{21})} ||P_{32}P_{21}L_1 - P_{32}L_2P_{21} \\
&\qquad + \quad P_{32}L_2P_{21} - L_3P_{32}P_{21}||_F \\
&\leq \inf_{P_{32}|\mathcal{C}(P_{32})} \inf_{P_{21}|\mathcal{C}(P_{21})} (||P_{32}P_{21}L_1 - P_{32}L_2P_{21}||_F \\
&\qquad + \quad ||P_{32}L_2P_{21} - L_3P_{32}P_{21}||_F) \\
&= \inf_{P_{32}|\mathcal{C}(P_{32})} \inf_{P_{21}|\mathcal{C}(P_{21})} (||P_{32}\left(P_{21}L_1 - L_2P_{21}\right)||_F \\
&\qquad + \quad ||\left(P_{32}L_2 - L_3P_{32}\right)P_{21}||_F)
\end{aligned} \tag{3.6}$$

By Lemma 3.2.1,

$$\begin{aligned}
\tilde{D}(G_1, G_3 \,|\alpha = 1) &\leq \inf_{P_{32}|\mathcal{C}(P_{32})} \inf_{P_{21}|\mathcal{C}(P_{21})} (||P_{21}L_1 - L_2P_{21}||_F \\
&\qquad + \quad ||P_{32}L_2 - L_3P_{32}||_F) \\
&= \inf_{P_{21}|\mathcal{C}(P_{21})} ||P_{21}L_1 - L_2P_{21}||_F \\
&\qquad + \quad \inf_{P_{32}|\mathcal{C}(P_{32})} ||P_{32}L_2 - L_3P_{32}||_F \\
&= \tilde{D}(G_1, G_2 \,|\alpha = 1) + \tilde{D}(G_2, G_3 \,|\alpha = 1)
\end{aligned} \tag{3.7}$$

□

We note that in this proof we use L_1, L_2, and L_3 (making this the small-t or linear version of the objective function), but the same argument holds when all three are replaced with e^{tL_i}, so we also have

Corollary 3.2.3. *D satisfies the triangle inequality for $\alpha = 1$.*

Proof. By the same calculation as in Theorem 3.2.2, with all L_i replaced by $e^{t_c L_i}$, we have

$$D\left(G_1, G_3 \middle| t_c, \alpha = 1\right) \leq D(G_1, G_2 \,|t_c, \alpha = 1) + D(G_2, G_3 \,|t_c, \alpha = 1) \tag{3.8}$$

for any constant t_c. Then, letting

$$t_{13} = \arg\sup_{t_c} D\left(G_1, G_3 \middle| t_c, \alpha = 1\right) \tag{3.9}$$

we have:

$$\begin{aligned} D\left(G_1, G_3 \middle| \alpha = 1\right) &= \sup_{t_c} D\left(G_1, G_3 \middle| t_c, \alpha = 1\right) \\ &= D\left(G_1, G_3 \middle| t_{13}, \alpha = 1\right) \\ &\leq D(G_1, G_2 \,|t_{13}, \alpha = 1) + D(G_2, G_3 \,|t_{13}, \alpha = 1) \\ &\leq \sup_{t_c} D\left(G_1, G_2 \middle| t_c, \alpha = 1\right) \\ &\qquad + \quad \sup_{t_c} D\left(G_2, G_3 \middle| t_c, \alpha = 1\right) \\ &= D\left(G_1, G_2 \middle| \alpha = 1\right) + D\left(G_2, G_3 \middle| \alpha = 1\right) \end{aligned} \tag{3.10}$$

□

Note that in the proofs of Theorem 3.2.2, Theorem 3.3.1, and Corollary 3.2.3, we assume that the constraint predicate $\mathcal{C}(P)$ includes at least orthogonality (so that we may apply Lemma 3.2.1). However, this constraint predicate could be more strict, e.g. include both orthogonality and sparsity. Hence these statements also apply to the $s > 0$ cases in Table 2.1, which we do not otherwise consider in this work: in our numerical experiments we (for reasons of computational simplicity) only require our optimization over P be orthogonally constrained.

3.3 Time-Scaled Graph Diffusion Distance

For any graphs G_1 and G_2, and some real nuber r, we define the Time-Scaled Graph Diffusion Distance (TSGDD) as a scaled version of the linear distance, with α fixed. Namely, let

$$\begin{aligned}\tilde{D}_r^2(G_1, G_2) &= (n_1 n_2)^{-2r} \tilde{D}^2 \left(G_1, G_2 \middle| \alpha = \left(\frac{n_1}{n_2}\right)^r \right) \\ &= \inf_{P|\mathcal{C}(P)} (n_1 n_2)^{-2r} \left\| \left(\frac{n_1}{n_2}\right)^{-r} P L_1 - \left(\frac{n_1}{n_2}\right)^r L_2 P \right\|_F^2 \end{aligned} \tag{3.11}$$

The intuition for this version of the distance measure is that we are constraining the time dilation, α, between G_1 and G_2 to be a power of the ratio of the two graph sizes. The factor $(n_1 n_2)^{-2r}$ is needed to ensure this version of the distance satisfies the triangle inequality, as seen in Theorem 3.3.1.

Theorem 3.3.1. *The TSGDD, as defined above, satisfies the triangle inequality.*

Proof. As above, let G_1, G_2, G_3 be three graphs with $n_i = |G_i|$ and $n_1 \leq n_2 \leq n_3$, and let L_i be the Laplacian of G_i. Let $\mathcal{C}(P)$ represent a transitive constraint predicate, also as described previously. Then, for a constant $r \in \mathbb{R}$, we have:

$$\begin{aligned}&\tilde{D}_r(G_1, G_3) = \\ &\quad \inf_{P|\mathcal{C}(P)} (n_1 n_3)^{-r} \left\| \left(\frac{n_1}{n_3}\right)^{-r} P L_1 - \left(\frac{n_1}{n_3}\right)^r L_3 P \right\|_F \\ &\quad \leq \inf_{P_{32}, P_{21}|\mathcal{C}(P_{32}P_{21})} (n_1 n_3)^{-r} \left\| \left(\frac{n_1}{n_3}\right)^{-r} P_{32} P_{21} L_1 - \left(\frac{n_1}{n_3}\right)^r L_3 P_{32} P_{21} \right\|_F \end{aligned}$$

under the assumption, as in Equation (3.12), that $\mathcal{C}(P_{32}) \wedge \mathcal{C}(P_{21}) \implies \mathcal{C}(P_{32}P_{21})$,

$$
\begin{aligned}
&\tilde{D}_r(G_1, G_3) \leq \\
&\inf_{P_{32},P_{21}|\mathcal{C}(P_{32})\wedge\mathcal{C}(P_{21})} (n_1 n_3)^{-r} \left|\left| \left(\frac{n_1}{n_3}\right)^{-r} P_{32}P_{21}L_1 - \left(\frac{n_1}{n_3}\right)^{r} L_3 P_{32}P_{21} \right|\right|_F \\
&= \inf_{P_{32},P_{21}|\mathcal{C}(P_{32})\wedge\mathcal{C}(P_{21})} (n_1 n_3)^{-r} \left|\left| \left(\frac{n_1}{n_3}\right)^{-r} P_{32}P_{21}L_1 - \left(\frac{n_1 n_3}{n_2^2}\right)^{r} P_{32}L_2 P_{21} \right.\right. \\
&\qquad + \qquad \left.\left. \left(\frac{n_1 n_3}{n_2^2}\right)^{r} P_{32}L_2 P_{21} - \left(\frac{n_1}{n_3}\right)^{r} L_3 P_{32}P_{21} \right|\right|_F \\
&\leq \inf_{P_{32},P_{21}|\mathcal{C}(P_{32})\wedge\mathcal{C}(P_{21})} (n_1 n_3)^{-r} \left|\left| \left(\frac{n_1}{n_3}\right)^{-r} P_{32}P_{21}L_1 - \left(\frac{n_1 n_3}{n_2^2}\right)^{r} P_{32}L_2 P_{21} \right|\right|_F \\
&\qquad + \qquad (n_1 n_3)^{-r} \left|\left| \left(\frac{n_1 n_3}{n_2^2}\right)^{r} P_{32}L_2 P_{21} - \left(\frac{n_1}{n_3}\right)^{r} L_3 P_{32}P_{21} \right|\right|_F \\
&= \inf_{P_{32},P_{21}|\mathcal{C}(P_{32})\wedge\mathcal{C}(P_{21})} (n_1 n_3)^{-r} \left(\frac{n_3}{n_2}\right)^{r} \left|\left| \left(\frac{n_1}{n_2}\right)^{-r} P_{32}P_{21}L_1 - \left(\frac{n_1}{n_2}\right)^{r} P_{32}L_2 P_{21} \right|\right|_F \\
&\qquad + \qquad (n_1 n_3)^{-r} \left(\frac{n_1}{n_2}\right)^{r} \left|\left| \left(\frac{n_2}{n_3}\right)^{-r} P_{32}L_2 P_{21} - \left(\frac{n_2}{n_3}\right)^{r} L_3 P_{32}P_{21} \right|\right|_F \\
&= \inf_{P_{32},P_{21}|\mathcal{C}(P_{32})\wedge\mathcal{C}(P_{21})} (n_1 n_2)^{-r} \left|\left| \left(\frac{n_1}{n_2}\right)^{-r} P_{32}P_{21}L_1 - \left(\frac{n_1}{n_2}\right)^{r} P_{32}L_2 P_{21} \right|\right|_F \\
&\qquad + \qquad (n_2 n_3)^{-r} \left|\left| \left(\frac{n_2}{n_3}\right)^{-r} P_{32}L_2 P_{21} - \left(\frac{n_2}{n_3}\right)^{r} L_3 P_{32}P_{21} \right|\right|_F
\end{aligned}
$$

By Lemma 3.2.1,

$$
\begin{aligned}
\tilde{D}_r(G_1, G_3) \leq & \inf_{P_{32}, P_{21} | \mathcal{C}(P_{32}) \wedge \mathcal{C}(P_{21})} (n_1 n_2)^{-r} \left\| \left(\frac{n_1}{n_2}\right)^{-r} P_{21} L_1 - \left(\frac{n_1}{n_2}\right)^{r} L_2 P_{21} \right\|_F \\
& + \quad (n_2 n_3)^{-r} \left\| \left(\frac{n_2}{n_3}\right)^{-r} P_{32} L_2 - \left(\frac{n_2}{n_3}\right)^{r} L_3 P_{32} \right\|_F \\
= & \inf_{P_{21} | \mathcal{C}(P_{21})} (n_1 n_2)^{-r} \left\| \left(\frac{n_1}{n_2}\right)^{-r} P_{21} L_1 - \left(\frac{n_1}{n_2}\right)^{r} L_2 P_{21} \right\|_F \\
& + \quad \inf_{P_{32} | \mathcal{C}(P_{32})} (n_2 n_3)^{-r} \left\| \left(\frac{n_2}{n_3}\right)^{-r} P_{32} L_2 - \left(\frac{n_2}{n_3}\right)^{r} L_3 P_{32} \right\|_F \\
= & \tilde{D}_r(G_1, G_2) + \tilde{D}_r(G_2, G_3)
\end{aligned}
$$

and so

$$
\tilde{D}_r(G_1, G_3) \leq \tilde{D}_r(G_1, G_2) + \tilde{D}_r(G_2, G_3)
$$

for any fixed $r \in \mathbb{R}$. □

3.4 Sparse-Diffusion Distance

Recall that we use the notation $\mathcal{C}(P)$ for a constraint predicate that must be satisfied by prolongation matrix P, which is transitive in the sense that:

$$
\mathcal{C}(P_{32}) \wedge \mathcal{C}(P_{21}) \implies \mathcal{C}(P_{32} P_{21}). \tag{3.12}
$$

The simplest example is $\mathcal{C}(P) = \mathcal{C}_{\text{orthog}}(P) \equiv (P^T P = I)$. Let $\text{degree}_{i,j}(M)$ is the total number of nonzero entries in row i or column j of M. Sparsity can be introduced in transitive form by $\mathcal{C}(P) = \mathcal{C}_{\text{orthog}}(P) \wedge C_{\text{sparsity}}(P)$ where

$$
\mathcal{C}_{\text{sparsity}}(P) \equiv \left(\max_{i,j} \text{degree}_{i,j}(P) \leq (n_{P\text{coarse}} / n_{P\text{fine}})^{-} s \right)
$$

for some real number $s \geq 0$. Here, $n_{P\text{fine}}$ and $n_{P\text{coarse}}$ are the dimensions of P. This predicate is transitive since

$$\max_{i,j} \text{degree}_{i,j}(P_{32}P_{21}) \leq \left(\max_{i,j} \text{degree}_{i,j}(P_{32}) \right) \left(\max_{i,j} \text{degree}_{i,j}(P_{21}) \right),$$

and since n_2 cancels out from the numerator and denominator of the product of the fanout bounds. This transitive sparsity constraint depends on a power-law parameter $s \geq 0$. When $s = 0$, there is no sparsity constraint.

Another form of sparsity constraints are those which specify a pattern on matrix entries which are allowed to be nonzero. Two simple examples (which are also transitive) are matrices which are constrained to be upper triangular, as well as matrices which are constrained to be of the form $A \otimes B$ where A and B are themselves both constrained to be sparse. More complicated are $n_1 \times n_2$ matrices which are constrained to be banded for some specified pattern of bands: more specifically, that there is a reordering of the rows and columns that the number of diagonal bands (of width 1, slope $\frac{n_1}{n_2}$) with nonzero entries is less than $\left(\frac{n_1}{n_2}\right)^q$ for some $0 \leq q < 1$. For example, linear interpolation matrices between d-dimensional grids, with non-overlapping source regions, follow this constraint.

As a final note on sparsity, we observe that any of the optimizations detailed in this work could also be performed including a sparsity term (for example, the $|\cdot|_1$-norm of the matrix P, calculated as $\sum_i \sum_j |p_{ij}|$ is one possibility, as are terms which penalize t or α far from 1), rather than explicit sparsity constraints. A potential method of performing this optimization would be to start by optimizing the non-sparse version of the objective function (as detailed in Section 4.1) and then slowly increasing the strength of the regularization term.

3.5 Upper Bounds for Graph Products (Linear Version)

We next consider the problem of finding optimal prolongations between two graphs $\mathbf{G}_{\square}^{(1)} = G_1^{(1)} \square G_1^{(2)}$ and $\mathbf{G}_{\square}^{(2)} = G_2^{(1)} \square G_2^{(2)}$ when optimal prolongations are known between $G_1^{(1)}$ and $G_2^{(1)}$, and $G_1^{(2)}$ and $G_2^{(2)}$. We show that under some reasonable assumptions, these two prolongation optimizations decouple - we may thus solve them separately and combine the solutions to obtain the optimal prolongations between the two product graphs.

From the definition of graph box product, we have

$$\begin{aligned}
L_{\square}^{(j)} &= L(G_1^{(j)} \square G_2^{(j)}) \\
&= A(G_1^{(j)} \square G_2^{(j)}) - D(G_1^{(j)} \square G_2^{(j)}) \\
&= \left(A(G_1^{(j)}) \otimes I_2^{(j)} + I_1^{(j)} \otimes A(G_2^{(j)}) \right) - \left(D(G_1^{(j)}) \otimes I_2^{(j)} + I_1^{(j)} \otimes D(G_2^{(j)}) \right) \\
&= \left(A(G_1^{(j)}) \otimes I_2^{(j)} - D(G_1^{(j)}) \otimes I_2^{(j)} \right) - \left(I_1^{(j)} \otimes A(G_2^{(j)}) - I_1^{(j)} \otimes D(G_2^{(j)}) \right) \\
&= (L_1^{(j)} \otimes I_2^{(j)}) + (I_1^{(j)} \otimes L_2^{(j)}) \\
&= L(G_1^{(j)}) \oplus L(G_2^{(j)})
\end{aligned}$$

where $\oplus$ is the Kronecker sum of matrices as previously defined. See [34], Item 3.4 for more details on Laplacians of graph products. We calculate

$$
\begin{aligned}
D^{P,\alpha}\left(G_{\square}^{(1)}, G_{\square}^{(2)}\right) &= \left|\left|\frac{1}{\sqrt{\alpha}} P L_{\square}^{(1)} - \sqrt{\alpha} L_{\square}^{(2)} P\right|\right|_F \\
&= \left|\left|\frac{1}{\sqrt{\alpha}} P \left(\left(L_1^{(1)} \otimes I_2^{(1)}\right) + \left(I_1^{(1)} \otimes L_2^{(1)}\right)\right)\right.\right. \\
&\qquad \left.\left. -\sqrt{\alpha}\left(\left(L_1^{(2)} \otimes I_2^{(2)}\right) + \left(I_1^{(2)} \otimes L_2^{(2)}\right)\right) P\right|\right|_F \\
&= \left|\left|\left(\frac{1}{\sqrt{\alpha}} P \left(L_1^{(1)} \otimes I_2^{(1)}\right) - \sqrt{\alpha}\left(L_1^{(2)} \otimes I_2^{(2)}\right) P\right)\right.\right. \\
&\qquad \left.\left. + \left(\frac{1}{\sqrt{\alpha}} P \left(I_1^{(1)} \otimes L_2^{(1)}\right) - \sqrt{\alpha}\left(I_1^{(2)} \otimes L_2^{(2)}\right) P\right)\right|\right|_F
\end{aligned}
$$

Now we try out the assumption that $P = P_1 \otimes P_2$, which restricts the search space over P and may increase the objective function:

$$
\begin{aligned}
D^{P,\alpha}\left(G_{\square}^{(1)}, G_{\square}^{(2)}\right) &= \left|\left|\left[\frac{1}{\sqrt{\alpha}} (P_1 \otimes P_2)\left(L_1^{(1)} \otimes I_2^{(1)}\right)\right.\right.\right. \\
&\qquad \left. - \sqrt{\alpha}\left(L_1^{(2)} \otimes I_2^{(2)}\right)(P_1 \otimes P_2)\right] \\
&\quad + \left[\frac{1}{\sqrt{\alpha}} (P_1 \otimes P_2)\left(I_1^{(1)} \otimes L_2^{(1)}\right)\right. \\
&\qquad \left.\left. -\sqrt{\alpha}\left(I_1^{(2)} \otimes L_2^{(2)}\right)(P_1 \otimes P_2)\right]\right|\right|_F \\
&= \left|\left|\left(\frac{1}{\sqrt{\alpha}}\left(P_1 L_1^{(1)} \otimes P_2\right) - \sqrt{\alpha}\left(L_1^{(2)} P_1 \otimes P_2\right)\right)\right.\right. \\
&\qquad \left.\left. + \left(\frac{1}{\sqrt{\alpha}}\left(P_1 \otimes P_2 L_2^{(1)}\right) - \sqrt{\alpha}\left(P_1 \otimes L_2^{(2)} P_2\right)\right)\right|\right|_F \\
&= \left|\left|\left(\left(\frac{1}{\sqrt{\alpha}} P_1 L_1^{(1)} - \sqrt{\alpha} L_1^{(2)} P_1\right) \otimes P_2\right)\right.\right. \\
&\qquad \left.\left. + \left(P_1 \otimes \left(\frac{1}{\sqrt{\alpha}} P_2 L_2^{(1)} - \sqrt{\alpha} L_2^{(2)} P_2\right)\right)\right|\right|_F
\end{aligned}
$$

Since $||A+B||_F \leq ||A||_F + ||B||_F$,

$$\begin{aligned}
&\leq \left|\left|\left(\left(\frac{1}{\sqrt{\alpha}}P_1 L_1^{(1)} - \sqrt{\alpha}L_1^{(2)}P_1\right)\otimes P_2\right)\right| \\
&\quad + \left|\left|\left(P_1 \otimes \left(\frac{1}{\sqrt{\alpha}}P_2 L_2^{(1)} - \sqrt{\alpha}L_2^{(2)}P_2\right)\right)\right|\right|_F \\
&= \left|\left|\frac{1}{\sqrt{\alpha}}P_1 L_1^{(1)} - \sqrt{\alpha}L_1^{(2)}P_1\right|\right|_F ||P_2||_F \\
&\quad + ||P_1||_F \left|\left|\frac{1}{\sqrt{\alpha}}P_2 L_2^{(1)} - \sqrt{\alpha}L_2^{(2)}P_2\right|\right|_F,
\end{aligned}$$

Thus assuming $P = P_1 \otimes P_2$

$$\begin{aligned}
D^{P,\alpha}\left(G_\square^{(1)}, G_\square^{(2)}\right) \leq &\left|\left|\tilde{P}_2\right|\right|_F D_{\alpha,P_1}\left(G_1^{(1)}, G_1^{(2)}\right) \\
&+ \left|\left|\tilde{P}_1\right|\right|_F D_{\alpha,P_2}\left(G_2^{(1)}, G_2^{(2)}\right),
\end{aligned}$$

which is a weighted sum of objectives of the optimizations for prolongation from $G_1^{(1)}$ to $G_1^{(2)}$ and $G_2^{(1)}$ to $G_2^{(2)}$. Recall that our original constraint on P was that $P^T P = I$; since $P = P_1 \otimes P_2$ this is equivalent (by a property of the Kronecker product; see Corollary 13.8 in [64]) to the coupled constraints on P_1 and P_2:

$$\left({P_1}^T P_1 = \frac{1}{\eta} I_1^{(1)}\right) \qquad \wedge \qquad \left({P_2}^T P_2 = \eta I_2^{(1)}\right) \tag{3.13}$$

for some $\eta \in \mathbb{R}$. For any P_1, P_2 which obey 3.13, we may rescale them by η to make them orthogonal without changing the value of the objective, so we take $\eta = 1$ in subsequent calculations. Noting that $||A||_F = \sqrt{\mathrm{Tr}(A^T A)}$, we see that

$$||P_1||_F = \sqrt{\mathrm{Tr}(I_1^{(1)})} = \sqrt{n_1^{(1)}} \quad \text{and similarly} \quad ||P_2||_F = \sqrt{n_2^{(1)}}.$$

Thus, we have proven the following:

Theorem 3.5.1. *Assuming that P decomposes as $P = P_1 \otimes P_2$, the diffusion distance $D_{P,\alpha}\left(G_\square^{(1)}, G_\square^{(2)}\right)$ between $G_\square^{(1)}$ and $G_\square^{(2)}$ is bounded above by the strictly monotonically increasing function of the two distances $D_{P_1,\alpha}$ and $D_{P_2,\alpha}$:*

$$\mathcal{F}(D_{P_1,\alpha}, D_{P_2,\alpha}) = \sqrt{n_2^{(1)}} D_{P_1,\alpha} + \sqrt{n_1^{(1)}} D_{P_2,\alpha},$$

Namely,

$$D_{P,\alpha}\left(G_\square^{(1)}, G_\square^{(2)}\right) \leq \mathcal{F}\left(D_{P_1,\alpha}\left(G_1^{(1)}, G_1^{(2)}\right), D_{P_2,\alpha}\left(G_2^{(1)}, G_2^{(2)}\right)\right)$$

Thus, the original optimization over the product graphs decouples into separate optimizations over the two sets of factors, constrained to have the same value of α. Additionally, since the requirement that $P = P_1 \otimes P_2$ is an additional constraint,

Corollary 3.5.2. *If (α_1, P_1) and (α_2, P_2), subject to orthogonality constraints, are optima of $D_{\alpha,P}\left(G_1^{(1)}, G_1^{(1)}\right)$ and $D_{\alpha,P}\left(G_2^{(1)}, G_2^{(1)}\right)$, and furthermore if $\alpha_1 = \alpha_2$, then the value of $D_{P,\alpha}(G_1^{(1)} \square G_2^{(1)}, G_1^{(2)} \square G_2^{(2)})$ for an optimal P is bounded above by $D_{P_1 \otimes P_2, \alpha_1}(G_1^{(1)} \square G_2^{(1)}, G_1^{(2)} \square G_2^{(2)})$.*

This upper bound on the original objective function is a monotonically increasing function of the objectives for the two smaller problems. A consequence of this upper bound is that if $D_{P_1,\alpha}\left(G_1^{(1)}, G_1^{(2)}\right) \leq \epsilon_1$ and $D_{P_2,\alpha}\left(G_2^{(1)}, G_2^{(2)}\right) \leq \epsilon_2$, then the composite solution $P_1 \otimes P_2$ must have $D_{P_1 \otimes P_2,\alpha}\left(G_\square^{(1)}, G_\square^{(2)}\right) \leq \epsilon = \left(\sqrt{n_1} + \sqrt{n_2}\right) \max(\epsilon_1, \epsilon_2)$. Thus if both of these distances are arbitrarily small then the composite distance must also be small. Furthermore, if only one of these is small, so that $D_{P_1,\alpha}\left(G_1^{(1)}, G_1^{(2)}\right) \approx 0$ or $D_{P_2,\alpha}\left(G_2^{(1)}, G_2^{(2)}\right) \approx 0$, then $D_{P_1 \otimes P_2,\alpha} \approx D_{P_2,\alpha}$ or $D_{P_1 \otimes P_2,\alpha} \approx D_{P_1,\alpha}$, respectively.

We have experimentally found that many families of graphs do not require scaling between the two diffusion processes: the optimal (α, P) pair has $\alpha = 1$. In particular, prolongation between path (cycle) graphs of size n and size $2n$ always have $\alpha_{\text{optimal}} = 1$, since the spectrum

of the former graph is a subset of that of the larger - therefore, there is a matching solution of cost 0 which by the argument above can be mapped to a graph-space P with objective function value 0 (we prove this in Section 3.7). In this case, the two terms of the upper bound are totally decoupled and may each be optimized separately (whereas in the form given above, they both depend on a α).

3.6 Upper Bounds for Graph Products (Exponential Version)

We now consider the case where we want to compute the distance of two graph box products, i.e. $D\left(\mathbf{G}_1, \mathbf{G}_2\right)$ where

$$\mathbf{G}_1 = G_1^{(1)} \square G_1^{(2)} \quad \text{and} \quad \mathbf{G}_2 = G_2^{(1)} \square G_2^{(2)} \tag{3.14}$$

and

$$\begin{aligned} P^{(1)} &= \arg \inf_{P_c | \mathcal{C}(P_c)} D\left(G_1^{(1)}, G_2^{(1)} | t_c, \alpha_c, P_c\right) \\ P^{(2)} &= \arg \inf_{P_c | \mathcal{C}(P_c)} D\left(G_1^{(2)}, G_2^{(2)} | t_c, \alpha_c, P_c\right) \end{aligned} \tag{3.15}$$

are known for some t_c, α_c.

Theorem 3.6.1. *Let $\mathbf{G}_1$ and $\mathbf{G}_2$ be graph box products as described above, and for a graph G let $L(G)$ be its Laplacian. For fixed $t = t_c$, $\alpha = \alpha_c$, $P^{(i)}$ as given above, for any $\lambda \in [0, 1]$,*

we have

$$\begin{aligned}\inf_{P_c|\mathcal{C}(P_c)} D\left(\mathbf{G}_1, \mathbf{G}_2\right) \quad \leq \\ \lambda\left(\left|\left|e^{\frac{t_c}{\alpha_c}L(G_1^{(2)})}\right|\right|_F + \left|\left|e^{t_c\alpha_c L(G_2^{(2)})}\right|\right|_F\right) D\left(G_1^{(1)}, G_2^{(1)}|P^{(1)}\right) \\ + (1-\lambda)\left(\left|\left|e^{\frac{t_c}{\alpha_c}L(G_1^{(1)})}\right|\right|_F + \left|\left|e^{t_c\alpha_c L(G_2^{(1)})}\right|\right|_F\right) D\left(G_1^{(2)}, G_2^{(2)}|P^{(2)}\right)\end{aligned} \tag{3.16}$$

where all distances are evaluated at $t = t_c$, $\alpha = \alpha_c$, but we have omitted that notation for simplicity.

Proof. For graph products $\mathbf{G}_i$, we have

$$\begin{aligned}L(\mathbf{G}_i) &= L(G_i^{(1)}) \oplus L(G_i^{(2)}) \\ &= \left(L(G_i^{(1)}) \otimes I_{\left|L(G_i^{(2)})\right|}\right) + \left(I_{\left|L(G_i^{(1)})\right|} \otimes L(G_i^{(2)})\right)\end{aligned} \tag{3.17}$$

(this fact can be easily verified from the formula for the adjacency matrix of a graph box product, given in the definition in Section 1.4.2), and so

$$\exp\left[cL(\mathbf{G}_i)\right] = \exp\left[c\left(L(G_i^{(1)}) \otimes I_{\left|L(G_i^{(2)})\right|}\right) + \left(I_{\left|L(G_i^{(1)})\right|} \otimes L(G_i^{(2)})\right)\right]. \tag{3.18}$$

Because $A \otimes I_{|B|}$ and $I_{|A|} \otimes B$ commute for any A and B,

$$\begin{aligned}\exp\left[cL(\mathbf{G}_i)\right] &= \exp\left[c\left(L(G_i^{(1)}) \otimes I_{\left|L(G_i^{(2)})\right|}\right)\right] \exp\left[c\left(I_{\left|L(G_i^{(1)})\right|} \otimes L(G_i^{(2)})\right)\right] \\ &= \left(\exp\left[cL(G_i^{(1)})\right] \otimes I_{\left|L(G_i^{(2)})\right|}\right)\left(I_{\left|L(G_i^{(1)})\right|} \otimes \exp\left[cL(G_i^{(2)})\right]\right) \\ &= \exp\left[cL(G_i^{(1)})\right] \otimes \exp\left[cL(G_i^{(2)})\right]\end{aligned} \tag{3.19}$$

We will make the following abbreviations:

$$\begin{aligned}
\mathbf{E}_1 &= e^{\frac{t_c}{\alpha_c}L(\mathbf{G}_1)} \quad E_1^{(1)} = e^{\frac{t_c}{\alpha_c}L(G_1^{(1)})} \quad E_1^{(2)} = e^{\frac{t_c}{\alpha_c}L(G_1^{(2)})} \\
\mathbf{E}_2 &= e^{t_c\alpha_c L(\mathbf{G}_2)} \quad E_2^{(1)} = e^{t_c\alpha_c L(G_2^{(1)})} \quad E_2^{(2)} = e^{t_c\alpha_c L(G_2^{(2)})}
\end{aligned}$$

Then,

$$\begin{aligned}
\inf_{P|\mathcal{C}(P)} D\left(\mathbf{G}_1, \mathbf{G}_2\right) &\leq D\left(\mathbf{G}_1, \mathbf{G}_2 | P^{(1)} \otimes P^{(2)}\right) \qquad (3.20)\\
&= \left|\left|\left(P^{(1)} \otimes P^{(2)}\right)\mathbf{E}_1 - \mathbf{E}_2\left(P^{(1)} \otimes P^{(2)}\right)\right|\right|_F
\end{aligned}$$

$$\begin{aligned}
&= \left|\left|\left(P^{(1)} \otimes P^{(2)}\right)\left(E_1^{(1)} \otimes E_1^{(2)}\right) - \left(E_2^{(1)} \otimes E_2^{(2)}\right)\left(P^{(1)} \otimes P^{(2)}\right)\right|\right|_F \\
&= \left|\left|\left(P^{(1)}E_1^{(1)} \otimes P^{(2)}E_1^{(2)}\right) - \left(E_2^{(1)}P^{(1)} \otimes E_2^{(2)}P^{(2)}\right)\right|\right|_F^2 \\
&= \left|\left|\left(P^{(1)}E_1^{(1)} \otimes P^{(2)}E_1^{(2)}\right) - \left(P^{(1)}E_1^{(1)} \otimes E_2^{(2)}P^{(2)}\right)\right.\right. \qquad (3.21)\\
&\quad \left.\left. + \left(P^{(1)}E_1^{(1)} \otimes E_2^{(2)}P^{(2)}\right) - \left(E_2^{(1)}P^{(1)} \otimes E_2^{(2)}P^{(2)}\right)\right|\right|_F \\
&\leq \left|\left|\left(P^{(1)}E_1^{(1)} \otimes P^{(2)}E_1^{(2)}\right) - \left(P^{(1)}E_1^{(1)} \otimes E_2^{(2)}P^{(2)}\right)\right|\right|_F \\
&\quad + \left|\left|\left(P^{(1)}E_1^{(1)} \otimes E_2^{(2)}P^{(2)}\right) - \left(E_2^{(1)}P^{(1)} \otimes E_2^{(2)}P^{(2)}\right)\right|\right|_F \\
&= \left|\left|P^{(1)}E_1^{(1)} \otimes \left(P^{(2)}E_1^{(2)} - E_2^{(2)}P^{(2)}\right)\right|\right|_F \qquad (3.22)\\
&\quad + \quad \left|\left|\left(P^{(1)}E_1^{(1)} - E_2^{(1)}P^{(1)}\right) \otimes E_2^{(2)}P^{(2)}\right|\right|_F \\
&= \left|\left|P^{(1)}E_1^{(1)}\right|\right|_F \left|\left|P^{(2)}E_1^{(2)} - E_2^{(2)}P^{(2)}\right|\right|_F \qquad (3.23)\\
&\quad + \quad \left|\left|P^{(1)}E_1^{(1)} - E_2^{(1)}P^{(1)}\right|\right|_F \left|\left|E_2^{(2)}P^{(2)}\right|\right|_F.
\end{aligned}$$

By Lemma 3.2.1,

$$\begin{aligned}
\inf_{P|\mathcal{C}(P)} D\left(\mathbf{G}_1, \mathbf{G}_2\right) &\leq \left|\left|E_1^{(1)}\right|\right|_F \left|\left|P^{(2)}E_1^{(2)} - E_2^{(2)}P^{(2)}\right|\right|_F \\
&\quad + \quad \left|\left|P^{(1)}E_1^{(1)} - E_2^{(1)}P^{(1)}\right|\right|_F \left|\left|E_2^{(2)}\right|\right|_F.
\end{aligned} \qquad (3.24)$$

If we instead use $\left(E_2^{(1)}P^{(1)} \otimes P^{(2)}E_1^{(2)}\right)$ as the cross term in Equation (3.21), we have

$$\begin{aligned}\inf_P D\left(\mathbf{G}_1, \mathbf{G}_2\right) \leq & \left|\left|E_2^{(1)}\right|\right|_F \left|\left|P^{(2)}E_1^{(2)} - E_2^{(2)}P^{(2)}\right|\right|_F \\ + & \left|\left|P^{(1)}E_1^{(1)} - E_2^{(1)}P^{(1)}\right|\right|_F \left|\left|E_1^{(2)}\right|\right|_F\end{aligned} \tag{3.25}$$

A linear combination of these two bounds gives us the desired bound. □

This has the additional consequence that

$$\begin{aligned}&\inf_{P_c|\mathcal{C}(P_c)} D\left(\mathbf{G}_1, \mathbf{G}_2\right) \quad \leq \\ &\min\left[\left(\left|\left|e^{\frac{t_c}{\alpha_c}L(G_1^{(2)})}\right|\right|_F + \left|\left|e^{t_c\alpha_c L(G_2^{(2)})}\right|\right|_F\right) D\left(G_1^{(1)}, G_2^{(1)}|P^{(1)}\right),\right. \\ &\left.\left(\left|\left|e^{\frac{t_c}{\alpha_c}L(G_1^{(1)})}\right|\right|_F + \left|\left|e^{t_c\alpha_c L(G_2^{(1)})}\right|\right|_F\right) D\left(G_1^{(2)}, G_2^{(2)}|P^{(2)}\right)\right]\end{aligned} \tag{3.26}$$

Additionally, if

$$E_i^{(1)} = E_i^{(2)} \text{ for } i \in 1, 2 \quad \text{and} \quad P^{(1)} = P^{(2)}, \tag{3.27}$$

This reduces further to

$$D\left(\mathbf{G}_1, \mathbf{G}_2|P^{(1)} \otimes P^{(1)}\right) \leq \min\left(\left|\left|E_1^{(1)}\right|\right|_F, \left|\left|E_2^{(1)}\right|\right|_F\right) \left|\left|P^{(1)}E_1^{(1)} - E_2^{(1)}P^{(1)}\right|\right|_F \tag{3.28}$$

and so

$$\begin{aligned}&D\left(G_1^{(1)}\square G_1^{(1)}, G_2^{(1)}\square G_2^{(1)}\middle| t_c, \alpha_c\right) \\ &\quad \leq \min\left(\left|\left|e^{\frac{t_c}{a_c}L(G_1^{(1)})}\right|\right|_F, \left|\left|e^{t_c a_c L(G_2^{(1)})}\right|\right|_F\right) D\left(G_1^{(1)}, G_2^{(1)} \mid t_c, \alpha_c\right)\end{aligned} \tag{3.29}$$

An example of such a graph sequence is the sequence of two-dimensional square grids, which are each the box product of two identical one-dimensional grids i.e., path graphs: $\mathrm{Sq}_n =$

$\mathrm{Pa}_n \square \mathrm{Pa}_n$.

3.7 Existence of Zero-Error P for Cycle Graphs

Theorem 3.7.1. *In this section, we give an example closed-form solution for a prolongation matrix which achieves zero error when prolonging between cycle graphs. Let $G^{(1)}$ and $G^{(2)}$ be graphs with spectra $\lambda_i^{(1)}$ and $\lambda_j^{(2)}$, respectively, with $n_1 = \left|G^{(1)}\right| \leq n_2 = \left|G^{(2)}\right|$. Suppose that for every $\lambda_i^{(1)} = r$ of multiplicity k, r is also an eigenvalue of $G^{(2)}$ of multiplicity $\geq k$. Then there is a zero-cost eigenvalue matching M between $G^{(1)}$ and $G^{(2)}$.*

Proof. Let $(i_1, j_1), (i_1, j_1) \ldots (i_{n_1}, j_{n_1})$ be a list of pairs of indices such that the following hold:

- All of the i_k are unique.
- All of the j_k are unique.
- For any pair (i_k, j_k), $\lambda_k^{(1)} = \lambda_k^{(2)}$.

Define P as follows:

$$P_{ij} = \begin{cases} 1 & (i,j) \text{ appears in the above list.} \\ 0 & \text{else.} \end{cases}$$

P is clearly orthogonal, since it has exactly one 1 in each row and each column and zeros elsewhere (P is a permutation matrix for $n_1 = n_2$ and a *subpermutation* matrix otherwise). Furthermore, we must have

$$\sum_{i=1}^{n_2} \sum_{j=1}^{n_1} P_{ij} \left(\lambda_j^{(1)} - \lambda_i^{(2)} \right) = 0$$

and therefore

$$\left|\left|P\Lambda^{(1)} - \Lambda^{(2)}P\right|\right|_F = 0 \tag{3.30}$$

□

Corollary 3.7.2. *For any n, there exist zero-error matchings between cycle graphs C_n and C_{2n}.*

Proof. The spectra of C_n are given by the formula (see [52] Section 39.3):

$$\lambda(C_n) = 2\cos(\frac{2\pi j}{n}) \qquad \text{for} \qquad (j = 0, 1, \ldots n-1)$$

Thus $\lambda(C_n)$ and $\lambda(C_{2n})$ clearly satisfy the conditions of Theorem 3.7.1 above. In particular, the matrix

$$P_{ij} = \begin{cases} 1 & \text{if} \quad i = 2j \\ 0 & \text{else.} \end{cases}$$

has 0 cost as in Equation 3.30 □

3.8 Spectral Version of Decoupling for the Diffusion Term of Graph Product Prolongations

In this section we derive an eigenspace version of the bound derived in Section 3.5.

Theorem 3.8.1. *The eigenspace version of the diffusion term of the objective function of a graph product prolongation also decouples into two smaller prolongation problems.*

Proof. From Theorem 3.5.1 of the main manuscript, we know that for

$$\mathbf{G}_{\square}^{(1)} = G_1^{(1)} \square G_1^{(2)} \qquad \text{and} \qquad \mathbf{G}_{\square}^{(2)} = G_2^{(1)} \square G_2^{(2)},$$

and assuming $P = P_1 \otimes P_2$,

$$\begin{aligned} D^{P,\alpha}\left(G_{\square}^{(1)}, G_{\square}^{(2)}\right) &= \left|\left|PL_{\square}^{(1)} - L_{\square}^{(2)}P\right|\right|_F \\ &\leq \sqrt{n_2^{(1)}} D_{P_1,\alpha}\left(G_1^{(1)}, G_1^{(2)}\right) + \sqrt{n_1^{(1)}} D_{P_2,\alpha}\left(G_2^{(1)}, G_2^{(2)}\right) \\ &= \sqrt{n_2^{(1)}} \left|\left|\frac{1}{\sqrt{\alpha}} P_1 L_1^{(1)} - \sqrt{\alpha} L_1^{(2)} P_1\right|\right|_F \\ &\quad + \sqrt{n_1^{(1)}} \left|\left|\frac{1}{\sqrt{\alpha}} P_2 L_2^{(1)} - \sqrt{\alpha} L_2^{(2)} P_2\right|\right|_F, \end{aligned}$$

Trivially, we can rewrite each of these Frobenius norms to be their spectral version, as in Equation 3.1. Thus,

$$\begin{aligned} \left|\left|PL_{\square}^{(1)} - L_{\square}^{(2)}P\right|\right|_F &= \left|\left|\tilde{P}\Lambda_{\square}^{(1)} - \Lambda_{\square}^{(2)}\tilde{P}\right|\right|_F \\ &\leq \sqrt{n_2^{(1)}} \left|\left|\frac{1}{\sqrt{\alpha}} \tilde{P}_1 \Lambda_1^{(1)} - \sqrt{\alpha} \Lambda_1^{(2)} \tilde{P}_1\right|\right|_F \\ &\quad + \sqrt{n_1^{(1)}} \left|\left|\frac{1}{\sqrt{\alpha}} \tilde{P}_2 \Lambda_2^{(1)} - \sqrt{\alpha} \Lambda_2^{(2)} \tilde{P}_2\right|\right|_F, \end{aligned}$$

which is a weighted sum of objectives of the two spectral prolongation problems for the two factor lineages. We have thus also decoupled this eigenvalue-matching version of the objective function into two separate prolongation problems. □

Finally, we show that if $\tilde{P}_1$ and $\tilde{P}_2$ are solutions to the eigenvalue matching problem $m*(L_1^{(1)}, L_1^{(2)})$ and $m*(L_2^{(1)}, L_2^{(2)})$ respectively, then $\tilde{P} = \tilde{P}_1 \otimes \tilde{P}_2$ is a valid, but not necessarily

optimal, solution to the eigenvalue matching problem $m*(L_{\square}^{(1)}, L_{\square}^{(2)})$. By valid we mean that $\tilde{P}$ satisfies the constraints given in the definition of matching problems in Section 1.4.2.

Proof. This fact follows directly from the constraints on $\tilde{P}_1$ and $\tilde{P}_2$. A matrix M is a valid matching matrix iff its entries are in $\{0, 1\}$ and it is orthogonal (this is an equivalent expression of the constraints given in Section 1.4.2. If $\tilde{P}_1$ and $\tilde{P}_2$ are both orthogonal and $\{0, 1\}$-valued, then we observe the following facts about their Kronecker product $\tilde{P}_1 \otimes \tilde{P}_2$:

- it is also $\{0, 1\}$-valued, since any of its entries is the product of an entry of $\tilde{P}_1$ and one of $\tilde{P}_2$.
- it is orthogonal, since $A \otimes B$ is orthogonal iff $A^T A = \zeta I$ and $B^T B = \frac{1}{\zeta} I$ for some $\zeta > 0$ [64]. $\tilde{P}_1$ and $\tilde{P}_2$ satisfy these conditions with $\zeta = 1$.

So $\tilde{P}_1 \otimes \tilde{P}_2$ satisfies the constraints given for the eigenvalue matching problem $m*(L_{\square}^{(1)}, L_{\square}^{(2)})$.

□

3.8.1 Distortion-penalized Distance

We can add a term to the graph diffusion distance which penalizes large distortions induced by α, as follows: define

$$D_{\text{reg}}(G_1, G_2) = \sup_t \inf_{P|\mathcal{C}(P)} \inf_{\alpha>0} \left\{ \left|\left| P e^{\frac{t}{\alpha} L_1} - e^{t\alpha L_2} P \right|\right|_F + \left|\left| e^{\frac{t}{\alpha} L_1} - e^{t L_1} \right|\right|_F + \left|\left| e^{t L_2} P - e^{t\alpha L_2} P \right|\right|_F \right\}$$

We can show analytically that this distance satisfies the triangle inequality:

Theorem 3.8.2. *D_{reg} satisfies the triangle inequality.*

Proof. For graphs G_1, G_2, G_3 and Laplacians L_1, L_2, L_3, for any fixed $t \geq 0$, we have:

$$\begin{aligned}
D_{\text{reg}}(G_1, G_3|t) = & \inf_{P|\mathcal{C}(P)} \inf_{\alpha>0} \Big\{ \Big|\Big| P e^{\frac{t}{\alpha} L_1} - e^{t\alpha L_3} P \Big|\Big|_F + \Big|\Big| e^{\frac{t}{\alpha} L_1} - e^{t L_1} \Big|\Big|_F \\
& + \big|\big| e^{tL_3} P - e^{t\alpha L_3} P \big|\big|_F \Big\} \\
\leq & \, D_{\text{reg}}(G_1, G_3|t, \alpha = 1) \\
= & \inf_{P|\mathcal{C}(P)} \big\{ \big|\big| P e^{tL_1} - e^{tL_3} P \big|\big|_F + \big|\big| e^{tL_1} - e^{tL_1} \big|\big|_F \\
& + \big|\big| e^{tL_3} P - e^{tL_3} P \big|\big|_F \big\} \\
= & \inf_{P|\mathcal{C}(P)} \big|\big| P e^{tL_1} - e^{tL_3} P \big|\big|_F
\end{aligned}$$

Suppose that

$$\begin{aligned}
\alpha_{32}, P_{32} = \arg \inf_{a>0} \inf_{P|\mathcal{C}(P)} & \Big\{ \Big|\Big| P e^{\frac{t}{\alpha} L_2} - e^{t\alpha L_3} P \Big|\Big|_F + \Big|\Big| e^{\frac{t}{\alpha} L_2} - e^{t L_2} \Big|\Big|_F \\
& + \big|\big| e^{tL_3} P - e^{t\alpha L_3} P \big|\big|_F \Big\} \\
\alpha_{21}, P_{21} = \arg \inf_{a>0} \inf_{P|\mathcal{C}(P)} & \Big\{ \Big|\Big| P e^{\frac{t}{\alpha} L_1} - e^{t\alpha L_2} P \Big|\Big|_F + \Big|\Big| e^{\frac{t}{\alpha} L_1} - e^{t L_1} \Big|\Big|_F \\
& + \big|\big| e^{tL_2} P - e^{t\alpha L_2} P \big|\big|_F \Big\}
\end{aligned}$$

Then,

$$\begin{aligned}
\inf_{P|\mathcal{C}(P)} \left|\left|Pe^{tL_1} - e^{tL_3}P\right|\right|_F &\leq \left|\left|P_{32}P_{21}e^{tL_1} - e^{tL_3}P_{32}P_{21}\right|\right|_F \\
\inf_{P|\mathcal{C}(P)} \left|\left|Pe^{tL_1} - e^{tL_3}P\right|\right|_F &\leq \Big|\Big|P_{32}P_{21}e^{tL_1} - P_{32}P_{21}e^{\frac{t}{\alpha_{21}}L_1} + P_{32}P_{21}e^{\frac{t}{\alpha_{21}}L_1} \\
&\quad - P_{32}e^{t\alpha_{21}L_2}P_{21} + P_{32}e^{t\alpha_{21}L_2}P_{21} - P_{32}e^{tL_2}P_{21} \\
&\quad + P_{32}e^{tL_2}P_{21} - P_{32}e^{\frac{t}{\alpha_{32}}L_2}P_{21} + P_{32}e^{\frac{t}{\alpha_{32}}L_2}P_{21} \\
&\quad - e^{t\alpha_{32}L_3}P_{32}P_{21} + e^{t\alpha_{32}L_3}P_{32}P_{21} - e^{tL_3}P_{32}P_{21}\Big|\Big|_F \\
&\leq \left|\left|P_{32}P_{21}e^{tL_1} - P_{32}P_{21}e^{\frac{t}{\alpha_{21}}L_1}\right|\right|_F \\
&\quad + \left|\left|P_{32}P_{21}e^{\frac{t}{\alpha_{21}}L_1} - P_{32}e^{t\alpha_{21}L_2}P_{21}\right|\right|_F \\
&\quad + \left|\left|P_{32}e^{t\alpha_{21}L_2}P_{21} - P_{32}e^{tL_2}P_{21}\right|\right|_F \\
&\quad + \left|\left|P_{32}e^{tL_2}P_{21} - P_{32}e^{\frac{t}{\alpha_{32}}L_2}P_{21}\right|\right|_F \\
&\quad + \left|\left|P_{32}e^{\frac{t}{\alpha_{32}}L_2}P_{21} - e^{t\alpha_{32}L_3}P_{32}P_{21}\right|\right|_F \\
&\quad + \left|\left|e^{t\alpha_{32}L_3}P_{32}P_{21} - e^{tL_3}P_{32}P_{21}\right|\right|_F
\end{aligned}$$

by Lemma 3.2.1,

$$\begin{aligned}
\inf_{P|\mathcal{C}(P)} \left|\left|Pe^{tL_1} - e^{tL_3}P\right|\right|_F &\leq \left|\left|e^{tL_1} - e^{\frac{t}{\alpha_{21}}L_1}\right|\right|_F + \left|\left|P_{21}e^{\frac{t}{\alpha_{21}}L_1} - e^{t\alpha_{21}L_2}P_{21}\right|\right|_F \\
&\quad + \left|\left|e^{t\alpha_{21}L_2}P_{21} - e^{tL_2}P_{21}\right|\right|_F \\
&\quad + \left|\left|e^{tL_2} - e^{\frac{t}{\alpha_{32}}L_2}\right|\right|_F + \left|\left|P_{32}e^{\frac{t}{\alpha_{32}}L_2} - e^{t\alpha_{32}L_3}P_{32}\right|\right|_F \\
&\quad + \left|\left|e^{t\alpha_{32}L_3}P_{32} - e^{tL_3}P_{32}\right|\right|_F \\
&= D_{\text{reg}}(G_1, G_2|t=c) + D_{\text{reg}}(G_2, G_3|t=c)
\end{aligned}$$

Since this is true for any fixed t, let

$$t^* = \arg\sup_t D_{\text{reg}}(G_1, G_3|t).$$

Then

$$\begin{aligned} D_{\text{reg}}(G_1, G_3) &= \sup_c D_{\text{reg}}(G_1, G_3|t) \\ &= D_{\text{reg}}(G_1, G_3|t^*) \\ &\leq D_{\text{reg}}(G_1, G_2|t^*) + D_{\text{reg}}(G_2, G_3|t = t^*) \\ &\leq \sup_{t_{21}} D_{\text{reg}}(G_1, G_2|t_{21}) + \sup_{t_{32}} D_{\text{reg}}(G_2, G_3|t_{32}) \\ &= D_{\text{reg}}(G_1, G_2) + D_{\text{reg}}(G_2, G_3) \end{aligned}$$

□

We can construct a similar regularized version of the linear objective function:

$$\tilde{D}_{\text{reg}}(G_1, G_2) = \left|\left|\frac{1}{\alpha} PL_1 - \alpha L_2 P\right|\right| + \left|\left|\frac{1}{\alpha} L_1 - L_1\right|\right| + \left|\left| PL_2 - \alpha L_2 P\right|\right|$$

The additional terms included in D_{reg} and $\tilde{D}_{\text{reg}}$ penalize α distorting the respective Laplacians far from their original values. In practice, many of the theoretical guarantees provided earlier in this manuscript may not apply to optimization of the augmented objective function. Hence, a major area of future work will be modification of our optimization procedure to compute this form of distance.

3.9 Theory Summary

Triangle inequalities are proven for some members of the proposed family of graph distortion or "distance" measures, including infinitesimal and finite diffusion time, a power law for sparsity, and/or a power law for the time scaling factor between coarse and fine scales. However, the case of an optimal (not power law) time conversion factor α needs to be investigated by numerical experiment, and that requires new algorithms, introduced in Section 4.3. We also show that in the case of distances between graph box products, optimization over P for the product graphs is bounded above by a monotonic function of the optimum over the component graphs.

Chapter 4

Efficiently Calculating GDD

In previous chapters we have introduced and defined diffusion distance. A crucial component of the optimization required to calculate GDD is the optimization of the α parameter for conversion between coarse and fine time scales. Optimizing α in addition to optimizing the prolongation matrix P under transitive constraints $\mathcal{C}(P)$, is a nontrivial numerical problem that in our experience seems to require new methods. We develop such methods here.

4.1 Algorithm Development

In this section, we describe the algorithm used to calculate upper bounds on graph distances as the joint optima (over P, t, and α) of the distance equations Equation 2.2 and Equation 2.3, under orthogonality constraints only, i.e. the case $\mathcal{C}(P) = \{P|P^TP = I\}$. At the core of both algorithms is a subroutine to solve the Linear Assignment Problem (LAP - see Equation (1.5)) repeatedly, in order to find the subpermutation matrix which is optimal at a particular value of α. Namely, we are interested in calculating $\tilde{D}$ as

$$\tilde{D}(G_1, G_2) = \min_{\alpha} f(\alpha) \quad \text{where} \quad f(\alpha) = \inf_{P|P^TP=I} \left|\left| \frac{1}{\alpha} PL(G_1) - \alpha L(G_2)P \right|\right| \tag{4.1}$$

which, for orthogonality or any other compact constraint

$$= \min_{P|P^TP=I} \left|\left| \frac{1}{\alpha} PL(G_1) - \alpha L(G_2)P \right|\right|.$$

However, we have found that the unique structure of this optimization problem admits a specialized procedure which is faster and more accurate than nested univariate optimization of α and t (where each innermost function evaluation consists of a full optimization over P at some t, α). We first briefly describe the algorithm used to find the optimal P and α for $\tilde{D}^2$. The formal description of the algorithm is given by Algorithm 1. In both cases, we reduce the computational complexity of the optimization over P by imposing the additional constraint that P must be a subpermutation matrix when rotated into the spectral basis (we define subpermutations in the proof of Theorem 4.4.1). This constraint is compatible with the orthogonality constraint (all subpermutation matrices are orthogonal, but not vice versa). The tradeoff of this reduction of computational complexity is that we can only guarantee that our optima are upper bounds of the optima over all orthogonal P. However, in practice, this bound seems to be tight: over a large number of empirical evaluations we did not find any example where orthogonally-constrained optimization was able to improve in objective function value over optimization constrained to subpermutation matrices. Therefore, we shall for the remainder of this paper refer to the optima calculated as distance values, when strictly they are distance upper bounds. We also note here that a distance lower bound is also possible to calculate by *relaxing* the constraints in $\mathcal{C}(P)$ (for instance, by replacing the optimization over all P with a less constrained matching problem - see Section 2.4).

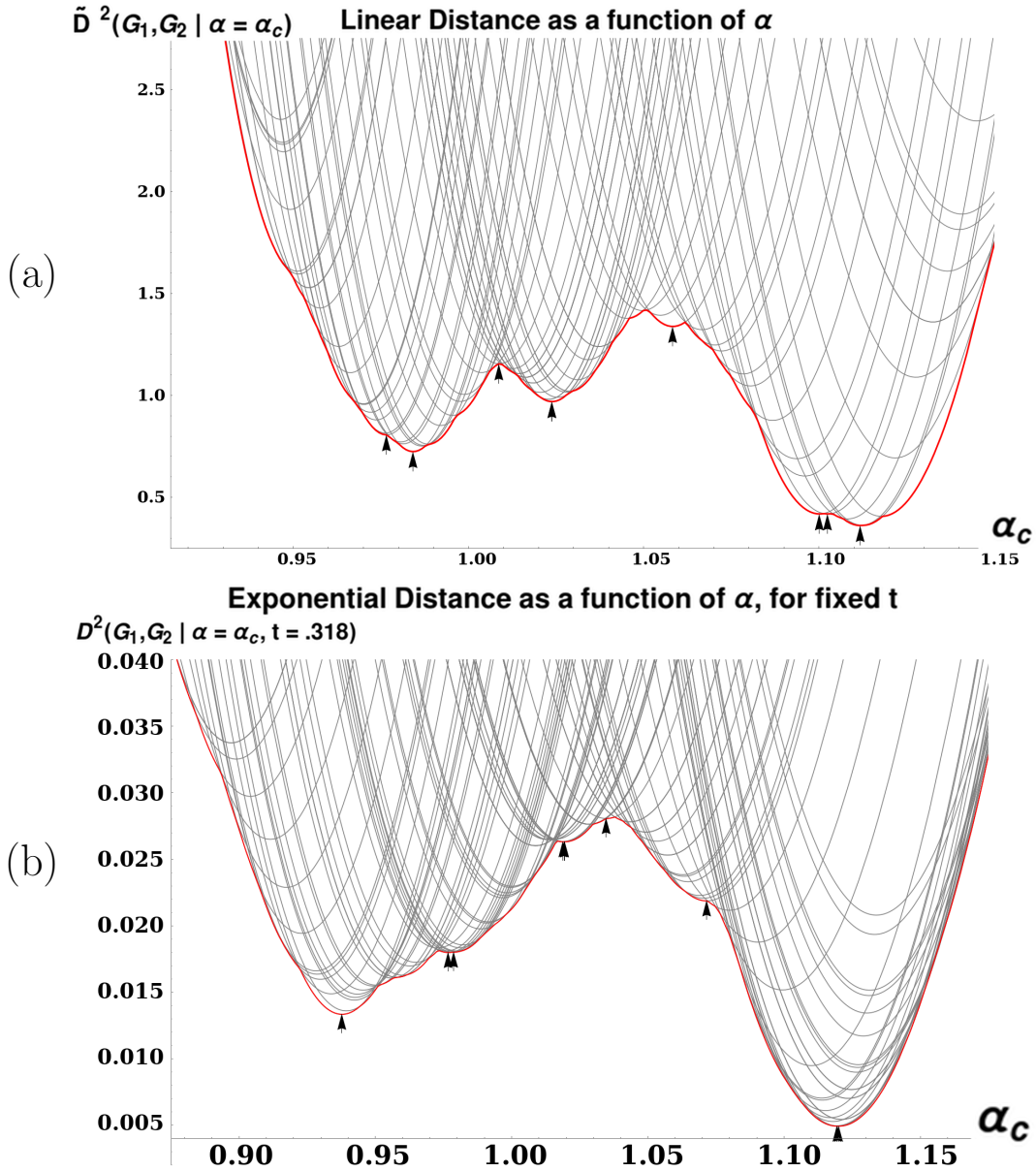

Figure 4.1: Two plots demonstrating characteristics of distance calculation between a (7×7) grid and an (8×8) grid.
(a): Plot illustrating the discontinuity and multimodality of the linear version of distance. Each gray curve represents a function $f_{P_c}(\alpha_c) = \tilde{D}^2\left(\mathrm{Sq}_7, \mathrm{Sq}_8 | \alpha_c, P_c\right)$. The thicker curve is the lower convex hull of the thinner curves as a function of α, that is: $\boldsymbol{f}(\alpha_c) = \inf_{P|\mathcal{C}(P)} \tilde{D}^2\left(\mathrm{Sq}_7, \mathrm{Sq}_8 | \alpha_c\right)$. We see that $\boldsymbol{f}(\alpha)$ is continuous, but has discontinuous slope, as well as several local minima (marked by arrowheads - note in this plot some of these look like inflection points or maxima because they are obscured by the red line, but the are all local *minima*). These properties make $\tilde{D}$ difficult to optimize, necessitating the development of Algorithm 1.
(b): As in (a), but with $D^2\left(\mathrm{Sq}_7, \mathrm{Sq}_8 | t = .318\right)$ plotted instead of $\tilde{D}^2$. This t value is the location of the maximum in Figure 2.1.

4.2 Optimization of $\tilde{D}^2$

Algorithm 1 Abbreviated pseudocode for the algorithm described in Section 4.2, for computing $\inf_{P,\alpha} \tilde{D}^2$.

procedure D-TILDE(L_1, L_2, α_{low}, α_{high}.)
    Compute $\lambda^{(1)}$, $\lambda^{(2)}$ as the eigenvalues of L_1 and L_2.
    Compute, by optimizing a linear assignment, M_{low} and M_{high} as the optimal matchings at α_{low}, α_{high} respectively. Initialize the list of optimal matchings as $\{M_{\text{low}}, M_{\text{high}}\}$.
    Until the current list of matchings is not expanded in the following step, or the entire interval $[\alpha_{\text{low}}, \alpha_{\text{high}}]$ is marked as explored:
        Attempt to expand the list of optimal matchings by solving a linear assignment problem at the α where the cost curves of two matchings (currently in the list) intersect. If no better assignment exists, then mark the interval covered by those matchings as explored, as guaranteed by Theorem 4.4.3.
    Return the lowest-cost M and its optimal α.
end procedure

Joint optimization of $\tilde{D}^2$ over α and P is a nested optimization problem (see [77] and [98] for a description of nested optimization), with potential combinatorial optimization over P dependent on each choice of α. Furthermore, the function $f(\alpha) = \inf_{P|\mathcal{C}(P)} \tilde{D}^2(G_1, G_2|\alpha)$ is both multimodal and continuous but, in general, with a discontinuous derivative (See Figure 4.1). Univariate optimization procedures such as Golden Section Search result in many loops of some procedure to optimize over P, which in our restricted case must each time compute a full solution to a LAP with $n_2 \times n_1$ weights. In our experience, this means that these univariate methods have a tendency to get stuck in local optima. We reduce the total number of calls to the LAP solver, as well as the size of the LAPs solved, by taking advantage of several unique properties of the optimization as a function of α. When the optimal $P^{(1)}$ and $P^{(2)}$ are known for α_1 and α_2, then for any α_c such that $\min(\alpha_1, \alpha_2) \leq \alpha_c \leq \max(\alpha_1, \alpha_2)$, the optimal $P^{(c)}$ at α_c must satisfy: $P^{(1)}_{ij} = 1 \wedge P^{(2)}_{ij} = 1 \implies P^{(c)}_{ij} = 1$ (see Theorem 4.4.3). Thus, the optimization over P at α_c is already partially solved given the solutions at α_1 and α_2, and so we need only re-compute the remaining (smaller) subproblem on the set of assignments where $P^{(1)}$ and $P^{(2)}$ disagree. This has two consequences for our search over α: First, the size of LAP problems which must be solved at each step decreases over time (as we

find P-optima for a denser and denser set of α). Secondly, these theoretical guarantees mean that we can mark intervals of α-values as being explored (meaning we have provably found the P which are optimal over the interval) and thus do not have to perform the relatively expensive optimization over P for any α in that interval.

4.3 Optimization of D^2

Algorithm 2 Abbreviated pseudocode for the algorithm described in Section 4.3, for computing $\sup_t \inf_{P,\alpha} D^2$.

procedure D(L_1, L_2, α_{low}, α_{high}, step size ϵ)
    Compute $\lambda^{(1)}$, $\lambda^{(2)}$ as the eigenvalues of L_1 and L_2.
    Solve the Linear Version of the problem using Algorithm 1, obtaining α^*, M^*. According to the argument presented in the definition of linear distance (Equation 2.3) this solution holds for very small t. Keep the entire frontier of matchings found during the execution of Algorithm 1. Set $t = 0$, $d(0) = D(G_1, G_2|\alpha*, M*, t)$
    Until $d(t + \epsilon) < d(t)$:
    $t = t + \epsilon$
        Use the linear algorithm with e^{tL_1} and e^{tL_2} as the input matrices, initializing the list of matchings with those found at the previous t.
        Set $d(t) = D(G_1, G_2|\alpha*, M*, t)$ where $\alpha*, M*$ are the optima from the previous step.
    Return the $\max_t d(t)$.
end procedure

Many of the theoretical guarantees underlying our algorithm for computing $\tilde{D}^2$ no longer hold for the exponential version of the distance. We adapt our linear-version procedure into an algorithm for computing this version, with the caveat that the lack of these guarantees means that our upper bound on the exponential version may be looser than that on the linear version. It is still clearly an upper bound, since the α and P found by this procedure satisfy the given constraints $\alpha > 0$ and $P^T P = I$. In particular, we have observed cases where the exponential-distance analog of Theorem 4.4.3 would not hold, meaning we cannot rule out α-intervals as we can in the linear version. Thus, this upper bound may be looser than that computed for the linear objective function.

For the exponential version of the algorithm, we first compute the list of optimal P for the linear version, assuming (since $e^{tL} \approx I + L$ for very small t) that this is also the list of optimal P for the exponential version of the objective function at some low t. We proceed to increment t with some step size Δt, in the manner of a continuation method [3]. At each new t value, we search for new optimal P along the currently known frontier of optima as a function of α. When a new P is found as the intersection of two known P_i, P_{i+1}, it is inserted into the list, which is kept in order of increasing α. For each P in this frontier, we find the optimal α, keeping P and t constant. Assuming $\inf_P \inf_\alpha D^2(G_1, G_2|t_c)$ is unimodal as a function of t_c, we increase t_c until $\inf_P \inf_\alpha D^2(G_1, G_2|t_c) \geq \inf_P \inf_\alpha D^2(G_1, G_2|t_c + \Delta t)$, storing all P matrices found as optima at each t_c value. P which were on the lower convex hull at some prior value of t but not the current value are retained, as they may regain optimality for some α-range at a future value of t (we have observed this, in practice). For this list $P_1, P_2 \ldots P_m$, we then compute $\sup_t \inf_\alpha \inf_i D^2(G_1, G_2|P_i)$. Since the exponential map is continuous, and we are incrementing t by very small steps, we also propose the further computational shortcut of storing the list of optimal α at time t to use as starting points for the optimization at $t + \Delta t$. In practice, this made little difference in the runtime of our optimization procedure.

4.4 Algorithm Correctness Proof

Theorem 4.4.1. *For any two graphs G_1 and G_2 with Laplacians $L(G_1)$ and $L(G_2)$, for fixed α, the optimization over P given in the innermost loop of Equation 2.3 is upper bounded by a Linear Assignment Problem as defined in Equation* (1.5)*. This LAP is given by taking R to be the eigenvalues $\lambda_j^{(1)}$ of $L(G_1)$ and S to be the eigenvalues $\lambda_i^{(2)}$ of $L(G_2)$, with the cost*

of a pair (equivalently, one entry of the cost matrix C) given by

$$C_{ij} = c(s_i, r_j) = c\left(\lambda_i^{(2)}, \lambda_j^{(1)}\right) = \left(\frac{1}{\alpha}\lambda_j^{(1)} - \alpha\lambda_i^{(2)}\right)^2 \tag{4.2}$$

Proof. $L(G_1)$ and $L(G_2)$ are both real symmetric matrices, so they may be diagonalized as $L(G_i) = U_i \Lambda_i U_i^T$, where the U_i are rotation matrices, and the Λ_i are diagonal matrices with the eigenvalues $\lambda_1^{(i)}, \lambda_2^{(i)} \ldots \lambda_{n_i}^{(i)}$ along the diagonal. Because the Frobenius norm is invariant under rotation, we have:

$$\begin{aligned}
\tilde{D}^2(G_1, G_2) &= \inf_{\alpha>0} \inf_{P^TP=I} \left|\left| \frac{1}{\alpha} PL(G_1) - \alpha L(G_2)P \right|\right|_F^2 \\
&= \inf_{\alpha>0} \inf_{P^TP=I} \left|\left| \frac{1}{\alpha} U_2^T PL(G_1)U_1 - \alpha U_2^T L(G_2)PU_1 \right|\right|_F^2 \\
&= \inf_{\alpha>0} \inf_{P^TP=I} \left|\left| \frac{1}{\alpha} U_2^T PU_1\Lambda_1 U_1^T U_1 - \alpha U_2^T U_2 \Lambda_2 U_2^T PU_1 \right|\right|_F^2 \\
&= \inf_{\alpha>0} \inf_{P^TP=I} \left|\left| \frac{1}{\alpha} U_2^T PU_1\Lambda_1 - \alpha \Lambda_2 U_2^T PU_1 \right|\right|_F^2.
\end{aligned} \tag{4.3}$$

Because the U_i are orthogonal, the transformation $\tilde{P} = U_2^T P U_1$ preserves orthogonality, so

$$\begin{aligned}
\tilde{D}^2(G_1, G_2) &= \inf_{\alpha>0} \inf_{P^TP=I} \left|\left| \frac{1}{\alpha} P\Lambda_1 - \alpha\Lambda_2 P \right|\right|_F^2 \\
&= \inf_{\alpha>0} \inf_{P^TP=I} \left|\left| \frac{1}{\alpha}\Lambda_1 \right|\right|_F^2 + \left|\left| \alpha\Lambda_2 P \right|\right|_F^2 - 2\mathrm{Tr}\left[P^T\Lambda_2 P\Lambda_1\right] \\
&= \inf_{\alpha>0} \inf_{P^TP=I} \left(\mathrm{Tr}\left[\frac{1}{\alpha^2}\Lambda_1^2\right] + \mathrm{Tr}\left[\alpha^2 P^T\Lambda_2^2 P\right] - 2\mathrm{Tr}\left[P^T\Lambda_2 P\Lambda_1\right] \right)
\end{aligned}$$

writing $P = [p_{ij}]$,

$$\tilde{D}^2(G_1, G_2) = \inf_{\alpha>0} \inf_{P^T P = I} \left(\frac{1}{\alpha^2} \sum_{j=1}^{n_1} {\lambda_j^{(1)}}^2 + \alpha^2 \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} p_{ij}^2 {\lambda_i^{(2)}}^2 \right. \tag{4.4}$$

$$\left. -2 \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} p_{ij}^2 \lambda_i^{(2)} \lambda_j^{(1)} \right)$$

$$= \inf_{\alpha>0} \inf_{P^T P = I} \left(\sum_{i=1}^{n_2} \sum_{j=1}^{n_1} p_{ij}^2 \left(\frac{1}{\alpha^2} {\lambda_j^{(1)}}^2 - 2\lambda_i^{(2)} \lambda_j^{(1)} + \alpha^2 {\lambda_i^{(2)}}^2 \right) \right)$$

$$= \inf_{\alpha>0} \inf_{P^T P = I} \left(\sum_{i=1}^{n_2} \sum_{j=1}^{n_1} p_{ij}^2 \left(\frac{1}{\alpha} \lambda_j^{(1)} - \alpha \lambda_i^{(2)} \right)^2 \right) \tag{4.5}$$

For any given α,

$$\inf_{P^T P = I} \left(\sum_{i=1}^{n_2} \sum_{j=1}^{n_1} p_{ij}^2 \left(\frac{\lambda_j^{(1)}}{\alpha} - \alpha \lambda_i^{(2)} \right)^2 \right) \leq \inf_{\tilde{P} | \mathrm{sub}(\tilde{P})} \left(\sum_{i=1}^{n_2} \sum_{j=1}^{n_1} \tilde{p}_{ij}^2 \left(\frac{\lambda_j^{(1)}}{\alpha} - \alpha \lambda_i^{(2)} \right)^2 \right) \quad ,$$

where subperm($\tilde{P}$) could be any other condition more strict than the constraint $P^T P = I$. Here we take this stricter constraint to be the condition that $\tilde{P}$ is a *subpermutation matrix*: an orthogonal matrix (i.e. $\tilde{P}^T \tilde{P} = I$) for which $\tilde{P} \in \{0,1\}^{n_2 \times n_1}$. Equivalently, a subpermutation matrix is a $\{0,1\}$-valued matrix $[\tilde{p}_{ij}]$ such that for each $i \in \{1, \ldots n_1 \leq n_2\}$, exactly one $j \in \{1, \ldots n_2 \geq n_1\}$ takes the value 1 rather than 0 (so $\sum_{j=1}^{n_2} \tilde{P}_{ji} = 1$), and for each $j \in \{1, \ldots n_2 \geq n_1\}$, either zero or one $i \in \{1, \ldots n_1 \leq n_2\}$ takes the value 1 rather than 0 (so $\sum_{i=1}^{n_1} \tilde{P}_{ji} \leq 1$).

Furthermore, this optimization is exactly a linear assignment problem of eigenvalues of $L(G_1)$ to $L(G_2)$, with the cost of a pair $\left(\lambda_j^{(1)}, \lambda_i^{(2)}\right)$ given by

$$c\left(\lambda_j^{(1)}, \lambda_i^{(2)}\right) = \left(\frac{1}{\alpha} \lambda_j^{(1)} - \alpha \lambda_i^{(2)} \right)^2$$

Note also that the same argument applies to the innermost two optimizations of the calculation of D^2 (the exponential version of the diffusion distance) as well as D_r^2. In the D^2 case the entries of the cost matrix are instead given by

$$c\left(\lambda_j^{(1)}, \lambda_i^{(2)}\right) = \left(e^{\frac{1}{\alpha}\lambda_j^{(1)}} - e^{\alpha\lambda_i^{(2)}}\right)^2$$

If we instead loosen the constraints on P, we can calculate a lower bound on the distance. See Section 2.4 for lower bound details.

□

Recall that our definition of a 'matching' in Section 1.4.2 was a P matrix representing a particular solution to the linear assignment problem with costs given as in Equation (4.2). For given G_1, G_2, and some matching M, let

$$f_M(\alpha) = \tilde{D}^2(G_1, G_2|\alpha, U_2^T M U_1) \tag{4.6}$$

where U_1, U_2 diagonalize L_1 and L_2 as in Equation (4.3).

Lemma 4.4.2. *For two unique matchings M_1 and M_2 (for the same G_1, G_2) the equation $f_{M_1}(\alpha) - f_{M_2}(\alpha) = 0$ has at most one real positive solution in α. This follows from the fact that when P and t are fixed, the objective function is a rational function in α (see Equation (4.4)), with a quadratic numerator and an asymptote at $\alpha = 0$.*

Proof. By Equation (4.4), we have

$$
\begin{aligned}
&f_{M_1}(\alpha) - f_{M_2}(\alpha) = \\
&\left(\frac{1}{\alpha^2} \sum_{j=1}^{n_1} {\lambda_j^{(1)}}^2 + \alpha^2 \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} [M_1]_{ij}^2 {\lambda_i^{(2)}}^2 - 2 \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} [M_1]_{ij}^2 \lambda_i^{(2)} \lambda_j^{(1)} \right) && (4.7) \\
&\quad - \left(\frac{1}{\alpha^2} \sum_{j=1}^{n_1} {\lambda_j^{(1)}}^2 + \alpha^2 \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} [M_2]_{ij}^2 {\lambda_i^{(2)}}^2 - 2 \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} [M_2]_{ij}^2 \lambda_i^{(2)} \lambda_j^{(1)} \right) && (4.8) \\
&= \alpha^2 \left(\sum_{i=1}^{n_2} \sum_{j=1}^{n_1} [M_1]_{ij}^2 {\lambda_i^{(2)}}^2 - \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} [M_2]_{ij}^2 {\lambda_i^{(2)}}^2 \right) && (4.9) \\
&\quad + \left(2 \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} [M_2]_{ij}^2 \lambda_i^{(2)} \lambda_j^{(1)} - 2 \sum_{i=1}^{n_2} \sum_{j=1}^{n_1} [M_1]_{ij}^2 \lambda_i^{(2)} \lambda_j^{(1)} \right) && (4.10)
\end{aligned}
$$

Abbreviating the sums, we have

$$\alpha^2 (A_1 - A_2) + (C_2 - C_1) = 0 \tag{4.11}$$

and so

$$\alpha = \pm \sqrt{\frac{C_2 - C_1}{A_1 - A_2}} \tag{4.12}$$

Since A_1, A_2, C_1, C_2 are all nonnegative reals, at most one of these roots is positive. □

We will say that a matching M "assigns" j to i if and only if $M_{ij} = 1$.

Theorem 4.4.3. *If two matchings M_1 and M_3 which yield optimal upper bounds for the linear distance $\tilde{D}^2$ (at $\alpha_1 \leq \alpha$ and $\alpha_3 \geq \alpha$ respectively) agree on a set of assignments, then the optimal M at α must also agree with that set of assignments.*

Proof. We need the following lemmas:

Lemma 4.4.4. *If an optimal matching assigns i to $m(i)$ (so that eigenvalue $\lambda_i^{(1)}$ of G_1 is paired with $\lambda_{f(i)}^{(2)}$ of G_2 in the sum of costs Equation (4.2)), then the sequence $m(1), m(2), \ldots m(n_1)$ is monotonic increasing.*

Proof. This follows from the fact that the two sequences of eigenvalues are monotonic nondecreasing, so if there's a 'crossing' ($i_1 < i_2$ but $m(i_2) < m(i_1)$) then the new matching obtained by uncrossing those two pairs (performing a 2-opt step as defined in [25]) has strictly lesser objective function value. Hence an optimal matching can't contain any such crossings. □

Lemma 4.4.5. *For all positive real $\alpha^* \geq \epsilon > 0$, let M_1 be an optimal matching at $\alpha^* - \epsilon$ and M_2 be optimal at $\alpha^* + \epsilon$. For $1 \leq i \leq n_1$, let $s_1(i)$ and $s_2(i)$ be the indices of $\lambda^{(2)}$ paired with i in M_1 and M_2, respectively. Then for all i, $s_1(i) \leq s_2(i)$.*

Proof. Define a "run" for s_1, s_2 as a sequence of consecutive indices $l, l+1, \ldots l+k$ in $[1, n_1]$ such that for any l, $l+1$: $\min(s_1(l+1), s_2(l+1)) < \max(s_1(l), s_2(l))$. The following must be true about a "run":

1. Within a run, either $s_1(l) < s_2(l)$ or $s_1(l) > s_2(l)$ for all l. Otherwise, we have one or more crossings (as in Lemma 4.4.4): for some l we have $s_1(l) > s_1(l+1)$ or $s_2(l) > s_2(l+1)$. Any crossing may be uncrossed for a strictly lower objective function value - violating optimality of M_1 or M_2.

2. Any pair of matchings as defined above consists of a sequence of runs, where we allow a run to be trivial i.e. be a single index.

Next, we show that within a run, we must have $s_1(i) < s_2(i)$ for all i. Let $S = \{l, l+1, \ldots l+k\}$ be a run. By optimality of M_1, M_2 at $\alpha^* - \epsilon$ and $\alpha^* + \epsilon$ respectively, we have:

$$\sum_{i\in S}\left(\frac{1}{\alpha^*-\epsilon}\lambda_i^{(1)}-(\alpha^*-\epsilon)\lambda_{s_1(i)}^{(2)}\right)^2<\sum_{i\in S}\left(\frac{1}{\alpha^*-\epsilon}\lambda_i^{(1)}-(\alpha^*-\epsilon)\lambda_{s_2(i)}^{(2)}\right)^2$$

and

$$\sum_{i\in S}\left(\frac{1}{\alpha^*+\epsilon}\lambda_i^{(1)}-(\alpha^*+\epsilon)\lambda_{s_2(i)}^{(2)}\right)^2<\sum_{i\in S}\left(\frac{1}{\alpha+\epsilon}\lambda_i^{(1)}-(\alpha+\epsilon)\lambda_{s_1(i)}^{(2)}\right)^2.$$

Respectively, these simplify to

$$-\sum_{i\in S}\left(\lambda_{s_1(i)}^{(2)}-\lambda_{s_2(i)}^{(2)}\right)\left(-2\lambda_i^{(i)}+(\alpha^*-\epsilon)^2\left(\lambda_{s_1(i)}^{(2)}+\lambda_{s_2(i)}^{(2)}\right)\right)>0$$

and

$$\sum_{i\in S}\left(\lambda_{s_1(i)}^{(2)}-\lambda_{s_2(i)}^{(2)}\right)\left(-2\lambda_i^{(i)}+(\alpha^*+\epsilon)^2\left(\lambda_{s_1(i)}^{(2)}+\lambda_{s_2(i)}^{(2)}\right)\right)>0.$$

Summing these inequalities and cancelling $-2\lambda_i^{(i)}$, we have:

$$\sum_{i\in S}\left\{(\alpha^*+\epsilon)^2\left(\left(\lambda_{s_1(i)}^{(2)}\right)^2+\left(\lambda_{s_2(i)}^{(2)}\right)^2\right)-(\alpha^*-\epsilon)^2\left(\left(\lambda_{s_1(i)}^{(2)}\right)^2+\left(\lambda_{s_2(i)}^{(2)}\right)^2\right)\right\}>0.$$

Summing and reducing gives us

$$4\alpha^*\epsilon\left(\sum_{i\in S}\left(\lambda_{s_1(i)}^{(2)}\right)^2-\sum_{i\in S}\left(\lambda_{s_2(i)}^{(2)}\right)^2\right)>0\quad\text{and so}\quad\sum_{i\in S}\left(\lambda_{s_1(i)}^{(2)}\right)^2>\sum_{i\in S}\left(\lambda_{s_2(i)}^{(2)}\right)^2.$$

However, since the $\lambda_j^{(2)}$ are monotonic nondecreasing, this means we cannot also have $s_1(i)>$

$s_2(i)$ for all $i \in S$, since that would imply

$$\sum_{i=1}^{n_1} \left(\lambda^{(2)}_{s_1(i)}\right)^2 < \sum_{i=1}^{n_1} \left(\lambda^{(2)}_{s_2(i)}\right)^2.$$

Therefore, in a run of arbitrary length, all indices must move 'forward' (meaning that $s_1(i) < s_2(i)$ for all i in the run), and so (since any pair of matchings optimal at such α define a set of runs) we must have $s_1(i) \leq s_2(i)$. This completes the proof of the lemma.

□

Thus, for three matchings M_1, M_2, M_3 which are optimal at a sequence of $\alpha_1 \leq \alpha_2 \leq \alpha_3$, we must have $s_1(i) \leq s_2(i) \leq s_3(i)$ for all i. In particular, if $s_1(i) = s_3(i)$, we must also have $s_1(i) = s_2(i) = s_3(i)$. □

Theorem 4.4.6. *If two matchings M_1 and M_3 yield optimal upper bounds for the linear distance $\tilde{D}^2$ at α_1 and α_3 respectively, and $f_{M_1}(\alpha_2) = f_{M_2}(\alpha_2)$ for some α_2 s.t. $\alpha_1 \leq \alpha_2 \leq \alpha_3$, then either (1) M_1 and M_3 are optimal over the entire interval $[\alpha_1, \alpha_3]$ or (1) some other matching M_2 improves over M_1 and M_3 at α_2.*

Proof. This follows directly from the facts that $f_{M_1}(\alpha)$ and $f_{M_2}(\alpha)$ (as defined in Equation (4.6)), can only meet at one real positive value of α (Lemma 4.4.2). Say that the cost curves for M_1 (known to be optimal at $\alpha = \alpha_1$) and M_3 (optimal at $\alpha = \alpha_3$) meet at $\alpha = \alpha_2$, and furthermore assume that $\alpha_1 \leq \alpha_2 \leq \alpha_3$. If some other matching M_2 improves over (meaning, has lesser obj. function value as a function of α) M_1 or M_3 anywhere in the interval $[\alpha_1, \alpha_3]$, it must improve over both at $\alpha = \alpha_2$, since it may intersect each of these cost curves at most once on this interval. If M_1 and M_3 are both optimal at their intersection point (meaning no such distinct M_2 exists) then we know that no other matching improves on either of them over the the interval $[\alpha_1, \alpha_3]$ and may therefore mark it as explored during the outermost

loop (otimization over α) of Algorithm 1. $\square$

Together, the preceeding properties verify that our algorithm will indeed find the joint optimum over all α and P (for fixed $t = c$, for $\tilde{D}$, subject to subpermutation constraints on P): it allows us to find the entire set of P subpermutation matrices which appear on the lower convex hull of distance as a function of alpha.

4.5 Implementation Details

We implement Algorithms 1 and 2 in the programming language Python (version 3.6.1) [87]. Numerical arrays were stored using the *numpy* package [109]. Our inner LAP solver was the package *lapsolver* [50]. Univariate optimization over t and α was performed with the 'bounded' method of the *scipy.optimize* package [30], with bounds set at $[0, 10.0]$ for each variable and a input tolerance of 10^{-12}. Laplacians were computed with the *laplacian* method from the package *networkX* [47], and their eigenvalues were computed with *scipy.linalg.eigh*.

Because of numerical precision issues arising during eigenvalue computation, it can be difficult to determine when two matchings agree, using eigenvalue comparison. In practice we ignore this issue and assume that two matchings are only identical if they associate the same indices of the two lists of eigenvalues. This means we may be accumulating multiple equivalent representations of the same matching (up to multiplicity of eigenvalues) during our sweeps through t and α. We leave mitigating this inefficiency for future work.

Code for computing diffusion distance, both with our algorithm and with naive univariate optimization, may be found in the Supplementary Information associated with this paper, as well as a maintained GitHub repository [93].

Chapter 5

Numerical Properties of GDD

Previous chapters have introduced graph diffusion distance and explored some of its theoretical properties, as well as demonstrating an efficient method for computing GDD. This chapter presents several experiments, using the machinery developed in Chapter 4 to explore numerical properties of GDD as well as numerical properties of Algorithm 1.

5.1 Graph Lineages

In this section we introduce several specific graph lineages for which we will compute various intra- and inter-lineage distances. Three of these are well-known lineages of graphs, and the fourth is defined in terms of a product of complete graphs:

Path Graphs (Pa_n): 1D grid graphs of length n, with aperiodic boundary conditions.

Cycle Graphs (Cy_n): 1D grid graphs of length n, with periodic boundary conditions.

Square Grid Graphs (Sq_n): 2D grid graphs of dimensions n, with aperiodic boundary con-

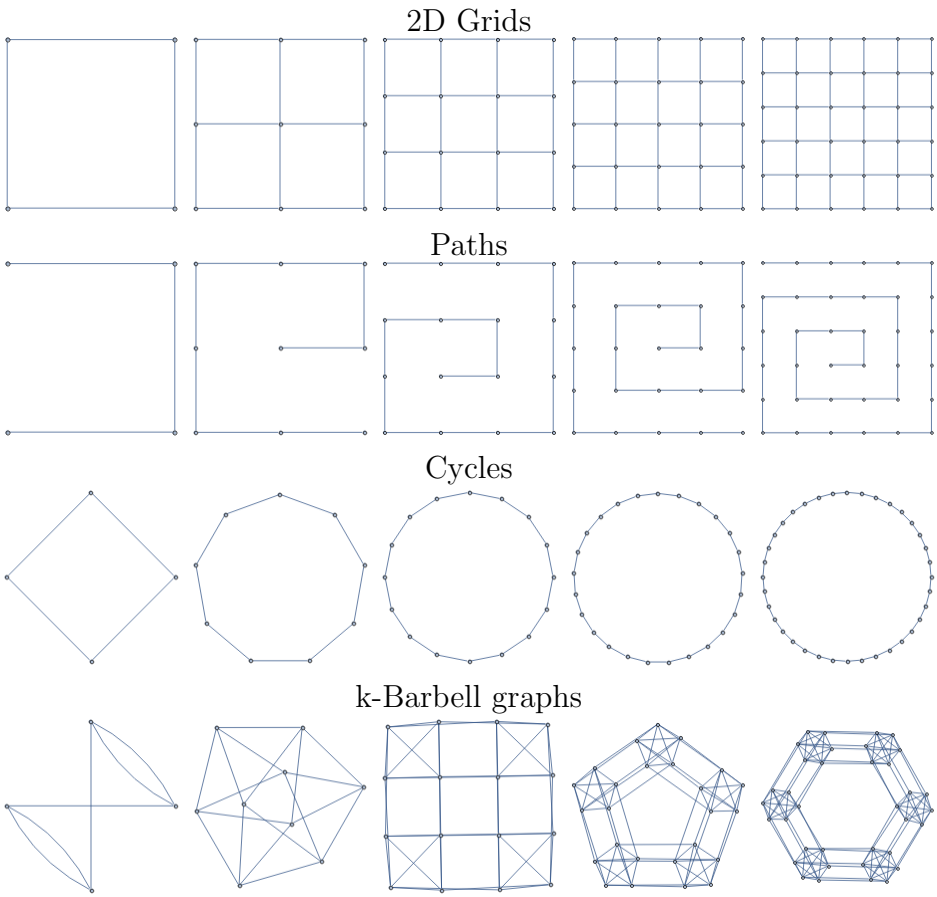

Figure 5.1: Graph lineages used in multiple numerical experiments in the main text.

ditions. $\mathrm{Sq}_n = \mathrm{Pa}_n \square \mathrm{Pa}_n$

"Multi-Barbell" Graphs (Ba_n): Constructed as $\mathrm{Cy}_n \square K_n$, where K_n is the complete graph on n vertices.

These familes are all illustrated in Figure 5.1.

Additionally, some examples distances between elements of these graph lineages are illustrated in Figure 5.2. In these tables we see that in general intra-lineage distances are small, and inter-lineage distances are large.

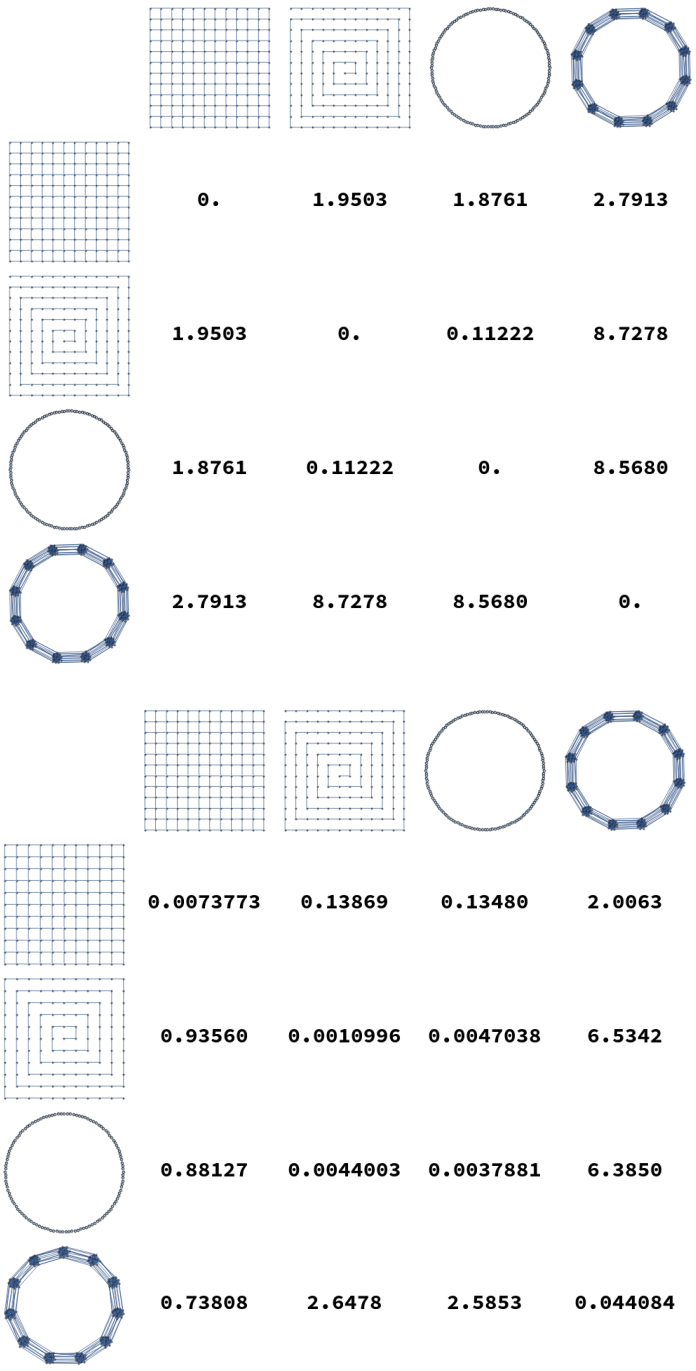

Figure 5.2: Distances $D^2(G, H)$ calculated for several pairs of graphs. The top plot shows distances where G and H are both chosen from $\{\text{Grid}_{13\times13}, P_{169}, C_{169}, \text{Ba}_{13}\}$. At bottom, distances are calculated from G chosen in $\{\text{Grid}_{12\times12}, P_{144}, C_{144}, \text{Ba}_{12}\}$ to H chosen in $\{\text{Grid}_{13\times13}, P_{169}, C_{169}, \text{Ba}_{13}\}$. As expected, diagonal entries are smallest.

5.2 Numerical Optimization Methods

We briefly discuss here the other numerical methods we have used to calculate $\tilde{D}^2$ and D^2. In general we have found these methods inferior to the algoithm presented in Chapter 4, but we present them here for completeness.

Nelder-Mead in Mathematica For very small graph pairs ($n_1 \times n_2 \leq 100$) we are able to find optimal P, α, t using constrained optimization in Mathematica 11.3 [53] using NMinimize, which uses Nelder-Mead as its backend by default. The size limitation made this approach unusable for any real experiments.

Orthogonally Constrained Opt. We also tried a variety of codes specialized for numeric optimization subject to orthogonality constraints. These included (1) the python package PyManopt [105], a code designed for manifold-constrained optimization; (2) gradient descent in Tensorflow using the penalty function $g(P) = c\left|\left|P^T P - I\right|\right|_F$ (with $c \ll 1$ a small positive constant weight) to maintain orthogonality, as well as (3) an implementation of the Cayley reparametrization method from [114] (written by the authors of that same paper). In our experience, these codes were slower, with poorer scaling with problem size, than combinatorial optimization over subpermutation matrices, and did not produce improved results on our optimization problem.

5.2.1 Black-Box Optimization Over α.

We compare in more detail two methods of joint optimization over α and P when P is constrained to be a subpermutation matrix in the diagonal basis for $L(G_1)$ and $L(G_2)$. Specifically, we compare our approach given in Algorithm 1 to univariate optimization over α, where each function evaluation consists of full optimization over P. Figure 5.3 shows the results of this experiment. We randomly sample pairs of graphs as follows:

1. n_1 is drawn uniformly from $[5, 120]$.

2. n_2 is drawn uniformly from $[n_1, n_1 + 60]$.

3. G_1 and G_2 are generated by adding edges according to a Bernoulli distribution with probability p. We ran 60 trials for each p in { .125, .25, .375, .5, .625, .75, .875 }.

We compute the linear version of distance for each pair. Because our algorithm finds all of the local minima as a function of alpha, we compute the cost of the golden section approach as the summed cost of multiple golden section searches in alpha: one GS search starting from the initial bracket $[0.618\alpha^*, 1.618\alpha^*]$ for each local minimum α^* found by our algorithm. We see that our algorithm is always faster by at least a factor of 10, and occasionally faster by as much as a factor of 10^3. This can be attributed to the fact that the golden section search is unaware of the structure of the linear assignment problem: it must solve a full $n_2 \times n_2$ linear assignment problem for each value of α it explores. In contrast, our algorithm is able to use information from prior calls to the LAP solver, and therefore solves a series of LAP problems whose sizes are monotonically nonincreasing.

5.3 Triangle Inequality violation of D (Exponential Distance) and $\tilde{D}$ (Linear Distance)

As stated in Section 2.3, our full graph dissimilarity measure does not necessarily obey the triangle inequality. In this section we systematically explore conditions under which the triangle inequality is satisfied or not satisfied. We generate triplets G_1, G_2, G_3 of random graphs of sizes n_i for $n_1 \in [5, 30]$, $n_2 \in [n_1, n_1 + 30]$, and $n_3 \in [n_2, n_2 + 30]$ by drawing edges from the Bernoulli distribution with probability p (we perform 4500 trials for each p value in [.125, .25, .375, .5, .625, .75, .875]). We compute the distance $\tilde{D}(G_i, G_k)$ (for

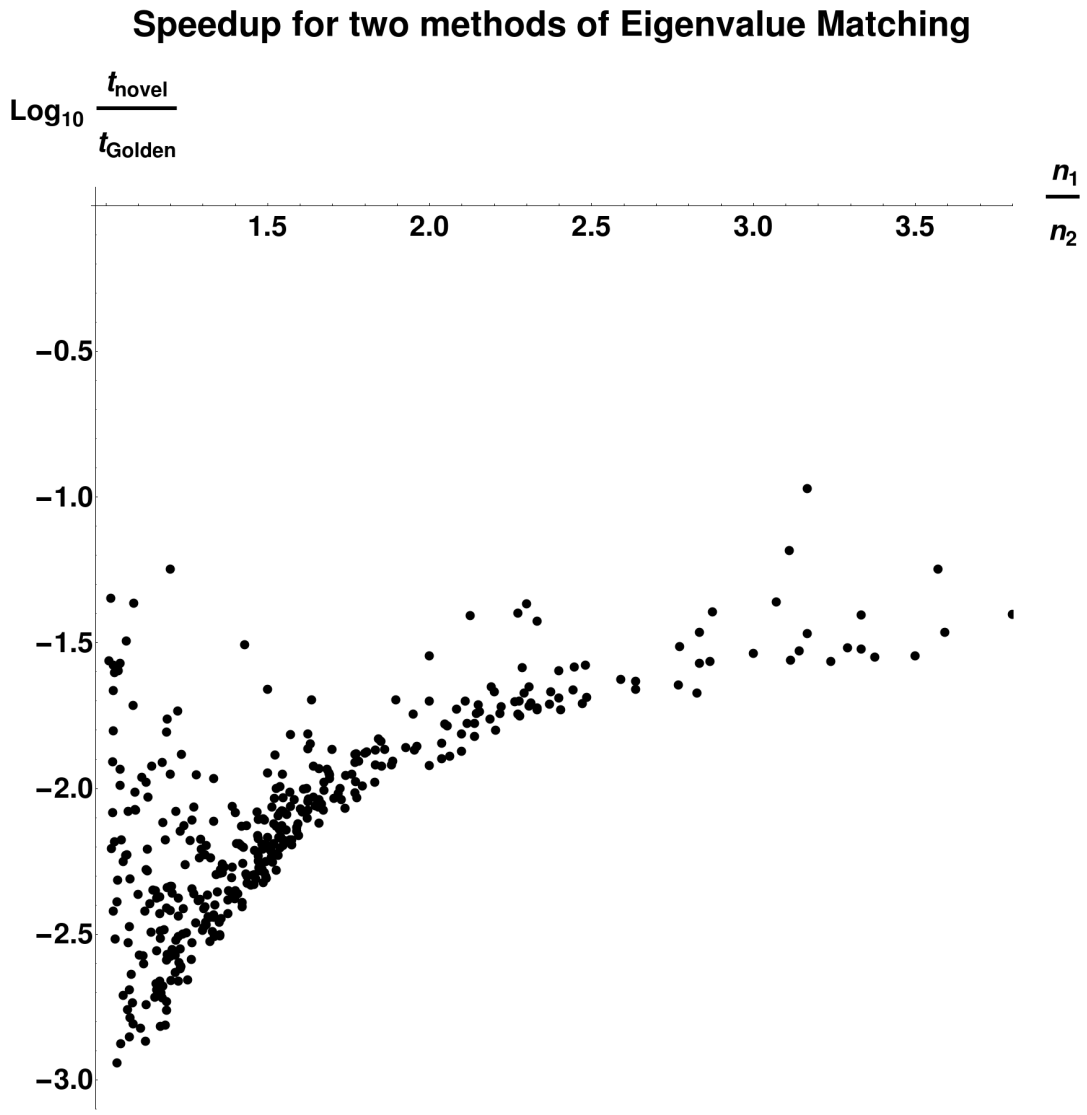

Figure 5.3: Comparison of runtimes for our algorithm and bounded golden section search over the same interval $[10^{-6}, 10]$. Runtimes were measured by a weighted count of evaluations of the Linear Assignment Problem solver, with an $n \times n$ linear assignment problem counted as n^3 units of cost. Because our algorithm recovers the entire lower convex hull of the objective function as a function of α, we compute the cost of the golden section search as the summed cost of multiple searches, starting from an interval bracketing each local optimum found by our algorithm. We see that our algorithm is much less computationally expensive, sometimes by a factor of 10^3. The most dramatic speedup occurs in the regime where $n_1 << n_2$. Graphs were generated by drawing n_1 uniformly from $[5, 120]$, drawing n_2 uniformly from $[n_1, n_1 + 60]$, and then adding edges according to a Bernoulli distribution with p in { .125, .25, .375, .5, .625, .75, .875 } (60 trials each).

$(i,k) \in \{(1,3),(1,2),(2,3)\}$). The results may be seen in Figure 5.4. In this figure we plot a histogram of the "discrepancy score"

$$\text{Disc}(G_1,G_2,G_3) = \tilde{D}(G_1,G_3)/(\tilde{D}(G_1,G_2)+\tilde{D}(G_2,G_3)), \tag{5.1}$$

which measures the degree to which a triplet of graphs violates the triangle inequality (i.e. falls outside of the unit interval [0,1]), for approximately 3×10^4 such triplets. It is clear that, especially for the linear definition of the distance, the triangle inequality is not always satisfied. However, we also observe that (for graphs of these sizes) the discrepancy score is bounded: no triple violates the triangle inequality by more than a factor of approximately 1.8. This is shown by the histogram of discrepancies in Figure 5.4. Additionally, the triangle inequality is satisfied in 28184 (95.2%) of cases.

We see similar but even stronger results when we run the same experiment with D^2 instead of $\tilde{D}^2$; these may also be seen in Figure 5.4. We calculated the discrepancy score analogously, but with D substituted for $\tilde{D}$. We see similarly that the degree of violation is bounded. In this case, no triple violated the triangle inequality by a factor of more than 5, and in this case the triangle inequality was satisfied in 99.8% of the triples. More work is needed to examine this trend; in particular, it would be interesting to examine whether other models of graph generation also satisfy the triangle inequality up to this constant. In both of these cases, the triangle inequality violations may be a result of our optimization procedure finding local minima/maxima for one or more of the three distances computed. We also repeat the above procedure for the same triplets of graphs, but with distances computed not in order of increasing vertex size: calculating $\text{Disc}(G_2,G_1,G_3)$ and $\text{Disc}(G_3,G_2,G_1)$. All of these results are plotted in Figure 5.4.

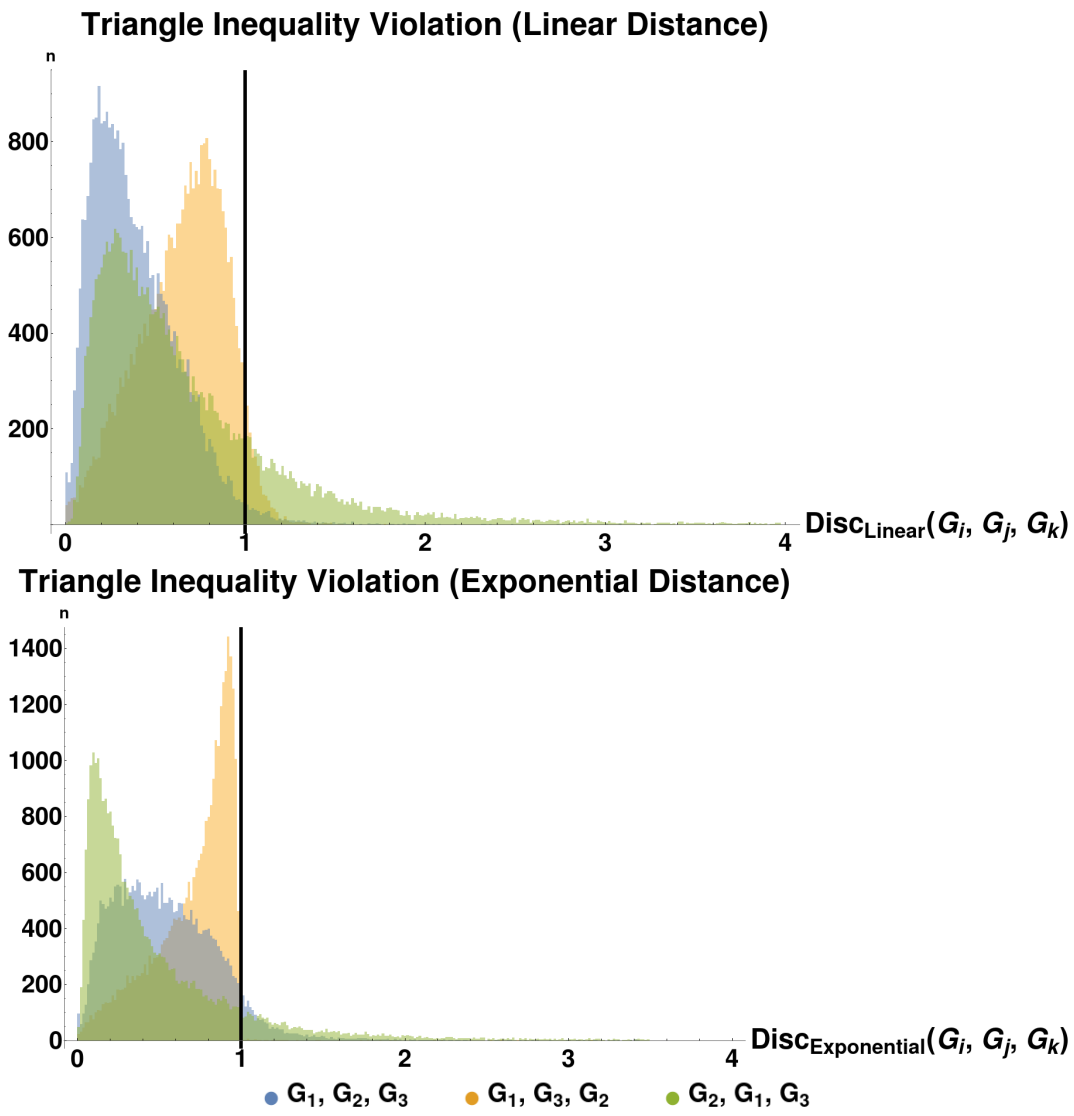

Figure 5.4: Histograms of triangle inequality violation. These plots show the distribution of $\mathrm{Disc}(G_1, G_2, G_3)$, as defined in the text, for the cases (a) top: the linear or small-time version of distance and (b) bottom: the exponential or arbitrary-time version of distance. We see that for the sizes of graph we consider, the largest violation of the triangle inequality is bounded, suggesting that our distance measure may be an infra-ρ-pseudometric for some value of $\rho \approx 1.8$ (linear version) or $\rho \approx 5.0$ (exponential version). See Table 2.1 for a summary of the distance metric variants introduced in this paper. We also plot the same histogram for out-of-order (by vertex size) graph sequences: $\mathrm{Disc}(G_2, G_1, G_3)$ and $\mathrm{Disc}(G_3, G_2, G_1)$. Each plot has a line at $x = 1$, the maximum discrepancy score for which the underlying distances satisfy the triangle inequality.

5.4 Intra- and Inter-Lineage Distances

We compute pairwise distances for sequences of graphs in the graph lineages displayed in Figure 5.1. For each pair of graph families (Square Grids, Paths, Cycles, and Multi-Barbells), we compute the distance from the ith member of one lineage to the $(i+1)$-st member of each other lineage, and take the average of the resulting distances from $i = 1$ to $i = 12$. These distances are listed in Table 5.1. As expected, average distances within a lineage are smaller than the distances from one lineage to another.

	Square Grids	Paths	Cycles	Multi-Barbells
Square Grids	0.0096700	0.048162	0.046841	0.63429
Paths	0.30256	0.0018735	0.010300	2.1483
Cycles	0.27150	0.0083606	0.0060738	2.0357
Multi-Barbells	0.21666	0.75212	0.72697	0.029317

Table 5.1: Mean distances between graphs in several lineages. For two lineages $G_1, G_2 \ldots$ (listed at left) and $H_!, H_2, \ldots$ (listed at the top), each entry shows the mean distance $D(G_i, H_{i+1})$ (where the average is taken over $i = 1$ to 12). As expected, we see that the distance from elements of a graph lineage to other members of the same lineage (the diagonal entries of the table) is smaller than distances taken between lineages. Furthermore as expected, 1D paths are more similar (but not equal) to 1D cycles than to other graph lineages.

We note here that the idea of computing intra- and inter- lineage distances is similar to recent work [49] computing distances between *graph ensembles*: certain classes of similarly-generated random graphs. Graph diffusion distance has been previously shown (in [49]) to capture key structural information about graphs; for example, GDD is known to be sensitive to certain critical transitions in ensembles of random graphs as the random parameters are varied. This is also true for our time dilated version of GDD. More formally: let G_p and $G_p^{'}$ represent random graphs on n vertices, drawn from the Erdős-Renyi distribution with edge probability p. Then $D(G_p, G_p^{'})$ has a local maximum at $p = \frac{1}{n}$, representing the transition

between disconnected and connected graphs. This is true for our distance as well as the original version due to Hammond.

5.5 Graph Limits

Here, we provide preliminary evidence that graph distance measures of this type may be used in the definition of a *graph limit* - a graphlike object which is the limit of an infinite sequence of graphs. This idea has been previously explored, most famously by Lovász [68], whose definition of a graph limit (called a *graphon*) is as follows: Recall the definition of graph cut-distance $D_{\text{cut}}(G, H)$ from Equation 1.3, namely: the cut distance is the maximum discrepancy in sizes of edge-cuts, taken over all possible subsets of vertices, between two graphs on the same vertex-set. A graphon is then an equivalence class of Cauchy sequences of graphs[1], under the equivalence relation that two sequences $G_1, G_2, \ldots$ and $H_1, H_2, \ldots$ are equivalent if $D_{\text{cut}}(G_i, H_i)$ approaches 0 as $n \to \infty$.

We propose a similar definition of graph limits, but with our diffusion distance substituted as the distance measure used in the definition of a Cauchy sequence of graphs. Hammond et. al. argue in [48] why their variant of diffusion distance may be a more descriptive distance measure than cut-distance. More specifically, they show that on some classes of graphs, some edge deletions 'matter' much more than others: removal of a single edge changes the diffusive properties of the graph significantly. However, the graph-cut distance between the new and old graphs is the same, regardless of which edge has been removed, while the diffusion distance captures this nuance. For graph limits, however, our generalization to *unequal-sized graphs* via P is of course essential. Furthermore, previous work [14] on sparse graph limits has shown that in the framework of Lovász all sequences of sparse graphs converge (in the infinite-

[1]Here we are calling a sequence of graphs "Cauchy" if for any $\epsilon > 0$ there is some N such that for all $n, m \geq N$, $D_{\text{cut}}(G_n, G_m) < \epsilon$.

size limit) to the zero graphon. Graph convergence results specific to sparse graphs include the Benjamini-Schramm framework [10], in which graph sequences are compared using the distributional limits of subgraph frequencies. These two graph comparison methods both have the characteristic that the "limit object" of a sequence of graphs is rigorously defined. It is unclear that density functions on the unit square are the best choice of limit objects for graphs; while graphons have many nice properties as detailed by Lovász, other underlying limit objects may be a more natural choice for sparse graphs. In this section we attempt to show empirically that such a limit object of graph sequences under GDD may exist, and therefore merit further investigation.

We examine several sequences of graphs of increasing size for the required Cauchy behavior (in terms of our distance measure) to justify this variant definition of a "graph limit". For each of the graph sequences defined in Section 5.1, we examine the distance between successive members of the sequence, plotting $D^2(G_n, H_{n+1})$ for each choice of G and H. These sequences of distances are plotted in Figure 5.7.

In this figure, we see that generally distance diverges between different graph lineages, and converges for successive members of the same lineage, as $n \to \infty$. We note the exceptions to this trend:

1. The distances between n-paths and $n+1$-cycles appear to be converging; this is intuitive, as we would expect that difference between the two spectra due to distortion from the ends of the path graph would decrease in effect as $n \to \infty$.

2. We also show analytically, under similar assumtions, that the distance between successive path graphs also shrinks to zero (Theorem 5.6.2).

We do not show that all similarly-constructed graph sequences display this Cauchy-like behavior. We hope to address this deeper question, as well as a more formal exploration of

the limit object, with one or more modified versions of the objective function (see Section 3.8.1).

5.6 Limit of Path Graph Distances

In this section, we demonstrate analytically that the sequence of path graphs of increasing size is Cauchy in the sense described by the previous section. In the following theorem (Theorem 5.6.2), we assume that the optimal value of t approaches some value $\tilde{t}$ as $n \to \infty$. We have not proven this to be the case, but have observed this behavior for both square grids and path graphs (see Figure 5.5 for an example of this behavior). Lemmas 5.6.1 and 5.6.2 show a related result for path graphs; we note that the spectrum of the Laplacian (as we define it in this paper) of a path graph of size n is given by

$$\lambda_k = -2 + 2\cos\frac{k\pi}{n-1} \qquad k \in \{0...n-1\}.$$

Lemma 5.6.1. *For any finite k, t, we have*

$$\lim_{n\to\infty} n\left(e^{t(-2+2\cos(\frac{\pi k}{n}))} - e^{t(-2+2\cos(\frac{\pi k}{n+1}))}\right)^2 = 0$$

Proof. Clearly for finite k, t

$$\lim_{n\to\infty} \left(e^{t(-2+2\cos(\frac{\pi k}{n}))} - e^{t(-2+2\cos(\frac{\pi k}{n+1}))}\right) = 0$$

Then,

$$\lim_{n\to\infty} n\left(e^{-2+2\cos(\frac{\pi k}{n})} - e^{-2+2\cos(\frac{\pi k}{n+1})}\right) = \lim_{n\to\infty} \frac{\left(e^{-2+2\cos(\frac{\pi k}{n})} - e^{-2+2\cos(\frac{\pi k}{n+1})}\right)}{\frac{1}{n}}$$

Evaluating this expression requires applying L'Hôpital's rule. Hence, we have:

$$\begin{aligned}
&\lim_{n\to\infty} \frac{\left(e^{-2+2\cos(\frac{\pi k}{n})} - e^{-2+2\cos(\frac{\pi k}{n+1})}\right)}{\frac{1}{n}} \\
&= \lim_{n\to\infty} \frac{2\pi kt\left(\frac{\sin\left(\frac{\pi k}{n}\right)e^{2t\left(\cos\left(\frac{\pi k}{n}\right)-1\right)}}{n^2} - \frac{\sin\left(\frac{\pi k}{n+1}\right)e^{2t\left(\cos\left(\frac{\pi k}{n+1}\right)-1\right)}}{(n+1)^2}\right)}{\frac{-1}{n^2}} \\
&= 2\pi kt \lim_{n\to\infty}\left(\frac{n^2\sin\left(\frac{\pi k}{n+1}\right)e^{2t\left(\cos\left(\frac{\pi k}{n+1}\right)-1\right)}}{(n+1)^2} - \sin\left(\frac{\pi k}{n}\right)e^{2t\left(\cos\left(\frac{\pi k}{n}\right)-1\right)}\right).
\end{aligned}$$

Since both of the limits

$$\lim_{n\to\infty}\left(\frac{n^2\sin\left(\frac{\pi k}{n+1}\right)e^{2t\left(\cos\left(\frac{\pi k}{n+1}\right)-1\right)}}{(n+1)^2}\right)$$

and

$$\lim_{n\to\infty}\left(-\sin\left(\frac{\pi k}{n}\right)e^{2t\left(\cos\left(\frac{\pi k}{n}\right)-1\right)}\right)$$

exist (and are 0),

$$2\pi kt \lim_{n\to\infty}\left(\frac{n^2\sin\left(\frac{\pi k}{n+1}\right)e^{2t\left(\cos\left(\frac{\pi k}{n+1}\right)-1\right)}}{(n+1)^2} - \sin\left(\frac{\pi k}{n}\right)e^{2t\left(\cos\left(\frac{\pi k}{n}\right)-1\right)}\right) = 0$$

and therefore

$$\lim_{n\to\infty} n\left(e^{t(-2+2\cos(\frac{\pi k}{n}))} - e^{t(-2+2\cos(\frac{\pi k}{n+1}))}\right)^2 = 0$$

□

Theorem 5.6.2. *If* $\lim_{n\to\infty} \arg\sup_t D^2\left(\mathrm{Pa}_n, \mathrm{Pa}_{n+1}|\, t\right)$ *exists, then:*

$$\lim_{n\to\infty} D^2\left(Pa_n, Pa_{n+1}\right) = 0.$$

Proof. Assume that $\lim_{n\to\infty} \arg\sup_t D^2\left(\mathrm{Pa}_n, \mathrm{Pa}_{n+1}|\, t\right) = \tilde{t}$. Then, we must have

$$\lim_{n\to\infty} D^2\left(\mathrm{Pa}_n, \mathrm{Pa}_{n+1}\right) \leq \lim_{n\to\infty} D^2\left(\mathrm{Pa}_n, \mathrm{Pa}_{n+1}|\, \tilde{t}\right)$$

Hence, it remains only to prove that

$$\lim_{n\to\infty} D^2\left(\mathrm{Pa}_n, \mathrm{Pa}_{n+1}|\, t\right) = 0$$

for any finite t (which will then include $\tilde{t}$). First, for any particular $(n+1)\times n$ subpermutation matrix S, note that

$$\begin{aligned} D^2\left(\mathrm{Pa}_n, \mathrm{Pa}_{n+1}|\, t\right) &= \inf_{\alpha>0} \inf_{P|\mathcal{C}(P)} D^2\left(\mathrm{Pa}_n, \mathrm{Pa}_{n+1}|\, t, P, \alpha\right) \\ &\leq D^2\left(\mathrm{Pa}_n, \mathrm{Pa}_{n+1}|\, t, \alpha = 1, U_{n+1}^T S U_n\right) \end{aligned}$$

Here, U_n and U_{n+1} are the matrices which diagonalize $L(\mathrm{Pa}_n)$ and $L(\mathrm{Pa}_{n+1})$ respectively (note also that a diagonalizer of a matrix L also diagonalizes e^L). If at each n we select S to be the subpermutation $S = \begin{bmatrix} I \\ 0 \end{bmatrix}$, then (using the same argument as in Theorem 4.4.1)

the objective function simplifies to:

$$\begin{aligned}
&D^2\left(\mathrm{Pa}_n, \mathrm{Pa}_{n+1} \middle| t, P = U_{n+1}^T S U_n, \alpha = 1\right) \\
&\quad = \left|\left| S e^{c\Lambda_{\mathrm{Pa}_n}} - e^{c\Lambda_{\mathrm{Pa}_{n+1}}} S \right|\right|_F^2 \\
&\quad = \sum_{k=0}^{n-1} \left(e^{c(-2+2\cos(\frac{\pi k}{n}))} - e^{c(-2+2\cos(\frac{\pi k}{n+1}))} \right)^2 \\
&\quad \leq \max_{0 \leq k \leq n-1} n \left(e^{c(-2+2\cos(\frac{\pi k}{n}))} - e^{c(-2+2\cos(\frac{\pi k}{n+1}))} \right)^2
\end{aligned}$$

By Lemma 5.6.1, for any finite k, t, we have

$$\lim_{n\to\infty} n \left(e^{t(-2+2\cos(\frac{\pi k}{n}))} - e^{t(-2+2\cos(\frac{\pi k}{n+1}))} \right)^2 = 0$$

So for any $\epsilon > 0$, $\exists N$ such that when $n \geq N$, for any c, k,

$$n \left(e^{c(-2+2\cos(\frac{\pi k}{n}))} - e^{c(-2+2\cos(\frac{\pi k}{n+1}))} \right)^2 < \epsilon$$

But then

$$\sum_{k=0}^{n-1} \left(e^{c(-2+2\cos(\frac{\pi k}{n}))} - e^{c(-2+2\cos(\frac{\pi k}{n+1}))} \right)^2 < \epsilon$$

as required. Thus, the Cauchy condition is satisfied for the lineage of path graphs Pa_n $\square$

Given a graph lineage which consists of levelwise box products between two lineages, it seems natural to use our upper bound on successive distances between graph box products to prove convergence of the sequence of products. As an example, the lineage consisting of square grids is the levelwise box product of the lineage of path graphs with itself. However, in this we see that this bound may not be very tight. Applying Equation (3.29) from Theorem

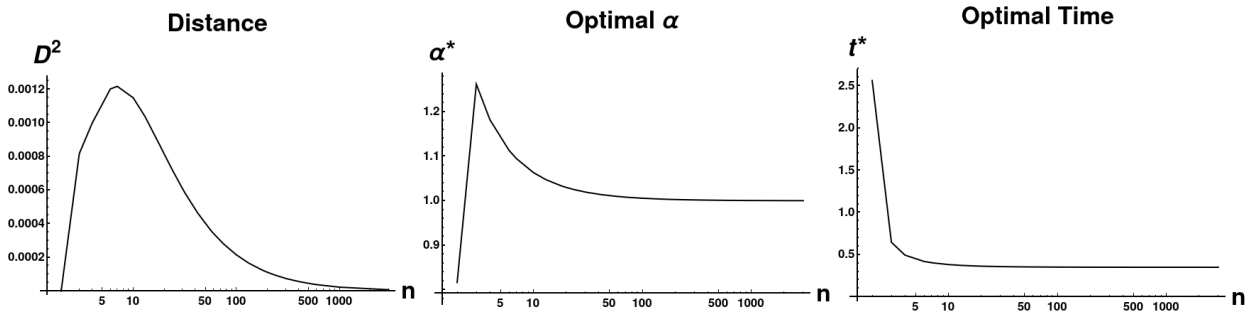

Figure 5.5: Limiting behavior of D and two parameters as path graph size approaches infinity. All distances were calculated between Path_n and Path_{n+1}. We plot the value of the objective function, as well as the optimal values of α and t, as $n \to \infty$. Optimal α rapidly approach 1 and the optimal distance tends to 0. Additionally, the optimal t value approaches a constant ($t \approx .316345$), providing experimental validation of the assumption we make in proving Theorem 5.6.2.

3.6.1, we have (for any t_c, α_c):

$$\begin{aligned} D\left(\text{Sq}_n, \text{Sq}_{n+1}\right) &\leq D\left(\text{Sq}_n, \text{Sq}_{n+1} \middle| t_c, \alpha_c\right) \\ &\leq D\left(\text{Pa}_{n+1}, \text{Pa}_{n+1} \,|\, t_c, \alpha_c\right) \left(\left|\left| e^{\frac{t_c}{a_c} L(\text{Pa}_n)} \right|\right|_F \right. \\ &\qquad \left. + \left|\left| e^{t_c a_c L(\text{Pa}_{n+1})} \right|\right|_F \right) \end{aligned}$$

As we can see in Figure 5.6, the right side of this inequality seems to be tending to a nonzero value as $n \to \infty$, whereas the actual distance (calculated by our optimization procedure) appears to be tending to zero.

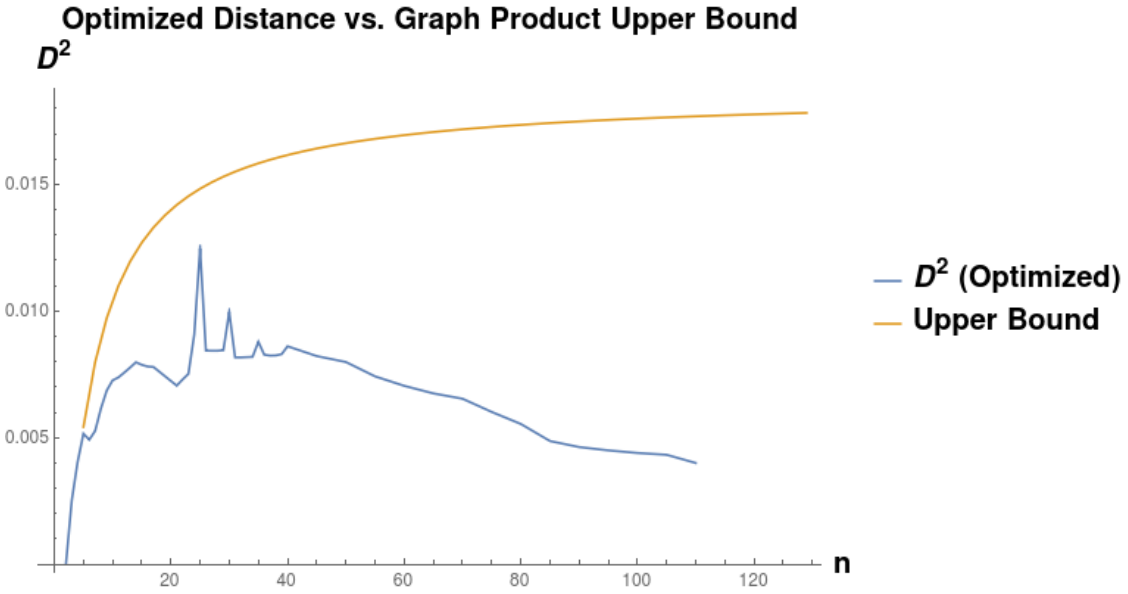

Figure 5.6: Comparison of the distance $D(\mathrm{Sq}_n, \mathrm{Sq}_{n+1})$ as a function of n, to the upper bound calculated as the optimum of distance between Pa_n and Pa_{n+1}. We see that the upper found converges to some constant $D \approx 0.01782$, whereas the actual distance appears to be converging to 0 as $n \to \infty$.

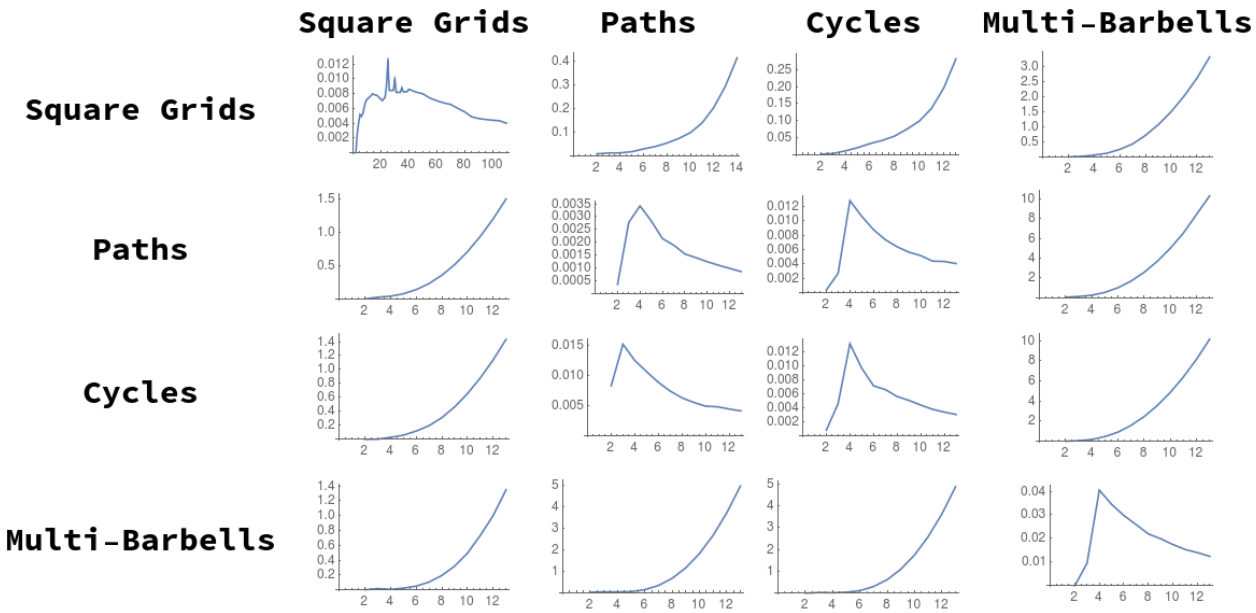

Figure 5.7: Cauchy-like behavior of graph distance as a function of sequence index, n. The distance between successive square grids and all other graph sequences appears to diverge (the same behavior is seen for k-barbells). Notably, the distance between $\mathrm{Grid}_{n\times n}$ and $\mathrm{Grid}_{(n+1)\times(n+1)}$ does not appear to converge, until much higher values of n ($n > 100$) than the other convergent series. This may be because the distances calculated are an upper bound, and may be converging more slowly than the 'true' optima.

Chapter 6

Application: Multiscale Neural Network Training

Previous chapters have explored the properties of Graph Diffusion Distance, as well as explained how to compute it efficiently. In this chapter, we use the P matrices which are a byproduct of computing Graph Diffusion Distance to accelerate the process of training a neural network. This is accomplished by defining a network which operates on multiple spatial scales of the input data, with the mapping in between scales performed by pre- and post-multiplication with P matrices. The final model shares some structural similarities with multigrid solvers for differential equations, which we discuss in the next section.

6.1 Prior Work

In this section, we discuss prior attempts to apply ideas from multigrid methods to neural network models. Broadly speaking, prior approaches to neural net multigrid can be categorized into two classes: (1) Neural network models which are "structurally multigrid", i.e. are

typical neural network models which make use of multiple scales of resolution; and (2) Neural network training processes which are hierarchical in some way, or use a coarsening-refinement procedure as part of the training process.

In the first class are approaches [43, 58, 95]. Ke et al [58] implement a convolutional network in which convolutions make use of a multigrid-like structure similar to a Gaussian pyramid, with the motivation that the network will learn features at multiple scales of resolution. Grais et. al [43] define a convolution operation, inspired by multigrid methods, that convolves at multiple levels of resolution simultaneously. Serban et. al [95] demonstrate a recurrent neural network model which similarly operates in multiple levels of some scale space; but in this work the scale space is a space of aggregated language models (specifically, the differing scales are different levels of generality in language models - for example, topic models are coarsest, word models are finest, with document models somewhere in between). Common to all three of these approaches is that they make use of a modified neural net structure while leaving the training process unchanged, except that the network accepts multiresolution inputs.

In contrast, multilevel neural network models [8, 91] in the second category present modified learning procedures which also use methodology similar to multilevel modeling. Reference [8] introduces a network which learns at coarse scales, and then gradually refines its decision making by increasing the resolution of the input space and learning "corrections" at each scale. However, that paper focuses on the capability of a particular family of basis functions for neural networks, and not on the capabilities of the multigrid approach. Reference [91] presents a reframing of the neural network training process as an evolution equation in time, and then applies a method called MGRIT (Multigrid Reduction in Time [33]) to achieve the same results as parallelizing over many runs of training.

Our approach is fundamentally different: we use coarsened versions of the network model to make coarse updates to the weight variables of our model, followed by 'smoothing steps'

in which the fine-scale weights are refined. This approach is more general than any of [43, 58, 95], since it can be applied to any feed-forward network and is not tied to a particular network structure. The approach in [91] is to parallelize the training process by reframing it as a continuous-in-time evolution equation, but it still uses the same base model and therefore only learns at one spatial scale.

Our method is both structurally multilevel and learns using a multilevel training procedure. Our hierarchical neural network architecture is the first to learn at all spatial scales simultaneously over the course of training, transitioning between neural networks of varying input resolution according to standard multigrid method schedules of coarsening and refinement. To our knowledge, this represents a fully novel approach to combining the powerful data analysis of neural networks with the model acceleration of multiscale modeling.

6.1.1 Outline

Building on the terminology in Chapters 1 and 2, in Section 6.2 we define an objective function which evaluates a map between two graphs, in terms of how well it preserves the behavior of some local process operating on those graphs (interpreting the smaller of the two graphs as a coarsened version of the larger). This is the core theory of this chapter: that of optimal prolongation maps between computational processes running on graph-based data structures, and hence between graphs. In this chapter we use a specific example of such a process, single-particle diffusion on graphs, to examine the behavior of these prolongation maps. Finally, we discuss numerical methods for finding (given two input graphs G_1 and G_2, and a process) prolongation and restriction maps which minimize the error of using G_1 as a surrogate structure for simulating the behavior of that process on G_2. We will define more rigorously what we mean by "process", "error", and "prolongation" in Section 6.2. In Subsection 6.3.2 we examine some properties of this objective function, including

presenting some projection matrices which are local optima for particular choices of graph structure and process. In Subsections 6.4 and 6.4.2, we define the *Multiscale Artificial Neural Network (MsANN)*, a hierarchically-structured neural network model which uses these optimized projection matrices to project network parameters between levels of the hierarchy, resulting in more efficient training. In Section 6.5, we demonstrate this efficiency by training a simple neural network model on a variety of datasets, comparing the cost of our approach to that of training only the finest network in the hierarchy.

6.2 Optimal Prolongation Maps Between Graphs

Given two graphs G_1 and G_2, we find the optimal prolongation map between them as follows: We first calculate the graph Laplacians L_1 and L_2, as well as pairwise vertex Manhattan distance matrices (i.e. the matrix with $T_{i,j}$ the minimal number of graph edges between vertices i and j in the graph), T_1 and T_2, of each graph. Calculating these matrices may not be trivial for arbitrary dense graphs; for example, calculating the pairwise Manhattan distance of a graph with m edges on n vertices can be accomplished in $O(m + n \log n)$ by the Fibonacci heap version of Dijkstra's algorithm [35]. Additionally, in Section 6.3 we discuss an optimization procedure which requires computing the eigenvalues of L_i (which are referred to as the *spectrum* of G_i). Computing graph spectra is a well studied problem; we direct the reader to [23, 79]. In practice, all of the graph spectra computed for experiments in this chapter took a negligible amount of time (< 1s) on a modern consumer-grade laptop using the scipy.linalg package [56], which in turn uses LAPACK routines for Schur decomposition

of the matrix [4]. The optimal map is defined as P which minimizes the matrix function

$$\begin{aligned} &\inf_{P|C(P),\alpha>0,\beta>0} && E(P) && (6.1)\\ =&\inf_{P|C(P),\alpha>0,\beta>0} && \left[(1-s)\left|\left|\frac{1}{\sqrt{\alpha}}PL_1-\sqrt{\alpha}L_2P\right|\right|_F^2 && \text{“Diffusion Term”}\right.\\ & && \left.+s\left|\left|\frac{1}{\sqrt{\beta}}PT_1-\sqrt{\beta}T_2P\right|\right|_F^2\right] && \text{“Locality Term”}^1 \end{aligned}$$

where $||\cdot||_F$ is the Frobenius norm, and $C(P)$ is a set of constraints on P (in particular, we require $P^TP = I_{n_1}$, but could also impose other restrictions such as sparsity, regularity, and/or bandedness). The manifold of real-valued orthogonal $n_2 \times n_1$ matrices with $n_1 \leq n_2$ is known as the Stiefel manifold; minimization constrained to this manifold is a well-studied problem [84, 106]. This optimization problem can be thought of as measuring the agreement between processes on each graph, as mapped through P. The expression $PX_1 - X_2P$ compares the end result of

1. Advancing process X_2 forward in time on G_2 and then using P to interpolate vertex states to the smaller graph, to:

2. Interpolating the initial state (the all-ones vector) using P and then advancing process X_1 on G_1.

Strictly speaking the above interpretation of our objective function does not apply to the Manhattan distance matrix T of a graph, since T is not a valid time evolution operator and thus is not a valid choice for X. However, the objective function term containing T may still be interpreted as comparing travel distance in one graph to travel distance in the other. That is, we are implicitly comparing the similarity of two ways of measuring the distance of two nodes v_k and v_l in G_1:

[1] By this we mean the notion that neighborhoods of G_1 should be mapped to neighborhoods of G_2 and vice versa.

1. The Manhattan distance, as defined above, and;

2. $\sum_{i=1}^{n_2} \sum_{j=1}^{n_2} p_{ik} d_{G_2}(u_i, u_j) p_{jl}$, a sum of path distances in G_2 weighted by how strongly v_k and v_l are connected, through P, to the endpoints of those paths, u_i and u_j.

Parameters α and β are rescaling parameters to compensate for different graph sizes; in other words, P must only ensure that processes 1 and 2 above agree up to some multiplicative constant. In operator theory terminology, the Laplacian is a time evolution operator for the single particle diffusion equation: $L_i = A(G_i) - \text{diag}(1 \cdot A(G_i))$. This operator evolves the probability distribution of states of a single-particle diffusion process on a graph G_i (but other processes could be used - for example, a chemical reaction network or multiple-particle diffusion). The process L defines a probability-conserving Master Equation of nonequilibrium statistical mechanics $dp/dt = L \cdot p$ which has formal solution $p(t) = \exp(tL) \cdot p(0)$. Pre-multiplication by the prolongation matrix P is clearly a linear operator i.e. linear transformation from $\mathbb{R}^{n_1}$ to $\mathbb{R}^{n_2}$. Thus, we are requiring P which minimizes the degree to which the operator diagram

$$\begin{array}{ccc} L_1 & \xrightarrow{\Delta t} & L_1{}' \\ \downarrow P & & \downarrow P \\ L_2 & \xrightarrow{\Delta t} & L_2{}' \end{array} \qquad \text{(Diagram 1)}$$

fails to commute. Δt of course refers to advancement in time. See [55], Figure 1, for a more complete version of this commutative diagram for model reduction.

We thus include in our objective function terms with 1) graph diffusion and 2) graph locality as the underlying process matrices (T, the Manhattan distance matrix, cannot be considered a time evolution operator because it is not probability-preserving). Parameter s adjusts the

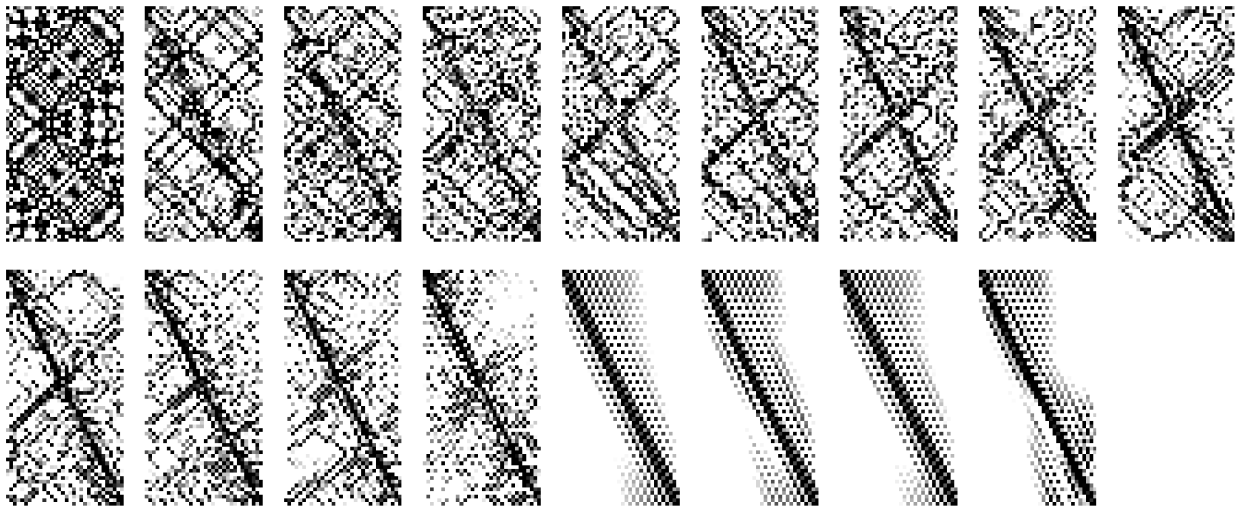

Figure 6.1: Several solutions of our objective function found by PyManOpt as s, the relative weight of the two terms of our objective function, is tuned from 0 (fully diffuse, top left) to 1 (fully local, bottom right). Within each subplot, grayscale indicates the magnitude of matrix entries. Note that the P matrices found with $s = 0$ do not appear to be structured in a way which respects the locality of the original graphs, whereas the matrices with $s = 1$ do.

relative strength of these terms to each other; so we may find "fully diffuse" P when $s = 0$ and "fully local" P when $s = 1$. Figure 6.1 illustrates this tradeoff for an example prolongation problem on a pair of grid graphs, including the transition from a global optimum of the diffusion term to a global optimum of the locality term. In each case, we only require P to map these processes into one another up to a multiplicative constant: α for the diffusion term and β for the locality term. Exhaustive grid search over α and β for a variety of prolongations between (a) path graphs and (b) 2D grid graphs of varying sizes has suggested that for prolongation problems where the G_i are both paths or both grids, the best values (up to the resolution of our search, 10^{-6}) for these parameters are $\alpha = 1.0$ and $\beta = n_1/n_2$. However, we do not expect this scaling law to hold for general graphs.

6.3 Comparison of Numerical Methods

To find minima of this objective function, we explore several numerical methods. For prototyping, we initially used Nelder-Mead [75] optimization with explicit orthogonality constraints, as implemented in the Mathematica commercial computer algebra program. However, this approach does not scale - in our hands Mathematica was not able to minimize this objective function with more than approximately 200 unknowns in a reasonable amount of time. Our next approach was to use a special-purpose code [114] for orthogonally-constrained gradient descent. While this software package scaled well to pairs of large graphs, it required many random restarts to find minima of our objective function. Motivated by its automatic differentiation capability and its ability to handle larger numbers of unknowns, we tried the TensorFlow minimization package [1]: first custom-written code and then a package called PyManOpt [105] which performs manifold-constrained optimization of arbitrary objective functions expressed as TensorFlow computation graphs. PyManOpt is able to perform first- and second-order minimization while staying within the constraint manifold (rather than our custom code, which takes gradient descent steps and then projects back to the constraint surface). These latter two approaches performed best in terms of optimization solution quality, and we compare them more throughly below.

To compare the performance of the TensorFlow method and the PyManOpt method, we explore the performance of both minimization methods as the relative weight s of the locality and diffusion terms is adjusted. Figure 6.2 shows the tradeoff plot of the optimized unweighted value of each term as the weight parameter s is tuned. The four subplots correspond to four runs of this experiment with differing sizes of graphs; in each we find optimal prolongations from a cycle graph of size n to one of size $2n$. The PyManOpt-based minimization code is clearly superior, as we see a clear linear tradeoff between objective function terms as a function of s. The TensorFlow code which maintains orthogonality by projecting

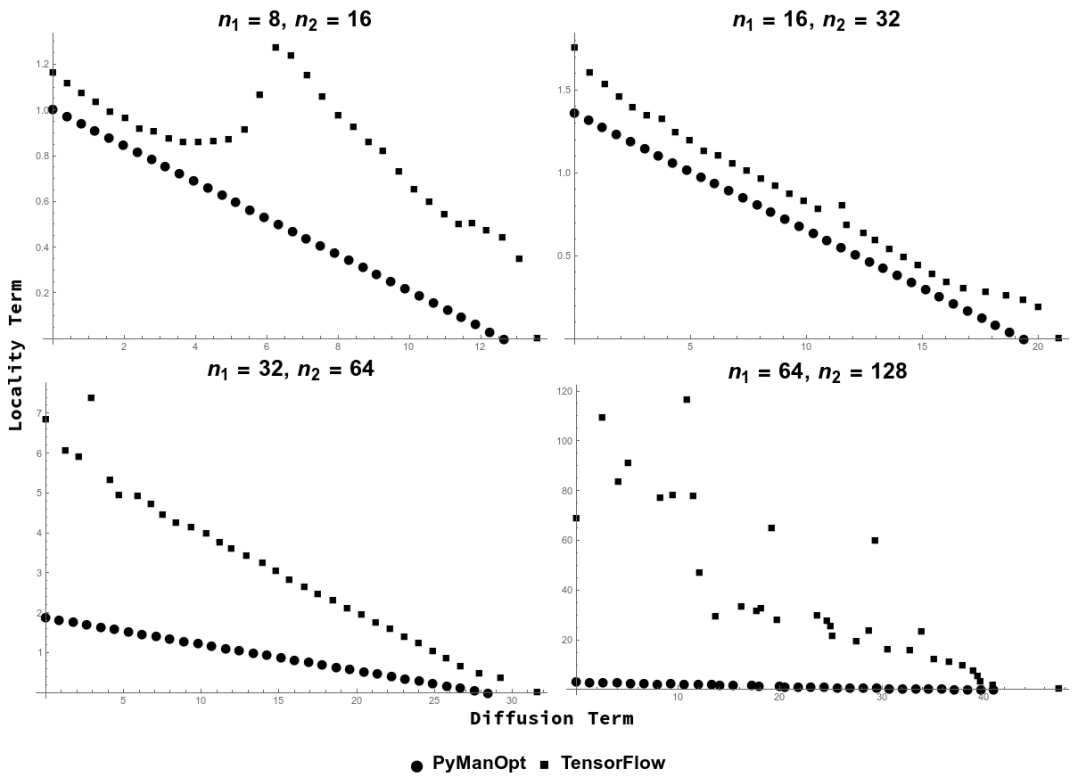

Figure 6.2: Tradeoff plot of locality vs. diffusion for several pairs of graphs. Multiple solutions are plotted in each subplot, representing the adjustment of the s parameter in our objective function from totally local to totally diffuse. We see that the PyManOpt boundary shows a linear tradeoff between the two terms of the objective function as their relative weight is tuned, whereas the Tensorflow boundary is more irregular. Furthermore, the PyManOpt method in general finds optima with lower objective function value than Tensorflow (for both objectives). We note that Nelder-Mead in Mathematica would not be able to tackle problems of this size, and the method due to Wen and Yin [114] produced points which are off of this plot by at least an order of magnitude (we do not present these points).

back to the Stiefel manifold falls short of this boundary in all cases. Therefore unless otherwise specified, for the rest of this chapter when we discuss solving for P matrices, we are reporting results of using the PyManOpt method.

6.3.1 Initialization

We initialize our minimization with an upper-bound solution given by the Munkres minimum-cost matching algorithm; the initial P is $m^*(L_1, L_2)$ as defined in equation 1.6, i.e. the binary matrix where an entry $P_{(i,j)}$ is 1 if the pair (i, j) is one of the minimal-cost pairs selected by the minimum-cost assignment algorithm, and 0 otherwise. While this solution is, strictly speaking, minimizing the error associated with mapping the spectrum of one graph into the spectrum of the other (rather than actually mapping a process running on one graph into a process on the other) we found it to be a reasonable initialization, outperforming both random restarts and initialization with the appropriately sized block matrix $\begin{pmatrix} I \\ 0 \end{pmatrix}$.

6.3.2 Precomputing P matrices

For some structured graph lineages it may be possible to derive formulaic expressions for optimal P and α, as a function of the lineage index. For example, during our experiments we discovered species of P which are local minima of prolongation between path graphs, cycle graphs, and grid graphs. A set of these outputs is shown in Figure 6.1. They feature various diagonal patterns as naturally idealized in Figure 6.3. These idealized versions of these patterns all are also empirical local minima of our optimization procedure, for $s = 0$ or $s = 1$, as indicated. Each column of Figure 6.3 provides a regular family of P structures for use in our subsequent experiments in Section 6.5. We have additionally derived closed-form expressions for global minima of the diffusion term of our objective function for some graph families (cycle graphs and grid graphs with periodic boundary conditions). However, in practice these global minima are nonlocal (in the sense that they are not close to optimizing the locality term) and thus may not preserve learned spatial rules between weights in levels of our hierarchy.

Examples of these formulaic P matrices can be seen in Figure 6.3. Each column of that figure shows increasing sizes of P generated by closed-form solutions which were initially found by solving smaller prolongation problems (for various graph pairs and choices of s) and generalizing the solution to higher n. Many of these examples are similar to what a human being would design as interpolation matrices between cycles and periodic grids. However, (a) they are valid local optima found by our optimization code and (b) our approach generalizes to processes running on more complicated or non-regular graphs, for which there may not be an obvious *a priori* choice of prolongation operator.

We highlight the best of these multiple species of closed-form solution, for both cycle graphs and grid graphs. The interpolation matrix-like P seen in the third column of the "Cycle Graphs" section, or the sixth column of the "Grid Graphs" section of Figure 6.3, were the local optima with lowest objective function value (with $s = 1$, i.e. they are fully local). As the best optima found by our method(s), these matrices were our choice for line graph and grid graph prolongation operators in our neural network experiments, detailed in Section 6.5. We reiterate that in those experiments we do not find the P matrices via any optimization method - since the neural networks in question have layer sizes of order 10^3, finding the prolongation matrices from scratch may be computationally difficult. Instead, we use the solutions found on smaller problems as a recipe for generating prolongation matrices of the proper size.

Furthermore, given two graph lineages $G_1^{(1)}, G_1^{(2)}, G_1^{(3)} \ldots$ and $G_2^{(1)}, G_2^{(2)}, G_2^{(3)} \ldots$, and sequences of optimal matrices $P_1^{(1)}, P_1^{(2)}, P_1^{(3)} \ldots$ and $P_2^{(1)}, P_2^{(2)}, P_2^{(3)} \ldots$ mapping between successive members of each, we can construct P which are related to the optima for prolonging between members of a new graph lineage which is comprised of the levelwise graph box product of the two sequences. We show in (Section 3.5, Corollary 3.5.2) conditions under which the value of the objective function at $P_{\text{box}}^{(i)} = P_1^{(i)} \otimes P_2^{(i)}$ is an upper bound of the optimal value for prolongations between members of the lineage $G_1^{(1)} \square G_2^{(1)}, G_1^{(2)} \square G_2^{(2)}, G_1^{(3)} \square G_2^{(3)}, \ldots$.

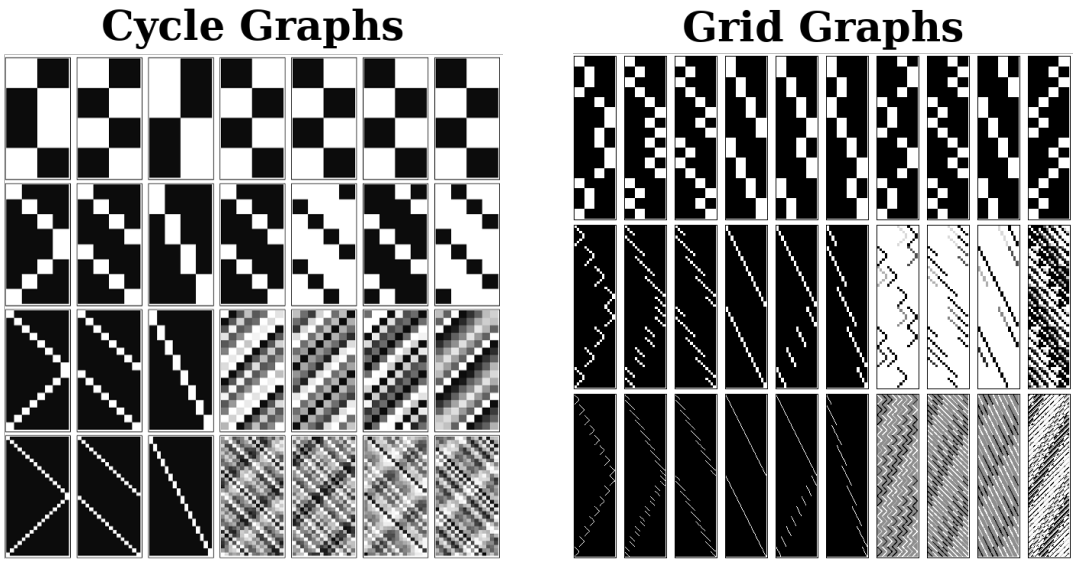

Figure 6.3: Examples of P matrices for cycle graph (left) and grid graph (right) prolongation problems of various sizes, which can be generated by closed-form representations dependent on problem size. Within each of the top and bottom plots, columns represent a series of matrices each generated by a particular numerical recipe, with rows representing increasing sizes of prolongation problem. Each matrix plot is a plot of the absolute value of matrix cell values. These closed-form representations were initially found as local minima of our objective function on small problems and then generalized to closed-form representations. For the "Cycle Graphs" section, the prolongation problems were between cycle graphs of sizes $n_1 = 2, 4, 8, 16$ and $n_2 = 2 * n_1$. Columns 1-3 were solutions found with $s = 1$ (fully local), and the rest were found with $s = 0$ (fully diffuse). For the "Grid Graphs" section, the prolongation problems were between grids of size (n_1, n_1) to grids of size $(2n_1, 2n_1)$ for n_1 in $4, 8, 16$. Columns 1-6 are fully local and columns 7-10 are fully diffuse, respectively. As in Figure 6.1, grayscale values indicate the magnitude of each matrix entry.

We leave open the question of whether such formulaic P exist for other families of structured graphs (complete graphs, k-partite graphs, etc.). Even in cases where formulaic P are not known, the computational cost of numerically optimizing over P may be amortized, in the sense that once a P-map is calculated, it may be used in many different hierarchical neural networks or indeed many different multiscale models.

6.4 Multiscale Artificial Neural Network Algorithm

In this section we describe the Multiscale Artificial Neural Network (MsANN) training procedure, both in prose and in pseudocode (Algorithm 3). Let $\mathcal{M}_0 \dots \mathcal{M}_L$ be a sequence of neural network models with identical "aspect ratios" (meaning the sizes of each layer relative to other layers in the same model) but differing input resolution, so that $\mathcal{M}_0$ operates at the finest scale and $\mathcal{M}_L$ at the coarsest. For each model $\mathcal{M}_l$, let $\theta_0^{(l)}, \theta_1^{(l)}, \dots \theta_{n_{\text{vars}}-1}^{(l)}$ be a list of the n_{vars} network parameters (each in matrix or vector form) in some canonical order which is maintained across all scales. Let the symbol $\mathcal{P}_j^{(l)}$ represent either:

- If the network parameters $\theta_j^{(i)}$ at levels $i = 0 \dots L$ are weight matrices between layers m_1 and m_2 of each hierarchy, then $\mathcal{P}_j^{(l)}$ represents a pair of matrices $\left(P_{\text{input}_j}^{(l)}, P_{\text{output}_j}^{(l)}\right)$, such that:
  - $P_{\text{input}_j}^{(l)}$ prolongs or restricts between possible values of nodes in layer m_1 of model $\mathcal{M}_l$, and values of nodes in layer m_1 of model $\mathcal{M}_{l+1}$.
  - $P_{\text{output}_j}^{(l)}$ does the same for possible values of nodes in layer m_2 of each model.
- If the network parameters $\theta_j^{(i)}$ at levels $i = 0 \dots L$ are bias vectors which are added to layer m of each hierarchy, then $\mathcal{P}_j^{(l)}$ represents a single $P_j^{(l)}$ which prolongs or restricts between possible values of nodes in layer m of model $\mathcal{M}_l$, and values of nodes in layer m of model $\mathcal{M}_{l+1}$.

As a concrete example, for a hierarchy of single-layer networks $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2$, each with one weight matrix $W^{(l)}$ and one bias vector $b^{(l)}$, we could have $\theta_0^{(l)} = W^{(l)}, \theta_1^{(l)} = b^{(l)}$ for each $\mathcal{M}_l$. $\mathcal{P}_0^{(0)}$ would represent a pair of matrices which map between the space of possible values of $W^{(0)}$ and the space of possible values of $W^{(1)}$ in a manner detailed in the next section. On the other hand, $\mathcal{P}_1^{(0)}$ would represent a single matrix which maps between $b^{(0)}$ and $b^{(1)}$.

Similarly, $\mathcal{P}_0^{(1)}$ would map between $W^{(1)}$ and $W^{(2)}$, and $\mathcal{P}_1^{(1)}$ between $b^{(1)}$ and $b^{(2)}$. In Section 6.4.2, we describe a general procedure for training such a hierarchy according to standard multilevel modeling schedules of refinement and coarsening, with the result that the finest network, informed by the weights of all coarser networks, requires fewer training examples.

6.4.1 Weight Prolongation and Restriction Operators

In this section we introduce the prolongation and restriction operators for neural network weight and bias optimization variables in matrix or vector form respectively.

For a 2D matrix of weights W, define

$$\begin{aligned} \text{Pro}_{\mathcal{P}} \circ W \equiv \text{Pro}_{(P_{\text{input}}, P_{\text{output}})} \circ W &\equiv P_{\text{input}} W P_{\text{output}}^T \\ \text{Res}_{\mathcal{P}} \circ W \equiv \text{Res}_{(P_{\text{input}}, P_{\text{output}})} \circ W &\equiv P_{\text{input}}^T W P_{\text{output}} \end{aligned} \tag{6.2}$$

where P_{input} and P_{output} are each prolongation maps between graphs which respect the structure of the spaces of inputs and outputs of W, i.e. whose structure is similar to the structure of correlations in that space. Further research is necessary to make this notion more precise. In our experiments on autoencoder networks in Section 6.5, we use example problems with an obvious choice of graph to use. In these 1D and 2D machine vision tasks, where we expect each pixel to be highly correlated with the activity of its immediate neighbors in the grid, 1D and 2D grids are clear choices of graphs for our prolongtion matrix calculation. Other choices may lead to similar results; for instance, we speculate that since neural network weight matrices may be interpreted as the weights of a multipartite graph of connected neurons in the network, these graphs could be an alternate choice of structure to prolong/restrict between. We leave for future work the development of automatic methods for determining these structures.

Note that the Pro and Res linear operators satisfy $\text{Res}_{\mathcal{P}} \circ \text{Pro}_{\mathcal{P}} = I$, the identity operator, so $\text{Pro}_{\mathcal{P}} \circ \text{Res}_{\mathcal{P}}$ is a projection operator.

For a 1D matrix of biases b, define

$$\begin{aligned} \text{Pro}_{\mathcal{P}} \circ b &= P \cdot b \\ \text{Res}_{\mathcal{P}} \circ b &= P^T \cdot b \end{aligned} \tag{6.3}$$

where, as before, we require that P be a prolongation matrix between graphs which are appropriate for the dynamics of the network layer where b is applied. Again $\text{Res}_{\mathcal{P}} \circ \text{Pro}_{\mathcal{P}} = I$.

Given such a hierarchy of models $\mathcal{M}_0 \dots \mathcal{M}_L$, and appropriate Pro and Res operators as defined above, we define a *Multiscale Artificial Neural Network (MsANN)* to be a neural network model with the same layer and parameter dimensions as the largest model in the hierarchy, where each layer parameter Θ_j is given by a sum of prolonged weight matrices from level j of each of the models defined above:

$$\Theta_j = \theta_j^{(0)} + \text{Pro}_{1\to 0} \circ \theta_j^{(1)} + \text{Pro}_{2\to 0} \circ \theta_j^{(2)} \dots \text{Pro}_{L\to 0} \circ \theta_j^{(L)} \tag{6.4}$$

Here we are using $\text{Pro}_{k\to 0}$ as a shorthand to indicate composed prolongation from model k to model 0, so if $\theta_j^{(i)}$ are weight variables we have (by Equation 6.2)

$$\begin{aligned} \Theta_j = \theta_j^{(0)} &+ P^{(0)}_{\text{input}_j} \theta_j^{(1)} \left(P^{(0)}_{\text{output}_j}\right)^T \\ &+ P^{(0)}_{\text{input}_j} P^{(1)}_{\text{input}_j} \theta_j^{(2)} \left(P^{(1)}_{\text{output}_j}\right)^T \left(P^{(0)}_{\text{output}_j}\right)^T \\ &+ \quad \dots \quad + \left(P^{(0)}_{\text{input}_j} \dots P^{(L-1)}_{\text{input}_j} \theta_j^{(L)} \left(P^{(L-1)}_{\text{output}_j}\right)^T \dots \left(P^{(0)}_{\text{output}_j}\right)^T\right) \end{aligned} \tag{6.5}$$

and if $\theta_j^{(i)}$ are bias variables we have (by Equation 6.3)

$$\Theta_j = \theta_j^{(0)} + P^{(0)}_{\text{bias}_j} \theta_j^{(1)} + P^{(0)}_{\text{bias}_j} P^{(1)}_{\text{bias}_j} \theta_j^{(2)} + \dots + \left(P^{(0)}_{\text{bias}_j} P^{(1)}_{\text{bias}_j} \dots P^{(L-1)}_{\text{bias}_j} \theta_j^{(L)}\right) \tag{6.6}$$

We note that matrix products such as $P^{(0)}_{\text{input}_j} \dots P^{(k)}_{\text{input}_j}$ need only be computed once, during model construction.

6.4.2 Multiscale Artificial Neural Network Training

The Multiscale Artificial Neural Network algorithm is defined in terms of a recursive 'cycle' that is analogous to one epoch of default neural network training. Starting with $\mathcal{M}_0$ (i.e. the finest model in the hierarchy), we call the routine MsANNCycle(0), which is defined recursively. At any level l, MsANNCycle trains the network at level l for k batches of training examples, recurses by calling MsANNCycle($l+1$), and then returns to train for k further batches at level l. The number of calls to MsANNCycle($l+1$) inside each call to MsANNCycle(l) is given by a parameter γ.

This is followed by additional training at the refined scale; this process is normally [111] referred to by the multigrid methods community as 'restriction' and 'prolongation' followed by 'smoothing'. The multigrid methods community additionally has special names for this type of recursive refining procedure with $\gamma = 1$ ("V-Cycles") and $\gamma = 2$ ("W-Cycles"). See Figure 6.4 for an illustration of these contraction and refinement schedules. In our numerical experiments below, we examine the effect of this parameter on multigrid network training.

Neural network training with gradient descent requires computing the gradient of the error E between the network output and target with regard to the network parameters. This gradient is computed by taking a vector of error for the nodes in the output layer, and *backpropagating* that error backward through the network layer by layer to compute the individual weight matrix and bias vector gradients. An individual network weight or bias term w is then adjusted using gradient descent, i.e. the new value w' is given by $w' = w - \eta \frac{dE}{dw}$, where η is a learning rate or step size. Several techniques can be used to dynamically change learning rate during model training - we refer the reader to [13] for a description of these techniques

and backpropagation in general.

Our construction of the MsANN model above did not make use of the Res (restriction) operator - we show here how this operator is used to compute the gradient of the coarsened variables in the hierarchy. This can be thought of as continuing the process of backpropagation through the Pro operator. For these calculations we assume Θ_j is a weight matrix, and derive the gradient for a particular $\theta_j^{(k)}$. For notational simplicity we rename these matrices W and V, respectively. We also collapse the matrix products

$$P^{(\text{input})} = P^{(0)}_{\text{input}_j} P^{(1)}_{\text{input}_j} \cdots P^{(k)}_{\text{input}_j} \tag{6.7}$$

$$\left(P^{(\text{output})}\right)^T = \left(P^{(L-1)}_{\text{output}_j}\right)^T \left(P^{(L-2)}_{\text{output}_j}\right)^T \cdots \left(P^{(0)}_{\text{output}_j}\right)^T \tag{6.8}$$

Let $\frac{dE}{dW}$ be a matrix where $\left(\frac{dE}{dW}\right)_{mn} = \frac{dE}{dw_{mn}}$, calculated via backpropagation as described above. Then, for some m, n:

$$\begin{aligned}
\frac{dw_{mn}}{dv_{kl}} &= \frac{d}{dv_{kl}}(\ldots + \text{Pro} \circ V + \ldots)_{mn} \\
&= \frac{d}{dv_{kl}}(\ldots + \text{Pro}_{k\to 0} \circ V + \ldots)_{mn} = \frac{d}{dv_{kl}}(\text{Pro}_{k\to 0} \circ V)_{mn} \\
&= \frac{d}{dv_{kl}}\left(P^{(\text{input})} V \left(P^{(\text{output})}\right)^T\right)_{mn} = \frac{d}{dv_{kl}}\left(\sum_{a,b} p^{(\text{input})}_{ma} v_{ab} p^{(\text{output})}_{nb}\right) \\
&= \left(p^{(\text{input})}_{mk} p^{(\text{output})}_{nl}\right)
\end{aligned} \tag{6.9}$$

Then,

$$\begin{aligned}
\frac{dE}{dv_{kl}} &= \sum_{m,n} \frac{dE}{dw_{mn}} \frac{dw_{mn}}{dv_{kl}} \\
&= \sum_{m,n} \frac{dE}{dw_{mn}} p^{(\text{input})}_{mk} p^{(\text{output})}_{nl} \\
&= \left(\left(P^{(\text{input})}\right)^T \frac{dE}{dW} P^{(\text{output})}\right)_{kl}
\end{aligned} \tag{6.10}$$

and so

$$\frac{dE}{dV} = \left(P^{(\text{input})}\right)^T \frac{dE}{dW} P^{(\text{output})}$$

and therefore finally

$$\frac{dE}{dV} = \text{Res}_{0 \to k} \circ \frac{dE}{dW} \tag{6.11}$$

where Res is as in 6.2.

Algorithm 3 One 'cycle' of the MsANN procedure.

Procedure `MsANNCycle(`l`):`

Train model $\mathcal{M}_l$ for k batches, where each consists of:

1. Feed examples through the network in feed-forward mode;
2. Compute error E between network output and target;
3. Use the classical backpropagation algorithm to compute the gradient of top-level parameter Θ_j w.r.t. this error;
4. Use the appropriate Res operations to compute the gradient of E w.r.t. the parameters in $\mathcal{M}_l$, as described in Equation 6.11.

if max_depth *has not been reached* **then**

  for $1 \leq i \leq \gamma$ **do**

    `MsANNCycle(`$l + 1$`);`

    Train model $\mathcal{M}_l$ for k batches, as above

  end

end

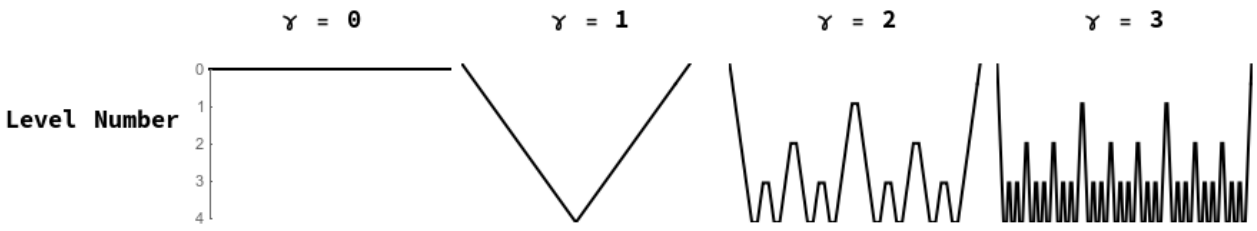

Figure 6.4: Visits to models in a hierarchy of neural networks realized by several values of the recursion frequency parameter γ. The $\gamma = 1$ case and the $\gamma = 2$ case are referred to as "V-cycles" and "W-cycles", respectively. Each time the multilevel training procedure visits a level, it performs some number, k, of smoothing steps (i.e. gradient descent at that resolution) at that model.

We also note here that our code implementation of this procedure does not make explicit use of the Res operator; instead, we use the automatic differentiation capability of Tensorflow [1] to compute this restricted gradient. This is necessary because data is supplied to the model, and error is calculated, at the finest scale only. Hence we calculate the gradient at this scale and restrict it to the coarser layers of the model. It may be possible to feed coarsened data through only the coarser layers of the model, eliminating the need for computing the gradient at the finest scale, but we do not explore this method in this thesis.

6.5 Machine Learning Experiments

We present four experiments using this Multiscale Neural Network method. All of the experiments below demonstrate that our multigrid method outperforms default training (i.e. training only the finest-scale network), in terms of the number of training examples (summed over all scales) needed to reach a particular mean-squared error (MSE) value. We perform two experiments with synthetic machine vision tasks, as well as two experiments with benchmark image datasets for machine learning. While all of the examples presented here are autoencoder networks (networks whose training task is to reproduce their input at the output layer, while passing through a bottleneck layer or layers), we do not mean to imply

that MsANN techniques are constrained to autoencoder networks. All network training uses the standard backpropagation algorithm to compute training gradients, and this is the expected application domain of our method. Autoencoding image data is a good choice of machine learning task for our experiments for two main reasons. First, autoencoders are symmetric and learn to reproduce their input at their output. Other ML models (for instance, neural networks for classification) have output whose nodes are not spatially correlated, and it is not yet clear if our approach will generalize to this type of model. Secondly, since the single and double-object machine vision tasks operate on synthetic data, we can easily generate an arbitrary number of samples from the data distribution, which was useful in the early development of this procedure. Our initial successes on this synthetic data led us to try the same task with a standard benchmark real-world dataset. For each experiment, we use the following measure of computational cost to compare relative performance. Let $|\mathcal{M}|$ be the number of trainable parameters in model $\mathcal{M}$. We compute the cost of a training step of the weights in model $\mathcal{M}_k$ using a batch of size b as $\frac{|\mathcal{M}_k|}{|\mathcal{M}_0|}b$. The total cost $C(t)$ of training at step t is the sum of this cost over all training steps thus far at all scales. This cost is motivated by the fact that the number of multiply operations for backpropagation is $O(nm)$ in the total number of network parameters m and training examples n, so we are adding up the relative cost of using a batch of size b to adjust the weights in model $\mathcal{M}_k$, as compared to the cost of using that same batch to adjust the weights in $\mathcal{M}_0$.

6.5.1 Simple Machine Vision Task

As an initial experiment in the capabilities of hierarchical neural networks, we first try two simple examples: finding lower-dimensional representation of two artificial datasets. In both cases, we generate synthetic data by uniformly sampling from

1. the set of binary-valued vectors with one "object" comprising a contiguous set of pixels

one-eighth as long as the entire vector set to 1, and the rest zero; and

2. the set of vectors with two such non-overlapping objects.

In each case, the number of possible unique data vectors is quite low: for inputs of size 1024, we have 1024 - 128 = 896 such vectors. Thus, for both of the synthetic datasets we add binary noise to each vector, where each "pixel" of the input has an independent chance of firing spuriously with $p = 0.05$. This noise in included only in the input vector, making these networks *Denoising Autoencoders*: models whose task is to remove noise from an input image.

Single-Object Autoencoder

We first test the performance of this procedure on a simple machine vision task. The neural networks in our hierarchy of models each have layer size specification (in number of units) $[2^n, 2^{n-2}, 2^{n-3}, 2^{n-2}, 2^n]$ for n in $\{10, \ldots 6\}$, with a bias term at each layer and sigmoid logistic activation. We present the network with binary vectors which are 0 everywhere except for a contiguous segment of indices of length 2^{n-3} which are set to 1, with added binary noise as described above. The objective function to minimize is the mean-squared error (MSE) between the input and output layers. Each model in the hierarchy is trained using RMSPropOptimizer in Tensorflow, with learning rate $\alpha = 0.0005$.

The results of this experiment are plotted in Figure 6.5 and summarized in Table 6.1. We perform multiple runs of the entire training procedure with differing values of k (the number of smoothing steps), γ (the multigrid cycle parameter), and L (depth of hierarchy). Notably, nearly all multigrid schedules demonstrate performance gains over the default network (i.e. the network which trains only at the $l = 0$ scale), with more improvement for higher values of k, L, and γ. The hierarchy which learned most rapidly was the deepest model ($L = 6$)

with $k = 4$ and $\gamma = 3$. Those multigrid models which did not improve over the default network were only slightly more computationally expensive per unit of accuracy than their default counterparts, and the multigrid models which did improve, improved significantly.

	Best MsANN	Worst MsANN	Default	Best MsANN params
Final MSE	6.612×10^{-4}	4.431×10^{-3}	3.654×10^{-3}	$(\gamma = 3, L = 5, k = 004)$
Cost to $\frac{1}{10}$ MSE	7.342×10^{3}	1.640×10^{5}	1.266×10^{5}	$(\gamma = 3, L = 6, k = 004)$

Table 6.1: Best performance (on validation dataset for the one-object autoencoding task) by any combination of parameters in our sweep over values for γ (recursion constant), L (depth of network), and k (number of batches processed at each visit to each level). We report the final Mean-Squared Error for both the best and worst combination of these parameters, as well as for default training. We also report the best combination of parameters. The second row indicates the cost $C(t)$ necessary to train each model to $\frac{1}{10}$ of the error at which it began. The best MsANN network reaches this threshhold in an order of magnitude less cost, and its final error is roughly half that of the default model, demonstrating clear improvement over training without multigrid.

Double-Object Autoencoder

We repeat the above experiment with a slightly more difficult machine vision task - the network must learn to de-noise an image with two (non-overlapping) 'objects' in the visual field. We use the same network structure and training procedure, and note that we see again (plotted in Figure 6.5 and summarized in Table 6.2) that the hierarchical model is more efficient, reaching lower error in the same amount of computational cost $C(t)$. The multigrid neural networks again typically learn much more rapidly than the non-multigrid models.

	Best MsANN	Worst MsANN	Default	Best MsANN params
Final MSE	2.576×10^{-3}	8.998×10^{-3}	8.816×10^{-3}	$(\gamma = 3, L = 6, k = 002)$
Cost to $\frac{1}{10}$ MSE	2.433×10^{4}	2.623×10^{5}	2.216×10^{5}	$(\gamma = 3, L = 6, k = 016)$

Table 6.2: Best performance (on validation dataset for the two-object autoencoding task). Again the MsANN network demonstrates performance and accuracy gains over neural network training alone. See Table 6.1.

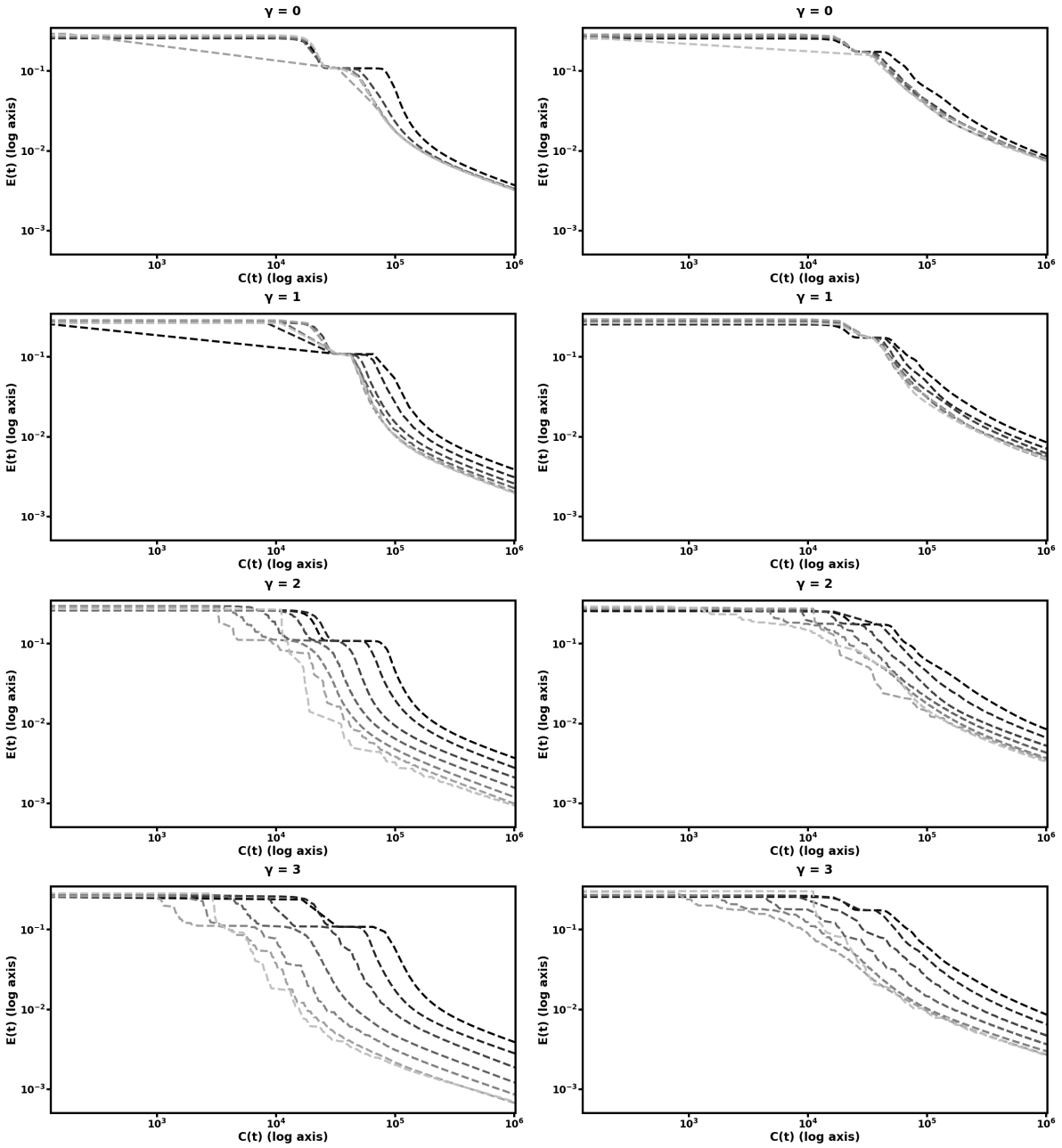

Figure 6.5: Log-log plots of accuracy $E(t)$ as a function of training cost $C(t)$ attained by a variety of hierarchical neural networks training on a simple machine vision task, demonstrating that deeper hierarchies with more mutligrid behavior learn more rapidly. Plots are ordered from top to bottom in increasing depth of recursion parameter γ; left plots are the single-object experiments and right plots are the double-object experiments. Within each plot, different curves represent different values of the depth of hierarchy, from $L = 6$ (lightest) to $L = 0$ (darkest). Each line is the best run for that pair (L, γ) over all choices of k (number of smoothing steps at each level) in $\{1, 2, 4, 8, 16, 32, 64, 128\}$.

6.5.2 MNIST

To supplement the above synthetic experiments with one using real-world data, we perform the same experiment with an autoencoder for the MNIST handwritten digit dataset [65, 66]. In this case, rather than the usual MNIST classification task, we use an autoencoder to map the MNIST images into a lower-dimensional ($d = 128$) space with good reconstruction. We use the same network structure as in the 1D vision example; also as in that example, each network in the hierarchy is constructed of fully connected layers with bias terms and sigmoid activation, and smoothing steps are performed with RMSProp [51] with learning rate 0.0005. The only difference is that in this example we do not add noise to the input images, since the dataset is larger by two orders of magnitude.

In this experiment, we see (in Figure 6.6 and Table 6.3) similar improvement in efficiency. Table 6.3 summarizes these results: the best multilevel models learned more rapidly and achieved lower error than their single-level counterparts, whereas the worst multilevel models performed on par with the default model. Because the MNIST data is comprised of 2D images, we tried using P matrices which were the optima of prolongation problems between grids of the appropriate sizes, in addition to the same 1D P used in the prior two experiments. The difference in performance between these two choices of underlying structure for the prolongation maps can be seen in Figure 6.6. With either approach, we see similar results to the synthetic data experiment, in that more training steps at the coarser layers results in improved learning performance of the finer networks in the hierarchy. However, the matrices optimized for 2D prolongation perform marginally better than their 1D cousins, - in particular, the multigrid hierarchy with 2D prolongations took 60% of the computational cost to reduce its error to $\frac{1}{10}$ of its original value, as compared to the 1D version. We explore the effect of varying the strategy used to pick P in Subsection 6.5.3.

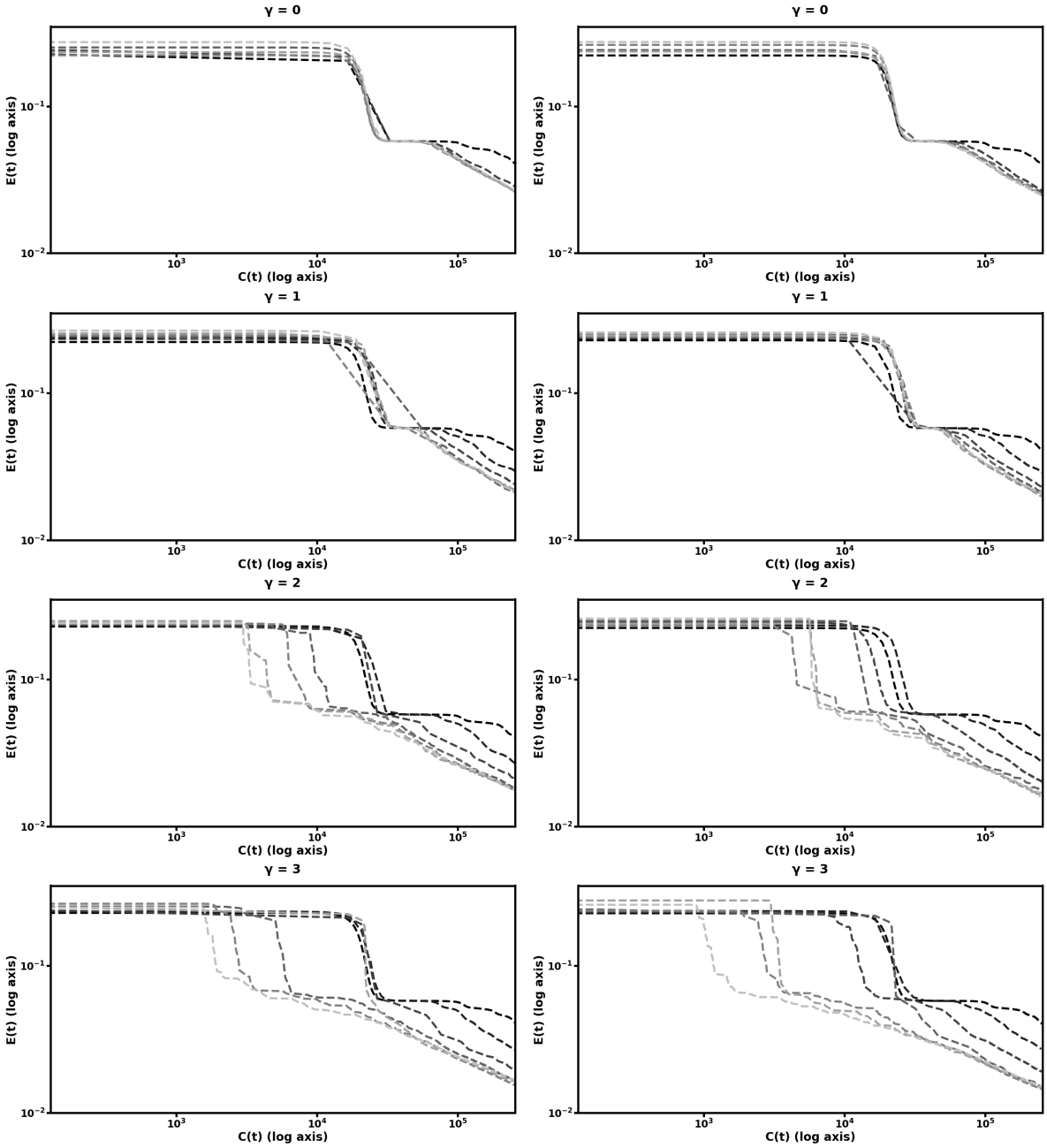

Figure 6.6: Log-log plots of mean-squared error $E(t)$ on MNIST autoencoding task as a function of computational cost $C(t)$; the left plots represent multigrid performed with path graph prolongations for each layer while the right plots used grid-based prolongation. While both approaches show gains over default learning in both speed of learning and final error value, the one which respects the spatial structure of the input data improves more rapidly. Subplot explanations are the same as in Figure 6.5.

Path-Based P Matrices				
	Best MsANN	Worst MsANN	Default	Best MsANN params
Final MSE	1.547×10^{-2}	4.605×10^{-2}	4.171×10^{-2}	$(\gamma = 3, L = 4, k = 008)$
Cost to $\frac{1}{10}$ MSE	7.207×10^{4}	N/A	N/A	$(\gamma = 3, L = 5, k = 032)$
Grid-Based P Matrices				
	Best MsANN	Worst MsANN	Default	Best MsANN params
Final MSE	1.436×10^{-2}	4.620×10^{-2}	4.132×10^{-2}	$(\gamma = 3, L = 4, k = 002)$
Cost to $\frac{1}{10}$ MSE	5.095×10^{4}	N/A	N/A	$(\gamma = 3, L = 6, k = 128)$

Table 6.3: Best performance (on validation dataset for the MNIST autoencoding task). See Table 6.1. Upper section represents scores attained by a MsANN with path-based prolongation, lower section represents grid-based prolongation. Entries marked N/A did not reach $\frac{1}{10}$ of their initial error during training.

6.5.3 Experiments of Choice of P

To further explore the role of the structure of P in these machine learning models, we compare the performance of several MsANN models with P generated according to various strategies. Our initial experiment on the MNIST dataset used the exact same hierarchical network structure and prolongation/restriction operators as the example with 1D data, and yielded marginal computational benefit. We were thus motivated to try this learning task with prolongations which are designed for for 2D grid-based model architectures, as well as trying unstructured (random orthogonal) matrices as a baseline. More precisely, our 1D experiments used P matrices resembling those in column 3 of the "Cycle Graphs" section of Figure 6.3. We instead, for the MNIST task, used P matrices like those in column 6 of the "Grid Graphs" section of the same figure. In Figure 6.7, we illustrate the difference in these choices for the MNIST training task, with the same choice of multigrid training parameters: $(L = 6, \gamma = 3, k = 1)$. We compare the following strategies for generating P:

1. As local optima of a prolongation problem between 1D grids, with periodic boundary conditions;

2. As local optima of a prolongation problem between 2D grids, with periodic boundary

conditions;

3. As in 2, but shuffled along the first index of the array.

Strategy 3 was chosen to provide the same degree of connectivity between each coarse variable and its related fine variables as strategy 2, but in random order i.e. connected in a way which is unrelated to the 2D correlation between neighboring pixels. We see in Figure 6.7 that the two strategies utilizing local optima outperform both the randomized strategy and default training (training only the finest scale). Furthermore, strategy 2 outperforms strategy 1, although the latter eventually catches up at the end of training, when coarse-scale weight training has diminishing marginal returns. The random strategy is initially on par with the two optimized ones (we speculate that this is due to the ability to affect many fine-scale variables at once, even in random order, which may make the gradient direction easier to travel), but eventually falls behind, at times being less efficient than default training. We leave for further work the question of whether there are choices of prolongation problem which are even more efficient for this machine learning task. We also compare all of the preceeding models to a model which has the same structure as a MsANN model (a hierarchy of coarsened variables with Pro and Res operators between them), but which was trained by training all variables in the model simultaneously. This model performs on par with the default model, illustrating the need for the multilevel training schedule dictated by the choice of γ.

6.5.4 Summary

We see uniform improvement (as the parameters L and γ are increased) in the rate of neural network learning when models are stacked in the type of multiscale hierarchy we define in equations 6.2 and 6.3, despite the diversity of machine learning tasks we examine. Furthermore, this improvement is marked: the hierarchical models both learn more rapidly

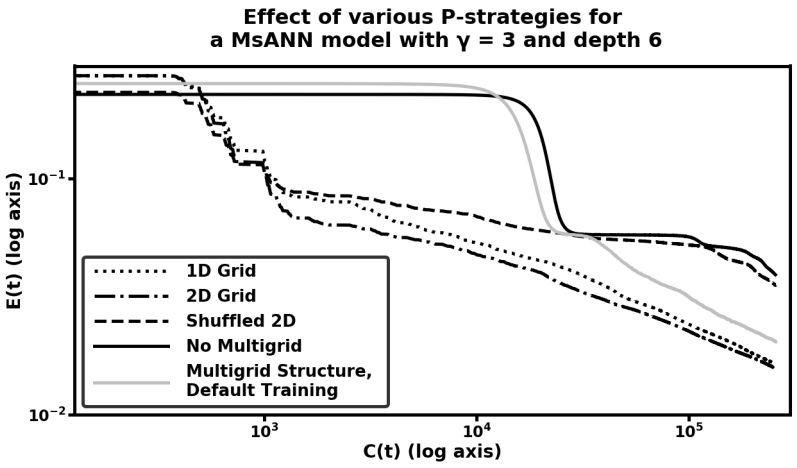

Figure 6.7: Comparison of several choices of Pro and Res operators for a Multiscale Neural Network training experiment, on MNIST data. Two choices for P which are local optima of prolongation problems demonstrate more efficient training than default, while two strategies perform worse: multigrid training with random P matrices, and training all varibles in the hierarchy simultaneously.

than training without multigrid and have final error lower than the default model. In many of our test cases, the hierarchical models reached the same level of MSE as the default in more than an order of magnitude fewer training examples, and continued to improve, surpassing the final level of error reached by the default network. Even in the worst case, our hierarchical model structure performed on par with neural networks which did not incorporate our weight prolongation and restriction operators. We leave the question of finding optimal (L, γ, k) for future work - see Section 6.6 for further discussion. Finally, we note that the model(s) in the experiments presented in section 6.5.1 were essentially the same MsANN models (same set of L, γ, k and same set of P matrices), and showed similar performance gains on two different machine vision problems, indicating that it may be possible to develop general MsANN model-creation procedures that are applicable to a variety of problems (rather than needing to be hand-tuned).

6.6 Conclusion and Future Work

We have introduced a novel method for multiscale modeling, which relies on a novel prolongation and restriction operator to move between models in a hierarchy. These prolongation

and restriction operators are the optima of an objective function we introduce which is a natural distance metric on graphs and graph lineages. We prove several important properties of this objective function, including an upper bound which allows us to decouple a difficult optimization into two smaller optimization problems under certain circumstances.

Additionally, we demonstrate an algorithm which makes use of such P and R operators to simultaneously train models in a hierarchy of neural networks (specifically, autoencoder neural networks). This Multiscale Artificial Neural Network (MsANN) approach statistically outperforms training only at the finest scale, achieving lower error than the default model and also reaching the default model's best performance in an order of magnitude fewer training examples. While in our experiments we saw uniform improvement as the parameters γ, k, and L were increased (meaning that the hierarchy is deeper, and the model spends more relative time training at the coarser scales), this may not always be the case, and we leave the question of finding optimal settings of these parameters for future work.

Promising directions for future work also include investigating the properties of the distance metric on graphs, and the use of those properties in graph lineage, as well as modifying the MsANN algorithm to perform the same type of hierarchical learning on more complicated ANN models, such as Convolutional Neural Networks (CNNs), as well as non-autoencoding tasks, for example classification.

Chapter 7

Application - Graph Prolongation Convolutional Networks

Building off of the multiscale neural network training ideas introduced in the previous chapter, we here apply a similar idea to train a Graph Convolutional Network, a model which excels at handling unstructured data. The problem we apply our model to is the prediction of energetic behavior of a rigid protein structure called a *microtubule*.

7.1 Convolution and Graph Convolution

Recent successes of deep learning have demonstrated that the inductive bias of Convolutional Neural Networks (CNNs) makes them extremely efficient for analyzing data with an inherent grid structure, such as images or video. In particular, many applications use these models to make per-node (per-pixel) predictions over grid graphs: examples include image segmentation, optical flow prediction, anticipating motion of objects in a scene, and facial detection/identification. Further work applies these methods to emulate physical models,

by discretizing the input domain. Computational Fluid Dynamics and other scientific tasks featuring PDEs or ODEs on a domain discretized by a rectangular lattice have seen recent breakthroughs applying machine learning models, like CNNs to handle data which is structured this way. These models learn a set of local filters whose size is much smaller than the size of the domain - these filters may then be applied simultaneously across the entire domain, leveraging the fact that at a given scale the local behavior of the neighborhood around a pixel (voxel) is likely to be similar at all grid points.

Graph Convolutional Networks (GCNs) are a natural extension of the above idea of image 'filters' to arbitrary graphs rather than nD grids, which may be more suitable in some scientific contexts. Intuitively, GCNs replace the image filtering operation of CNNs with repeated passes of: 1) aggregation of information between nodes according to some structure matrix 2) nonlinear processing of data at each node according to some rule (most commonly a flat neural network which takes as separate input(s) the current vector at each node). We refer the reader to a recent survey by Bacciu et al [6] for a more complete exploration of the taxonomy graph neural networks.

7.2 Microtubules

As an example of a dataset whose underlying graph is not a grid, we consider a coarse-grained simulation of a microtubule. Microtubules (MTs) are self-assembling nanostructures, ubiquitous in living cells, that along with actin filaments comprise a major portion of the dynamic cytoskeleton governing cell shape and mechanics. Whole-MT biomechanical models would be a useful tool for modeling cytoskeletal dynamics at the cellular scale. MTs play important structural roles during cell division, cell growth, and separation of chromosomes (in eukaryotic cells) [20]. Microtubules are comprised of a lattice structure of two conformations (α and β) of tubulin. Free-floating tubulin monomers associate energetically into dimer subunits,

which then associate head-to-tail to form long chain-like complexes called *protofilaments*. Protofilaments associate side-to side in a sheet; at some critical number of protofilaments (which varies between species and cell type) the sheet wraps closed to form a repeating helical lattice with a seam. See [78], Page 303, Figure 1. Key properties of microtubules are:

Dynamic instability: microtubules grow from one end by attracting free-floating tubulin monomers [110]. Microtubules can spontaneously enter a "catastrophe" phase, in which they rapidly unravel, but can also "rescue" themselves from the catastrophe state and resume growth [39, 97].

Interactions: Microtubules interact with one another: they can dynamically avoid one another during the growth phase, or collide and bundle up, or collide and enter catastrophe [104]. The exact mechanism governing these interactions is an area of current research.

Structural strength: microtubules are very stiff, with a Young's Modulus estimated at $\approx$1GPa for some cases [78]. This stiffness is thought to play a role in reinforcing cell walls [63].

In this work we introduce a model which learns to reproduce the dynamics of a graph signal (defined as an association of each node in the network with a vector of discrete or real-valued labels) at multiple scales of graph resolution. We apply this model framework to predict the potential energy of each tubulin monomer in a simplified mechanochemical simulation of a microtubule. This trial dataset illustrates the efficiency of our proposed type of graph convolutional network and is a solid proof-of-concept for applying this model to more biologically accurate microtubule models in the future. In the next section, we discuss the wide variety of microtubule simulations which have been previously studied.

7.2.1 Simulation of MTs and Prior Work

Non-continuum, non-event-based simulation of large molecules is typically done by representing some molecular subunit as a particle/rigid body, and then defining rules for how these subunits interact energetically. Molecular Dynamics (MD) simulation is an expansive area of study and a detailed overview is beyond the scope of this paper. Instead, we describe in general terms some basic ideas relevant to the numerical simulation detailed in Section 7.4.1. Simulation of microtubules is an area of active research, and there are many fundamental questions yet to be answered. A brief review of previous MT simulation studies [99, 38, 72, 110, 113, 70] finds a wide variety of different simulation techniques and assumptions. For this reason, we choose a simple model which is in a qualitative sense the "lowest common denominator" of many of these models. Our microtubule simulation is a fixed structure of tubulin with energy terms defined only for tubulin-tubulin associations (consisting of angle and edge length constraints between monomers). We simulated the behavior of this structure under bending load in the MD software package LAMMPS [82] using Verlet integration [112] and an implicit surrounding solvent [90]. For more details of our simulation, see Section 7.4.1 and the source code, available in the Supplementary Material accompanying this paper. Each timestep of our simulator produces a vector consisting of each monomer's contribution to the total potential energy of the structure at that timestep, as detailed in Section 7.4.1. This vector is the target output we want our machine learning model to predict. In this work, we apply graph convolutional networks, trained via a method we introduce, to predict these energy values for a section of microtubule.

7.3 Model Architecture and Mathematical Details

7.3.1 Model Description

Many approaches to scientific problems benefit from the use of *multiscale* analysis: separating the behavior at hand into multiple scale lengths and analyzing each separately. We expect in general to have different phenomena at different scales, therefore necessitating varying treatments; a typical example would be a hybrid computational mechanics solver which uses both a continuum model at the largest spatial scale, but models spatially smaller interactions with an all-atom simulation [100, 115]. Even when phenomena are the same across multiple spatial scales (i.e. solving the Navier-Stokes equations on irregular domains [85]) we expect to see acceleration of simulations when we use a multiscale architecture, as in the case of Multigrid solvers for iterative systems. These methods work on the premise that it if the wavelength of an error is large in comparison to the scale length considered by a solver, it may take many iterative steps at that scale to resolve the error. It is therefore advantageous to resolve errors at a scale similar to their characteristic wavelength, which is accomplished by building a hierarchy of solvers which each address error at a particular scale length. The exact method for reduction in error (a "smoothing" step) is problem dependent; however, strategies for stepping between spatial scales have been invented, with good theoretical guarantees for accelerated error reduction of the entire system.

It is here necessary to note that the scheduling dictates which scale of error is reduced at a given step in the algorithm. In multigrid methods, the actual fine-to-coarse mapping (or vice versa) is given by multiplying the current solution by either a restriction or prolongation matrix, respectively. Typically these matrices are constrained, for example to be norm-preserving. This is similar in both motivation and practice to the matrix multiplication we use in our model architecture, detailed below and in Section 7.3.4.

Multiscale architectures are also a staple of machine learning methods. Convolutional Neural Networks, as described in section 7.1, are an example of such a system: features are propagated through the network so that the nodes in the final layer are aggregating information from a wide visual area. Motivated by both CNNs and the multiscale method literature, we develop a model which uses a multiscale architecture to learn molecular dynamics at multiple spatial scales. Input is coarsened to each of these scales by applying an optimized linear projection (for details of this optimization, see Section 7.4.2). At each scale, a graph convolutional network processes that scale's information, analogous to the lateral connections in U-Net [86]. Again analogously to the "upscaling" connection in U-Net, the output of these GCNs is upsampled using the inverse of the same optimized linear projection used in the prior downsampling step. These outputs are all summed to produce a final model prediction at the finest scale. In the rest of this section, we first provide some general mathematical background (Section 7.3.2), formally define Graph Convolution (Section 7.3.3), and finally use these definitions to formally specify our model architecture in (Section 7.3.4)

7.3.2 Mathematical Background

Definitions: For all basic terms (graph, edge, vertex, degree) we use standard definitions. We use the notation $\{x_i\}_{i=a}^{b}$ to represent the sequence of x_i indexed by the integers $a, a+1, a+2, \ldots b$. When X is a matrix, we will write $[X]_{ij}$ to denote the entry in the ith row, jth column.

Graph Laplacian: The graph Laplacian is the matrix given by $L(G) = A(G) - \text{diag}(A(G) \cdot \mathbf{1})$ where $A(G)$ is the adjacency matrix of G, and $\mathbf{1}$ is an appropriately sized vector of 1s. The graph Laplacian is given by some authors as the opposite sign.

Linear Graph Diffusion Distance (GDD): Given two graphs G_1 and G_2, with $|G_1| \leq$

$|G_2|$ the Linear Graph Diffusion Distance $D(G_1, G_2)$ is given by:

$$D(G_1, G_2) = \inf_{\substack{P|\mathcal{C}(P) \\ \alpha>0}} \left|\left| \frac{1}{\alpha} PL(G_1) - \alpha L(G_2)P \right|\right|_F \tag{7.1}$$

where $\mathcal{C}(P)$ represents some set of constraints on P, α is a scalar with $\alpha > 0$, and $||\cdot||_F$ represents the Frobenius norm. We take $\mathcal{C}(P)$ to be orthogonality: $P^T P = I$. Note that since in general P is a rectangular matrix, it may not be the case that $PP^T = I$. Unless stated otherwise all P matrices detailed in this work were calculated with $\alpha = 1$, using the procedure laid out in the following section, in which we briefly detail an algorithm for efficiently computing the distance in the case where α is allowed to vary. The efficiency of this algorithm is necessary to enable the computation of the LGDD between very large graphs, as discussed in Section 7.6.3.

Prolongation matrix: we use the term "prolongation matrix" to refer to a matrix which is the optimum of the minimization given in the definition of the LGDD.

7.3.3 Graph Convolutional Layer Definition

We follow the GCN formulation given by Kipf and Welling [62]. Assuming an input tensor X of dimensions $n \times F$ (where n is the number of nodes in the graph and F is the dimension of the label at each node), we inductively define the layerwise update rules for a graph convolutional network $\text{GCN}\left(Z_i, X, \left\{\theta_l^{(i)}\right\}_{l=1}^{m}\right)$ as:

$$\begin{aligned} X_0 &= X \\ X_m &= g_m\left(Z_i X_{m-1} W_m^{(i)} + b_m^{(i)}\right), \end{aligned}$$

where g_m is the activation function of the mth layer.

7.3.4 Graph Prolongation Convolutional Networks

The model we propose is an ensemble of GCNs at multiple scales, with optimized projection matrices performing the mapping in between scales (i.e. between ensemble members). More formally, Let $\{G_i\}_{i=1}^k$ represent a sequence of graphs with $|G_1| \geq |G_2| \ldots \geq |G_k|$, and let $\{Z_i = z(G_i)\}_{i=1}^k$ be their structure matrices (for some chosen method z of calculating the structure matrix given the graph). In all experiments in this paper, we take $z(G) = L(G)$, the graph Laplacian, as previously defined [1]. In an ensemble of Graph Convolutional Networks, let $\theta_l^{(i)} = \left\{W_l^{(i)}, b_l^{(i)}\right\}$ represent the parameters (filter matrix and bias vector) in layer l of the ith network.

When $i = j - 1$, let $P_{i,j}$ be an optimal (in either the sense of Graph Diffusion Distance, or in the sense we detail in section 7.5.5) prolongation matrix from $L(G_j)$ to $L(G_i)$, i.e. $P_{i,j} = \arg\inf_{P|\mathcal{C}(P)} ||PL(G_j) - L(G_i)P||_F$. Then, for $i < j - 1$, let $P_{i,j}$ be shorthand for the matrix product $P_{i,i+1}P_{i+1,i+2}\ldots P_{j-1,j}$. For example, $P_{1,4} = P_{1,2}P_{2,3}P_{3,4}$.

Our multiscale ensemble model is then constructed as:

$$\begin{aligned}
&\textbf{GPCN}\left(\{Z_i\}_{i=1}^k, X, \left\{\left\{\theta_l^{(i)}\right\}_{l=1}^{m_i}\right\}_{i=1}^k, \{P_{i,i+1}\}_{i=1}^{k-1}\right) \\
&\quad = \text{GCN}\left(Z_1, X, \left\{\theta_l^{(1)}\right\}_{l=1}^{m_1}\right) \\
&\qquad + \sum_{i=2}^{k} P_{1i}\text{GCN}\left(Z_i, P_{1i}^T X, \left\{\theta_l^{(i)}\right\}_{l=1}^{m_i}\right)
\end{aligned} \tag{7.2}$$

This model architecture is illustrated in Figure 7.1. When the P matrices are constant/fixed, we will refer to this model as a GPCN, for Graph Prolongation-Convolutional Network. However, we find in our experiments in Section 7.5.5 that validation error is further reduced

[1] Other GCN research uses powers of the Laplacian, the normalized Laplacian, the symmetric normalized Laplacian, etc. Comparison of these structure matrices is beyond the scope of this paper.

when the P operators are tuned during the same gradient update step which updates the filter weights, which we refer to as an "adaptive" GPCN or A-GPCN. We explain our method for choosing Z_i and optimizing P matrices in Section 7.5.5.

7.4 Dataset Generation and Reduced Model Construction

In this section we describe some of the ancillary numerical results needed to reproduce and understand our main machine learning results in Section 7.5.

7.4.1 Dataset

In this Section we detail the process for generating the simulated microtubule data for comparison of our model with other GCN ensemble models. Our microtubule structure has 13 protofilaments (each 48 tubulin monomers long). As in a biological microtubule, each tubulin monomer is offset (along the axis parallel to the protofilaments) from its neighbors in adjacent protofilaments, resulting in a helical structrure with a pitch of 3 tubulin units. We refer to this pitch as the "offset" in Section 7.4.3. Each monomer subunit (624 total) is represented as a point mass of 50 Dalton (8.30×10^{-15} ng). The diameter of the whole structure is 26 nm, and the length is ≈ 260 nm. The model itself was constructed using Moltemplate [54], a tool for constructing large regular molecules to be used in LAMMPS simulations. Our Moltemplate structure files were organized hierarchically, with: tubulin monomers arranged into α-β dimer pairs; which were then arranged into rings of thirteen dimers; which were then stacked to create a molecule 48 dimers long. Note that this organization has no effect on the final LAMMPS simulation: we report it here for reproducibility, as well as providing

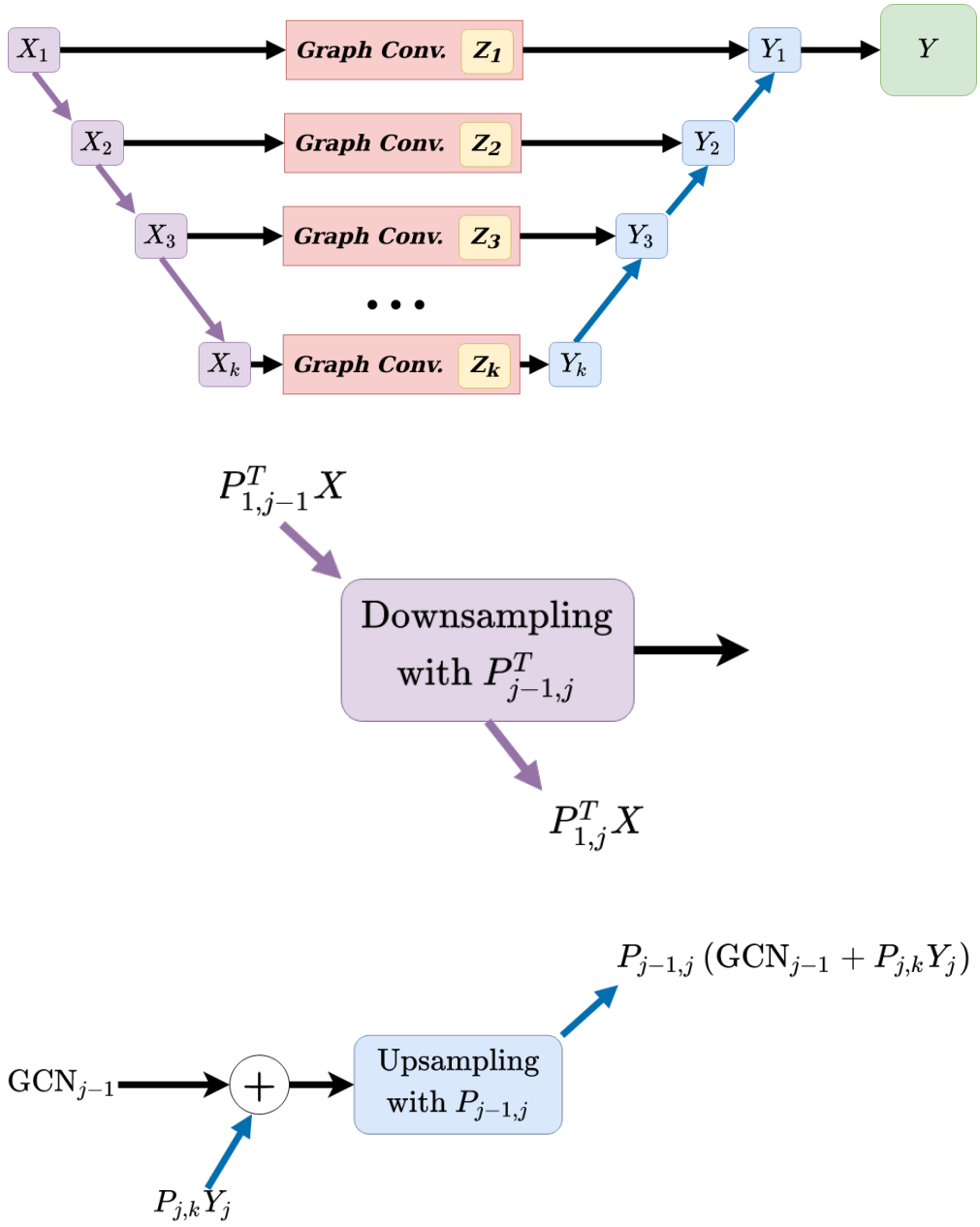

Figure 7.1: Top: Schematic of GPCN model. Data matrix X is fed into the model and repeatedly coarsened using optimized projection matrices P_{ik}, illustrated by purple arrows. These coarsened data matrices are separately fed into GCN models, producing predictions at each scale. Each blue arrow represents an upsample-and-add operation, where the upsampling is performed with the transpose of the P_{ik}. The final output of the ensemble is the projected sum of the outputs of each component GCN.Middle and bottom: mathematical details of upsampling and downsampling steps from top diagram. See Equation 7.2 for details.

the template files in the supplementary material accompanying this paper.

For this model, we define energetic interactions for angles and associations only. No steric or dihedral interactions were used: for dihedrals, this was because the lattice structure of the tube meant any set of four molecules contributed to multiple, contradictory dihedral interactions [2]. Interaction energy of an association b was calculated using the "harmonic" bond style in LAMMPS, i.e. $E(b) = L_{\text{type}(b)}(\text{length}(b) - b_0)^2$, where b_0 is the resting length and L is the strength of that interaction (L varies according to bond type). The energy of an angle ϕ was similarly calculated using the "harmonic" angle style, i.e. $E(\phi) = L_{\text{type}(\phi)}(\phi - \phi_0)^2$, where ϕ_0 is the resting angle and k is again the interaction strength, and L again depends on the angle type of ϕ[3]. The resting lengths and angles for all energetic interactions were calculated using the resting geometry of our microtubule graph G_{mt}: a LAMMPS script was used to print the value of every angle interaction in the model, and these were collected and grouped based on value (all 153° angles, all 102° angles, etc). Each strength parameter was varied over the values in $\{3.0, 9.0, 18.0, 30.0, 39.0, 48.0, 57.0\}$, producing 7^5 parameter combinations. Langevin dynamics were used, but with small temperature, to ensure stability and emphasize mechanical interactions. See Table 7.1 and Figure 7.3 for details on each strength parameter. See Figure 7.4 for an illustration of varying resting positions and final energies as a result of varying these interaction parameters.

GNU Parallel [103] was used to run a simulation for each combination of interaction parameters, using the particle dynamics simulation engine LAMMPS. In each simulation, we clamp the first two rings of tubulin monomers (nodes 1-26) in place, and apply force (in the negative y direction) to the final two rings of monomers (nodes 599-624). This force starts at 0 and ramps up during the first 128000 timesteps (one step = 0.018 ns) to its max-

[2]Association and angle constraints were sufficient to replicate the bending resistance behavior of microtubules. We hope to run a similar experiment using higher-order particle interactions (which may be more biologically plausible), in future work.

[3]the LAMMPS manual uses the character K to represent the interaction coefficient; we have used L to distinguish it from the spring constant k, for which we have $L = \frac{k}{2}$.

imum value of 9×10^{-14} N. Once maximum force is reached, the simulation runs for 256000 additional timesteps, which in our experience was long enough for all particles to come to rest. See Figure 7.2 for an illustration (visualized with Ovito [101]) of the potential energy per-particle at the final frame of a typical simulation run. Every $K = 32000$ timesteps, we save the following for every particle: the position x, y, z; components of velocity v_x, v_y, v_z; components of force F_x, F_y, F_z; and the potential energy of the particle E. The dataset is then a concatenation of the 12 saved frames from every simulation run, comprising all combinations of input parameter values, where for each frame we have:
x_i, the input graph signal, a 624×10 matrix holding the position and velocity of each particle, as well as values of the four interaction coefficients; and
y_i, the output graph signal, a 624×1 matrix holding the potential energy calculated for each particle.

We note here that none of the inputs to the model encode information about any of the statistics of the system as a whole (for example, the total energy, the temperature or density of the surrounding solvent, etc). This was not necessary in our example simulations because these factors did not vary in our experiment. A more detailed data input would likely be necessary for our model to be implemented in a more complicated simulation scenario that tuned any of these system quantities between runs.

During training, after a training/validation split, we normalize the data by taking the mean and standard deviation of the $N_{\text{train}} \times 624 \times 10$ input and $N_{\text{train}} \times 624 \times 1$ output tensors along their first axis. Each data tensor is then reduced by the mean and divided by the standard deviation so that all 624×10 inputs to the network have zero mean and unit standard deviation. We normalize using the training data only.

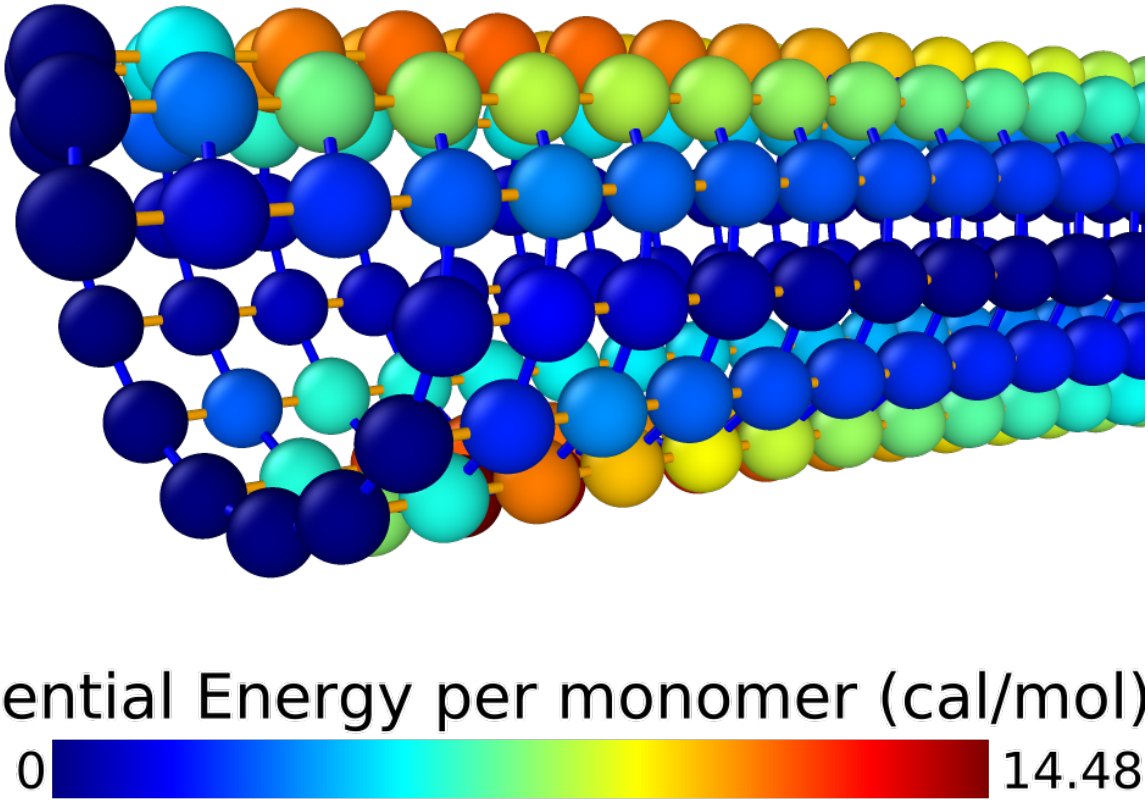

Figure 7.2: Microtubule model under bending load. Color of each particle indicates the sum of that particle's share of all of the energetic interactions in which it participates. This view is on the clamed end; the other end, out of view, has a constant force applied. The flexural rigidity (EI) we measure from the stiffest MTs we simulate is within the (broad) range of values found by prior work for taxol-stabilized MTs (both simulated and measured; see [60, 102, 110]).

7.4.2 Efficient Calculation of Graph Diffusion Distance

The joint optimization given in the definition of Linear Graph Diffusion Distance (Equation 7.1) is a nested optimization problem. If we set

$$\begin{aligned} f(\alpha) &= D(G_1, G_2|\alpha) \\ &= \inf_{P|\mathcal{C}(P)} \left|\left| \frac{1}{\alpha} PL(G_1) - \alpha L(G_2)P \right|\right|_F, \end{aligned}$$

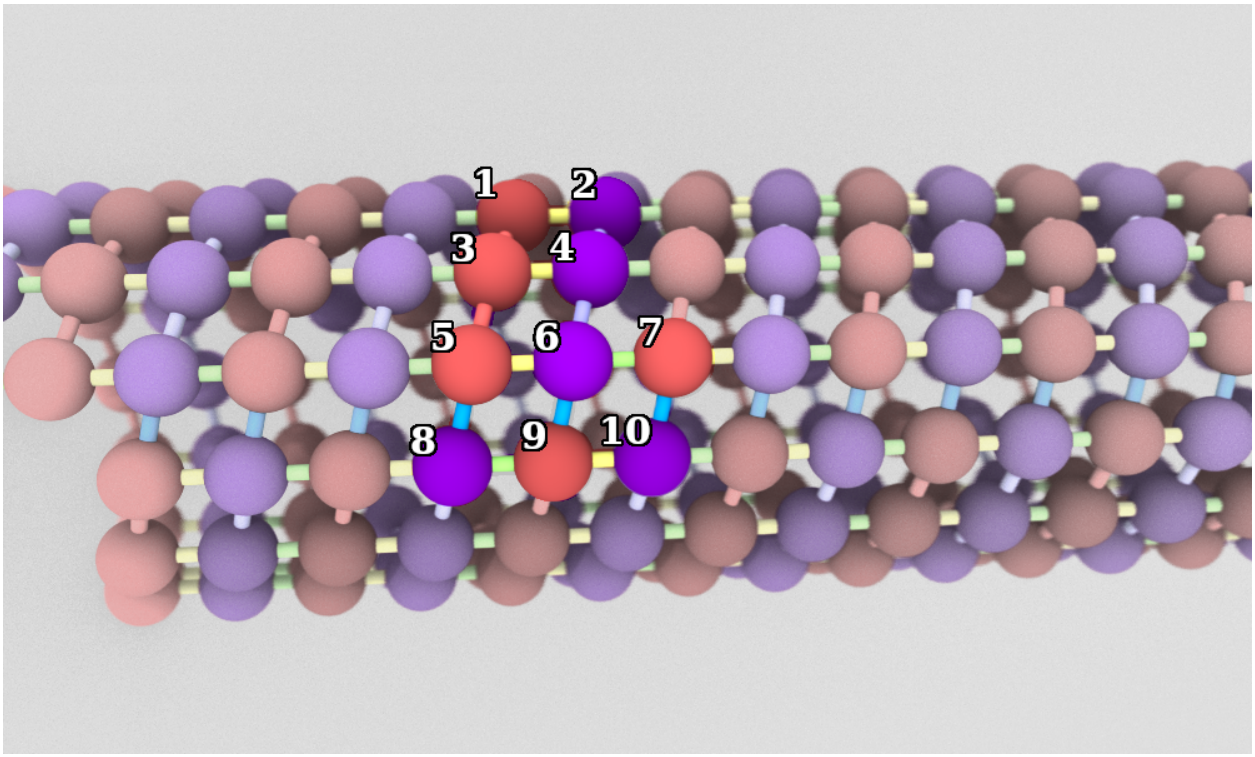

Figure 7.3: Microtubule model structure. Red spheres represent α-tubulin; purple spheres represent β-tubulin. Highlighted atoms at center are labelled to show example energetic interactions: each type of interaction indicated in Table 7.1 (using the particle labels in this image) is applied everywhere in the model where that arrangement of particle and association types occurs in that position.

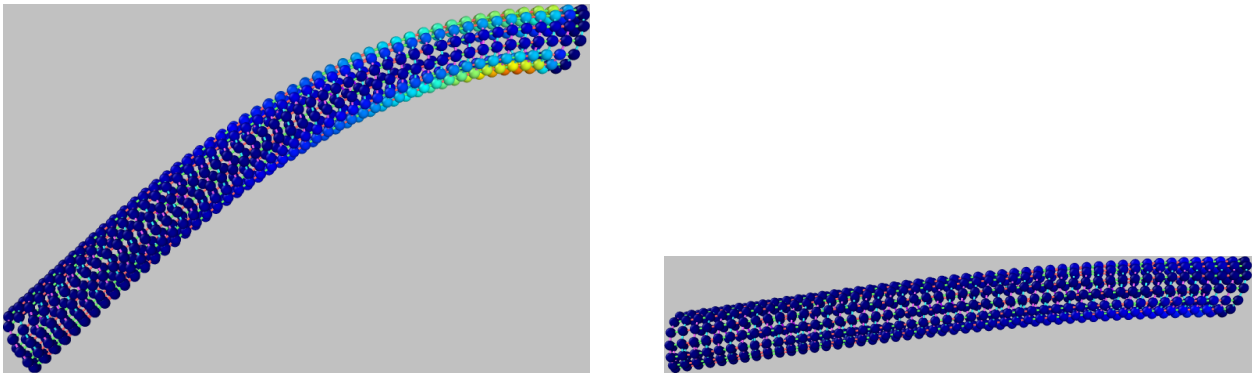

Figure 7.4: Changes in stiffness of microtubule model under constant load, as parameters controlling interaction strength are varied. We see qualitiative differences in behavior as spring constants are adjusted between 0.1 and 1.9. The left and right images show the final timestep of simulations where all spring constants were set to the minimum and maximum strength, respectively. Particles (tubulin monomers) are colored according to their contribution to total potential energy of the configuration, identically to Figure 7.2. All pictures show the microtubule at rest e.g. at the end of the simulation run using that parameter set.

Table 7.1: Description of energetic interactions in microtubule simulation, according to the labels in Figure 7.3.

Association interactions			
Description	Examples	Resting Length	Strength Param.
Lateral association inside lattice	(1,3),(2,4)	5.15639nm	LatAssoc
Lateral association across seam	(5,8),(6,9)	5.15639nm	LatAssoc
Longitudinal association	(1,2),(3,4)	5.0nm	LongAssoc
angle interactions			
Description	Examples	Resting Angle	Strength Param.
Pitch angle inside lattice	(1,3,5),(2,4,6)	153.023°	LatAngle
Longitudinal angle	(5,6,7),(8,9,10)	180°	LongAngle
Lattice cell acute angle	(3,4,6),(3,5,6),(5,8,9),(6,9,10)	77.0694°	QuadAngles
Lattice cell obtuse angle	(4,3,5),(4,6,5),(6,5,8),(6,9,8)	102.931°	QuadAngles

then each evaluation of f requires a full optimization of the matrix P subject to constraints $\mathcal{C}$. When $L(G_1)$ and $L(G_2)$ are Graph Laplacians, $f(\alpha)$ is continuous, but with discontinuous derivative, and has many local minima (see Figure 7.5). As a result, the naive approach of optimizing $f(\alpha)$ using a univariate optimization method like Golden Section Search is inefficient. In this section we briefly describe a procedure for performing this joint optimization more efficiently. For a discussion of variants of the LGDD, as well as the theoretical justification of this algorithm, see [94].

First, we note that by making the constraints on P more restrictive, we upper-bound the original distance:

$$\begin{aligned} D(G_1, G_2) &= \inf_{\substack{P|\mathcal{C}(P) \\ \alpha>0}} \left|\left| \frac{1}{\alpha} PL(G_1) - \alpha L(G_2)P \right|\right|_F \\ &\leq \inf_{\substack{P|\mathcal{S}(P) \\ \alpha>0}} \left|\left| \frac{1}{\alpha} PL(G_1) - \alpha L(G_2)P \right|\right|_F . \end{aligned} \tag{7.3}$$

In our case, $\mathcal{C}(P)$ represents orthogonality. As a restriction of our constraints we specify that P must be related to a *subpermutation* matrix (an orthogonal matrix having only 0 and

1 entries) $\tilde{P}$ as follows: $P = U_2 \tilde{P} U_1^T$, where the U_i are the fixed matrices which diagonalize $L(G_i)$: $L(G_i) = U_i \Lambda_i U_i^T$. Then,

$$\begin{aligned} D(G_1, G_2) &\leq \inf_{\substack{P|\mathcal{S}(P) \\ \alpha>0}} \left|\left| \frac{1}{\alpha} P L(G_1) - \alpha L(G_2) P \right|\right|_F \\ &= \inf_{\substack{\tilde{P}|\text{subperm}(\tilde{P}) \\ \alpha>0}} \Big|\Big| \frac{1}{\alpha} U_2 \tilde{P} U_1^T U_1 \Lambda_1 U_1^T \\ &\qquad\qquad - \alpha U_2 \Lambda_2 U_2^T U_2 \tilde{P} U_1^T \Big|\Big|_F \\ &= \inf_{\substack{\tilde{P}|\text{subperm}(\tilde{P}) \\ \alpha>0}} \left|\left| \frac{1}{\alpha} U_2 \tilde{P} \Lambda_1 U_1^T - \alpha U_2 \Lambda_2 \tilde{P} U_1^T \right|\right|_F \\ &= \inf_{\substack{\tilde{P}|\text{subperm}(\tilde{P}) \\ \alpha>0}} \left|\left| U_2 \left(\frac{1}{\alpha} \tilde{P} \Lambda_1 - \alpha \Lambda_2 \tilde{P} \right) U_1^T \right|\right|_F . \end{aligned}$$

Because the U_i are rotation matrices (under which the Frobenius norm is invariant), this further simplifies to

$$D(G_1, G_2) \leq \inf_{\substack{\tilde{P}|\text{subperm}(\tilde{P}) \\ \alpha>0}} \left|\left| \frac{1}{\alpha} \tilde{P} \Lambda_1 - \alpha \Lambda_2 \tilde{P} \right|\right|_F .$$

Furthermore, because the Λ_i are diagonal, this optimization is equivalent to a Rectangular Linear Assignment Problem (RLAP) [12], between the diagonal entries $\lambda_j^{(1)}$ and $\lambda_l^{(2)}$ of Λ_1 and Λ_2, respectively, with the α-dependent cost of an assignment given by:

$$c_\alpha(\lambda_j^{(1)}, \lambda_l^{(2)}) = \left(\frac{1}{\alpha} \lambda_j^{(1)} - \alpha \lambda_l^{(2)} \right)^2 . \tag{7.4}$$

.

RLAPs are extensively studied. We use the general LAP solving package lapsolver [50] to comute $\tilde{P}$. In practice (and indeed in this paper) we set often set $\alpha = 1$, in which case

the solution $\tilde{P}$ of the RLAP only acts as a preconditioner for the orthogonally-constrained optimization over P. More generally, when alpha is allowed to vary (and therefore many RLAPs must be solved), a further speedup is attained by re-using partial RLAP solutions from previously-tested values of α to find the optimal assignment at α'. We detail how this may be done in out recent work [94].

For the P matrices used in the experiments in this work, we set $\alpha = 1$ and used lapsolver to find an optimal assignment $\tilde{P}$. We then initialized an orthogonally-constrained optimization of 7.1 with $P = U_2 \tilde{P} U_1^T$. This constrained optimization was performed using Pymanopt [105].

7.4.3 Graph Coarsening

In this Section we outline a procedure for determining the coarsened structure matrices to use in the hierarchy of GCN models comprising a GPCN. We use our microtubule graph as an example. In this case, we have two a-priori guidelines for producing the reduced-order graphs: 1) the reduced models should still be a tube and 2) it makes sense from a biological point of view to coarsen by combining the α-β pairs into single subunits. Given these restrictions, we can explore the space of coarsened graphs and find the coarse graph which is nearest to our original graph (under the GDD).

Our microtubule model is a tube of length 48 units, 13 units per complete "turn", and with the seam offset by three units. We generalize this notion as follows: Let p be the offset, and k be the number of monomers in one turn of the tube, and n the number of turns of a tube graph $G_{\text{Tube}(n,k,p)}$. The graph used in our simulation is thus $G_{\text{mt}} = G_{\text{Tube}(48,13,3)}$. We pick the medium scale model G_{inter} to be $G_{\text{Tube}(24,13,1)}$, as this is the result of combining each α-β pair of tubulin monomer units in the fine scale, into one tubulin dimer unit in the medium scale. We pick the coarsest graph G_{coarse} by searching over possible offset tube

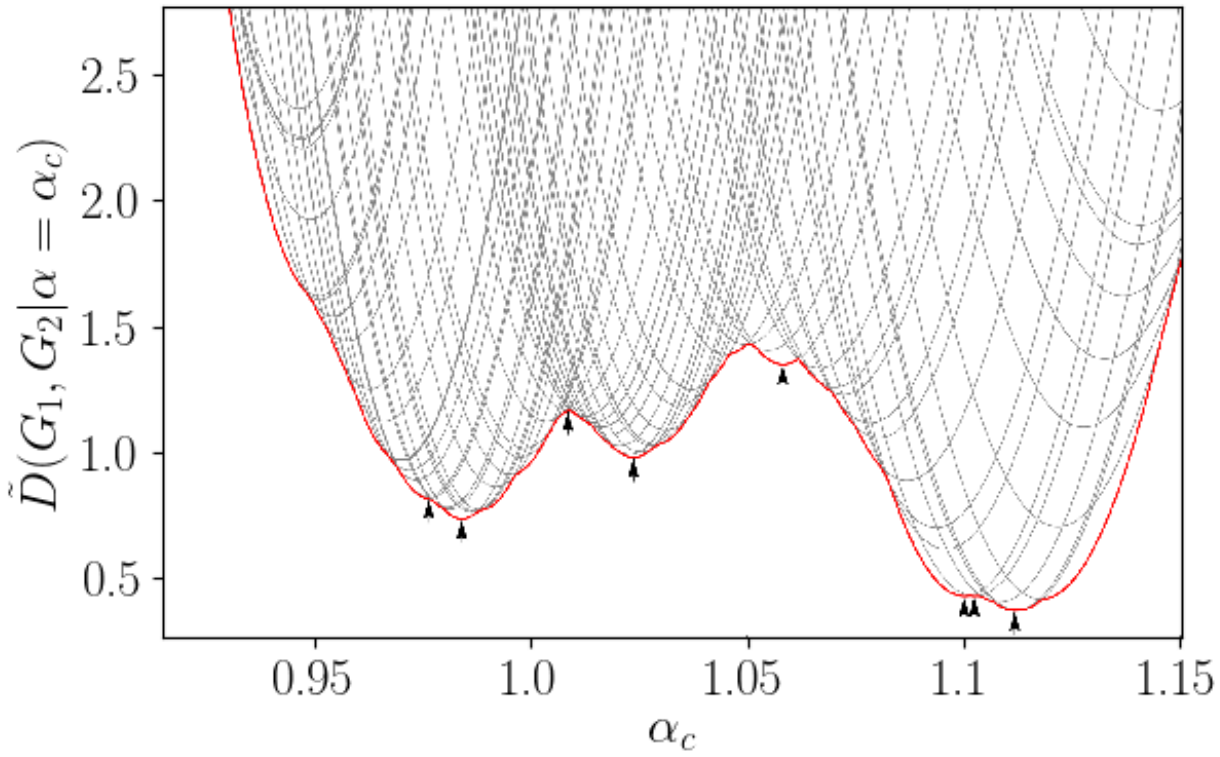

Figure 7.5: Plot of Linear Graph Diffusion Distance between two small random graphs, as α is varied. Each grey curve shows the objective function when P is fixed, as a function of α, and each curve represents a P matrix which is optimal at any value of α in the plotted range. The red curve shows the lower convex hull of all grey curves. Note that it is continuous but has discontinuous slope. Black arrows represent local optima. The discontinuous slope and high number of local optima illustrate why optimizing this function using univariate search over α is inefficient.

graphs. Namely, we vary $k \in \{3, 4, \ldots 12\}$ and $p \in \{0, 1, 2, 3\}$, and compute the optimal P^* and its associated distance $D(G_{\text{Tube(24,k,p)}}, G_{\text{mt}}|P = P^*)$. Figure 7.6 shows the distance between G_{mt} and various other tube graphs as parameters p and k are varied. The nearest $G_{\text{Tube(24,k,p)}}$ to G_{mt} is that with $p = 0$ and $k = 3$. Note that Figure 7.6 has two columns for each value of k: these represent the coarse edges along the seam having weight (relative to the other edges) 1 (marked with an S) or having weight 2 (no S). This is motivated by the fact that our initial condensing of each dimer pair condensed pairs of seam edges into single edges.

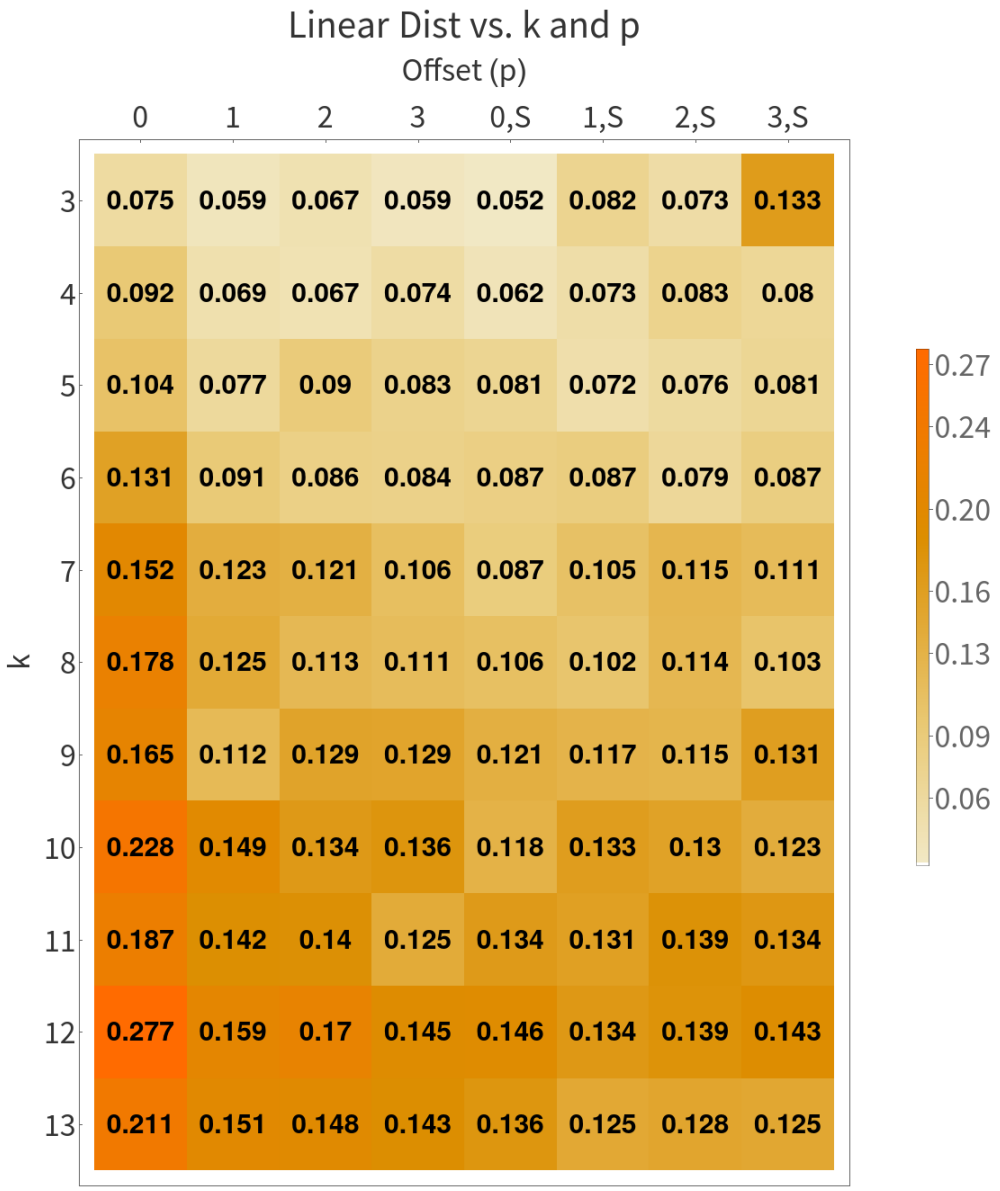

Figure 7.6: Directed Graph Diffusion Distance (GDD) between offset tube graphs and G_{mt}. Table cells colored by value. We see from this comparison that the two graphs which are closest to G_{mt} are $G_{\text{Tube}(24,3,0)}$ and $G_{\text{Tube}(24,3,0)}$ with an edge weight of 2 for connections along the seam, motivating our choice of $G_{\text{Tube}(24,3,0)}$ (unweighted) as the coarsest graph in our hierarchy.

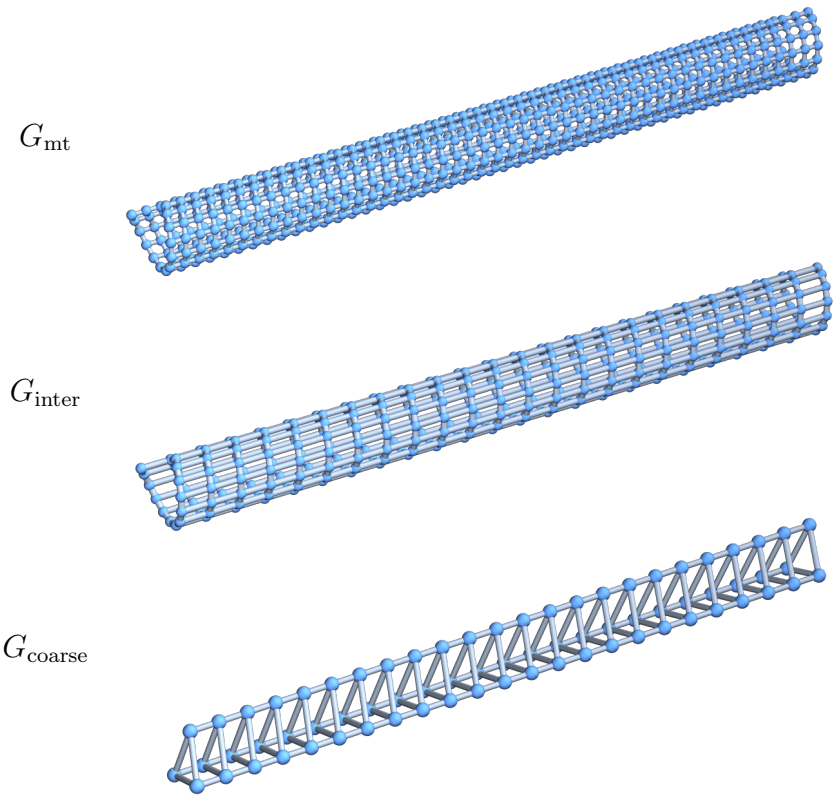

Figure 7.7: Three graphs used to create structure matrices for our GPCN model. Top: microtubule graph. Center: Offset tube with 13 subunits per turn, length 24, and offset 1. Bottom: Tube with 3 subunits per turn, no offset, and length 24.

7.5 Machine Learning Experiments

7.5.1 Experimental Procedure

This section contains several experiments comparing our model, and its variants, to other types of Graph Convolutional Networks. All models were trained using ADAM with default hyperparameters, in TensorFlow [1]. Random seeds for Python, TensorFlow, Numpy, and Scipy were all initialized to the same value for each training run, to ensure that the train/validation split is the same across all experiments, and the batches of drawn data are the same. See supplementary material for version numbers of all software packages used. Training batch size was set to 8, all GCN layers have ReLU activation, and all dense layers

have sigmoidal activation with the exception of the output layer of each network (which is linear). All modes were trained for 1000 epochs of 20 batches each. The time per batch of each model is listed in Table 7.4.

Since hardware implementations may differ, we estimate the computational cost in Floating Point OPerations (FLOPs) of each operation in our models. The cost of a graph convolutional layer with $n \times n$ structure matrix Z, $n \times F$ input data X, and $F \times C$ filter matrix W is estimated as: $nF(|Z|+C)$, where $|Z|$ is the number of nonzero entries of Z. This is calculated as the sum of the costs of the two matrix multiplications $X \cdot W$ and $Z \cdot XW$, with the latter assumed to be implemented as sparse matrix multiplication and therefore requiring $O(|Z|nF)$ operations. For implementation reasons, our GCN layers (across all models) do not use sparse multiplication; if support for arbitrary-dimensional sparse tensor outer products is included in TensorFlow in the future, we would expect the wall-clock times in Table 7.4 to decrease. The cost of a dense layer (with $n \times F$ input data X, and $F \times C$ filter matrix W) applied to every node separately is estimated as: $O(nFC)$. The cost of taking the dot product between a $n \times k$ matrix and a $k \times m$ matrix (for example, the restriction/prolongation by P) is estimated as $O(nmk)$.

For GPCN models, P matrices were calculated using Pymanopt [105] to optimize Equation 7.1 subject to orthogonality constraints. The same P were used to initialize the (variable) P matrices of A-GPCN models.

7.5.2 Evaluation of GPCN Variants

Our proposed model uses a hierarchy of graph convolutional networks to predict energy of a molecule at several spatial scales. The computational cost of a graph convolutional layer is approximately quadratic in the number of nodes in the underlying graph. We would therefore expect to see efficiency gains when some number of graph convolution layers are

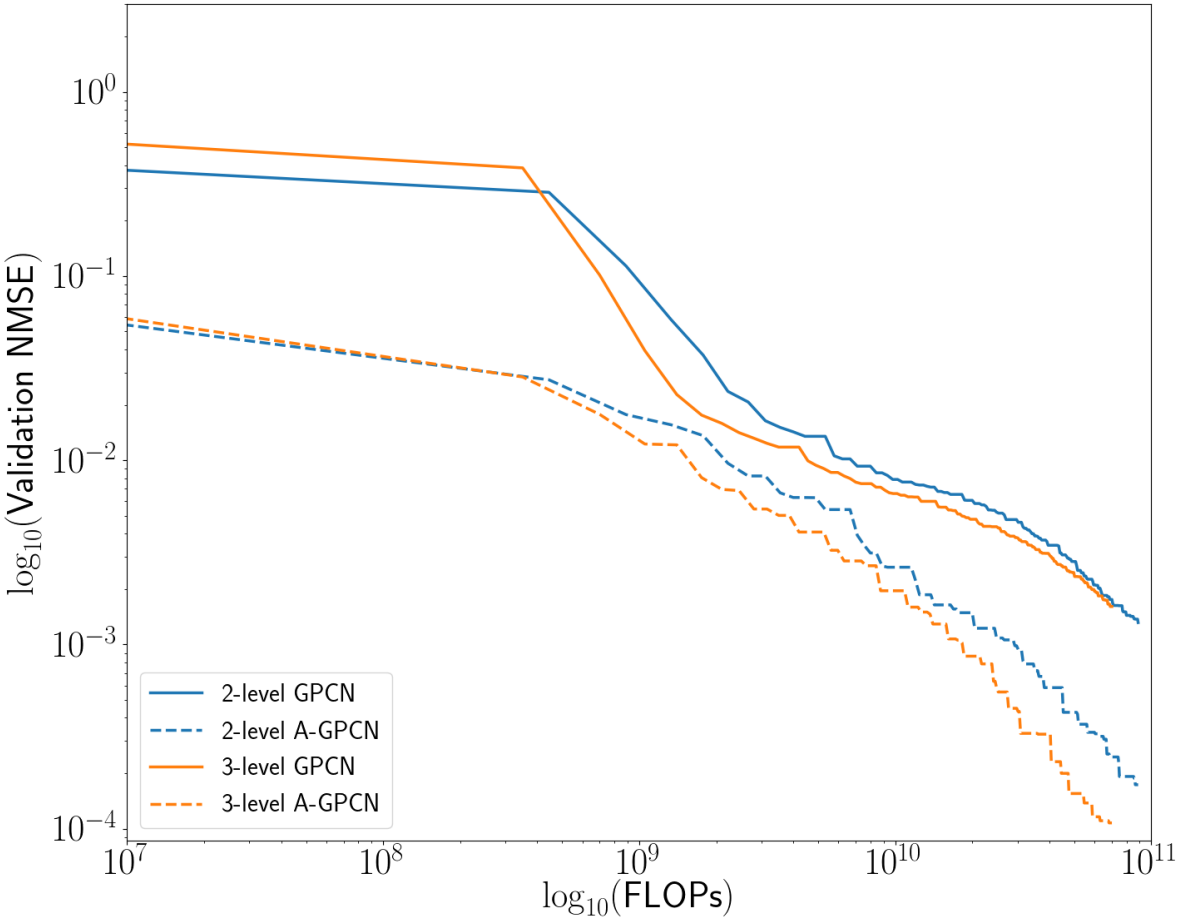

Figure 7.8: Comparison of mean squared error (MSE) on held-out validation data (normalized by averaging over the validation set) as a function of FLOPs expended, for variants of our model. We see that the adaptive and non-adaptive models occupy separate regimes (the adaptive models are superior), and within each the depth-3 model outperforms the depth-2 one.

operating on a reduced graph. In this subsection we present numerical experiments showing that this is indeed the case: the accuracy gained (per unit of computational expenditure) is higher for deeper hierarchies. Additionally, the adaptive model(s) universally outperform their non-adaptive counterparts.

We compare the following versions of our model:

- a two-level GPCN with static P-matrices;

- a three-level GPCN with static P-matrices;
- both of the above, but with P matrices allowed to vary during training (adjusted with the same backpropagation signals which are used to modify the convolution weights).

Figure 7.8 and Table 7.3 summarize these results.

7.5.3 Evaluation of Training Schedules

In contrast to the prior section, where we use the same training strategy and evaluate the efficiency of different variants of our model, in this section we fix the model architecture and evaluate the effect of different training schedules. Specifically, we compare the computational cost of training the entire GPCN at once, versus training the different 'resolutions' (meaning the different GCNs in the hierarchy) of the network according to a more complicated training schedule. This approach is motivated by recent work in coarse-to-fine training of both flat and convolutional neural networks [92, 118, 45, 29, 59], as well as the extensive literature on Algebraic MultiGrid (AMG) methods [111].

AMG solvers for differential equations on a mesh (which arises as the discretization of some volume to be simulated) proceed by performing numerical "smoothing steps" at multiple resolutions of discretization. The intuition behind this approach is that modes of error should be smooth at a spatial scale which is equivalent to their wavelength, i.e. the solver shouldn't spend many cycles resolving long-wavelength errors at the finest scale, since they can be resolved more efficiently at the coarse scale. Given a solver and a hierarchy of discretizations, the AMG literature defines several types of training procedures or "cycle" types (F-cycle, V-cycle, W-cycle). These cycles can be understood as being specified by a recursion parameter γ, which controls how many times the smoothing or training algorithm visits all of the coarser levels of the hierarchy in between smoothing steps at a given scale. For

example, when $\gamma = 1$ the algorithm proceeds from fine to coarse and back again, performing one smoothing step at each resolution - a 'V' cycle.

We investigate the efficiency of training 3-level GPCN and A-GPCN (as described in Section 7.5.2), using multigrid-like training schedules with $\gamma \in \{0, 1, 2, 3\}$, as well as "coarse-to-fine" training: training the coarse model to convergence, then training the coarse and intermediate models together (until convergence), then finally training all three models at once. Error was calculated at the fine-scale. For coarse-to-fine training convergence was defined to have occurred once 10 epochs had passed without improvement of the validation error.

Our experiments (see Figure 7.9) show that these training schedules do result in a slight increase in efficiency of the GPCN model, especially during the early phase of training. The increase is especially pronounced for the schedules with $\gamma = 2$ and $\gamma = 3$. Furthermore, these multigrid training schedules produce models which are more accurate than the GPCN and A-GPCN models trained in the default manner. As a final note, previous work [92] has shown that these types of multiscale neural network architectures, with this type of multigrid training schedule may also be more efficient in a "statistical" sense - that is, require much less data to find an equivalent or better local minimum of error. A third type of efficiency results from the fact that once trained, querying the machine learning model is faster than running an entire simulation. This means that the cost of generating the initial dataset and training the model is amortized over the time gained by using the machine learning model as an approximator. We would expect our model to also perform well under both of these latter measures of efficiency - one run of our fine-scale simulation took approximately 20 minutes, whereas querying the trained GPCN takes tenths of milliseconds. However, quantifying this possibility further is beyond the scope of this paper.

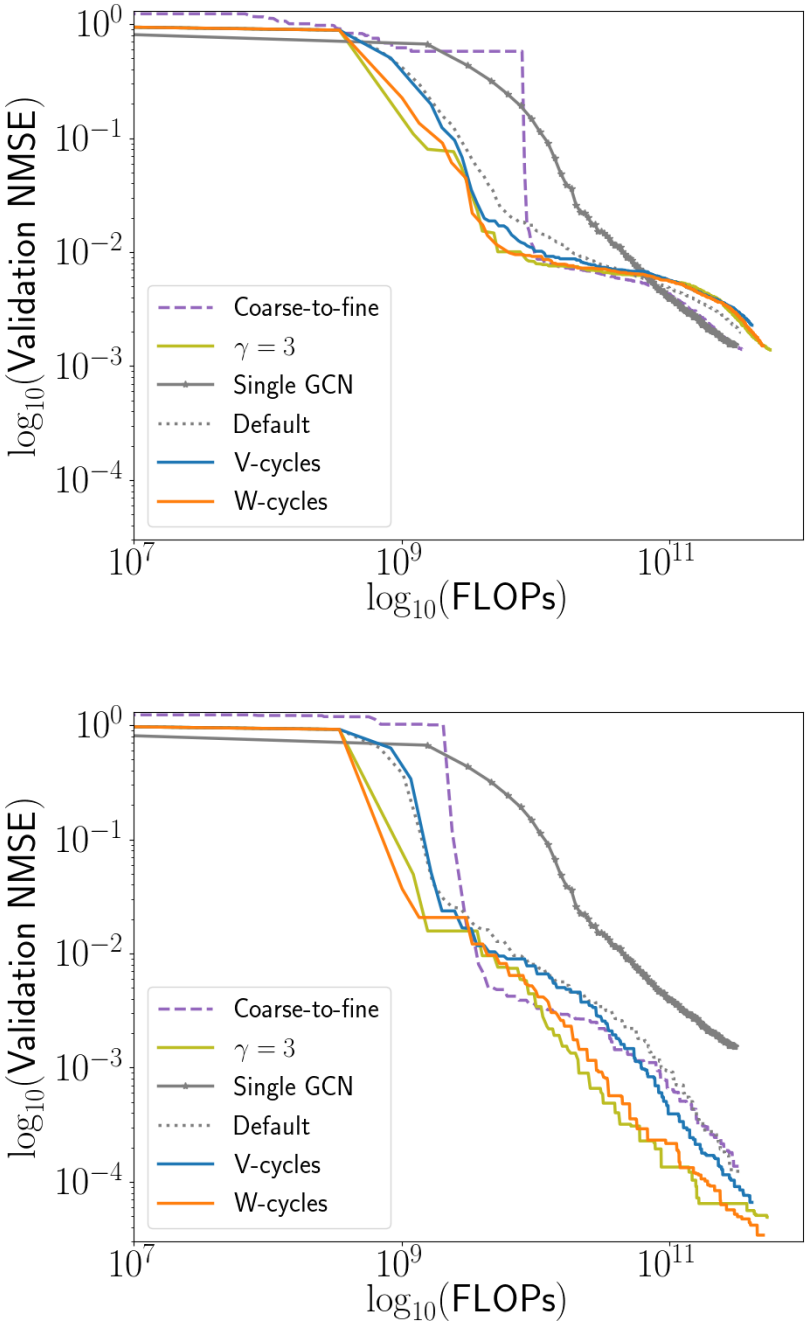

Figure 7.9: Effect of varying training schedule for training a GPCN model. Notably, The various multigrid training cycles result in models which are more accurate, and do so more efficiently. Top: FLOPs vs. NMSE for training GPCNs with multigrid training schedules. Bottom: same, but with A-GPCNs.

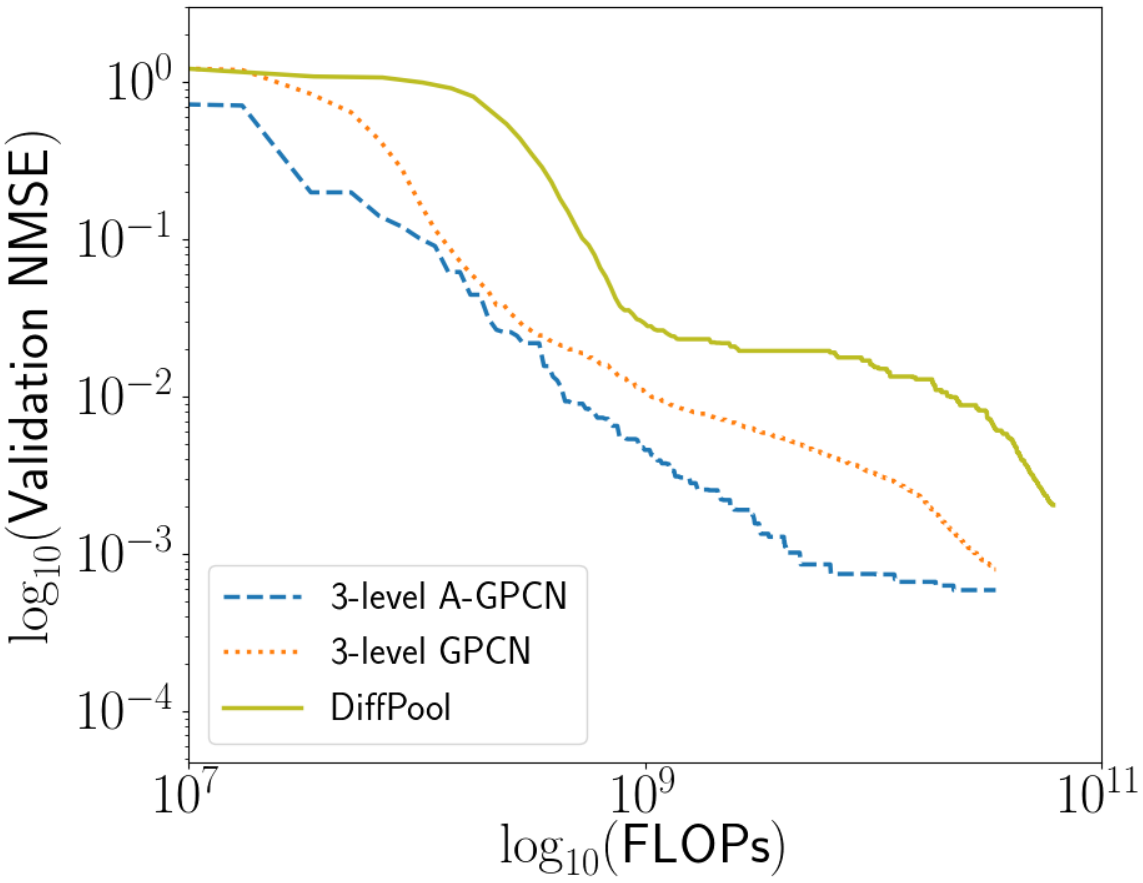

Figure 7.10: Comparison of 3-level GPCN and A-GPCN models to a 3-level GPCN which uses DiffPool modules to coarsen the input graph and data. Our models improve over DiffPool in terms of both efficiency and final error.

7.5.4 Comparison with DiffPool

Graph coarsening procedures are in general not differentiable. DiffPool [116] aims to address this by constructing an auxiliary GCN, whose output is a pooling matrix. Formally: Suppose that at layer l of a GCN we have a $n_l \times n_l$ structure matrix $Z^{(l)}$ and a $n \times F$ data matrix $X^{(l)}$. In addition to GCN layers as described in Section 7.3, Ying et. al define a pooling

operation at layer l as:

$$S^{(l)} = \sigma\left(\text{GCN}_{\text{pool}}\left(Z^{(l)}, X^{(l)}, \left\{\theta_1^{(i)}\right\}_{l=1}^{m}\right)\right)$$

where GCN_{pool} is an auxillary GCN with its own set of parameters $\left\{\theta_1^{(i)}\right\}_{l=1}^{m}$, and σ is the softmax function. The output of GCN_{pool} is a $n \times n_{\text{coarse}}$ matrix, each row of which is softmaxed to produce an affinity matrix S whose rows each sum to 1, representing each fine-scale node being connected to one unit's worth of coarse-scale nodes. The coarsened structural and data matrices for the next layer are then calculated as:

$$\begin{aligned} X^{(l+1)} &= {S^{(l)}}^T X^{(l)} \\ Z^{(l+1)} &= {S^{(l)}}^T Z^{(l)} S^{(l)} \end{aligned} \tag{7.5}$$

Clearly, the additional GCN layers required to produce $S^{(l)}$ incur additional computational cost. We compare our 3-level GPCN (adaptive and not) models from the experiment in Section 7.5.5 to a model which has the same structure, but in which each P matrix is replaced by the appropriately-sized output of a DiffPool module, and furthermore the coarsened structure matrices are produced as in Equation 7.5.

We see that our GPCN model achieves comparable validation loss with less computational work, and our A-GPCN model additionally achieves lower absolute validation loss.

7.5.5 Comparison to Other GCN Ensemble Models

In this experiment we demonstrate the efficiency advantages of our model by comparing our approach to other ensemble Graph Convolutional Networks. Within each ensemble, ours and others, each GCN model consists of several graph convolution layers, followed by several dense layers which are applied to each node separately (node-wise dense layers can be

Table 7.2: Filter specifications for ensemble models in comparison experiment.

Structure Matrix	GCN Filters	Dense Filters
Single GCN		
L_{mt}	64,64,64	256, 32, 8, 1
2-GCN Ensemble		
L_{mt}	64,64,64	256, 32, 8, 1
L_{mt}	32,32,32	256, 32, 8, 1
3-GCN Ensemble		
L_{mt}	64,64,64	256, 32, 8, 1
L_{mt}	32,32,32	256, 32, 8, 1
L_{mt}	16,16,16	256, 32, 8, 1
2-level GPCN		
L_{inter}	64,64,64	256, 32, 8, 1
L_{mt}	32,32,32	256, 32, 8, 1
3-level GPCN		
L_{coarse}	64,64,64	256, 32, 8, 1
L_{inter}	32,32,32	256, 32, 8, 1
L_{mt}	16,16,16	256, 32, 8, 1
N-GCN (radii 1,2,4)		
L_{mt}^r	64,64,64	256, 32, 8, 1
N-GCN (radii 1,2,4,8,16)		
L_{mt}^r	64,64,64	256, 32, 8, 1

alternatively understood as a GCN layer with $Z = I$, although we implement it differently for efficiency reasons). The input to the dense layers is the node-wise concatenation of the output of each GCN layer. Each ensemble is the sum output of several such GCNs. We compare our models to 1, 2, and 3- member GCN ensembles with the same number of filters (but all using the original fine-scale structure matrix).

We also compare our model to the work of Abu-El-Haija et. al [2], who introduce the N-GCN model: an ensemble GCN in which each ensemble member uses a different power Z^r of the structure matrix (to aggregate information from neighborhoods of radius r). We include a N-GCN with radii (1,2,4) and a N-GCN with radii (1,2,4,8,16).

Table 7.3: Mean error and uncertainty of several GCN ensemble models across ten random trials. For each trial, the random seed was set to the same value for each model. Reported values are the minimum error on the validation set during training (not the error at the final epoch). Normalized Mean Squared Error (NMSE) values are unitless. Only one trial was performed with the DiffPool model.

Model Name	Mean NMSE ± Std. Dev ($\times 10^{-3}$)	Min NMSE ($\times 10^{-3}$)
Single GCN	1.55 ± 0.10	1.45914
Ensemble - 2 GCNs	1.44 ± 0.07	1.38313
Ensemble - 3 GCNs	1.71 ± 0.20	1.43059
2-level GPCN	1.43 ± 0.12	1.24838
2-level A-GPCN	0.17 ± 0.05	0.08963
3-level GPCN	2.09 ± 0.32	1.57199
3-level A-GPCN	0.131 ± 0.030	0.10148
N-GCN radii (1,2,4)	1.30 ± 0.05	1.23875
N-GCN radii (1,2,4,8,16)	1.30 ± 0.06	1.22023
DiffPool	2.041 ± n/a	2.041

Table 7.4: Mean wall-clock time to perform feed-forward and backpropagation for one batch of data, for various GCN ensemble models. Times were collected on a single Intel(R) Xeon(R) CPU core and an NVIDIA TITAN X GPU.

Model Name	Mean time per batch (s)
Single GCN	0.042
Ensemble - 2 GCNs	0.047
Ensemble - 3 GCNs	0.056
2-level GPCN	0.056
2-level A-GPCN	0.056
3-level GPCN	0.061
3-level A-GPCN	0.059
N-GCN, radii (1,2,4)	0.067
N-GCN, radii (1,2,4,8,16)	0.086
DiffPool	0.0934

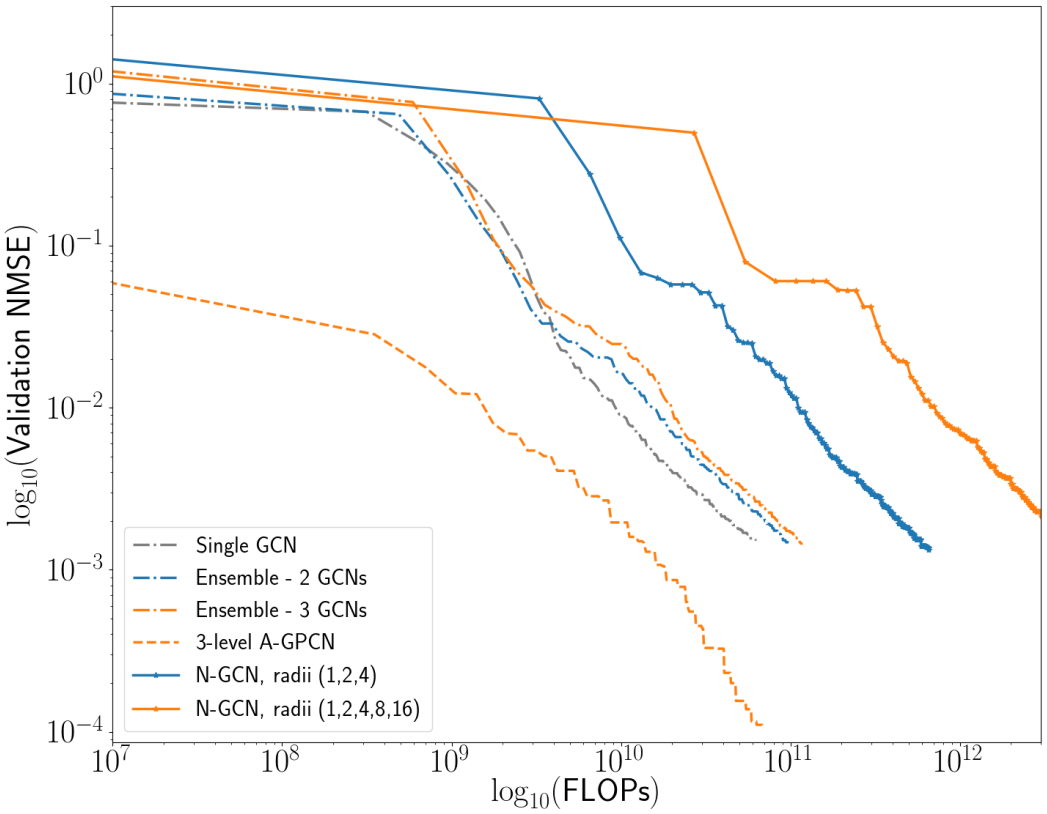

Figure 7.11: Comparison of Normalized MSE on held-out validation data as a function of FLOPs expended for a variety of ensemble Graph Convolutional Network Models. Plotted error is is the minimum validation error of the model over training thus far. We see that especially in early stages of training, our model formulation learns faster (e.g. requires fewer FLOPs) than an ensemble of 2, 3 or 5 GCNs with the same number of filters.

We summarize the structure of each of our models in Table 7.2. In Figure 7.11 we show a comparison between each of these models, for one particular random seed (42). Error on the validation set is tracked as a function of computational cost expended to train the model (under our cost assumption given above). We see that all four GPCN models outperform the other types of ensemble model during early training, in the sense that they reach lower levels of error for the same amount of computational work performed. Additionally, the adaptive GPCN models outperform all other models in terms of absolute error: after the same number of training epochs (using the same random seed) they reach an order of magnitude lower error. Table 7.3 shows summary statistics for several runs of this experiment with varying random

seeds; we see that the A-GPCN models consistently outperform all other models considered. Note that Figures 7.11,7.10, and 7.9 plot the Normalize Mean Squared Error (NMSE). This unitless value compares the output signal to the target after both are normalized by the procedure described in section 7.4.1.

7.5.6 Machine Learning Summary

The machine learning model presented in Section 7.3.4 is validated through numerical experiments on an evaluation dataset. First, variations of our architecture are compared in Section 7.5.2, demonstrating that deeper versions of this architecture perform significantly better, and that re-training the P matrices leads to further accuracy gains. In Section 7.5.3, we fix the model architecture to be the best-performing of those considered in Section 7.5.2, and examine the effect of varying training schedules, including multigrid-like and coarse-to-fine training. These experiments demonstrate that our model achieves comparable error in less computation when trained in a multigrid fashion. Finally in Sections 7.5.4 and 7.5.5, we validate our model by training other types of graph convolutional network models on the same learning task. We show significant accuracy gains over previous GCN ensemble models such as [2] and also outperform DiffPool [116], which learns pooling maps during the training process. All results comparing our model to other GCN models are summarized in Tables 7.3 and 7.4. Together these experiments demonstrate the superior accuracy and efficiency of our machine learning architecture.

7.6 Future Work

7.6.1 Differentiable Models of Molecular Dynamics

This work demonstrates the use of feed-forward neural networks to approximate the energetic potentials of a mechanochemical model of an organic molecule. Per-timestep, GCN models may not be as fast as highly-parallelized, optimized MD codes. However, neural networks are highly flexible function approximators: the GCN training approach outlined in this paper could also be used to train a GCN which predicts the energy levels per particle at the end of a simulation (once equilibrium is reached), given the boundary conditions and initial conditions of each particle. In the case of our MT experiments, approximately 3×10^5 steps were required to reach equilibrium. The computational work to generate a suitably large and diverse training set would then be amortized by the GCN's ability to generalize to initial conditions, boundary conditions, and hyperparameters outside of this data set. Furthermore, this GCN reduced model would be fully differentiable, making it possible to perform gradient descent with respect to any of these inputs. In particular, we derive here the gradient of the input to a GCN model with respect to its inputs.

Derivation of Energy Gradient w.r.t Position

As described above, the output of our GCN (or GPCN) model is a $n \times 1$ matrix (or vector) Y, representing the energy of each simulated particle.. The total energy of the molecule at position X is given by a sum over monomers, $E = \sum_{i=1}^{n} [Y]_i$. Note that any GCN's initial layer update is given by the update rule:

$$X^{`} = g_1 \left(ZXW_1 + b_1 \right).$$

During backpropagation, as an intermediate step of computing the partial derivatives of energy with respect to W_1 and b_1, we must compute the partial $\frac{\partial E}{\partial A_1}$ of energy with respect to the input to the activation function g_1:

$$A_1 = ZXW_1 + b_1$$
$$X^{`} = g_1(A_1).$$

We therefore assume we have this derivative. By the Chain Rule for matrix derivatives:

$$\left[\frac{\partial E}{\partial X}\right]_{ij} = \frac{\partial E}{\partial [X]_{ij}} = \sum_{k,p} \frac{\partial E}{\partial [A_1]_{kp}} \frac{\partial [A_1]_{kp}}{\partial [x_{ij}]}.$$

Since

$$[A_1]_{kp} = \left(\sum_{c,d} [Z]_{kc}[X]_{cd}[W_1]_{dp}\right) + [b_1]_{kp}$$

and therefore

$$\begin{aligned}
\frac{\partial [A_1]_{kp}}{\partial [X]_{ij}} &= [Z]_{ki}[W_1]_{jp}, \\
\frac{\partial E}{\partial [X]_{ij}} &= \sum_{k,p} \frac{\partial E}{\partial [A_1]_{kp}} [Z]_{ki}[W_1]_{jp} \\
\frac{\partial E}{\partial X} &= Z^T \frac{\partial E}{\partial A_1} W_1^T.
\end{aligned} \tag{7.6}$$

Furthermore, since our GPCN model is a sum of the output of several GCNs, we can also derive a backpropagation equation for the gradient of the fine-scale input, $\mathbf{X}$, with respect to the energy prediction of the entire ensemble. Let $E^{(i)}$ represent the total [4] fine-scale energy

[4] meaning summed over all monomers, in contrast to the per-monomer predictions made in Section 7.5.

prediction of the ith member of the ensemble, so that $E = \sum_{i=1}^{k} E^{(i)}$. Then, let

$$\frac{\partial E^{(i)}}{\partial X^{(i)}} = Z^{(i)^T} \frac{\partial E^{(i)}}{\partial A_1^{(i)}} W_1^{(i)^T} \tag{7.7}$$

be the application of Equation 7.6 to each GCN in the ensemble. Since the input to the ith member of the ensemble is given by $X^{(i)} = P_{1,i}^T \mathbf{X}$, we can calculate the gradient of $E^{(i)}$ with respect to $\mathbf{X}$, again using the Chain Rule:

$$\begin{aligned}
\frac{\partial E^{(i)}}{\partial [\mathbf{X}]_{mn}} &= \sum_{s=1}^{N_s} \sum_{t=1}^{N_t} \frac{\partial E^{(i)}}{\partial \left[X^{(i)}\right]_{st}} \frac{\partial \left[X^{(i)}\right]_{st}}{\partial [\mathbf{X}]_{mn}} \\
&= \sum_{s=1}^{N_s} \sum_{t=1}^{N_t} \frac{\partial E^{(i)}}{\partial \left[X^{(i)}\right]_{st}} \frac{\partial \left[P_{1,i}^T \mathbf{X}\right]_{st}}{\partial [\mathbf{X}]_{mn}} \\
&= \sum_{s=1}^{N_s} \sum_{t=1}^{N_t} \frac{\partial E^{(i)}}{\partial \left[X^{(i)}\right]_{st}} \delta_{tm} [P_{1,i}]_{ns} \\
&= \sum_{s=1}^{N_s} \frac{\partial E^{(i)}}{\partial \left[X^{(i)}\right]_{sm}} [P_{1,i}]_{ns}
\end{aligned}$$

Therefore,

$$\frac{\partial E^{(i)}}{\partial [\mathbf{X}]_{mn}} = P_{1,i} \frac{\partial E^{(i)}}{\partial X^{(i)}}$$

and so

$$\frac{\partial E}{\partial \mathbf{X}} = \sum_{i=1}^{k} \frac{\partial E^{(i)}}{\partial \mathbf{X}} = \sum_{i=1}^{k} P_{1,i} \frac{\partial E^{(i)}}{\partial X^{(i)}}$$

This backpropagation rule may then be used to adjust $\mathbf{X}$, and thereby find low-energy configurations of the molecular graph. Additionally, analogous to the GCN training procedure outlined in Section 7.5.3, this optimization over molecule positions could start at the coarse

scale and be gradually refined.

7.6.2 Tensor Factorization

Recent work has re-examined GCNs in the context of the extensive literature on tensor decompositions. LanczosNet [67], uses QR decomposition of the structure matrix to aggregate information from large neighborhoods of the graph. The "Tensor Graph Convolutional Network" of Zhang et. al [117], is a different decomposition method, based on graph factorization; a product of GCNs operating on each factor graph can be as accurate as a single GCN acting on the product graph. Since Theorem 3.7.1 shows that the GDD of a graph product is bounded by the distances between the factor graphs, it seems reasonable to combine both ideas into a model which uses a separate GPCN for each factor. One major benefit of this approach would be that a transfer-learning style approach can be used. For example, we could train a product of two GCN models on a short section of microtubule; and then re-use the weights in a model that predicts energetic potentials for a longer microtubule. This would allow us to extend our approach to MT models whose lengths are biologically relevant, e.g. 10^3 tubulin monomers.

7.6.3 Graph Limits

Given that *in vivo* microtubules are longer than the one simulated in this paper by a factor of as much as 200x, future work will focus on scaling these methods to the limit of very large graphs. In particular, this means repeating the experiments of Sections 7.5, but with longer tube graphs. We hypothesise that tube graphs which are closer to the microtubule graph (under the LGDD) as their length $n \to \infty$ will be more efficient reduced-order models for a GPCN hierarchy. This idea is similar to the "graphons" (which are the limits of sequences

of graphs which are Cauchy under the Cut-Distance of graphs) introduced by Lovász [68]. To show that it is reasonable to define a "graph limit" of microtubule graphs in this way, we plot the distance between successively longer microtubule graphs. Using the same notation as in Section 7.4.3, we define three families of graphs:

- $G_{\text{Grid}}(n, 13)$: Grids of dimensions $n \times 13$, and;
- $G_{\text{Tube}(n,13,1)}$: Microtubule graphs with 13 protofilaments, of length n, with offset 1, and;
- $G_{\text{Tube}(2n,13,3)}$: Microtubule graphs with 13 protofilaments, of length $2n$, with offset 3.

In this preliminary example, as n is increased, we see a clear distinction in the distances $D(G_{\text{Tube}(n,13,1)}, G_{\text{Tube}(2n,13,3)})$ and $D(G_{\text{Grid}(n,13)}, G_{\text{Tube}(2n,13,3)})$, with the former clearly limiting to a larger value as $n \to \infty$.

7.7 Conclusion

We introduce a new type of graph ensemble model which explicitly learns to approximate behavior at multiple levels of coarsening. Our model outperforms several other types of GCN, including both other ensemble models and a model which coarsens the original graph using DiffPool. We also explore the effect of various training schedules, discovering that A-GPCNs can be effectively trained using a coarse-to-fine training schedule. We present the first use of GCNs to approximate energetic potentials in a model of a microtubule.

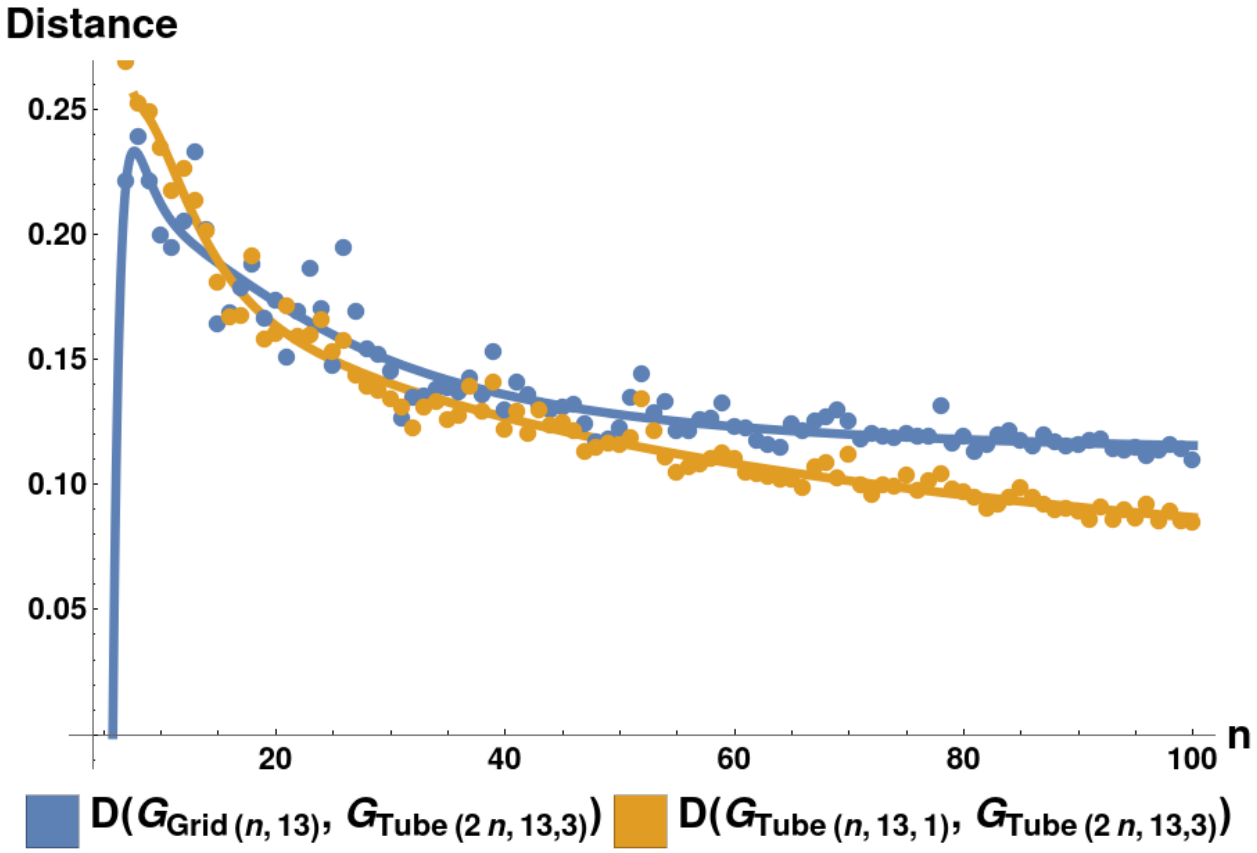

Figure 7.12: Limiting behavior of two classes of distances between graphs, as a function of graph size. We plot $D(G_{\text{Tube}(n,13,1)}, G_{\text{Tube}(2n,13,3)})$ and $D(G_{\text{Grid}(n,13)}, G_{\text{Tube}(2n,13,3)})$ as a function of n, along with seventh-degree polynomial fit curves of each. The smaller tube graphs are closer than the grid graphs to the larger tube, even in the large-graph limit.

Chapter 8

Other Applications

This chapter presents two shorter investigations which use Graph Diffusion Distance.

8.1 Shape Analysis for Discretized Meshes

In this section we demonstrate that graph diffusion distance captures structural properties of 3D point clouds. Ten 3D meshes (see Figure 8.1 for an illustration of the meshes used) were chosen to represent an array of objects with varying structural and topological properties. Not all of the mesh files chosen are simple manifolds: for example, the "y-tube" is an open-ended cylinder with a fin around its equator. Each mesh was used to produce multiple graphs, via the following procedure:

1. Subsampling the mesh to 1000 points;
2. Performing a clustering step on the new point cloud to identify 256 cluster centers;
3. Connecting each cluster center to its 16 nearest neighbors in the set of cluster centers.

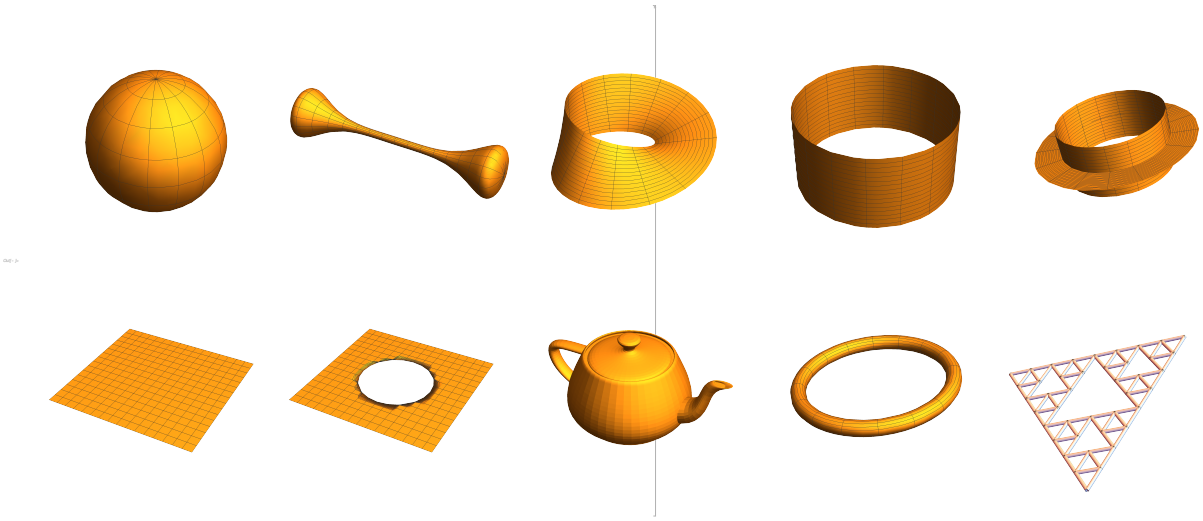

Figure 8.1: 3D meshes used in the shape analysis experiment. Each mesh was used to produce several sampled discretizations, which were then compared using GDD.

Since each pass of this procedure (with different random seeds) varied in Step 1, each pass produced a different graph. We generated 20 graphs for each mesh, and compared the graphs using GDD.

The results of this experiment can be seen in Figure 8.2. This Figure shows the three first principal components of the distance matrix of GDD on the dataset of graphs produced as described above. Each point represents one graph in the dataset, and is colored according to the mesh which was used to generate it. Most notably, all the clusters are tight and do not overlap. Close clusters represent structurally similar objects: for example, the cluster of graphs from the tube mesh is very close to the cluster derived from the tube with an equatorial fin. This synthetic dataset example demonstrates that graph diffusion distance is able to compare structural information about point clouds and meshes.

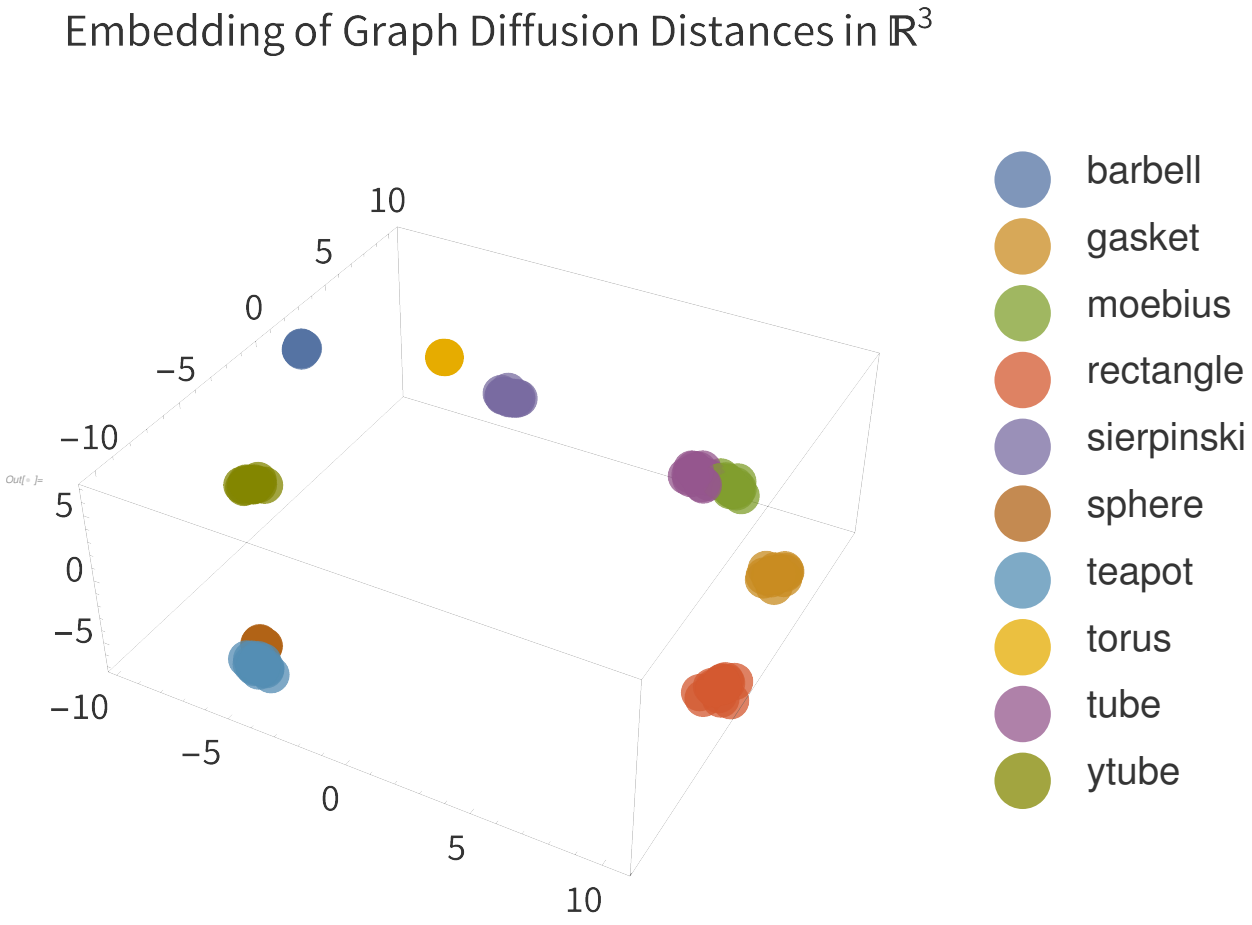

Figure 8.2: Embedding of pairwise distances between mesh discretizations. We see that GDD clusters each category of mesh tightly, and furthermore that clusters are nearby when they are structurally similar meshes, and distant otherwise. Axes represent the three principal components of the distance matrix and are thus unitless.

8.2 Morphological Analysis of Cell Networks

In this section, we present an application of GDD to biological image analysis, and a generalization of GDD that makes it more suitable for machine learning tasks.

8.2.1 Biological Background

Species in the plant genus *Arabidopsis* are of high interest in plant morphology studies, since 1) its genome was fully sequenced in 1996, relatively early [57], and 2) its structure makes it relatively easy to capture images of the area of active cell division: the *shoot apical meristem (SAM)*. Recent work [89] has found that mutant *Arabidopsis* specimens with a decreased level of expression of genes trm6,trm7, and trm8 demonstrate more variance in the placement of new cell walls during cell division.

We prepare a dataset of *Arabidopsis* images with the following procedure:

1. Two varieties of *Arabidopsis* (wild type as well as "trm678 mutants": mutants with decreased expression of all three of TRM6,TRM7,and TRM8) were planted and kept in the short-day condition (8 hours of light, 16 hours of dark) for 6 weeks.

2. Plants were transferred to long-day conditions and kept there until the SAM had formed. This took two weeks for wild-type plants and three weeks for TRM678 mutants.

3. The SAM of each pant was then was then dissected and observed with a confocal microscope, Leica SP8 upright scanning confocal microscope equipped with a water immersion objective (HC FLUOTAR L 25x/0.95 W VISIR).

4. This resulted in 3D images of the SAM from above (e.g. perpendicular to the plane of cell division) collected for both types of specimens.

5. Each 3D image was converted to a 2D image showing only the cell wall of the top layer of cells in the SAM.

We take 20 confocal microscope images (13 from wild-type plants and 7 from trm678 mutants) of shoot apical meristems, and process them to extract graphs representing local

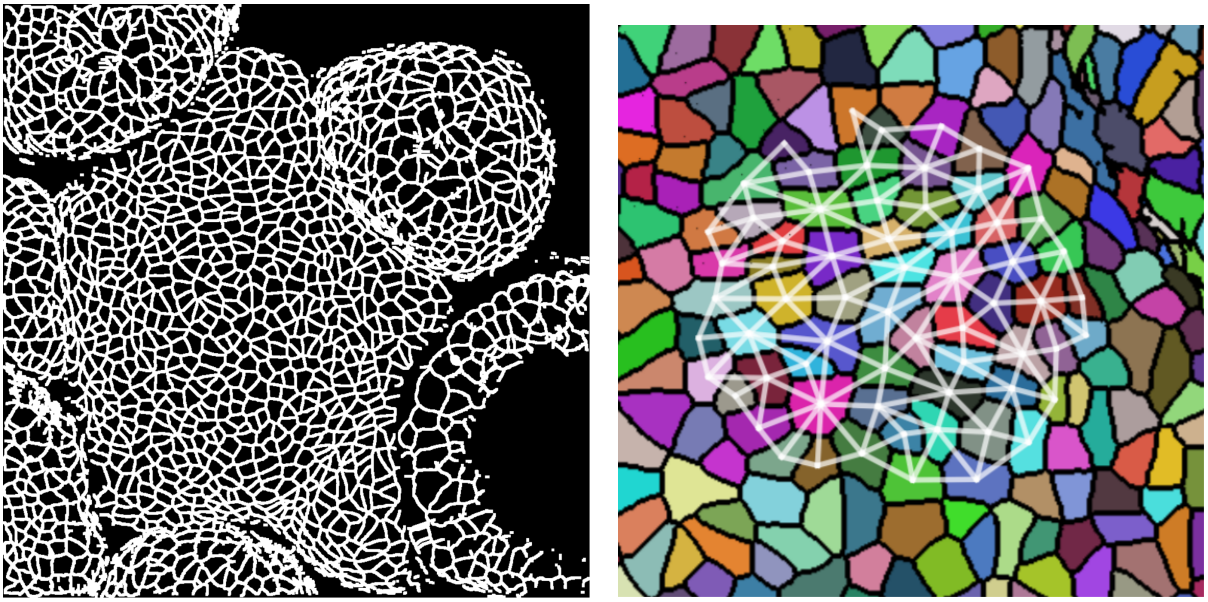

Figure 8.3: Left: a microscope image of the SAM of a mutant *Arabidopsis* specimen. Right: the same specimen, with separate cells false-colored and an example extracted cell neighborhood graph overlaid.

neighborhoods of cells. Each graph consists of a cell and its 63 closest neighbors (64 cells total). Cell neighborhood selection was limited to the central region of each SAM image, since the primordia surrounding the SAM are known to have different morphological properties. For each cell neighborhood, we produce a graph by connecting two cells iff their shared boundary is 30 pixels or longer. For each edge, we save the length of this shared boundary, as well as the angle of the edge from horizontal and the edge length. We extracted 600 cell neighborhoods for each type, for a total of 1200 graphs. See Figure 8.3 for an example SAM image and resulting graph, and see Figure 8.4.

8.2.2 GDD is a differentiable function of t and edge weights

Once all of the eigenvalues λ_i and eigenvectors v_i (of a matrix L) are computed, we may backpropagate through the eigendecomposition as described in [76] and [5]. If our edge weights (and therefore the values in the Laplacian matrix L) are parametrized by some

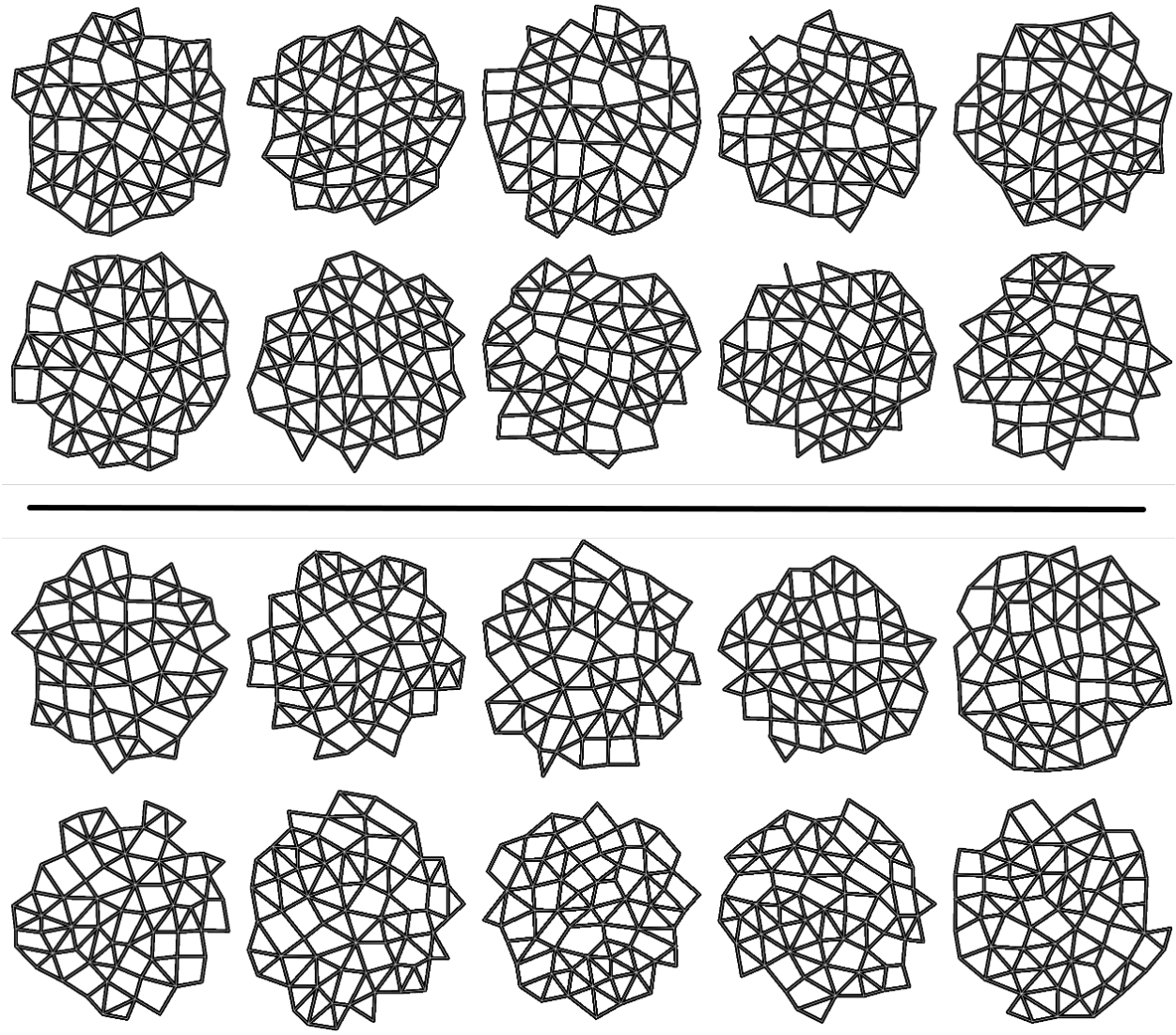

Figure 8.4: Top: ten example morphological graphs extracted from wild-type SAM images. Bottom: ten morphological graphs drawn from trm678 mutant images.

value θ, and our loss function $\mathcal{L}$ is dependent on the eigenvalues of L, then we can collect the gradient $\frac{\partial \mathcal{L}}{\partial \theta}$ as:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_k \left(\frac{\partial \mathcal{L}}{\partial \lambda_k} \frac{\partial \lambda_k}{\partial \theta} \right) = \sum_k \left(\frac{\partial \mathcal{L}}{\partial \lambda_k} v_k^T \frac{\partial L}{\partial \theta} v_k \right). \tag{8.1}$$

In practice, if the entries of L are computed as a function of θ using an automatic differentiation package (such as PyTorch [80]) the gradient matrix $\frac{\partial L}{\partial \theta}$ is already known before eigendecomposition. We note here that for any fixed value of t, all of the operations needed

to compute GDD are either simple linear algebra or continuous or both. Therefore, for any loss function $\mathcal{L}$ which takes the GDD between two graphs as input, we may optimize $\mathcal{L}$ by backpropagation through the calculation of GDD using Equation 8.1.

8.2.3 Weighted Diffusion Distance

We make two main changes to GDD to make it capable of being tuned to specific graph data. First, we replace the real-valued optimization over t with a maximum over an explicit list of t values $t_1, t_2, \ldots t_p$. This removes the need for an optimization step inside the GDD calculation. Second, we re-weight the Frobenius norm in the GDD calculation with a vector of weights β_j which is the same length as the list of eigenvalues (these weights are normalized to sum to 1). The resulting GDD calculation is then:

$$D(G_1, G_2) = \max_{t \in t_1, t_2, \ldots t_p} \sqrt{\sum_{j=1}^{n} \beta_j \left(e^{t\lambda_j^{(1)}} - e^{t\lambda_j^{(2)}} \right)^2}. \tag{8.2}$$

This distance calculation may then be explicitly included in the computation graph (e.g. in PyTorch) of a machine learning model, without needing to invoke some external optimizer to find the supremum over all t. t_n and β_j may be tuned by gradient descent or some other optimization algorithm to minimize a loss function which takes d as input. Tuning the t_n values results in a list of values of t for which GDD is most informative for a given dataset, while tuning β_j reweights GDD to pay most attention to the eigenvalues which are most discriminative. In the experiments in Section 8.2.4 we demonstrate the efficacy of tuning these parameters using contrastive loss.

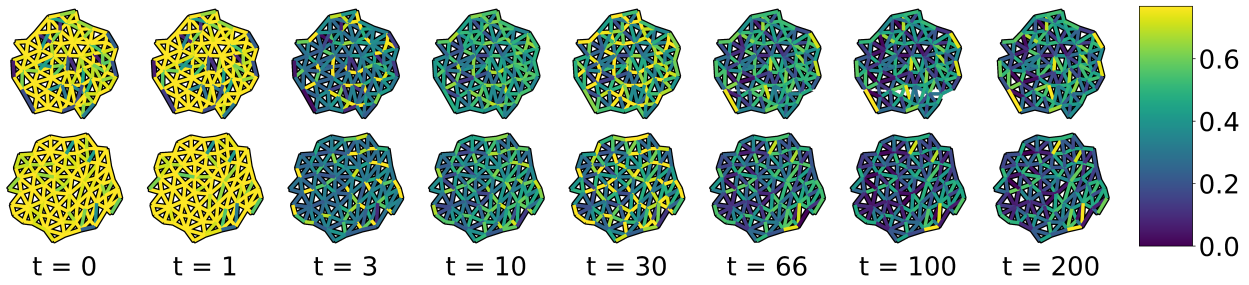

Figure 8.5: A neural network model learns edge weights which distinguish two classes of graphs. Each row shows the weight values assigned by the network at different times during the training process, from pre-training (far left) to convergence (right). The top row represents a patch of wild-type *Arabidopsis* cells, and the bottom row represents mutants. The pictured edge weights cause these two categories of graph to have distinct spectra.

8.2.4 Learning Edge Weighting Functions

Here, we note that if graph edge weights are determined by some function f parametrized by θ, we may still apply all of the machinery of Sections 8.2.2 and 8.2.3. A common edge weighting function for graphs embedded in Euclidean space is the *Gaussian Distance Kernel*, $w_{ij} = \exp\left(\frac{-1}{2\sigma^2} d_{ij}\right)$, where d_{ij} is the distance between nodes i and j in the embedding. σ is the 'radius' of the distance kernel and can be tuned in the same way as β and t. In cases like the data discussed in this section, our edge weights are vector-valued, and it is therefore advantageous to replace this hand-picked edge weight with weights chosen by a general function approximator, e.g. an artificial neural network [36]. As before, the parameters of this ANN could be tuned using gradients backpropagated through the GDD calculation and eigendecomposition. Example weights learned by an ANN, trained with the constrastive loss function, can be seen in Figure 8.5. We test each of the GDD generalizations proposed, on each dataset. Both datasets were split 85/15 % train/validation; All metrics we report are calculated on the validation set. For each dataset, we compare the following four methods: 1) GDD on unweighted graphs, with no tuning of t or other parameters; 2) Gaussian kernel edge weights (fixed σ), with t and β_i tuned; 3) Gaussian kernel edge weights, with t, σ, and β_i 4) General edge weights parametrized by a small neural network. For methods 2 and 3,

the input to the distance kernel was the distance between nodes in the original image. All parameters were tuned using ADAMOpt [61] (with default PyTorch hyperparameters and batch size 256) to minimize the *contrastive loss* function [46]. Training took 200 epochs. For the neural network approach, edge weights were chosen as the final output of a neural network with three layers of sizes $\{32, 128, 1\}$ with SiLU activations on the first two layers and no activation function on the last layer. Results of these experiments can be found in Table 8.1, and distance matrices (along with a distance-preserving embedding [24] of all points in the dataset) for the cell morphology dataset can be seen in Figure 8.6. The distance matrices developed using the ANN approach clearly show better separation between the two categories. The validation accuracy on the cell morphology dataset is best for the ANN method.

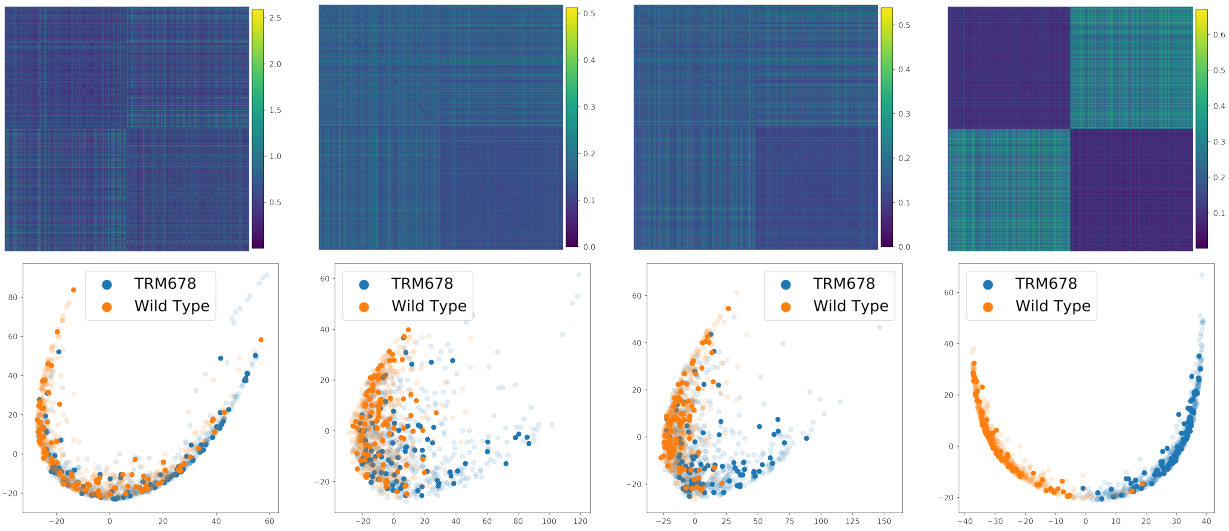

Figure 8.6: Top row: distance matrices between cell morphology graphs produced using our methods 1-4, as described in Section 8.2.4. Bottom row: the result of embedding each distance matrix in 2D using its first two principal components. Training dataset points are semitransparent, validation points are opaque. Note: the principal components used for this embedding are calculated only on the submatrix which corresponds to training data.

Method	Validation Accuracy % (morpho)
GDD only	77.7
t-tuning and β-weights	85.0
t and σ-tuning,β-weights	85.5
ANN Parametrization	**98.3**

Table 8.1: Validation set accuracy for a simple K-nearest neighbors classifier for both datasets, for all four methods. The validation set was the same for each of these tests. The value reported is the highest value over all $K \in \{3, 50\}$.

8.3 Conclusion and Future Work

This section presents two applications of GDD to the classification of graphs embedded in 2D and 3D. In both, we demonstrate that GDD (and a parametrized variant) produce clusters which draw out structural differences in our dataset(s). Additionally, we demonstrate a method to compute distance metrics between edge-labelled graphs, in such a way as to respect class labels. This approach is flexible and can be implemented entirely in PyTorch, making it possible to learn a distance metric between graphs that were previously not able to be discriminated by Graph Diffusion Distance. In the future we hope to apply this method to more heterogenous graph datasets by including the varying-size version of GDD. We also note here that our neural network approach, as described, is not a Graph Neural Network in the sense described by prior works like [62, 7], as there is no message-passing step. We expect message-passing layers to directly improve these results and hope to include them in a future version of differentiable GDD.

Chapter 9

Conclusion

This thesis presents an in-depth examination of the properties and efficient computation of Graph Diffusion Distance (GDD). This distance metric, defined for undirected graphs of unequal size, uses the eigenvalues of the graph Laplacian to quantify the difference in behavior between diffusion of heat running on the nodes of each graph. In Chapter 2, we defined a class of distance measures which use the eigenvalues of a graph Laplacian, and its matrix exponential, as a distinctive measurement of the properties of diffusion on the nodes of the graph. This family of related distance measures have several nice theoretical properties, which we examine in detail in Chapter 3. One specific nice property is that GDD and several of its variants are bounded above and below by expressions which depend on the eigenvalues of the two graphs (the spectral lower and upper bounds), making them possible to compute with real-valued (rather than combinatorial) optimization. However, these measures typically still require expensive matrix-valued optimizations to calculate. In Chapters 4 and 5 we present, and examine the numeric behavior of, a novel optimization algorithm which greatly reduces the number and size of matrix optimizations which need to be performed, resulting in a speedup of up to 1000x.

With this computational speedup, GDD can differentiate digits of MNIST, distinguish which of several 3D meshes a given graph is a discretization of, and be used to classify biological data (morphological graphs); we show some of these applications in Chapter 8. Furthermore, the P matrix which is produced during the GDD calculation is useful as a prolongation/restriction operator to coarsen and refine computational graphs. An example of this latter application is the use of coarsening and refinement operators as part of a neural network architecture; these multiscale machine learning models are more accurate and more efficient than their single-level counterparts. In Chapters 6 and 7 we demonstrate the advantages of these machine learning approaches through a variety of numerical experiments. Specifically, P matrices can be used to automatically coarsen the model architecture of a machine learning model. The resulting coarsened model learns more efficiently, and in the case of the experiments in Chapter 7 learns to emulate a dataset with lower error than a model operating only at one scale.

The graph Laplacian (along with its matrix exponential), is a fundamental object which captures structural information about a graph. This thesis presents a variety of methods for comparing such operators and accelerating machine learning models which are constructed around the graph Laplacian.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A System for Large-Scale Machine Learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] S. Abu-El-Haija, A. Kapoor, B. Perozzi, and J. Lee. N-GCN: Multi-Scale Graph Convolution for Semi-supervised Node Classification, 2018.

[3] E. L. Allgower and K. Georg. *Numerical continuation methods: an introduction*, volume 13. Springer Science & Business Media, 2012.

[4] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[5] A. L. Andrew, K.-W. E. Chu, and P. Lancaster. Derivatives of eigenvalues and eigenvectors of matrix functions. *SIAM journal on matrix analysis and applications*, 14(4):903–926, 1993.

[6] D. Bacciu, F. Errica, A. Micheli, and M. Podda. A Gentle Introduction to Deep Learning for Graphs. *arXiv preprint arXiv:1912.12693*, 2019.

[7] D. Bacciu, F. Errica, A. Micheli, and M. Podda. A gentle introduction to deep learning for graphs. *Neural Networks*, 2020.

[8] B. R. Bakshi and G. Stephanopoulos. Wave-Net: A Multiresolution, Hierarchical Neural Network with Localized Learning. *AIChE Journal*, 39(1):57–81, 1993.

[9] M. Belkin and P. Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *Advances in Neural Information Processing Systems*, pages 585–591, 2002.

[10] I. Benjamini and O. Schramm. Recurrence of distributional limits of finite planar graphs. *Electron. J. Probab.*, 6:13 pp., 2001.

[11] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[12] J. Bijsterbosch and A. Volgenant. Solving the Rectangular Assignment Problem and Applications. *Annals of Operations Research*, 181(1):443–462, 2010.

[13] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[14] C. Borgs, J. Chayes, H. Cohn, and Y. Zhao. An $l^{\{p\}}$ theory of sparse graph convergence i: Limits, sparse random graph models, and power law distributions. *Transactions of the American Mathematical Society*, 372(5):3019–3062, 2019.

[15] A. E. Brouwer and W. H. Haemers. *Spectra of Graphs*. Springer Science & Business Media, 2011.

[16] A. E. Brouwer and W. H. Haemers. Distance-regular graphs. In *Spectra of Graphs*, pages 177–185. Springer, 2012.

[17] A. E. Brouwer and E. Spence. Cospectral Graphs on 12 Vertices. *The Electronic Journal of Combinatorics*, 16(1):N20, 2009.

[18] R. E. Burkard and E. Cela. Linear Assignment Problems and Extensions. In *Handbook of Combinatorial Optimization*, pages 75–149. Springer, 1999.

[19] R. E. Burkard, M. Dell'Amico, and S. Martello. *Assignment Problems*. Springer, 2009.

[20] B. Chakrabortty, I. Blilou, B. Scheres, and B. M. Mulder. A Computational Framework for Cortical Microtubule Dynamics in Realistically Shaped Plant Cells. *PLoS Computational Biology*, 14(2):e1005959, 2018.

[21] L. Chen, Z. Gan, Y. Cheng, L. Li, L. Carin, and J. Liu. Graph optimal transport for cross-domain alignment. In *International Conference on Machine Learning*, pages 1542–1553. PMLR, 2020.

[22] B. M. Clapper. Munkres Implementation for Python, 2008–. [Online; accessed June 10, 2018].

[23] D. Cohen-Steiner, W. Kong, C. Sohler, and G. Valiant. Approximating the spectrum of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1263–1271. ACM, 2018.

[24] M. A. Cox and T. F. Cox. Multidimensional scaling. In *Handbook of data visualization*, pages 315–347. Springer, 2008.

[25] G. A. Croes. A method for solving traveling-salesman problems. *Operations research*, 6(6):791–812, 1958.

[26] D. M. Cvetkovic, P. Rowlinson, and S. Simic. *An Introduction to the Theory of Graph Spectra*. Cambridge University Press Cambridge, UK, 2010.

[27] Y. Dodonova, M. Belyaev, A. Tkachev, D. Petrov, and L. Zhukov. Kernel Classification of Connectomes Based on Earth Mover's Distance between Graph Spectra. *arXiv preprint arXiv:1611.08812*, 2016.

[28] C. Donnat and S. Holmes. Tracking Network Dynamics: A Survey of Distances and Similarity Metrics. *arXiv preprint arXiv:1801.07351*, 2018.

[29] H. Dou and X. Wu. Coarse-to-Fine Trained Multi-Scale Convolutional Neural Networks for Image Classification. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2015.

[30] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open Source Scientific Tools for Python, 2001–. [Online; accessed June 20 2019].

[31] M. Eshera and K.-S. Fu. A Graph Distance Measure for Image Analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14(3):398–408, 1984.

[32] M. Eshera and K.-S. Fu. An Image Understanding System using Attributed Symbolic Representation and Inexact Graph-Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(5):604–618, 1986.

[33] R. D. Falgout, S. Friedhoff, T. V. Kolev, S. P. MacLachlan, and J. B. Schroder. Parallel Time Integration with Multigrid. *SIAM Journal on Scientific Computing*, 36(6):C635–C661, 2014.

[34] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

[35] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[36] K. Fukushima and S. Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[37] X. Gao, B. Xiao, D. Tao, and X. Li. A Survey of Graph Edit Distance. *Pattern Analysis and Applications*, 13(1):113–129, 2010.

[38] M. K. Gardner, A. J. Hunt, H. V. Goodson, and D. J. Odde. Microtubule assembly dynamics: new insights at the nanoscale. *Current opinion in cell biology*, 20(1):64–70, 2008.

[39] M. K. Gardner, M. Zanic, and J. Howard. Microtubule Catastrophe and Rescue. *Current Opinion in Cell Biology*, 25(1):14–22, 2013.

[40] C. D. Godsil and B. McKay. Constructing Cospectral Graphs. *Aequationes Mathematicae*, 25(1):257–268, 1982.

[41] S. Gold, C.-P. Lu, A. Rangarajan, S. Pappu, and E. Mjolsness. New Algorithms for 2d and 3d Point Matching: Pose Estimation and Correspondence. In *Advances in neural information processing systems*, pages 957–964, 1995.

[42] S. Gold, A. Rangarajan, and E. Mjolsness. Learning with preknowledge: Clustering with Point and Graph Matching Distance Measures. In *Advances in Neural Information Processing Systems*, pages 713–720, 1995.

[43] E. M. Grais, H. Wierstorf, D. Ward, and M. D. Plumbley. Multi-Resolution Fully Convolutional Neural Networks for Monaural Audio Source Separation. *arXiv preprint arXiv:1710.11473*, 2017.

[44] J. Gu, B. Hua, and S. Liu. Spectral Distances on Graphs. *Discrete Applied Mathematics*, 190:56–74, 2015.

[45] E. Haber, L. Ruthotto, E. Holtham, and S.-H. Jun. Learning Across Scales - Multiscale Methods for Convolution Neural Networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[46] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[47] A. Hagberg, P. Swart, and D. S Chult. Exploring Network Structure, Dynamics, and Function using NetworkX. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[48] D. K. Hammond, Y. Gur, and C. R. Johnson. Graph Diffusion Distance: A Difference Measure for Weighted Graphs based on the Graph Laplacian Exponential Kernel. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 419–422. IEEE, 2013.

[49] H. Hartle, B. Klein, S. McCabe, A. Daniels, G. St-Onge, C. Murphy, and L. Hébert-Dufresne. Network comparison and the within-ensemble graph distance. *Proceedings of the Royal Society A*, 476(2243):20190744, 2020.

[50] C. Heindl. lapsolver: Fast Linear Assignment Problem (LAP) Solvers for Python Based on c-extensions. `https://github.com/cheind/py-lapsolver`, 2018.

[51] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

[52] L. Hogben. *Handbook of Linear Algebra*. CRC Press, 2006.

[53] W. R. Inc. Mathematica, Version 11.3. Champaign, IL, 2018.

[54] A. I. Jewett, Z. Zhuang, and J.-E. Shea. Moltemplate: a Coarse-Grained Model Assembly Tool. *Biophysical Journal*, 104(2):169a, 2013.

[55] T. Johnson, T. Bartol, T. Sejnowski, and E. Mjolsness. Model reduction for stochastic camkii reaction kinetics in synapses by graph-constrained correlation dynamics. *Physical biology*, 12(4):045005, 2015.

[56] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open Source Scientific Tools for Python, 2001–. [Online; accessed June 10, 2018].

[57] S. Kaul, H. L. Koo, J. Jenkins, M. Rizzo, T. Rooney, L. J. Tallon, T. Feldblyum, W. Nierman, M. I. Benito, X. Lin, et al. Analysis of the genome sequence of the flowering plant arabidopsis thaliana. *nature*, 408(6814):796–815, 2000.

[58] T.-W. Ke, M. Maire, and S. X. Yu. Neural Multigrid. *arXiv preprint arXiv:1611.07661*, 2016.

[59] T.-W. Ke, M. Maire, and S. X. Yu. Multigrid Neural Architectures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6665–6673, 2017.

[60] M. Kikumoto, M. Kurachi, V. Tosa, and H. Tashiro. Flexural rigidity of individual microtubules measured by a buckling force with optical traps. *Biophysical journal*, 90(5):1687–1696, 2006.

[61] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[62] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[63] A. Kis, S. Kasas, B. Babić, A. Kulik, W. Benoit, G. Briggs, C. Schönenberger, S. Catsicas, and L. Forro. Nanomechanics of Microtubules. *Physical Review Letters*, 89(24):248101, 2002.

[64] A. J. Laub. *Matrix Analysis for Scientists and Engineers*, volume 91. SIAM, 2005.

[65] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[66] Y. LeCun, C. Cortes, and C. Burges. MNIST Handwritten Digit Database. *AT&T Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010. [Accessed June 10, 2018.].

[67] R. Liao, Z. Zhao, R. Urtasun, and R. S. Zemel. LanczosNet: Multi-Scale Deep Graph Convolutional Networks. *arXiv preprint arXiv:1901.01484*, 2019.

[68] L. Lovász. *Large Networks and Graph Limits*, volume 60. American Mathematical Soc., 2012.

[69] H. P. Maretic, M. El Gheche, G. Chierchia, and P. Frossard. Got: An optimal transport framework for graph comparison. In *Advances in Neural Information Processing Systems*, pages 13876–13887, 2019.

[70] G. Margolin, I. V. Gregoretti, T. M. Cickovski, C. Li, W. Shi, M. S. Alber, and H. V. Goodson. The mechanisms of microtubule catastrophe and rescue: implications from analysis of a dimer-scale computational model. *Molecular biology of the cell*, 23(4):642–656, 2012.

[71] A. McGregor and D. Stubbs. Sketching Earth-Mover Distance on Graph Metrics. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 274–286. Springer, 2013.

[72] M. I. Molodtsov, E. A. Ermakova, E. E. Shnol, E. L. Grishchuk, J. R. McIntosh, and F. I. Ataullakhanov. A molecular-mechanical model of the microtubule. *Biophysical journal*, 88(5):3167–3179, 2005.

[73] J.-M. Morvan and B. Thibert. On the approximation of a smooth surface with a triangulated mesh. *Computational Geometry*, 23(3):337–352, 2002.

[74] J. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[75] J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 1965.

[76] R. B. Nelson. Simplified calculation of eigenvector derivatives. *AIAA journal*, 14(9):1201–1205, 1976.

[77] T. Onoyama, S. Kubota, K. Oyanagi, and S. Tsuruta. A Method for Solving Nested Combinatorial Optimization Problems-a Case of Optimizing a Large-Scale Distribution Network. In *SMC 2000 Conference Proceedings. 2000 IEEE International Conference on Systems, Man and Cybernetics. 'Cybernetics Evolving to Systems, Humans, Organizations, and their Complex Interactions' (cat. no. 0*, volume 1, pages 340–345. IEEE, 2000.

[78] F. Pampaloni and E.-L. Florin. Microtubule Architecture: Inspiration for Novel Carbon Nanotube-based Biomimetic Materials. *Trends in Biotechnology*, 26(6):302–310, 2008.

[79] V. Y. Pan and Z. Q. Chen. The complexity of the matrix eigenproblem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 507–516. ACM, 1999.

[80] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[81] A. Patney, M. S. Ebeida, and J. D. Owens. Parallel view-dependent tessellation of catmull-clark subdivision surfaces. In *Proceedings of the conference on high performance graphics 2009*, pages 99–108, 2009.

[82] S. Plimpton. Fast Parallel Algorithms For Short-Range Molecular Dynamics. Technical report, Sandia National Labs., Albuquerque, NM (United States), 1993.

[83] A. Rangarajan, S. Gold, and E. Mjolsness. A Novel Optimizing Network Architecture with Applications. *Neural Computation*, 8(5):1041–1060, 1996.

[84] T. Rapcsák. On Minimization on Stiefel Manifolds. *European Journal of Operational Research*, 143(2):365–376, 2002.

[85] M. Raw. Robustness of coupled algebraic multigrid for the navier-stokes equations. In *34th Aerospace sciences meeting and exhibit*, page 297, 1996.

[86] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[87] G. Rossum. Python Reference Manual. *Technical Report*, 1995.

[88] A. Sanfeliu and K.-S. Fu. A Distance Measure between Attributed Relational Graphs for Pattern Recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(3):353–362, 1983.

[89] E. Schaefer, K. Belcram, M. Uyttewaal, Y. Duroc, M. Goussot, D. Legland, E. Laruelle, M.-L. de Tauzia-Moreau, M. Pastuglia, and D. Bouchez. The preprophase band of microtubules controls the robustness of division orientation in plants. *Science*, 356(6334):186–189, 2017.

[90] T. Schneider and E. Stoll. Molecular-Dynamics Study of a Three-Dimensional One-Component Model for Distortive Phase Transitions. *Physical Review B*, 17(3):1302, 1978.

[91] J. B. Schroder. Parallelizing Over Artificial Neural Network Training Runs with Multigrid. *arXiv preprint arXiv:1708.02276*, 2017.

[92] C. Scott and E. Mjolsness. Multilevel Artificial Neural Network Training for Spatially Correlated Learning. *SIAM Journal on Scientific Computing*, 41(5):S297–S320, 2019.

[93] C. B. Scott. DiffusionDistance: Efficient Calculation of Graph Diffusion Distance in Python. `https://github.com/scottcb/DiffusionDistance`, 2021.

[94] C. B. Scott and E. Mjolsness. Novel diffusion-derived distance measures for graphs, 2019.

[95] I. V. Serban, T. Klinger, G. Tesauro, K. Talamadupula, B. Zhou, Y. Bengio, and A. C. Courville. Multiresolution Recurrent Neural Networks: An Application to Dialogue Response Generation. In *AAAI*, pages 3288–3294, 2017.

[96] A. Shamir. A survey on mesh segmentation techniques. In *Computer graphics forum*, volume 27, pages 1539–1556. Wiley Online Library, 2008.

[97] S. L. Shaw, R. Kamyar, and D. W. Ehrhardt. Sustained Microtubule Treadmilling in Arabidopsis Cortical Arrays. *Science*, 300(5626):1715–1718, 2003.

[98] A. Sinha, P. Malo, and K. Deb. A Review on Bilevel Optimization: from Classical to Evolutionary Approaches and Applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2018.

[99] S. F. Stewman and A. Ma. A structural mechano-chemical model for dynamic instability of microtubule. *bioRxiv*, page 291682, 2018.

[100] K. Stüben. A review of algebraic multigrid. In *Numerical Analysis: Historical Developments in the 20th Century*, pages 331–359. Elsevier, 2001.

[101] A. Stukowski. Visualization and Analysis of Atomistic Simulation Data with OVITO - the Open Visualization Tool. *Modelling Simulation in Materials Science and Engineering*, 18(1), JAN 2010.

[102] T. Takasone, S. Juodkazis, Y. Kawagishi, A. Yamaguchi, S. Matsuo, H. Sakakibara, H. Nakayama, and H. Misawa. Flexural rigidity of a single microtubule. *Japanese journal of applied physics*, 41(5R):3015, 2002.

[103] O. Tange. GNU Parallel - The Command-Line Power Tool. *;login: The USENIX Magazine*, 36(1):42–47, Feb 2011.

[104] S. H. Tindemans, E. E. Deinum, J. J. Lindeboom, and B. Mulder. Efficient Event-Driven Simulations Shed New Light on Microtubule Organization in the Plant Cortical Array. *Frontiers in Physics*, 2:19, 2014.

[105] J. Townsend, N. Koep, and S. Weichwald. Pymanopt: A Python Toolbox for Optimization on Manifolds using Automatic Differentiation. *arXiv preprint arXiv:1603.03236*, 2016.

[106] P. Turaga, A. Veeraraghavan, and R. Chellappa. Statistical Analysis on Stiefel and Grassmann Manifolds with Applications in Computer Vision. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[107] E. R. Van Dam and W. H. Haemers. Which Graphs are Determined by their Spectrum? *Linear Algebra and its applications*, 373:241–272, 2003.

[108] E. R. Van Dam and W. H. Haemers. Developments on Spectral Characterizations of Graphs. *Discrete Mathematics*, 309(3):576–586, 2009.

[109] S. Van Der Walt, S. C. Colbert, and G. Varoquaux. The NumPy Array: a Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[110] V. VanBuren, L. Cassimeris, and D. J. Odde. Mechanochemical Model of Microtubule Structure and Self-Assembly Kinetics. *Biophysical Journal*, 89(5):2911–2926, 2005.

[111] P. Vaněk, J. Mandel, and M. Brezina. Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems. *Computing*, 56(3):179–196, 1996.

[112] L. Verlet. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Physical Review*, 159(1):98, 1967.

[113] H.-W. Wang and E. Nogales. Nucleotide-dependent bending flexibility of tubulin regulates microtubule assembly. *Nature*, 435(7044):911–915, 2005.

[114] Z. Wen and W. Yin. A Feasible Method for Optimization with Orthogonality Constraints. *Mathematical Programming*, 142(1-2):397–434, 2013.

[115] P. Wesseling and C. W. Oosterlee. Geometric multigrid with applications to computational fluid dynamics. *Journal of computational and applied mathematics*, 128(1-2):311–334, 2001.

[116] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*, pages 4800–4810, 2018.

[117] T. Zhang, W. Zheng, Z. Cui, and Y. Li. Tensor Graph Convolutional Neural Network. *arXiv preprint arXiv:1803.10071*, 2018.

[118] J. Zhao, L. Dai, M. Zhang, F. Yu, M. Li, H. Li, W. Wang, and L. Zhang. PGU-net+: Progressive Growing of U-net+ for Automated Cervical Nuclei Segmentation. *Lecture Notes in Computer Science*, page 51–58, Dec 2019.