ABSTRACT

Title of dissertation: COMPUTATIONALLY COMPARING BIOLOGICAL NETWORKS AND RECONSTRUCTING THEIR EVOLUTION

Robert Patro, Doctor of Philosophy, 2012

Dissertation directed by: Professor Carl Kingsford
Department of Computer Science

Biological networks, such as protein-protein interaction, regulatory, or metabolic networks, provide information about biological function, beyond what can be gleaned from sequence alone. Unfortunately, most computational problems associated with these networks are NP-hard. In this dissertation, we develop algorithms to tackle numerous fundamental problems in the study of biological networks.

First, we present a system for classifying the binding affinity of peptides to a diverse array of immunoglobulin antibodies. Computational approaches to this problem are integral to virtual screening and modern drug discovery. Our system is based on an ensemble of support vector machines and exhibits state-of-the-art performance. It placed 1st in the 2010 DREAM5 competition.

Second, we investigate the problem of biological network alignment. Aligning the biological networks of different species allows for the discovery of shared structures and conserved pathways. We introduce an original procedure for network alignment based on a novel topological node signature. The pairwise global alignments of biological networks

produced by our procedure, when evaluated under multiple metrics, are both more accurate and more robust to noise than those of previous work.

Next, we explore the problem of ancestral network reconstruction. Knowing the state of ancestral networks allows us to examine how biological pathways have evolved, and how pathways in extant species have diverged from that of their common ancestor. We describe a novel framework for representing the evolutionary histories of biological networks and present efficient algorithms for reconstructing either a single parsimonious evolutionary history, or an ensemble of near-optimal histories. Under multiple models of network evolution, our approaches are effective at inferring the ancestral network interactions. Additionally, the ensemble approach is robust to noisy input, and can be used to impute missing interactions in experimental data.

Finally, we introduce a framework, GrowCode, for learning network growth models. While previous work focuses on developing growth models manually, or on procedures for learning parameters for existing models, GrowCode learns fundamentally new growth models that match target networks in a flexible and user-defined way. We show that models learned by GrowCode produce networks whose target properties match those of real-world networks more closely than existing models.

Computationally Comparing Biological Networks and Reconstructing Their Evolution

by

Robert Patro

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
2012

Advisory Committee:

Professor Carl Kingsford, Chair
Professor Héctor Corrada Bravo
Professor Wojciech Czaja, Dean's representative
Professor Mihai Pop
Professor Amitabh Varshney

Acknowledgments

My graduate experience at Maryland has truly been one of personal and professional growth, and it is a journey that I will never forget. A journey is not just about where you go, but about who you meet along the way. I've met some wonderful people who I must thank. First and foremost, I'd like to thank my advisor, Carl Kingsford, for inspiring an appreciation in me for the theory and practice of Computer Science that I might not have otherwise enjoyed. Working with Carl has been an exhilarating experience; he makes difficult problems look tractable by making brilliant ideas seem obvious. I have learned from him a clarity of thought and a direct approach to solving problems that I hope to keep for the rest of my career.

I'd also like to thank Amitabh Varshney, with whom I had the pleasure of working for four years. Amitabh taught me a lot, both about how to do science and about how to interact with the people who make up any scientific community. Most importantly, his philosophy of anticipating the future and asking the big questions has permanently impacted my approach to research, and I'm very thankful for that. I was originally introduced to Amitabh by Michelle Hugue, to whom I owe a debt of gratitude for guiding me into the graduate school process and connecting me with an advisor who was such a good fit for me.

The other members of my committee, Héctor Corrada Bravo, Wojtek Czaja and Mihai

Pop, have all been important contributors to my academic development while at the University of Maryland. I'm honored to have such an excellent committee and I can't think of another set of people whose scientific opinions I value more.

I'm also very grateful to the wonderful friends and colleagues with whom I've worked during the course of my graduate career. In particular, Sujal Bista, Horace Ip and André Maximo made the graphics lab a fun and interesting place to work and were able to provide insightful discussion on almost any topic. While in the Kingsford Lab, I've had the honor of being surrounded by great people; specifically my lab mates and co-authors Geet Duggal, Darya Filippova, Justin Malin, Guillaume Marçais, Saket Navlakha, Emre Sefer and Hao Wang. From the morning lattés to the afternoon philosophy debates to the 3 a.m. paper deadlines, these colleagues and friends have made the past few years the most enjoyable and interesting I could imagine. The other members of the CBCB with whom I've had regular interaction and discussion, including Irina Astrovskaya, Jeremy Bellay, Mohammad Ghodsi, Ted Gibbons, Chris Hill, Daehwan Kim, Henry Lin, Bo Liu, Joe Paulson and Praveen Vaddadi have made it a great place to work and learn.

Before I began a proper academic career, my family always nurtured my sense of curiosity and my interest in science. For this, and for the comfortable and loving environment they provided for me growing up, I can never repay them. In particular, I want to thank my parents, Laura Teal and Robert Patro, and my grandparents Genny (Nana) and Jerry (Pop) Clarke.

Finally, I'd like to thank my wife, Kim, who constantly reminds me that there is more to life than my research. She has acted as a much-needed counterweight to my sometimes extreme working habits. Most of all, however, she has been a friend and partner through

all of my trials and triumphs; she makes my life better.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The analysis of biological networks holds the promise of fundamentally improving our understanding of how life operates at the molecular scale. As new technologies and experimental protocols provide a constant stream of higher quality experimental data, computer science must meet the challenge of developing algorithms and building tools which make the analysis and understanding of this raw data possible.

The goal of this work is to develop novel algorithms that allow us to answer important questions about the structure, origin and function of interactions between biological molecules. We can answer some of these questions by analyzing a single network, while others require a comparative approach in which two or more networks must be analyzed in tandem.

In particular, we explore four problems where the aim is to uncover the relationships within and among biological networks. In Chapter 2, we describe a system which predicts binding interactions between peptides and human immunoglobulin antibodies. Computational approaches to predict antibody binding are an integral part of modern drug discovery, helping to target new and effective medications by enabling the virtual screening of an immense number of candidate molecules. Therefore, we wish to develop methods that are as accurate, as fast and as scalable as possible.

Uncovering antibody binding interactions can be viewed as an edge prediction task in a bipartite network between the set of peptides and the set of antibodies, where an edge

exists between an antibody and any peptide that binds to it. We develop a system, based on an ensemble of diverse classifiers, that exhibits state-of-the-art performance in this epitope classification task. Our system is able to predict these interactions with high accuracy (86%), and also performs well with respect to other common classification metrics (area under ROC and PR curves).

Next, we turn our attention to the problem of comparing extant biological networks directly. The use of network data, in conjunction with sequence, allows us to gain more insight into protein homology across species. Discovering shared structure in the biological networks of different species allows us to transfer biological knowledge, such as the functional annotations of proteins, between these species. Potential applications exist in the clinical field as well. Network comparison can provide evidence about what mechanisms and targets for drugs or treatments might have a higher probability of being transferred from model or test organisms to humans.

In Chapter 3 we explore the problem of biological network alignment. To tackle this alignment problem, we introduce a novel node descriptor based on the eigenvalues of the normalized graph Laplacian for subgraphs of different radii around the node. This descriptor is robust to small changes in graph structure, and for similar graphs, correlates highly with true graph edit distance. We explore different alignment strategies and show that combining the appropriate strategy with our new descriptor and a local search procedure leads to state-of-the art performance in aligning various protein interaction networks. Additionally, we suggest two novel metrics to gauge the quality of pairwise network alignments; one which measures the topological quality of the alignment, and the other the biological relevance.

We can address questions about the shared structure and evolutionary similarities and differences between related organisms even more directly by attempting to reconstruct the network topology of their common ancestor. We take this approach in Chapter 4, where we present a solution to the problem of uncovering biological network interactions in an-

cestral species. Knowing the topology of ancestral biological networks allows us to answer questions about the conservation and divergence of protein interaction sub-networks and regulatory and metabolic pathways. We infer ancestral network state by comparing the interactions and gene duplication histories between extant biological networks. We develop a framework to represent gene duplication histories that allows an efficient encoding of the interactions between the constituent genes. This framework admits an efficient $O(n^2)$ dynamic programming algorithm to determine the most parsimonious interaction history, and hence, the ancestral network state, if one allows certain temporally inconsistent events. We present a post-processing method that can remove these inconsistencies, but which may possibly sacrifice the optimality of the computed solution. We test the effectiveness of this framework under multiple models of regulatory network evolution, and show that we are able to recover the topology of the ancestral regulatory network with high precision and moderate to high recall under a broad range of network evolution parameters.

In Chapter 5, we substantially extend our approach to ancestral network reconstruction. The improved approach sums over a large number of parsimonious and nearly parsimonious evolutionary histories to obtain a posterior probability for the existence of ancestral interactions. It provides three main benefits over the method of Chapter 4. First, since ancestral interactions are given a relative weight, they can be ranked, allowing us to posit their existence with varying degrees of confidence. Second, by considering an ensemble of parsimonious histories rather than a single evolutionary history, the new method is made more robust to noise in the input network. This is important since existing techniques for measuring molecular interactions (specifically protein-protein interactions) are known to produce a substantial number of false-negative and false-positive interactions. Finally, the improved method yields a posterior probability for both ancestral and extant interactions. This allows us to quantify the level of surprise we have in either observing or not observing an extant interaction given the structure of the duplication history and the presence or absence of homologous interactions in related species. Thus, we can use our method to

identify potential false-positive or false-negative interactions in existing experimental data and suggest candidates for future targeted experiments.

Chapter 6 focuses on a new framework for representing and learning models of network growth. We introduce a framework, GrowCode, in which network growth models are represented as programs, composed of primitive instructions, which run on a virtual machine. The instruction set we propose is capable of representing a number of existing network growth models which produce networks with substantially different topological properties. We then show how genetic programming can be used to effectively search the space of programs and learn a growth model that matches a user-defined set of properties of some target graph. Unlike previous work, which has focused primarily on the manual development of network growth models or on the creation of automated procedures to learn parameters for existing growth models, GrowCode is capable of automatically learning new models of network growth. We demonstrate that, for multiple different classes of target graphs, automatically learned GrowCode models produce graphs with topological properties more closely matching those of the real-world target graphs than existing network growth models.

Chapter 7 suggests some interesting directions for future work. Specifically, we explore the relationship between the network alignment problem, the ancestral network reconstruction problem and network growth model inference problem, and suggest how some of our approaches might be combined to produce new methods that further improve the results presented in this dissertation.

Chapter 2

Epitope Prediction with Sequence and Structure-Based Features using an Ensemble of SVMs

2.1 Introduction

In this chapter, we present an effective computational method to predict the binding affinity of peptides to antibodies. In particular, our system predicts high binding affinity to intravenous immunoglobulin (IVIg), and was developed in response to DREAM5 Challenge 1, where it was the top performing solution. This is a particularly difficult task, due in part to the great diversity exhibited by the challenge's IVIg fractions, which are isolated from up to $100{,}000$ individuals. Understanding the binding of peptides to IVIg antibodies is an important problem with numerous implications in the study of immune and autoimmune disorders.

Our approach computes a wide array of different peptide features. Some of these features arise from the peptide sequence alone, such as measures of localized physicochemical properties, amino acid composition, and features derived from existing string kernel functions. Other features are computed from inferred shape complementarity of the peptide with experimentally measured immunoglobulin protein structures. The goal of considering such a diverse set of features is to capture as much relevant information as possible to assist in determining the binding affinity of a given peptide. We train a probabilistic support vector machine (SVM) classifier on each feature independently, optimizing parameters automatically by means of cross-validation.

This procedure yields an ensemble of classifiers, the predictions of which are weighted by a regularized cross-validation score and combined to classify novel peptide sequences. One of the primary strengths of our approach is its ease of extensibility. New features and even new classifiers can be easily incorporated into the ensemble. Our approach shows promising results. By training efficient classifiers on a diverse set of features, our approach obtains an area under the receiver operator characteristic (AUROC) curve of 0.893 and an area under the precision recall curve (AUPR) of 0.772 on the withheld challenge testing examples.

2.2 Related Work

While the particular challenge issued in the DREAM5 competition was new, there has been a significant amount of work on predicting the binding affinity of peptides to various target molecules. Various machine learning classifiers such as artificial neural networks [134], hidden Markov models [16], and support vector machines [13] have been explored in tackling the problem of predicting Human Leukocyte Antigen (HLA) binding peptides.

Much work has also focused on the prediction of T-cell and B-cell binding peptides. Zhao *et al.* [138] explore various classifiers to predict peptide T-cell binding. Using a 10 dimensional feature vector to represent each amino acid, they discover that SVMs provide the best classification performance in their task. Huang *et al.* [55] also explore the classification of peptide binding to T-cells using a support vector machine classifier. They present a novel peptide feature based on combining a 20-dimensional indicator vector with amino acid similarity information encoded by the `BLOSUM50` [54] matrix. Zhang *et al.* [135] consider 3D features and a random forest based classifier to predict B-cell epitopes.

Nanni and Lumini, introduced the MppS system [90], which relies on an ensemble of support vector machines, trained on various physicochemical properties, to classify peptide binding to HIV-protease and T-cells. They use sequential floating forward selection to select a subset of features, and combine the individual classifier predictions using the

max rule [63]. More recently, Nanni and Lumini [91] have explored the use of a novel peptide encoding scheme which relies on the use of nonlinear dimensionality reduction to extract the information encoded across a large number of physicochemical properties. They demonstrate that this novel feature representation, when used in conjunction with a support vector machine classifier, exhibits state-of-the-art performance in predicting peptide T-cell binding.

2.3 Approach

Our approach to the epitope classification task is based on an ensemble of learners. A study of previous literature yields a wide variety of useful features for related epitope classification tasks; though none of the previous work deals with such a wide variety of paratopes — the regions of antibodies which recognize antigens — as is found in intravenous immunoglobulin fractions. The features we consider range from simple sequence-based features, such as the average value of some physicochemical property over a sliding window of amino acids, to more complicated structural features based on estimated docking accuracy of the conjectured peptide conformation to a measured immunoglobulin structure.

Each of the features we consider are used, either separately or in small groups, to train Support Vector Machine (SVM) models. Explicitly constructed features, which can be represented as numerical vectors (see section 2.5.1 below), are trained using a radial basis function (RBF) kernel. To perform the SVM training and classification, we use the libsvm software [19].

For all features we consider, the optimal SVM parameters are discovered via a grid search and cross-validation. For a given SVM model, the cross-validation accuracy for the optimal set of parameters is used as a weight to combine the corresponding model's predictions with the others from the ensemble. To test our approach, we train on a subset of $13{,}638$ peptides. The features we consider obtain cross-validation accuracies on our training subset ranging from $\sim 80\% - \sim 83\%$. When we combine the predictions of these

classifiers on the testing subset, we obtain an accuracy above 86%. Furthermore, on the testing subset, our ensemble achieves an area under the Receiver Operating Characteristic (AUROC) curve of 0.893, and an area under the Precision Recall (AUPR) curve of 0.772, both of which represent a substantial gain over the area under the respective curves of any individual classifier.

2.4 Probabilistic Support Vector Machines

Motivated by the success of previous work in various protein prediction tasks [13, 90, 91, 138, 77], we chose to use a Support Vector Machine (SVM) as our classifier. Originally introduced by Cortes and Vapnik [29], SVMs are efficient and highly accurate classifiers, especially when one expects non-linear separability between classes and high-dimensional training features. While the soft-margin formulation of SVMs allow for a maximum-margin classifier of data, even when the training set is not perfectly separable, it still results in a binary classification scheme. During the classification stage, instances are assigned a hard label, as belonging to the negative or positive class. Such a hard labeling poses no problem when only a single classifier is used to label test data. However, when an ensemble of classifiers is used, it is useful to have extra information about the degree to which the label assigned by each individual classifier should be trusted.

For this reason, we chose to use Platt's extension [106], which provides probabilistic outputs for a support vector machine's classifications. Instead of receiving a 0-1 label, each instance is given an *a posteriori* estimate of the probability with which it belongs to the positive class, as illustrated in figure 2.1. Thus, we expect that instances which clearly belong to the negative class will be given a value close to 0, while instances which belong to the positive class will be given values close to 1. One can then devise a hard classification rule by imposing a cutoff τ. Instances with *a posteriori* probabilities above τ are considered to belong to the positive class. All other instances are then simply assigned to the negative class.

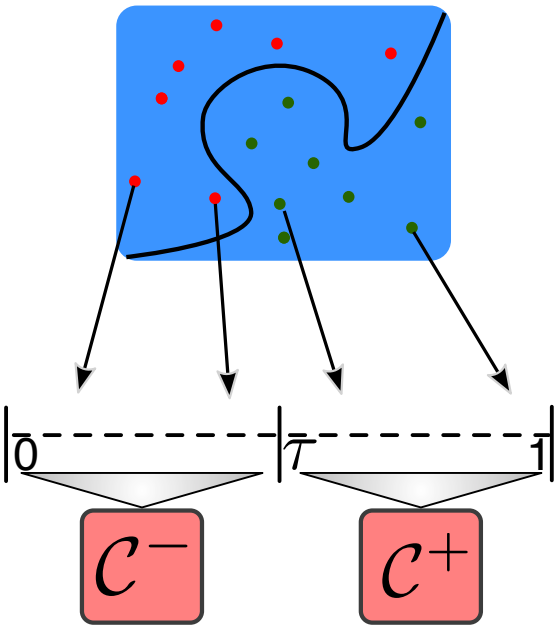


Figure 2.1: *EAR: Probabilistic SVM*

Probabilistic support vector machines assign each instance an a posteriori probability of belonging to the positive class. These probabilities can then be used to obtain a hard classification by imposing a decision rule which classifies instances whose probability is greater than some threshold τ as belonging to the positive class, while other instances are labeled as belonging to the negative class.

Having a probabilistic interpretation of the classification for data instances makes combining the output of different classifiers a simple task. We use a variant of the sum rule, where the predictions of the individual classifiers are summed and normalized to yield the prediction of the ensemble. Specifically, the prediction of the ensemble for a particular instance $\mathbf{x}^i$ is computed using equation **??**.

$$p_+^{\text{ens}}(\mathbf{x}^i) = \frac{1}{A}\sum_{j=0}^{M} a^j p_+^j(\mathbf{x}^i) \tag{2.1}$$

Where $p_+^j(\cdot)$ is the *a posteriori* probability output by classifier j, and a^j is classifier j's cross-validation accuracy. A is a normalization factor equal to $\sum_{j=0}^{M} a^j$. We can then simply take p_+^{ens} to be the probability with which the ensemble predicts $\mathbf{x}^i$ to belong to the

positive class, or we can obtain a discrete class prediction with the decision rule:

$$\mathbf{x}^i \in \begin{cases} \mathcal{C}^+ & \text{if } p_+(\mathbf{x}^i) \geq \tau, \\ \mathcal{C}^- & \text{otherwise.} \end{cases}$$

In our experiments, we set τ as 0.5, but other values may be reasonable. In fact, one may one may even learn the value of τ which yields the best performance by using a held-out subset of the training data.

2.5 Features

2.5.1 Numerically Encoded Sequence Features

Some of the features we consider encode the peptide sequence directly as a numerical vector. In this case, for each peptide, we can record the relevant features directly, and train our SVM model using the RBF kernel.

There are two distinct types of sequence features that we encode numerically. First, we consider a simple variation on the peptide encoding scheme presented by Huang and Dai [55]. Essentially, we will encode each amino acid in the peptide by replacing its single

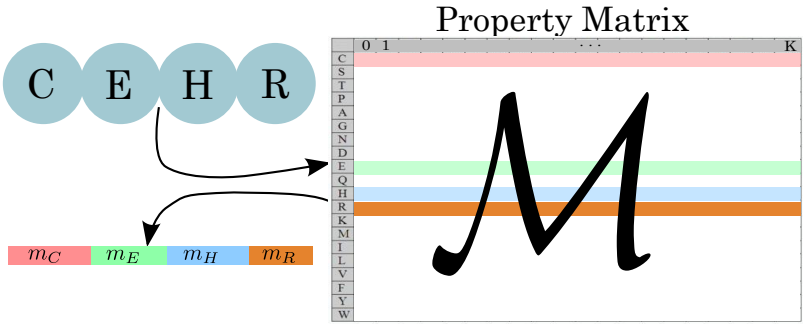


Figure 2.2: *EAR: Generic Encoding*

The property matrix encoding scheme allows us to associate each amino acid with a row of a $20 \times K$ property matrix, $\mathcal{M}$. We may construct $\mathcal{M}$ to account for any amino acid properties we deem to be relevant. Under this scheme, the representation for each peptide is given by the concatenation of the encoding of its constituent amino acids; leading to length Kd encoding for a peptide of length d.

letter code with its corresponding row in the BLOSUM50 matrix. The BLOSUM50 matrix contains empirically derived log-odds scores which encode the frequency of different amino acid substitutions and is commonly used to measure the similarity between different amino acids. Let the peptide of length d be given as $\mathbf{x} = (a_0, a_1, \ldots, a_d)$, where a_i is the amino acid in the i^{th} position of the peptide. Further, let $\text{row}(a)$ map the amino acid a to its corresponding row in the BLOSUM50 matrix. We encode the peptide as $\text{enc}(\mathbf{x}) = (\text{row}(a_0), \text{row}(a_1), \ldots, \text{row}(a_d))$. For the length d peptide $\mathbf{x}$, $\text{enc}(\mathbf{x})$ will be a $20k$ dimensional feature vector. In addition to the peptide encoding using the BLOSUM matrix, we also consider the encoding using the **nlf** and **sa** matrices suggested by Nanni and Lumini [91]. These matrices are derived by performing dimensionality reduction on a large, rectangular (i.e. $20\times k$ with $k \gg 20$) matrix, where each row corresponds to an amino acid and each column to some physicochemical property. The goal of the dimensionality reduction is to decorrelate the physicochemical properties, reducing the column space of the matrix significantly. The **nlf** matrix is a 20×18 matrix obtained using a nonlinear fisher transform, while the **na** matrix is a 20×10 matrix obtained using a combination of clustering and principal component analysis. As is shown in Figure 2.2, this scheme generalizes naturally to any feature matrix.

The second type of sequence feature we encode numerically involves various physicochemical properties of the constituent amino acids of each peptide. In particular, we analyze the amino acid properties present in the Amino Acid Index (AAIndex) [60]. Each AAIndex property provides a mapping from each of the 20 amino acids to a numerical scale measuring some physicochemical attribute (e.g. hydrophobicity, antigenicity). Currently the AAIndex lists 544 different amino acid properties. We use an approach based on a sliding window and histograms to turn each AAIndex property into a numerical feature vector for a peptide. Consider a single AAIndex property AAI^j, and let $\text{AAI}^j(a)$ represent the numerical value to which the amino acid a is mapped under AAIndex property j.

To form a representation for the entire peptide $\mathbf{x}$ under the property AAI^j, we could

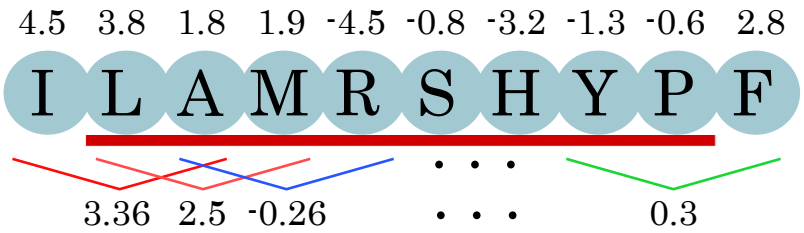


Figure 2.3: *EAR: Sliding Window Scheme*

The sliding window encoding scheme, with $w = 3$, is used to produce a representation of the peptide in this figure based on each amino acid's hydrophobicity. As the window slides from left to right, each amino acid is visited in turn, and it's hydrophobicity index is averaged with that of the preceding and subsequent amino acids.

simply average the contributions of each amino acid (i.e. $\mathrm{AAI}^j(\mathbf{x}) = \frac{1}{k}\sum_{i=0}^{k}\mathrm{AAI}^j(a_i)$). However, we find that this characterization of the peptide is too coarse-grained. Averaging a particular amino acid property over the entire peptide prevents the detection of the spatially localized signatures of the epitopes, which constitute only a sub-region of the whole peptide. Thus, instead of using a single scalar to represent the entire peptide, we will consider a sliding window of length w. Figure 2.3 illustrates this encoding procedure. As we slide the window from left to right across the peptide, we will produce a separate average for each window position. Thus, for a peptide of length d, we will produce a $(d - w + 1)$-dimensional vector. By varying w, we can change the coarseness of this representation. Through a process of experimenting with different values for this classification task, we consider the computation of these features for $w \in [3, 5]$.

2.5.2 String Kernel Features

String kernels are common in natural language processing, and have recently been adopted in bioinformatics applications. Despite the many different variants, the intuition behind most string kernels is the same — encode the relationships between a collection of words or strings by counting the occurrences of shared substrings. However, despite their conceptual simplicity, string kernels allow us to overcome the computationally difficult problem of measuring the similarity of strings in the exponential space of possible substrings. The key

benefit of string kernels, in fact, is the property shared by all kernel methods; they allow us to measure the similarity between two data instances in some very large feature space without requiring us to record feature vectors explicitly.

There are many different varieties of string kernels, ranging from the somewhat simple k-spectrum kernel, which essentially counts the occurrence of all k-mers in each peptide, to the more complex substring-mismatch kernel [78], which considers all shared subsequences between two peptides, allowing for gaps and mismatches. Since we make direct use of these string kernels, and do not alter their implementation or output in any way, we simply list the kernels we consider and the relevant reference for each in Table 2.1.

Kernel	Parameters	Reference
k-spectrum	$k = 3,4,5,6$	[77]
SSSK (triple kernel)	$d = 6$	[72]
bounded range substring	$r = 8$	[122]

Table 2.1: *EAR: String Kernels*

The output of each of these methods is a matrix, known as the kernel matrix, in which the entry at row i and column j is the result of the kernel evaluation between data instances i and j. To train a SVM model for each of these string kernels, we simply compute the kernel matrix, and then make use of the ability of libsvm to train a model using a precomputed kernel.

2.5.3 Structure Features

The features described above allow us to measure many diverse properties of peptides. Yet, they are based overwhelmingly on sequence information. We know, however, that binding relies, in part, on conformation complementarity between an epitope and antibody. While the sequence certainly informs the peptide's conformation, we also attempt to measure this complementarity directly. We first hypothesize a structure for each peptide, and then com-

pute the complementarity of this structure to the measured structure of an immunoglobulin (IgG1) molecule.

First, we compute a hypothesized 3-dimensional structure for each peptide. To perform this task, we make use of the Biochemical Algorithms Library (BALL) [10]. For a particular peptide $\mathbf{x}$, we place the amino acids into the 3D structure in sequence. We position the side chains for each amino acid by choosing the most frequently occurring rotamer position from a rotamer library. The peptide constructed in this manner may not have a globally consistent structure. For example, self-intersections and physically unlikely positions may occur since the peptide was constructed sequentially without accounting for the global conformations. Thus, we optimize the initial structure of each peptide, by performing an energy minimization using the AMBER [107] force field. This procedure alters the conformation of the peptide; relaxing the structure until a (possibly local) energy minimum is achieved. After this process completes, we expect the peptide to be in a globally consistent state, if not necessarily in its native conformation.

Additionally, we obtained an experimentally measured 3D structure for IgG1, the most prevalent class of IgG antibody present in intravenous immunoglobulin. Finally, we measure the conformational complementarity of each of our hypothesized peptide structures with the immunoglobulin structure. To compute this complementarity, we perform a protein-protein docking simulation for each of the constructed peptides against IgG1 using the ZDock software [21]. Each ZDock run produces a list of the 2000 top-ranked (according to ZDock's criteria) docking predictions for each peptide. Each prediction consists of a location and orientation for the peptide, describing the location on the immunoglobulin molecule where the docking occurred, as well as a ZDock score. The ZDock score provides a measure of the complementarity of the peptide and immunoglobulin conformation in the docking region and is used as a proxy for the overall quality of the docking. For each peptide, we form a histogram from the 2000 ZDock scores, and use this histogram as a feature vector with which to train the SVM model. Intuitively, we expect peptides

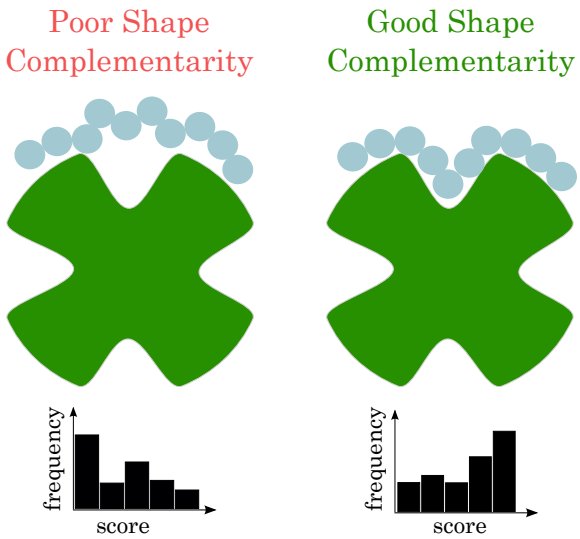


Figure 2.4: *EAR: Structural Features*

Our structural features are composed of a histogram of ZDock scores between a hypothesized peptide conformation and a measured immunoglobulin conformation. The peptide on the left has poor shape complementarity with the immunoglobulin molecule, and therefore, a distribution of docking scores skewed toward low values. Conversely, the peptide on the right has good shape complementarity with the immunoglobulin molecule and a distribution of docking scores skewed toward high values. We expect that peptide on the right, having good shape complementarity with the target antibody, is a more likely binder than the peptide on the left.

whose ZDock score distributions are skewed toward high scores to have better shape complementarity and, therefore, to be more likely binders than peptides whose ZDock score distributions are skewed toward low scores (see Figure 2.4).

2.6 Feature Combination

Applying the procedures described above yields an ensemble of SVM models trained on a medley of diverse features; ranging from running averages of physicochemical features to scores from protein docking simulations. In addition to the feature diversity obtained by considering such a diverse set of features, each SVM model is trained separately; leading, also, to a high degree of parametric diversity (i.e. the parameters with which the models are trained).

Each SVM model will yield a prediction for each peptide in the testing set. Let $p_+^j(\mathbf{x}^i)$

be the probability that the i^{th} peptide is in the positive class for classifier j. Recall that because we are using the probabilistic extension to SVMs [106], these probabilities are given directly. We combine the predictions for all of the classifiers in the ensemble using a variation on the approach presented by Nanni and Lumini [90], which is itself an extension of the sum-rule. We normalize the predictions for each classifier to have a standard deviation 1. Next, we combine the predictions from each of the j classifiers according to **??**. By simply sorting each peptide in the testing set according to this value, we produce a rank ordered list of the peptides which we believe belong to the positive (high binding affinity) class.

2.7 Results

We test the performance of our method on a set of $13{,}640$ held-out peptides. We are interested in examining both the overall performance of the ensemble classifier, as well as the relative improvement we obtain by employing the ensemble as opposed to its constituent classifiers. To analyze the classifiers' performance, we use two standard metrics, the area under the receiver operating characteristic (ROC) curve, and the area under the precision/recall (PR) curve.

The ROC curve measures how the true positive rate of a classifier performs as the false positive rate is increased. An ideal classifier will recall all of the true positive data instances before recalling even a single false positive. Hence, the perfect ROC curve obtains an ordinate value of 1 with the abscissa at 0, and the value remains there as the false positive rate is increased. Analyzing the ROC curves for the classifiers in our ensemble (Figure 2.5a), we observe that many of the classifiers show similar performance, with the exception of the structural classifier which displays significantly lower classification performance. The ensemble, however, yields superior performance compared to any of its constituent classifiers. Thus, at any given false positive rate, the ensemble classifier will obtain a higher true positive rate than any of the other classifiers. This also implies that the ensemble classifier

ranks first when we consider the aggregate metric of the total area under the ROC curve (AUROC). This metric distills the information contained in the ROC curve into a commensurate number which can be analyzed across different classifiers. A perfect classifier has an AUROC of 1; our ensemble classifier has an AUROC of 0.893.

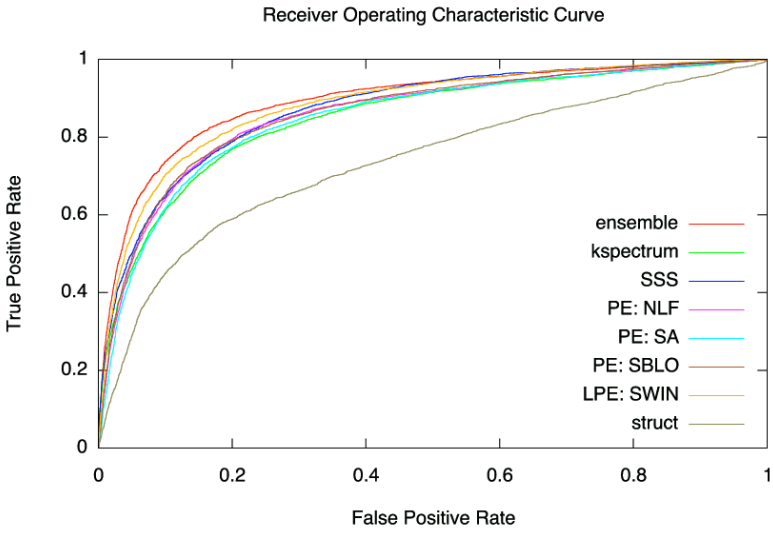
Table 2.2: *EAR: Single vs. Ensemble*

This table shows the performance of a number of different classifiers, as well as the ensemble classifier, with respect to the AUROC and AUPR metrics. The ensemble obtains the best AUROC and AUPR scores. The best single classifier (marked with a *), under both the AUROC and AUPR metrics is the local composition classifier, which performs a sliding window encoding of a number of different physicochemical properties as described in Section 2.5.1. While this classifier performs almost as well as the ensemble under the AUROC metric, the ensemble yields a substantial performance boost ($> 3\%$) under the AUPR metric.

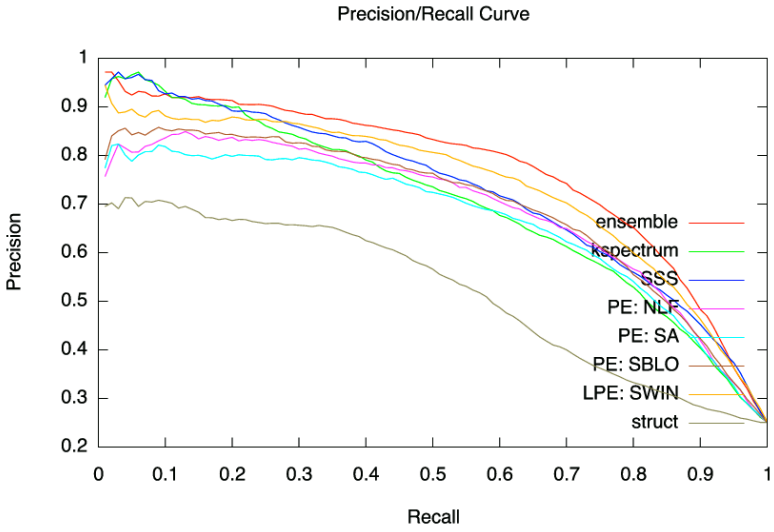
				vs. ensemble	
	Features	AUROC	AUPR	ΔAUROC	ΔAUPR
	k-spectrum	0.85	0.70	-0.043	-0.072
	Sparse Spatial Sample	0.87	0.73	-0.023	-0.042
	Nonlinear Fisher Mat.	0.86	0.69	-0.024	-0.082
	Statistical Analysis Mat.	0.85	0.67	-0.025	-0.102
	BLOSUM Encoding	0.86	0.70	-0.024	-0.072
	Local Composition*	0.88	0.74	-0.013	-0.032
	Structure	0.74	0.53	-0.153	-0.242
	ensemble	0.89	0.77		

The PR curve presents a related view of classifier performance to the ROC curve. It measures how the precision changes as the recall is increased. However, the ROC and PR curves measure different quantities, and the relationship between them is more complex than one might first expect. For example, Davis and Goadrich [31] show that algorithms which maximize the AUROC do not necessarily maximize the AUPR. Indeed, an inspection of the PR curves of our classifiers (figure 2.5b) highlights some differences in classifier performance that are not apparent in the ROC curves. For very small recall values (i.e. recall ≤ 0.1), the sparse spatial sample and k-spectrum string kernels yield the best (and very similar) precision. However, for the vast majority of recall values, the ensemble classi-

fier yields the highest precision. Just as was the case with the ROC curves, the ensemble again achieved the maximum area under the PR curve. While the AUPRs were generally lower than the AUROCs, we did observe that the benefit of the ensemble was larger with respect to the PR curves than the ROC curves. Table 2.2 provides a numeric comparison between a number of different classifiers, showing how they compare to each other and to the ensemble with regard to the AUROC and AUPR metrics.



(a) *Receiver Operating Characteristic (ROC) Perf.*



(b) *Precision / Recall (PR) Perf.*

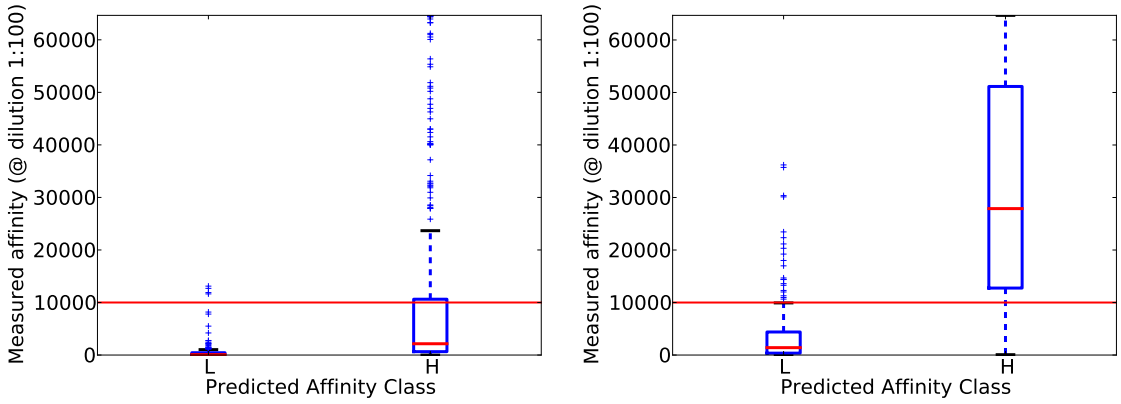
Figure 2.5: *EAR: AUROC & AUPR Results*

Figure 2.5a shows the performance of various classifiers as well as that of the ensemble, as characterized by the receiver operating characteristic (ROC) curve. The ensemble obtains a higher area under the ROC curve (a common metric of overall classification performance) than any of the other classifiers. Additionally, we observe that the ensemble demonstrates uniformly superior performance with respect to the ROC curve. Figure 2.5b illustrates the performance of the same set of classifiers as characterized by their precision/recall (PR) curves. We note that while two string kernels (the k-spectrum and sparse spatial sample kernels) show the best performance at very low recall values, the ensemble again obtains the highest area under the PR curve. It also obtains the highest precision over a large range of recall values.

2.8 Generating Novel Peptides

We also participated in the bonus round of DREAM5 challenge 1. This required the submission of novel peptides along with their predicted reactivity category (high (H), medium (M), or low(L)). These *de novo* predictions had to adhere to a set of rules meant to enforce sequence diversity from the original data set. Specifically, the *de novo* sequences predicted to be in the H or L reactivity category could not share any 4-mer, or exhibit a sequence identity of greater than 6 amino acids in any subsequence of length 11, with any sequence in the same reactivity category in the training set. Our predictions were actually more stringent, as we extended these restrictions to those sequences in the testing set as well. Subject to these constraints, we submitted 1500, 3000 and 1500 peptides in each of the H, M and L categories respectively.

Since our classifier is discriminative, rather than generative, in nature, we made our predictions by first generating the *de novo* peptide sequences and then assigning them a reactivity category according to the predictions of our classifier. We generated the *de novo* sequences using a sampling approach that corresponds, intuitively, to a seeded random walk in sequence space. To obtain a putative sequence for reactivity class $\mathcal{C}$, we choose a seed sequence $s \in \mathcal{C}$ from the training set, and randomly mutate its constituent amino acids until it adheres to the sequence diversity rules. We seeded our *de novo* predictions with 6000 sequences from the training set — 3000 sequences with the highest experimentally measured reactivity and 3000 sequences with the lowest experimentally measured reactivity. We classified these 6000 sequences and sorted them according to their *a priori* probability of belonging to the positive class. There were 2468 peptides with an *a priori* probability greater than or equal to 0.5, and 3542 with an *a priori* probability less than 0.5. The 1500 sequences with the highest probabilities were predicted belong to class H, while the 1500 sequences with the lowest probabilities were predicted to belong to class L. The remaining 3000 peptides, with *a priori* probabilities closest to 0.5, were predicted to belong to the reactivity class M.



(a) *Team Pavia's* de novo *predictions*

(b) *Our* de novo *predictions*

Figure 2.6: *EAR: De Novo Predictions*

The distribution of measured affinities for the de novo peptides predicted to belong to the low (L) and high (H) binding affinity classes. The horizontal line at $10{,}000$ indicates the binding affinity cutoff above which a peptide is considered to have a high binding affinity.

The top 400 and bottom 200 of our *de novo* predictions, as well as the same number generated by the team that placed second (Pavia) in the original challenge, were synthesized. The binding affinity of these *de novo* predictions were then experimentally measured. We discover that both our classifier and that of the Pavia team seem to be doing well in producing both binding and non-binding peptides. However, our *de novo* approach seems to do much better at generating binding peptides compared with that of the other team (Figure 2.6).

Furthermore, the set of peptides produced using our approach is much more diverse than the set generated by team Pavia's approach. To quantify the diversity, we create a graph from the set of predicted high and low binding affinity peptides, where each peptide is a vertex in the graph and two different peptides are connected by an edge if they have less than a specified number x of differences. For each graph, we consider each vertex v, and compute a maximal independent set that is guaranteed to contain v. This yields n maximal independent sets (where n is the order of the graph). We compute the average size of these maximal independent sets, and observe how this value changes as we vary the cutoff

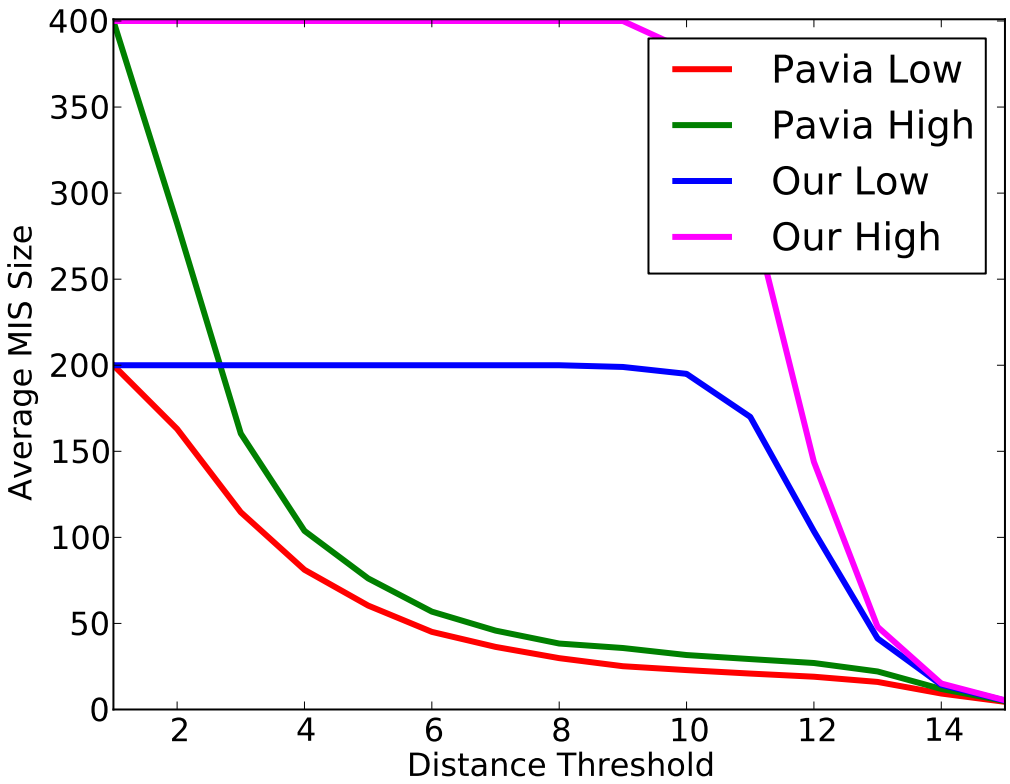


Figure 2.7: *EAR: De Novo Diversity*

The diversity of the de novo high and low predicted binding affinity peptides. The sequence diversity among our peptides is significantly higher — almost all of them differ in at least 9 of their 15 possible positions. The peptides submitted for experimental validation by team Pavia, however, are highly redundant in terms of their sequence.

parameter x. At a given cutoff level x, the larger the average size of the maximal independent sets, the more independent peptides exist (i.e. peptides having x or more differences). According to this metric, our *de novo* predictions exhibit substantially more diversity than the predictions of team Pavia (Figure 2.7). In particular, note that until a distance cutoff of 9, almost all of our peptides (in both the high and low affinity sets) belong to a single independent set which spans the entire graph. The *de novo* predictions of the Pavia team, however, share a great deal of sequence similarity, representing a very dense sampling of the sequence space near only a few particular points.

2.9 Conclusion and Future Work

The ensemble classification scheme we have presented exhibits state-of-the-art performance in the task of classifying the binding affinity of peptides to human immunoglobulin fractions. Further, we are able to combine our classifier with a seeded random walk in sequence space to predict *de novo* high and low binding affinity peptides with high accuracy. While the existing classifier performs well, there are promising directions for future work. Currently, the single classifier with the poorest performance is the one based on structural features. It is likely the case that this performance deficit is due more to particular implementation details than to the nature of the features being considered. For example, ZDock considers only rigid docking of the peptides to immunoglobulin. This means that any error in the hypothesized structure which decreases the peptides shape complementarity with the immunoglobulin structure, will result in lower docking scores. However, our confidence in the hypothesized peptide structures is not particularly high, and we should admit conformational changes in these structures if they lead to better docking. Thus, the structural features may change significantly if we consider a non-rigid docking procedure, where conformational changes in the paratope, epitope, or both are allowed. It is actually quite surprising that, using only a single IGg1 model and considering only perfectly rigid docking, the structure based classifier obtained such respectable performance. This indicates that improving the computation of the structural features is a promising way to increase prediction accuracy.

Another venue for future improvement is to further expand the diversity of the ensemble classifier. Our method is designed so that it is simple to incorporate the predictions of new classifiers into the ensemble's predictions. It will be interesting to see if the performance of our classifier can be improved simply by considering an even larger and more diverse set of features and classifiers.

Chapter 3

Network Alignment

This chapter is based on the paper "Global Network Alignment Using Multiscale Spectral Signatures", written in collaboration with C. Kingsford, that is currently under review.

3.1 Introduction and Related Work

In this chapter, we explore methods for the comparative analysis of biological networks by means of network alignment. Algorithms for the accurate and efficient alignment and comparison of biological sequences were among the first major successes of computational biology, and these methods are in wide-spread use today. Yet, genomic sequence provides only a partial view of the biological system that it encodes. For example, it has been shown that, across species, the protein with the most similar sequence does not always play the same functional role [112], and that topological information can be used to disambiguate sequence-similar proteins and determine functional orthology [6].

While the technologies and experimental techniques used to obtain biological networks are not yet as prevalent or cost-effective as those used to obtain biological sequences, a number of high-throughput techniques such as yeast two- hybrid (Y2H) screening [41] and tandem affinity purification mass spectrometry (TAP-MS) [50], and ChIP-seq [59] exist. As a result, the available biological network data has been steadily increasing. For example, the number of interactions cataloged in `BioGRID` [15], a popular repository for biological interaction datasets, has increased from 157,123 in 2006 to 392,218 in June 2011. As new

techniques are developed, and the cost of experiments continues to fall, we expect an even more rapid growth of biological network data.

A solution to the global network alignment problem is an injective mapping f from the nodes of one network $G = (V_G, E_G)$ into another network $H = (V_H, E_H)$ such that the structure of G is well preserved. This global mapping allows us to measure the similarity between proteins in G and those in H in terms of shared interaction patterns. By exposing large subnetworks with shared interactions patterns across species, a network alignment allows us to transfer protein function annotations from one organism to another using more information than can be captured by sequence alone. For example, it has been shown that, across species, the protein with the most similar sequence does not always play the same functional role [112], and that topological information can be used to disambiguate sequence-similar proteins and determine functional orthology [6]. Additionally, by looking at the magnitude of structure conserved between G and H, we can measure the similarity between these networks and infer phylogenetic relationships between the corresponding species [69]. We can also hypothesize the existence of unobserved interactions (missing edges), remove noise from error-prone, high-throughput experiments, and track the evolution of pathways.

Our approach to the global network alignment problem uses a novel measure of topological node similarity that is based on multiscale spectral signatures. These signatures are composed from the spectra of the normalized Laplacian for subgraphs of varying sizes centered around a node. We combine this highly specific yet robust node signature with a seed-and-extend alignment strategy that explicitly enforces the proximity of aligned neighborhoods. The initial alignment is improved by means of a local search procedure. We implement these ideas in our network alignment software, GHOST, which exceeds state-of-the-art accuracy under several different metrics of alignment quality.

There has been significant interest in the network alignment problem, and previous work can naturally be divided into three main categories: approaches to local network

alignment, approaches to network querying, and approaches to global network alignment. Because we are introducing a system for global network alignment, we restrict our discussion to the relevant work in this area.

Singh et al. [115] introduced IsoRank that uses a recursively defined measure of topological similarity between nodes in different networks. They proposed an eigenvector-based formulation to discover a high-scoring matching. Liao et al. [80] developed IsoRankN, which extends IsoRank with a new algorithm for multiple network alignment based on spectral clustering. Chindelevitch et al. [22] use a local search heuristic, which they call PISWAP, to iteratively improve an initial alignment that is based solely on sequence data. The Graemlin aligner was originally developed by Flannick et al. [43] to discover evolutionarily conserved modules across multiple biological networks. Later, it was extended [44] to perform global multiple network alignment (Graemlin2). However, this approach relies on a variety of additional information about the networks being aligned, including phylogenetic information. Further, sample alignments are required for the parameter learning phase of Graemlin2.

The GRAAL family of programs, like IsoRank, perform unconstrained and global pairwise alignments of biological networks. Kuchaiev et al. [68] originally introduced GRAAL, which measures the topological similarity of nodes in different networks based on the distance between their graphlet degree signatures and aligns the networks using a seed-and-extend strategy. Milenkoviç et al. [87] then introduced H-GRAAL, which relies on the same graphlet degree signatures used by GRAAL but performs the alignment of the networks by solving the linear assignment problem via the Hungarian algorithm [71]. Finally, Kuchaiev and Pržulj [69] introduced MI-GRAAL, which combines these two alignment strategies. It relies on a seed-and-extend alignment procedure but uses the Hungarian algorithm to compute the assignment between local neighborhoods of the two graphs that maximizes the sum of their linear scoring function. MI-GRAAL also incorporates a number of other topological metrics, in addition to the graphlet degree signatures, to help

quantify the topological similarity between nodes.

Recently, multiple attempts have been made to tackle the biological network alignment problem using graph matching. Klau [64] introduced a non-linear integer program to maximize a structural matching score between two given networks and then showed how the problem can be linearized, yielding an integer linear program (ILP), and finally suggested a Lagrangian relaxation approach to the ILP. Later, El-Kebir et al. [37] extended this approach and improved the upper and lower bounds of the relaxation, implementing their approach in the Natalie 2.0 software package. The HopeMap approach of Tian and Samatova [124] used an algorithm that iteratively merges conserved connected components. Zaslavskiy et al. [133] explore the use of a number of graph matching methods, particularly the PATH and GA methods, which attempt to find a permutation matrix between vertices of the networks being aligned that maximizes a score that is a combination of the structural similarity and conserved interactions of the matched vertices. This optimization is NP-hard and they must rely on a relaxation to discover an approximate solution. Many similar graph-matching approaches have been applied to shape matching in computer graphics and computer vision [126, 94, 34]. All of these matching-based approaches require a large number of constraints to be placed on the set of potential alignments, usually in the form of homology information between the proteins of the networks being aligned, in order to run in a reasonable amount of time. These constraints vastly reduce the search space and help bring these computationally burdensome methods into the realm of tractability. However, the hard constraints introduced by the homology information can have a negative effect on the ability of these methods to discover truly novel functional homologs between highly divergent species. In a way, these methods focus more on discovering conserved patterns of interactions between proteins that are already posited to be homologous, rather than on performing a truly *de novo* and unconstrained alignment of biological networks that is merely guided by homology information. GHOST takes a hybrid approach, where the initial alignment can be constrained by some aspect of the scoring function, but the local

search procedure allows exploration into regions of the alignment space that do not adhere to the original constraints.

Our network aligner, GHOST, combines a novel spectral signature to measure topological similarity with a seed-and-extend alignment procedure, and an iterative local search step. In sections 3.3.1 and 3.3.2, we show that GHOST performs much better than current aligners at the network self-alignment task. In section 3.3.3, we compare an ensemble of alignments produced by different aligners as we vary their parameter settings to trade off between the topological and biological quality of the alignments they produce. GHOST consistently outperforms the other aligners in these tests and is able to produce alignments higher overall quality. This improved quality will be useful for more accurate comparative systems biology.

3.2 Methods

3.2.1 Measuring Alignment Quality

It is challenging to state the global network alignment problem formally and precisely because a "good" alignment balances two, often disparate, goals. A high-quality global alignment between two biological networks should reveal shared topological structure between the networks being aligned, while also respecting the strong evidence for homology revealed via sequence analysis.

Neither of these goals, however, should act as hard constraints when aligning two networks, and a high-quality global network alignment should strive to satisfy both the topological and sequence requirements. This naturally leads to two distinct measures for the quality of network alignments; one quantifies *topological quality*, the degree of shared structure revealed between the two networks, and the other quantifies *biological quality*, how well the alignment respects the biological and functional similarities of the proteins.

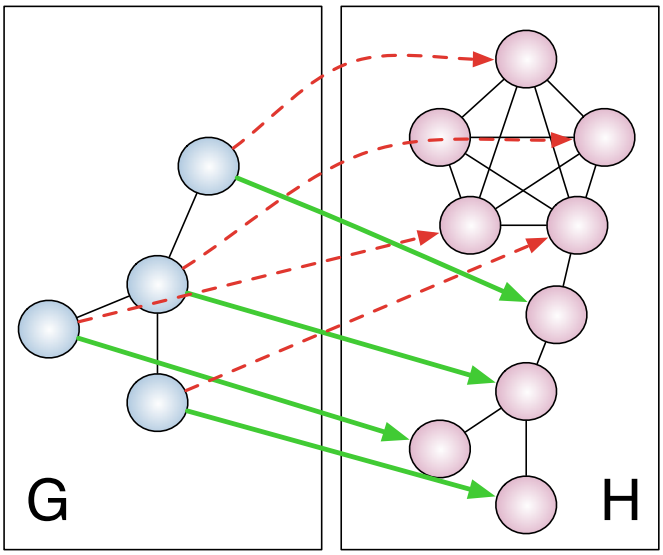


Figure 3.1: *ALIGNMENT: Induced Conserved Substructure vs. Edge Correctness*

The mapping from G to H given by the solid green arrows can be considered a better alignment than that given by the dashed red arrows, despite the fact that they both have the same edge correctness.

Topological Quality. A topological quality metric should measure the degree to which the structure of G is preserved, under f (the computed injective mapping from V_G to V_H), when mapped into H. For example, we expect that an alignment of high topological quality will map interacting proteins in G to interacting proteins in H. The most common measure of topological quality is edge correctness, which measures the percentage of edges from G that are aligned to edges in H. Let $G[V]$ be the induced subgraph of G on the vertex set V, $f(V) = \{f(v) \mid v \in V\}$, $f(E) = \{(f(u), f(v)) \mid (u, v) \in E\}$ and $f(G) = (f(V_G), f(E_G))$. Then, the edge correctness (EC) is defined as

$$\mathrm{EC}(G, H, f) = \frac{\left|f(E_G) \cap E_H\right|}{|E_G|} . \tag{3.1}$$

Despite its prevalence, edge correctness fails to differentiate alignments that one might intuitively consider to be of different topological quality (see Section 3.2.1) because it accounts only for the number of edges from G that are mapped into H and incorporates no

notion of the similarity between G and the induced subgraph of $f(G)$.

We introduce a new measure of topological quality, the induced conserved structure (ICS) score, that uses a more discriminative notion of conserved structure than EC. We define the ICS score between G and H induced by the alignment f as

$$\text{ICS}(G, H, f) = \frac{\left|f(E_G) \cap E_H\right|}{\left|E_{H[f(V_G)]}\right|}. \tag{3.2}$$

Notice that, for the example given in Section 3.2.1, while the edge EC score of both the green and red mappings is 1, the ICS successfully distinguishes the two cases. In particular, the ICS of the green mapping remains 1, while the ICS of the red mapping becomes 0.4, agreeing with the intuition that the green mapping conserves more structure than does the red mapping. Also, note that the ICS score is 1 if and only if G is isomorphic to $H[f(V_G)]$. Thus, alignments that map subgraphs of G into denser subgraphs of H, where there are potentially many more mappings, will be punished under the ICS score while they will not be punished under the standard edge correctness score. Note that optimizing EC and ICS directly is, in general, $\mathcal{NP}$-hard. This can be shown by reduction from `CLIQUE`, since when G is a clique, both EC and ICS are 1 if and only if H contains a clique of order $|V_G|$.

Biological Quality. Given an alignment, $f : G \to H$, a measure of biological quality should evaluate the similarity of p and $f(p)$ in terms of biological function. The most common measure of similarity computes the enrichment of shared Gene Ontology [123] (GO) annotations between the mapped proteins. The greater the enrichment, the higher the biological quality of the alignment. In most previous work [115, 69], two GO annotations are considered the same only if they are identical.

This common metric has two main disadvantages. First, many GO terms are assigned largely based on sequence homology to proteins with verified annotations, which strongly biases the results in favor of alignments that ignore topology completely and align proteins based solely on sequence similarity. Additionally, measuring the functional enrichment be-

tween proteins by considering only exact overlap between their associated GO annotations ignores the hierarchical structure of annotation similarity encoded in the ontology. Only recently has the literature on network alignment [37] started to use methods [58] that use the hierarchical structure of GO. Most previous work [115, 80, 68, 69] considers only the exact overlap metric, and it is potentially misleading.

While the issue that annotations often come from sequence remains a concern, we address the second concern by using an additional metric of protein function similarity that takes into account the relationships between annotations encoded by the GO hierarchy. Pesquita et al. [104] recently compared a number of methods for computing protein similarities based on GO annotations. They find that one of the best performing methods computes the similarity of GO terms using the Resnik ontological similarity measure and combines annotation similarities using the best-match average strategy to obtain a functional similarity measure on proteins. We adopted an implementation of this measure provided in the csbl.go R-project package [95]. We denote this similarity measure by $s_a(p_1, p_2)$, where a is an aspect — Biological Process (BP), Molecular Function (MF) and Cellular Component (CC) — of GO. The similarity measure between networks G and H induced by the alignment f under the GO aspect a is given by $s_a(G, H, f) = \frac{1}{|V_G|} \sum_{p \in V_G} s_a(p, f(p))$.

3.2.2 Spectral Signature

One of the primary contributions of our work is the introduction of a novel topological signature for nodes in a network. We use these signatures to guide our network alignment and to provide a measure of the similarity, or topological context, of nodes within their respective networks. Useful topological signatures should be precise, robust to topological variation, and fast to compute. Spectral graph theory provides tools that allow us to develop a signature having all of these properties.

There is a well-studied and strong relationship between the structure of a graph and the spectrum of its adjacency matrix and other related matrices. For example, isomorphic

graphs are necessarily cospectral, though cospectral graphs are not necessarily isomorphic. However, simple comparison of spectra provide a powerful isomorphism filter in practice. In fact, using the eigenvalues and associated eigenvectors of graphs, Babai et al. [5] developed an algorithm for graph isomorphism that is polynomial in the algebraic multiplicity of the graph.

The spectra of graphs are also robust to topological variations. Wilson and Zhu [131] show that the distance between the spectra of the normalized Laplacian of graphs correlates well, at least for small perturbations, with the true edit distance between the graphs. Further, such spectra are efficient to compute. It takes $O(n^3)$ time to compute the spectrum for dense graphs with n vertices. However, for sparse graphs, like the biological graphs in which we are interested, faster algorithms exist [98]. For any subgraph, the computation of the spectrum is an independent operation and can be parallelized.

Our vertex signature is based on the spectrum of the normalized Laplacian for subgraphs of various radii centered around a vertex. Consider a graph $G = (V_G, E_G)$ and vertex v. We denote by G_v^k the induced subgraph on all nodes whose unweighted shortest-path length from v is less than or equal to k. We denote by W_v^k the adjacency matrix of G_v^k. In all experiments performed in this paper, we use the unweighted adjacency matrix, though using a weighted adjacency matrix is also possible. Finally, let the matrix D_v^k be given by

$$D_v^k[i,j] = \begin{cases} \sum_{\ell=1}^{|V_G|} W_v^k[i,\ell] & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

Then, the normalized Laplacian of G_v^k is $\mathcal{L}_v^k = (D_v^k)^{\frac{1}{2}}(I - W_v^k)(D_v^k)^{\frac{1}{2}}$, where I is the appropriately-sized identity matrix. The eigendecomposition of this normalized Laplacian yields $\mathcal{L}_v^k V = \Lambda V$, where the sizes of V and Λ are the same as that of $\mathcal{L}_v^k$, but Λ is a diagonal matrix. We denote spectrum of $\mathcal{L}_v^k$ by $\sigma(\mathcal{L}_v^k)$, which is simply the entries along the main diagonal of Λ.

Many properties of $\sigma(\mathcal{L}_v^k)$ make it an enticing candidate for a vertex signature. Since the

$\mathcal{L}_v^k$ is a positive, symmetric, semi-definite matrix with real entries, $\sigma(\mathcal{L}_v^k)$ consists entirely of non-negative real numbers. Further, the entries of $\sigma(\mathcal{L}_v^k)$ are bounded below by 0 and above by 2. Finally, many topological properties of a graph, such as the number of spanning trees, the Cheeger constant, the distribution of path lengths [23], and the frequency of motifs [109] are known to be related to the spectrum of its Laplacian.

However, for different vertices, the size of their k-hop neighborhoods will vary and thus the length of their spectra will be different and so the spectra cannot be directly compared. To overcome this difficulty, we consider the densities of the spectra rather than the spectra themselves. The spectral density simply measures how eigenvalues are distributed over their potential range ($[0, 2]$ in the case of the normalized Laplacian). This yields a commensurate signature that is independent of the order of the graph, but is nonetheless effective in measuring the structural similarity of graphs [7]. For each G_v^k, which we will use this spectral density, denoted by $\mathcal{S}_v^k$, as a signature.

To compare the topological context of vertices at different scales, we simply consider the induced subgraphs for a range of different radii centered about v (i.e. $G_v^1, G_v^2, \ldots, G_v^k$). This leads, in turn, to a set of different spectra, and subsequently, different signatures. However, since the radii have the same meaning across different vertices and graphs (it is just the diameter of the neighborhood), the corresponding signatures can be compared directly and independently of the signatures at other radii. This leads to a simple scheme for comparing the topological contexts of two vertices at multiple scales using our signature. Given two graphs, $G = (V_G, E_G)$ and $H = (V_H, E_H)$, with $u \in V_G$ and $v \in V_H$, and a sequence of radii $R = [1, 2, \ldots, k]$ (for all experiments performed in this paper, we set $k = 4$), we compute the distance between the signatures of u and v for this sequence of radii as

$$D_{\text{topo}}(\mathcal{S}_u^R, \mathcal{S}_v^R) = \sum_{r \in R} d(\mathcal{S}_u^r, \mathcal{S}_v^r), \tag{3.4}$$

where $d(\cdot, \cdot)$ can be any desired distance between the two signatures. We use $d = d_{\text{struct}}$, the structural distance as defined by [7]. The structural distance is a symmetric information

theoretic distance defined on the smoothed spectral densities of two graphs. Specifically, the structural distance between signatures, $\mathcal{S}_u^i$ and $\mathcal{S}_v^i$, for a particular i, is given by:

$$d_{\text{struct}}(\mathcal{S}_u^i, \mathcal{S}_v^i) = JS(\mathcal{N}(0, \eta^2) \star \mathcal{S}_u^i, \mathcal{N}(0, \eta^2) \star \mathcal{S}_v^i), \tag{3.5}$$

where $\mathcal{N}(0, \eta^2)$ is the normal distribution with mean 0 and standard deviation η (we used a value of $\eta = 0.01$ as suggested in [7]), $\star$ is the convolution operator, and JS is the Jensen-Shannon divergence. In the case that the maximum radius of the subgraph centered around a node u is some $k' < k$, then we define $\mathcal{S}_u^r = \mathcal{S}_u^{k'}, \forall k' < r < k$. The motivation behind this distance is to measure structural similarity of a pair of vertices by comparing the spectral distributions of their surrounding subgraphs. By considering the subgraphs at multiple scales, we first compare the most immediate and then the broader-scale topological contexts of the two vertices. Both empirical results [7] and intuition lead us to believe that comparing the spectral distributions of graphs, like comparing their spectra directly [131], is an effective way to measure their topological similarity.

In a manner similar to IsoRank [115], we can incorporate sequence information into our distance measure between two proteins u and v by using a simple combination of the topological distance — $D_{\text{topo}}(\mathcal{S}_u^R, \mathcal{S}_v^R)$ as defined in **??** — and a sequence distance, $D_{\text{seq}}(u, v)$, such as the symmetrized BLAST E-value. The total distance measure is a linear combination of the topological and sequence distance, parameterized by some weight α and is given by

$$D_\alpha(u, v) = \alpha D_{\text{topo}}(\mathcal{S}_u^R, \mathcal{S}_v^R) + (1.0 - \alpha) D_{\text{seq}}(u, v). \tag{3.6}$$

If no user-suggested α is provided, GHOST automatically computes α by scaling the sequence and topological distances so that the $|V_G|^{\text{th}}$ smallest sequence and topological distances match.

3.2.3 Alignment Procedure

GHOST aligns networks using a two-phase approach. Much like the strategy used in the sequence alignment tool `BLAST` [3], GHOST's initial phase employs a seed-and-extend strategy that seeds regions of an alignment with high scoring pairs of nodes from the different networks and then extends the alignments around the neighborhoods of these two nodes. The neighborhoods are matched by computing an approximate solution to the quadratic assignment problem (QAP). This procedure executes in rounds until all nodes from the smaller of the two networks have been aligned with some node from the larger network. GHOST's second phase uses a local search strategy to explore regions of the solution space around the initial alignment for a potentially better solution.

The algorithm is given formally in Algorithms 1 and 2. First, an alignment is seeded with a high-scoring match $\hat{M}^0 = (\hat{M}^0_G, \hat{M}^0_H)$. This is a pair of vertices between which the specified D_α (**??**) is minimal. Then, we consider all pairwise matches between the 1-hop neighborhoods of these two vertices, $M = \left[(i,j) \mid i \in \mathcal{N}(\hat{M}^0_G)), j \in \mathcal{N}(\hat{M}^0_H))\right]$, and form a quadratic assignment matrix Q given by:

$$Q[(g_1, h_1, g_2, h_2)] = \begin{cases} 1 - D_\alpha(g_1, h_1) & \text{if } (g_1, h_1) = (g_2, h_2) \\ C(g_1, h_1, g_2, h_h) & \text{otherwise.} \end{cases}$$

The arguments to Q, (g_1, h_1) and (g_2, h_2), are matches from M, where g_1 and g_2 are vertices in G and h_1 and h_2 are the vertices in H with which they are matched. The pairwise consistency between potential matches (g_1, h_1) and (g_2, h_2) is given by

$$C(g_1, h_1, g_2, h_2) = \exp(\frac{-\left|D_{\text{topo}}(g_1, h_1) - D_{\text{topo}}(g_2, h_2)\right|}{D_{\text{topo}}(g_1, h_1) + D_{\text{topo}}(g_2, h_2)}).$$

We approximate the solution to the quadratic assignment problem by finding the leading eigenvector of Q and binarizing this vector to select matches that adhere to the

matching constraints (further details on this QAP approximation algorithm can be found in Leordeanu and Hebert [74]). The solution to the QAP assigns each protein from the smaller of the two neighborhoods to exactly one protein in the larger neighborhood. This mapping is used to align the currently unmapped proteins in these neighborhoods, and the matches are inserted into a priority queue as potential seeds by which to further extend the alignment between these local neighborhoods. However, we only accept mappings that align proteins with a sequence distance less than a certain (user defined) value β. This is because a seed-and-extend approach is implicitly biased in favor of extending topological alignments, and may otherwise match proteins with very little evidence of sequence homology, simply because they reside in the neighborhoods of already aligned proteins. Biologically, it is more plausible that a pair of proteins with very low sequence similarity happen to be adjacent to a pair of currently aligned proteins by chance, or as the result of spurious edges in the measured networks, than it is that they are truly functional homologs.

We continue extending the alignment in this manner by picking a new pair of center nodes in the now-aligned topological neighborhoods of the original seed nodes, and aligning their 1-hop neighborhoods using the QAP procedure. This process continues until no further extension of the alignment between the current neighborhoods is possible. Then, the next seed pair, $\hat{M}^1$, is chosen from among the unaligned nodes and the same procedure is applied to extend the alignment around this seed. This process continues until all nodes from V_G (assumed, w.l.o.g., to be smaller than V_H) have been aligned.

There are two parameters that govern this alignment phase. First, α determines the relative weight of the sequence and topological distances when performing the seed-and-extend procedure (**??**). Second, β acts as a hard constraint on sequence similarity of aligned pairs: no pair, (u, v) of proteins will be aligned if $D_{\text{seq}}(u, v) > \beta$. This ensures that, when extending the alignments between local neighborhoods, no pair of proteins with sequences too divergent are aligned simply because the alignment can be extended by aligning them.

Algorithm 1: ALIGNMENT: Seed & Extend Algorithm

input : Networks G and H
output: Alignment f

$P \leftarrow \{\}$; *// Initialize (min) heap*
$f \leftarrow \{\}$; *// Initialize empty alignment*
foreach $(x, y) \in V_G \times V_P$ **do**
  push$(P,\ (x, y, D_\alpha(x, y)))$;
while P *is not empty* **do**
  $(t_G, t_H) \leftarrow$ **pop**(P);
  if t_G *and* t_H *are not already aligned* **then**
    GreedyQAPExtend$(G, P, (t_G, t_H), f)$;
return *LocalImprove(f)*;
SeedAndExtend

Algorithm 2: ALIGNMENT: QAP Extend Algorithm

input : Networks G and H, seed pair (u_G, u_H), current alignment f
side-effect: f extended with some neighbors of u_G, u_H

$P \leftarrow \{(u_G, u_H)\}$; *// Initialize (max) heap*
while P *is not empty* **do**
  $(t_G, t_H) \leftarrow$ **pop**(P);
  if t_G *and* t_H *are not already aligned* **then**
    // Align neighborhoods using the approximate
    // quadratic assignment procedure, QA
    $s \leftarrow QA(\mathcal{N}(t_G), \mathcal{N}(t_H))$;
    foreach $(x, y) \in s \setminus (f(G) \times f(H))$ **do**
      if $D_{seq} \leq \beta$ **then**
        push$(P,\ (x, y, D_\alpha(x, y)))$;
        $f(x) \leftarrow y$;
GreedyQAPExtend

Once we have computed an initial alignment using the seed-and-extend procedure, we attempt to improve this alignment using a local search. The moves of the local search procedure are similar to those employed by PISWAP [22], but the evaluation strategy and application of rules is different. Given an alignment, f, we seek f' similar to f that is superior. Consider a pair of aligned proteins, $u \in G$ and $f(u) = w \in H$, and a third vertex $v \neq w \in H$. It is possible that we may improve the quality of our alignment by realigning u so that $f'(u) = v$ if the topological and or biological quality is improved by performing

this realignment. When realigning u, there are two cases to consider. Either v is unaligned, in which case we assign $f'(u) = v$, or v is aligned by f, in which case aligning u to v requires realigning $u' = f^{-1}(v)$. In this case, we consider swapping the aligned protein pairs, so that $f'(u) = v$ and $f'(u') = w$. In either case, we will call this realignment a move from (u, w), denoted by $m = (u, w) \rightarrow (u, v)$. Each move can be given a score, $S(m) = (s_0^m, s_1^m, s_2^m)$ where

$$
\begin{aligned}
s_0^m &= EC(G, H, f') - EC(G, H, f) \\
s_1^m &= D_{\text{seq}}(u, w) - D_{\text{seq}}(u, v) \\
s_2^m &= \begin{cases} D_{\text{seq}}(u', v) - D_{\text{seq}}(u', w) & \text{if } f^{-1}(v) = u' \\ 0 & \text{if } v \notin \text{im}(f). \end{cases}
\end{aligned}
$$

For each mapping, (u, w), in the current alignment, the local search procedure scores the potential moves from (u, w), and performs the highest scoring feasible move. The scores are ordered first by s_0^m, then s_1^m, and finally s_2^m. Any remaining ties are broken arbitrarily. We call a move feasible if $s_0^m > 0$, $s_1^m \geq 0$ and either $s_2^m \geq 0$ or we have decided to allow a move from (u, w) that potentially lowers the sequence score of the pair that is displaced by the move. The purpose of allowing such a move from (u, v) is that it may allow us to escape a local minimum of the alignment space.

During the local search procedure, we allow some number of exceptions to the hard constraint given by β. We define a parameter $b \in \mathbb{R}$ that governs the probability that we will allow a move that improves the topology and sequence scores of one pair while hurting the sequence score of the pair it displaces. The higher this value, the more likely GHOST will be to accept local moves that increase the topological quality of the alignment at the expense of realigning a pair of proteins with lower sequence similarity than the original pair. We distribute b across local search iterations so that we initially allow many such moves, but allow far fewer in later iterations. In particular, during iteration i, we compute

$b_i = b\frac{\exp(-i)}{\mathcal{Z}}$, where $\mathcal{Z} = \sum_{i=1}^{L} \exp(-i)$ is a partition function that normalizes the per-iteration weights. Within iteration i, we consider each mapped pair of the current alignment in turn and draw a number $p \sim U[0, 1]$. If $p \leq b_i$, then we will allow a move that results in a potentially lower sequence score when realigning this mapped pair; otherwise, such moves will not be considered. The practical effect of choosing a larger b is to reduce the importance of sequence similarity in the alignment.

3.2.4 Network Data

We performed an alignment of the high-confidence protein interaction networks of *Campylobacter jejuni* (*C. jejuni*) and *Escherichia Coli* (*E. coli*). Both of these bacterial species are well-studied model organisms. In order to draw the most appropriate comparisons to MIGRAAL, we use the same versions of the interaction networks that were used by Kuchaiev and Pržulj [69]. Thus, we used *E. coli* network composed of interactions from the data of Peregrín-Alvarez et al. [102], consisting of 1941 proteins among which there are 3989 interactions. We consider the *C. jejuni* network which consists of the high-confidence interaction from the data of Parrish et al. [99], containing of 2988 interactions among 1111 proteins.

We also explored the ability of GHOST to align the protein interaction networks of distant eukaryotes by performing an alignment of the protein interaction networks of *Arabidopsis thaliana* and *Drosophila melanogaster*. We obtained the interactions for these networks from the HitPredict website [101]. HitPredict places interaction data for each species into three categories: high-confidence small-scale interactions (HCSS), high-confidence high-throughput interactions (HCHT), and low-confidence high-throughput interactions (LCHT). The high-confidence small-scale interactions are identified directly in small-scale experiments considering fewer than 100 interactions each. The HCHT interactions are those interactions identified in high-throughput experiments with a likelihood ratio greater than 1, or predicted from protein complex data. The low-confidence high-throughput interactions are those having a likelihood ratio less than 1. In our experiments, we considered

only the high-confidence interactions — the union of those interactions in the HCSS and HCHT sets. This resulted in a network for *A. thaliana* having 2082 proteins and 4145 interactions. The *D. melanogaster* network consisted of 7615 interactions among 3792 different proteins.

3.2.5 Comparison with Other Aligners

To investigate the quality of the solutions produced by the different aligners we consider, we explore how they trade off between topological and biological quality at different points in their parameter spaces. The alignments are compared using the novel measures of the topological and biological quality introduced in Section 3.2.1. To calculate GO similarities, we rely on the set of GO annotations for each protein retrieved from the European Bioinformatics Institute website in June of 2011, and the gene ontology retrieved on Nov. 10, 2011. When producing alignments using MI-GRAAL, we included graphlet degree signatures, clustering coefficients and sequence similarity scores — the topological features that Kuchaiev and Pržulj [69] found to lead to the highest scoring and most stable alignments. MI-GRAAL determines the value of α — the parameter that trades off between functional and sequence similarity — internally, and so no α value was provided. For IsoRank and Natalie 2.0, we varied α between 0 and 1 in increments of 0.1. The rest of Natalie 2.0's parameters were left at their default values. For GHOST, α was determined automatically using the procedure specified in section 3.2.2, 10 iterations of the local improvement procedure were performed, β was set to 10, and b was varied over $\{0\} \cup \{2^j\}_{j=-2}^{7}$.

3.3 Results

We evaluated the performance of GHOST in several different scenarios, and compare against IsoRank, GRAAL, MI-GRAAL, H-GRAAL and Natalie 2.0. First, we perform two tests that have been used in the past to assess topological alignment quality. These tests, self-alignment and self-alignment with noise, are instructive because the correct node mapping is known when aligning a network to itself. This allows us to measure accuracy

in a way that is not possible when comparing networks from different species. The results of these experiments provide important evidence about the robustness and specificity of different topological signatures and the ability of different global alignment approaches to align two networks based solely on topological information. In sections 3.3.1 and 3.3.2, we are interested primarily in the utility of the local topological signatures and the basic alignment procedures. Thus, we do not perform the local search phase of GHOST described above. Further, because we cannot use biological sequence information to constrain the space of alignments, we do not consider the performance of the graph-matching approach (i.e. Natalie 2.0) on this task.

Subsequently, we consider the alignment between high-confidence protein-protein interaction networks of a pair of bacteria and a pair of eukaryotes. Here, we use the new metrics described in Section 3.2.1 to measure the topological and biological quality of our alignments. Considering unconstrained alignments using graph-matching approaches either exhausted the memory of our machines [37], or failed to finish aligning the networks within 16 hours [133]. Thus, when comparing against graph matching approaches, we use Natalie 2.0 [37] to produce a constrained alignment.

3.3.1 Self-Alignment

For networks with many similar sub-regions, even a self-alignment in the absence of noise can be difficult. To demonstrate this difficulty, we consider a self-alignment of the largest connected component of a high-confidence network of the bacterium *Mesorhizobium loti* (*M. loti*). This network was obtained from the interactions reported in the study by Shimoda et al. [113] and consists of 3006 interactions among 1655 proteins. The alignment produced by GHOST is an automorphism of the graph, with an edge correctness of 100% and a node correctness (the fraction of nodes that were aligned with themselves) of 79%. The alignment produced by IsoRank had an edge correctness of 76% and a node correctness of 53%, while the alignment produced by MI-GRAAL had a edge correctness of 38% and node correctness of only 0.3%. Because MI-GRAAL is probabilistic in nature, we performed this

alignment multiple times, using a wide variety and combination of the topological features suggested in Kuchaiev and Pržulj [69], to ensure that this failure of self-alignment was not coincidental. None of these subsequent MI-GRAAL alignments differed in topological quality — either node or edge correctness — by more than a fraction of a percent. IsoRank produced an alignment of significantly higher topological quality than the one discovered by MI-GRAAL; this is different from what we see in the rest of the tests described below.

Despite the fact that its node correctness is only 79%, GHOST's alignment is structurally perfect. Without more information beyond what is provided by the network itself, one cannot hope to obtain a better alignment than the one produced by GHOST.

3.3.2 Self-Alignment Under Noise

We also re-performed the experiment originally carried out by Milenkoviç et al. [87], where progressively noisier variants of the *S. cerevisiae* interaction network are aligned to the high-confidence network of Collins et al. [26]. The higher noise networks are created by starting with the highest confidence network, and then adding interactions (constrained to the original, high-confidence set) in decreasing order of experimental confidence. Since this is again a self-alignment, and sequence information would allow the almost perfect identification of the correspondences between nodes, we consider a purely topological alignment (i.e. $\alpha = 1.0$ and $\beta = \infty$). We explore how the fraction of correctly aligned nodes changes as larger quantities of noisy interactions are added to the high-confidence network (Figure 3.2).

In the case with the fewest noisy interactions, most of the programs achieve similar performance. However, as the number of noisy interactions increases, GHOST outperforms all of the other approaches by an increasing margin. By the time 20% of the noisy interactions have been included in the network, the node correctness of GHOST is more than twice that of the next-best-performing aligner, while the edge correctness is over 30% greater. There also seems to be a substantial gap between IsoRank and the rest of the alignment

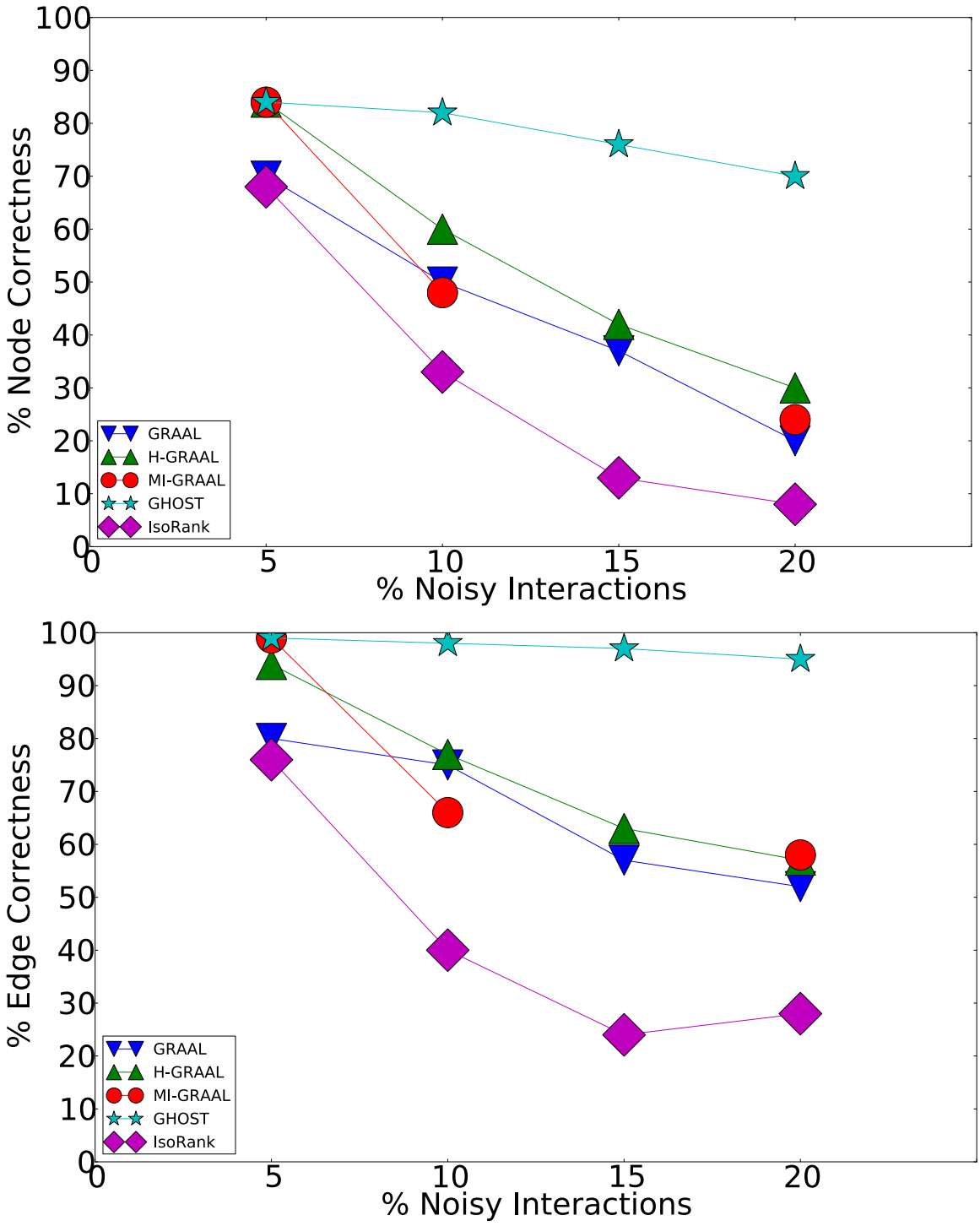


Figure 3.2: *ALIGNMENT: S. cerevisiae Self Alignment*

Performance of various aligners on a noisy yeast PPI under the node (top) and edge (bottom) correctness metrics. Note: In the 15% noise case, the performance numbers of MI-GRAAL are not given because it failed to run to completion.

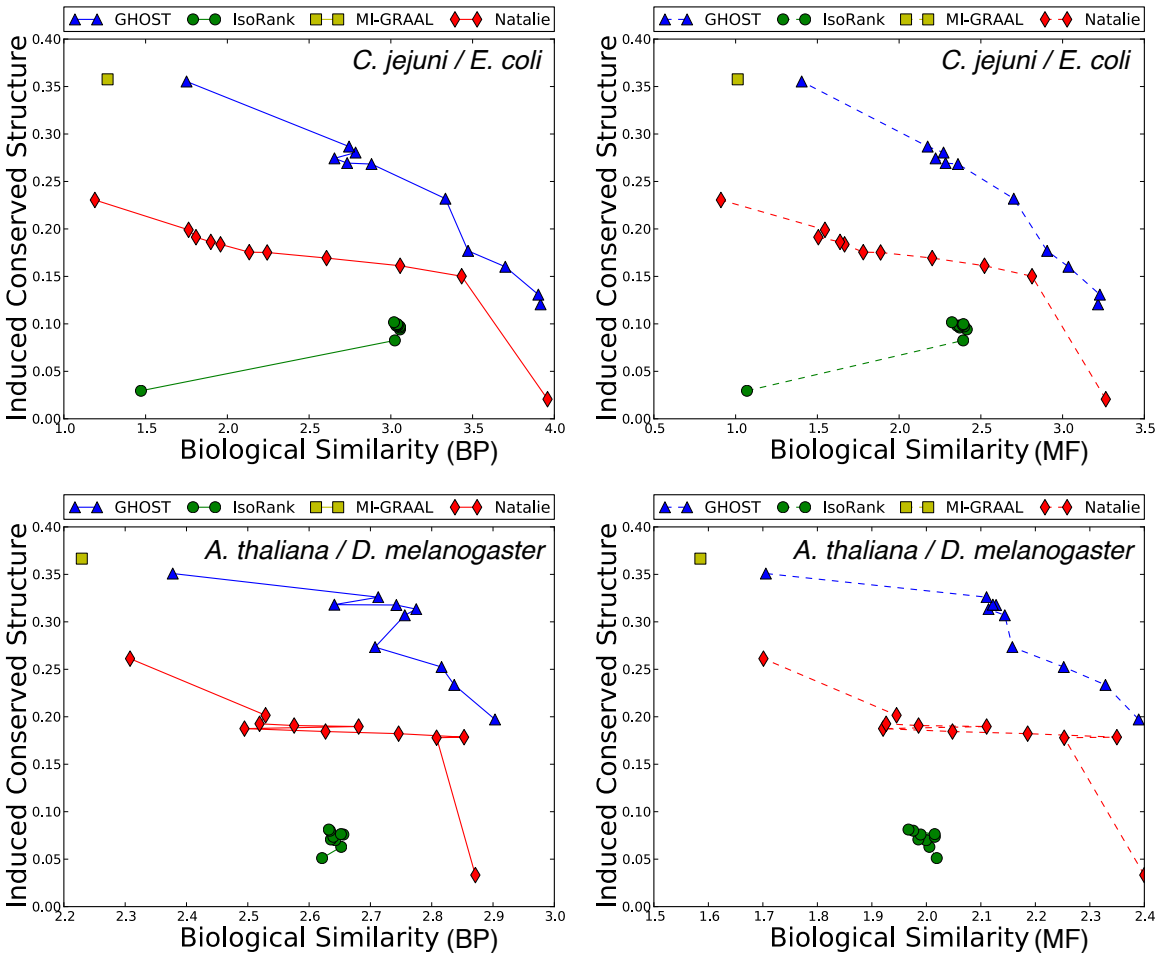


Figure 3.3: *ALIGNMENT: C. jejuni vs. E. coli & A. thaliana vs. D. melanogaster Alignment Qualities*

Under both biological process (BP) and molecular function (MF) GO aspects and both alignments, we observe a consistent trend in the quality of the solutions produced by the different aligners. IsoRank produces alignments of reasonable biological but poor topological quality, while MI-GRAAL exhibits the opposite behavior (i.e. high topological but poor biological quality). Natalie 2.0 and GHOST consistently produce alignments with competitive trade-offs between the competing goals of topological and biological quality, though GHOST's alignments exhibit consistently higher topological quality.

procedures in terms of both the node and edge correctness. This is indicative of a trend we observe when aligning networks from different organisms as well (see below), where the topological quality of the alignments produced by IsoRank, even with a large weight being placed on the topological score, seems to fall behind those produced by the other aligners.

The performance of GHOST in this set of experiments suggests that the spectral signature is robust to the presence of noise in the network, significantly more so than the graphlet degree signatures used in the GRAAL aligners. These results agree with existing evidence, such as that presented by Wilson and Zhu [131], that the spectral distance between graphs is robust to small topological changes. Both this robustness and the specificity of the spectra seem to carry over to our topological signatures, and do not appear to be negatively affected by the use of spectral densities to deal with graphs of different order.

3.3.3 Alignments Between Different Species

The same general performance trend holds under the *C. jejuni* / *E. coli* and *A. thaliana* / *D. melanogaster* alignments we considered, as well as under both the biological process and molecular function aspects of the gene ontology (see figure 3.3) — due to sparseness of annotation, the cellular component aspect was not included in this analysis. GHOST produces alignments with very high biological and topological qualities, and seems capable of trading off between these two goals more effectively than the other aligners. When placing the most weight on the biological quality of the alignment (i.e. $b = 0$ for GHOST and $\alpha = 0$ for Natalie 2.0), GHOST and Natalie 2.0 produce alignments with substantially higher biological quality than the other aligners. However, GHOST's alignments exhibit a much higher ICS score than Natalie 2.0's. As we vary the corresponding parameters and place more weight on topology, GHOST produces alignments with topological quality very close to those obtained by MI-GRAAL, but with significantly higher biological quality. In general, at a similar biological quality under both GO aspects, GHOST is capable of producing alignments with much greater topological quality other aligners.

For IsoRank, the precise value of α seems to matter very little. It produced alignments

of reasonable biological quality but very low topological quality. In fact, the highest ICS score achieved by IsoRank was ~ 0.1, when aligning *C. jejuni* and *E. coli*. MI-GRAAL performed very differently from IsoRank, producing alignments of excellent topological quality but generally poor biological quality.

The alignments obtained by Natalie 2.0 dominate those of IsoRank in terms of topological and biological quality for a large range of α. At an approximately equal biological similarity, Natalie 2.0 is capable of obtaining solutions with ICS scores between 50% and 120% higher. When aligning the *A. thaliana* and *D. melanogaster* networks, Natalie 2.0 can produce alignments with topological quality 120% greater than that of IsoRank that simultaneously exhibit $\sim 10\%$ greater biological similarity under the GO biological process aspect and $\sim 20\%$ greater biological similarity under the GO molecular function aspect. However, at the same biological quality, GHOST dominates Natalie 2.0, with topological quality improvements ranging from a few percent to a factor of 2 or more.

3.3.4 Runtime

Solving the spectral relaxation of the quadratic assignment problem is the step of GHOST with the largest potential asymptotic complexity. This step has worst-case running time $O((d_G d_H)^2)$ where d_G and d_H are the largest degrees in G and H respectively. This complexity results from the need to find the dominant eigenvector of the largest quadratic assignment matrix, which is quadratic in the size of the matrix [70]. Despite the potential worst-case complexity, we find that GHOST is fast in practice. First, we note that the computation of the spectral signatures are independent of the alignment being performed. Thus, the signatures need only be extracted once, and they can be reused for all alignments involving that organism. This also allows for a quicker exploration of the parameter space, because alignments can be performed under different parameter settings without recomputing the spectral signatures. Extracting the spectral signatures took 0.5 minutes for *E. coli*, 14 minutes for *C. jejuni*, 1 minute for *S. cerevisiae*, 1 minute for *A. thaliana* and 218 minutes for *D. melanogaster*.

The time to perform the actual alignments given the spectral signatures ranged between 1 and 6 minutes depending on the networks being compared. All timings were measured using 20 threads on a JVM instance given 16GB of heap space. The testing machine had 8 Opteron 8356 processors and 256GB of memory.

3.4 Conclusion

We have introduced GHOST, a novel framework for the global alignment of biological networks. At the heart of GHOST is a new spectral, multiscale node signature that we combine with a seed-and-extend approach and a local search procedure to perform global network alignment. The spectral signature is highly discriminative and robust to small topological variations. We verify this robustness in Section 3.3.2, showing that GHOST outstrips the competition in aligning the *S. cerevisiae* protein interaction network to noisier variants of itself. In these experiments, as well as the self-alignment of the *M. loti* network, the accuracy of GHOST is significantly higher than that of either IsoRank or MI-GRAAL. These experiments are of particular interest, because the ground truth is known and the ability of different aligners to uncover shared topological structure can be accurately measured.

We find that the alignments produced by GHOST consistently dominate those produced by the other aligners. When producing an alignment of approximately the same biological quality, GHOST yields alignments with substantially higher topological quality than either IsoRank or Natalie 2.0. Furthermore, at a similar level of topological quality, GHOST produces alignments that have far more biological relevance than those produced by MI-GRAAL. Finally, GHOST consistently produces alignments which exhibit a more competitive trade-off between topological and biological quality than the other aligners we considered (see Figure 3.3).

Chapter 4

Parsimonious Reconstruction of Network Evolution

This chapter is based on the paper "Parsimonious Reconstruction of Network Evolution", written in collaboration with E. Sefer, J. Malin, G. Marçais, S. Navlakha and C. Kingsford that appeared in the Workshop on Algorithms for Bioinformatics (WABI) 2011. A journal version will also appear in Algorithms for Molecular Biology.

4.1 Introduction

Evolution provides a powerful lens through which to view biological relationships. Many relationships between extant species and between these species and their environments, can be understood by analyzing and comparing their phenotypic traits [30], often leading to a hypothesis about the phenotype of their common ancestor. The problem of inferring the genome of ancestral species has likewise been explored [96]. With the growing prevalence of high-throughput regulatory and protein-protein interaction data, we are now well poised to ask what ancestral species looked like at the critical level of their interactomes. We present a framework to predict the topology of ancestral pathways, complexes, and regulatory programs by observing the structure of their descendants in multiple extant species.

Exploring the biological networks of present-day species provokes many questions about how these networks have evolved to have the structure and function we find. Generating plausible ancestral networks can often help to answer these questions. For example, joint histories can be used to compare the conservation and the route to divergence of

corresponding processes in two species. This allows us to more finely quantify how modularity has changed over time [67] and how interactions within a protein complex may have reconfigured across species starting from a single shared state [103]. Understanding the state of ancestral networks and how their interactions have evolved into those of the present-day species also allows for the development of better network alignment algorithms [42, 44, 114, 36]. While it has been shown that phylogenetic relationships can be inferred based on the analysis of conserved interaction modules [39] and through the topological alignment of interaction networks [68], a reasonable estimate the ancestral network topology will help to improve both the quality of such alignments and the accuracy of network-based phylogenies. Contrasting the topology of extant and ancestral networks can also shed light on the nature of robustness and evolvability [2, 40, 111]. Further, inferred changes in metabolic networks can be linked to changes in the biochemical environment in which each species has evolved, and this can reveal novel mechanisms of ecological adaptation [12, 11]. Finally, comparing network histories inferred using different model parameters can be used to estimate the likelihoods of various evolutionary events [86, 92].

There has been some recent work on reconstructing ancestral interactions. Gibson and Goldberg [52] presented a framework for estimating ancestral protein interaction networks that handles gene duplication and interaction loss using gene trees reconciled against a species phylogeny. However, their approach assumes that interaction losses occur immediately after duplication and does not support interaction gain outside of gene duplication. These assumptions are limiting because interaction loses may occur well after duplication, and independent gains are believed to occur at non-trivial rates [79]. Dutkowski and Tiuryn [36] provided a probabilistic method for inferring ancestral interactions with the goal of improved network alignment. Their approach is based on constructing a Bayesian network with a tree topology where binary random variables represent existence or non-existence of potential interactions. A similar graphical model was proposed by Pinney et al. [105], who applied it to inferring ancestral interactions between bZIP proteins. In the

former method, interaction addition and deletion is assumed to occur only immediately following a duplication or speciation event. Further, both methods assume the relative ordering of duplication events is known even between events in unrelated homology groups. Pinney et al. [105] also explore a parsimony-based approach [88] and find it to work well; however, it too assumes a known ordering of unrelated duplication events. The main drawback of these approaches is that the assumed ordering comes from sequence-derived branch lengths, which do not necessarily agree with rates that would be estimated based on network evolution [137]. This motivates an approach such as we describe below that does not use branch lengths as input.

Zhang and Moret [137, 136] use a maximal likelihood method to reconstruct ancestral regulatory networks as a means to improve estimation of regulatory networks in extant species. Mithani et al. [89] study the evolution of metabolic networks, but they only model the gain and loss of interactions amongst a fixed set of metabolites, whereas we also consider node duplication and loss encoded by a tree. Navlakha and Kingsford [92] present greedy algorithms for finding high-likelihood ancestral networks under several assumed models of network growth. They applied these methods to a yeast protein interaction network and a social network to estimate relative arrival times of nodes and interactions and found that the inferred histories matched many independently studied properties of network growth. This attests to the feasibility of using networks to study evolution. The authors, however, only consider a single network at a time, and there is no guarantee that independent reconstruction of two networks will converge to a common ancestor.

Here, we introduce a combinatorial framework for representing histories of network evolution that can encode gene duplication, gene loss, interaction gain and interaction loss at arbitrary times and does not assume a known total ordering of duplication events. We show that almost parsimonious histories of interaction gain and loss can be computed in practice quickly given a duplication history. In simulated settings, we show that these parsimonious histories can be used to accurately reconstruct a common ancestral regulatory

network of two extant regulatory networks. We also show that our approach can infer, with high accuracy, the interactions among the bZIP family of proteins in several ancestral organisms.

4.2 A Framework for Representing Network Histories

Any natural model of network evolution will include events for gene duplication, gene loss, interaction gain, and interaction loss. Many such growth models have been studied (e.g. [24, 121, 100, 57, 2, 136]). We describe below how these events can be encoded in a history graph. We note that there are other evolutionary events that affect the growth and structure of biological networks. For example, Toll-Riera et al. [125] provide evidence for *de novo* gene birth originating from non-coding genomic regions. While such events play a role in shaping the evolutionary history and current structure of biological networks; they are less common than the gene duplication and loss and interaction gain and loss, and are not explicitly modeled in the current framework.

Consider a set V of proteins or genes (henceforth "nodes") descended from a common ancestor by duplication events. Those duplication events can be encoded in a binary *duplication tree* T with the items of V as the leaves. An internal node u in T represents a duplication event of u into its left and right children, u_L and u_R. In this representation, after a duplication event, the node represented by u conceptually does not exist anymore and has been replaced by its two children. The leaves of a duplication tree are labeled *Present* or *Absent*. Absent leaves represent products of duplication events that were subsequently lost. A collection of such trees is a *duplication forest* F.

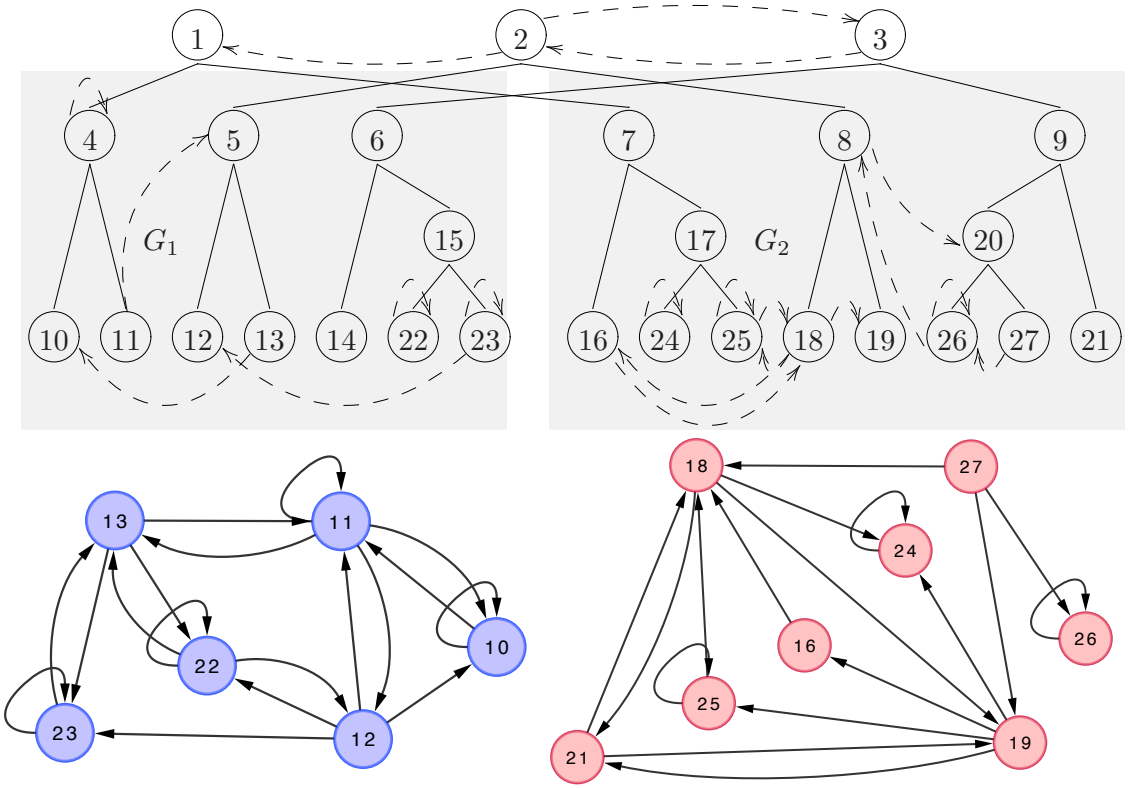


Figure 4.1: *ANR: Example History*

A duplication forest (solid edges at top) with the non-tree edges (dashed) necessary to construct G_1 and G_2 (shown at bottom). Nodes 1, 2, and 3 represent the 3 homology groups present in the ancestral graph. Node 14 was lost. As an example of the connectivity induced by the non-tree edges, consider edge $(27, 18)$ in G_2 which is implied by the directed non-tree edge from $(3, 2)$. However, the reverse edge $(18, 27)$, which is implied by $(2, 3)$, does not exist because its state is flipped by $(8, 20)$.

The gain and loss of interactions can be represented with additional non-tree edges placed on a duplication forest. A non-tree edge $\{u, v\}$ represents an *edge flip event*, where the interaction between u and v is created if the interaction is currently absent or removed if the interaction is currently present. Let P_u and P_v be the paths from nodes u and v to the root. An interaction exists between u and v if there are an odd number of such flip non-tree edges between nodes in P_u and P_v. Every non-tree edge between P_u and P_v, therefore, represents alternatively interaction creation or deletion between nodes u and v in the evolution of the biological network.

A graph H consisting of the union of a duplication forest and flip non-tree edges is a *network history*. A history H *constructs* a graph G when the Present leaves of the duplication forest in H correspond to the nodes of G and the flip edges of H imply an interaction between u and v if and only if $\{u, v\}$ is an interaction in G. See Figure 4.1 for an example history.

Not all placements of non-tree edges lead to a valid network history. The interaction histories must be consistent with some temporal embedding of the tree. Let t_u^c and t_u^d be respectively the time of creation and duplication of node u. Naturally, $t_u^c < t_u^d$, $t_u^d = \infty$ if u is a Present leaf, and if v is the child of u, then by definition we have

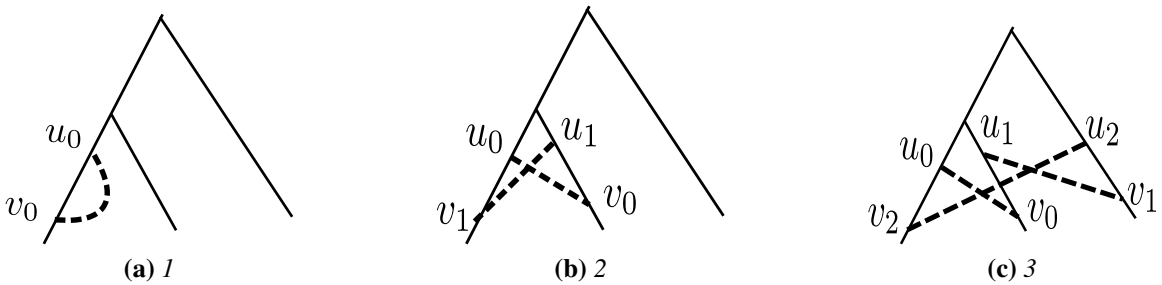
$$t_u^c < t_u^d = t_v^c < t_v^d. \tag{4.1}$$

If $\{u, w\}$ is a flip edge, then the time $t_{\{u,w\}}$ of appearance of this edge must satisfy

$$t_u^c \leq t_{\{u,w\}} < t_u^d \quad \text{and} \quad t_w^c \leq t_{\{u,w\}} < t_w^d, \tag{4.2}$$

because an event between u and w can only occur when both u and w exist. A history graph H is said to be *valid* if there exist t_u^c, t_u^d for every node u such that conditions (4.1) and (4.2) are satisfied for every non-tree edge.

Whether a particular history is valid can be checked combinatorially using the fol-



(a) *1* **(b)** *2* **(c)** *3*

Figure 4.2: *ANR: Blocking Loops*

Blocking loops of size 1, 2 and 3. The solid lines represent a subset of the tree T. The dashed lines are non-tree edges representing interaction flip events.

lowing alternative characterization of validity. A *k-blocking loop* is a set of flip edges $\{\{u_i, v_i\}\}_{0 \leq i < k}$ such that u_{i+1} is an ancestor of v_i in the tree for $0 \leq i < k$ (where the index $i + 1$ is taken modulo k). See Figure 4.2 for examples. Blocking loops are not permitted in valid histories and, conversely, the non-existence of blocking loops implies that a history is valid.

4.3 Parsimonious Reconstruction of a Network History

Traditional phylogenetic inference algorithms and reconciliation between gene and species trees can be used to obtain duplication and speciation histories [20, 35, 4]. What remains is the reconstruction of interaction gain and loss events. This leads to the following problem:

> **Problem 1** (Minimum Flips) Given a duplication forest F and an extant network G, find H, a valid history constructing G, with a minimum number of flip edges.

We will show that nearly optimal solutions to this problem for a large range of instances can be solved in polynomial time in practice. Whether Problem 1 is NP-hard or admits a polynomial-time algorithm for all instances remains open.

4.3.1 A Fast Heuristic Algorithm

The challenge of Problem 1 comes from avoiding the creation of blocking loops. A polynomial-time algorithm can find a minimum set of flip edges that reconstructs a graph G and does not contain 1- and 2-blocking loops but allows longer blocking loops. We define an *interaction encoding* of $G = (V, E)$ as a function $f_G : V \times V \to \{0, 1\}$ such that $f_G(u, v) = 1$ if $\{u, v\}$ is an interaction in G and $f_G(u, v) = 0$ otherwise. We omit the subscript on f_G if G is clear from the context.

The following intertwined dynamic programming recurrences find the minimum number of flip edges required for H to construct a given graph G if blocking loops of length ≥ 3 are allowed. First, $S(u, f)$ finds the minimum number of flip edges for the subtree rooted at u and interaction encoding f:

$$S(u, f) = S(u_L, f) + S(u_R, f) + A(u_L, u_R, f). \tag{4.3}$$

The expression $A(u, v, f)$ gives the minimum number of flip edges that should be placed between the subtree rooted at u and the subtree rooted at v. This can be computed using the recurrence:

$$A(u, v, f) = \min \begin{cases} A(u_L, v, f) + A(u_R, v, f) \\ A(u, v_L, f) + A(u, v_R, f) \\ 1 + A(u_L, v, \bar{f}) + A(u_R, v, \bar{f}) \\ 1 + A(u, v_L, \bar{f}) + A(u, v_R, \bar{f}). \end{cases} \tag{4.4}$$

In the above, if one of u or v is a leaf but the other is not, the options that look at non-existent children are disallowed.

The function $\bar{f}$ in Eqn. (4.4) is defined as $1 - f$ and thus represents a function such that $\bar{f}(x)$ has opposite parity from $f(x)$ for all x. The A recurrence considers two possible

options: (1) We connect u and v with a non-tree edge, this costs us 1 and flips the parity of all interactions going between the subtree rooted at u and the subtree rooted at v; or (2) We do not connect u and v with a flip edge. This costs 0 and keeps the parity requirement the same. Regardless of the choice to create an edge, because we are not allowed to have a 2-blocking loop, either (a) we possibly connect u to some descendant of v (and do not connect v to a descendant of u) or (b) we possibly connect v to some descendant of u (and do not connect u to a descendant of v).

The base case for the S recurrence when u is a leaf and the base case for the A recurrence when u and v are leaves are:

$$S(u, f) = 0 \quad \text{and} \quad A(u, v, f) = f(u, v).$$

The minimum number of flip edges needed to turn a duplication forest F into a history constructing G (allowing blocking loops of ≥ 3) is then given by $\sum_r S(r, d_G) + \sum_{r,q} A(r, q, d_G)$, where d_G is the interaction encoding of G, and the sums are over roots r, q of the trees in F. Standard backtracking can be used to recover the actual minimum edge set. If n is the number of nodes in the forest, the dynamic program runs in $O(n^2)$ time and space because only two functions f are ever considered: d_G, and $\bar{d_G}$. This yields $\approx n \times n \times 2$ subproblems, each of which can be solved in constant time.

The heuristic also can be extended to handle different costs for interaction addition and deletion by changing the constants in the recurrences to be a function of the parity of each flip. Only two values of f (d_G and $\bar{d_G}$) are ever considered, and every flip switches f between these two states. Thus, by examining f, and determining if its current states corresponds to d_G or $\bar{d_G}$, one can determine if an odd or even number of flips have occurred, and thus, whether the current flip corresponds to the addition or deletion of an interaction. If the current flip represents the addition of an interaction, then it incurs the cost c_{add}. Otherwise, the flip encodes the loss of an interaction, and incurs the loss cost, c_{loss}.

4.3.2 Identifying and Removing Blocking Loops

To identify blocking loops, we use a modified depth-first search procedure in which tree edges are traversed according to their direction (i.e. away from the root) while non-tree edges can be traversed in either direction. Whenever a node is encountered twice during the depth first search, a cycle has been discovered and is checked for the blocking loop condition given above. If the cycle is not blocking loop, we can safely ignore it. Otherwise, one of the non-tree edges of this loop is chosen at random, and we forbid that edge from appearing in the solution and rerun the dynamic program. Because there are $O(n^2)$ possible non-tree edges, iterating this procedure will terminate in polynomial time. We repeat the process of identifying blocking loops and forbidding non-tree edges until a valid solution is obtained. In the worst case, one may obtain a solution where all non-tree edges are placed at leaves, but in practice long blocking loops do not often arise, and the obtained solutions are close to optimal.

4.3.3 Reconstruction of a Common Ancestor of Two Graphs

Given extant networks of several species, in addition to the reconstructed history, we seek a parsimonious estimate for their common ancestor network. Specifically, Given extant networks G_1 and G_2, with interaction encodings d_1 and d_2, and their duplication forests F_1 and F_2, we want to find an ancestral network $X = (V_X, E_X)$ such that the cost of X evolving into G_1 and G_2 after speciation is minimized. V_X is the set of roots of the homology forests. We assume that the networks of the two species evolved independently after speciation. Therefore, we can use the recurrence above applied to F_1 and F_2 to compute $A_{F_1}(r, q, d_1)$ and $A_{F_2}(r, q, d_2)$ independently for $r, q \in V_X$, and then select interactions in X as follows. E_X of X is given by the pairs $r, q \in V_X \times V_X$ for which creating an interaction leads to a lower total cost than not creating an interaction. Formally, we place an

interaction $\{r, q\}$ in E_X if

$$1 + A_{F_1}(r, q, \bar{d}_1) + A_{F_2}(r, q, \bar{d}_2) < A_{F_1}(r, q, d_1) + A_{F_2}(r, q, d_2). \tag{4.5}$$

Rule (4.5) creates an interaction in X if doing so causes the cost of parsimonious histories inferred for G_1 and G_2 between the homology groups associated with r and q to be smaller than if no interaction was created.

Modifications for self-loops

Self-loops (homodimers) can be accommodated by modifying recurrence (4.3):

$$S'(u, f) = \min \begin{cases} S'(u_L, f) + S'(u_R, f) + A(u_L, u_R, f) \\ 1 + S'(u_L, \bar{f}) + S'(u_R, \bar{f}) + A(u_L, u_R, \bar{f}). \end{cases} \tag{4.6}$$

The intuition here is that paying cost 1 to create a self-loop on node u creates (or removes) interactions, including self-loops, among all the descendants of u.

Modifications for directed graphs

The algorithm can be modified to handle evolutionary histories of directed graphs. For this, only the recurrence A need be modified. When computing $A'(u, v, f)$, a non-tree edge can be included from u to v, from v to u, both, or neither. Each of these cases modifies the function f in a different way. Specifically:

$$A'(u, v, f) = \min \begin{cases} 0 + A'(u_L, v, f) + A'(u_R, v, f) \\ 1 + A'(u_L, v, \overleftarrow{f}) + A'(u_R, v, \overleftarrow{f}) \\ 1 + A'(u_L, v, \overrightarrow{f}) + A'(u_R, v, \overrightarrow{f}) \\ 2 + A'(u_L, v, \overleftrightarrow{f}) + A'(u_R, v, \overleftrightarrow{f}), \\ \vdots \end{cases}$$

where the vertical ellipsis indicates the symmetric cases involving v_L and v_R, and where $\overrightarrow{f}, \overleftarrow{f}, \overleftrightarrow{f}$ are defined, depending on u and v, as follows:

$$\overrightarrow{f}(x,y) = \min \begin{cases} 1 - f(x,y) & \text{if } x \in \text{ST}(u) \text{ and } y \in \text{ST}(v) \\ f(x,y) & \text{otherwise} \end{cases} \tag{4.7}$$

$$\overleftrightarrow{f}(x,y) = \min \begin{cases} 1 - f(x,y) & \text{if } x \in \text{ST}(u) \text{ and } y \in \text{ST}(v) \text{ or vice versa} \\ f(x,y) & \text{otherwise,} \end{cases} \tag{4.8}$$

with $\overleftarrow{f}$ defined analogously to $\overrightarrow{f}$. Here, $\text{ST}(u)$ indicates the set of nodes in the subtree rooted at u.

Accounting for phylogenetic branch lengths

One of the strengths of our proposed method is that it does not require the user to specify the lengths of the edges in a duplication history. The estimation of such phylogenetic branch lengths relies on the molecular clock assumption, and these lengths can easily be misestimated, especially those for distant ancestors. However, previous approaches [105, 88] relied crucially upon the phylogenetic branch lengths to impose a specific ordering on the set of potential ancestral interactions. Small errors in the estimates of phylogenetic branch lengths can lead these approaches to disallow potentially high probability or high parsimony ancestral interactions.

Yet, the branch lengths in the duplication history do encode potentially useful information. For example, two ancestral proteins for which the intervals of existence are separated by a significant amount of time are unlikely to have interacted, even if branch length estimates are imprecise. The algorithm we defined above can be further modified to account for phylogenetic branch lengths, using them to penalize unlikely ancestral states without explicitly disallowing potentially important interactions. This can be achieved by modify-

ing the recurrence as follows:

$$A(u,v,f) = \min \begin{cases} A(u_L,v,f) + A(u_R,v,f) \\ A(u,v_L,f) + A(u,v_R,f) \\ \alpha\delta(u,v) + 1 + A(u_L,v,\bar{f}) + A(u_R,v,\bar{f}) \\ \alpha\delta(u,v) + 1 + A(u,v_L,\bar{f}) + A(u,v_R,\bar{f}). \end{cases} \tag{4.9}$$

where

$$\delta(u,v) = \begin{cases} t_v^c - t_u^d & \text{if } t_u^d < t_v^c \\ t_u^c - t_v^d & \text{if } t_v^d < t_u^c \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

The analogous modification applies to the directed recurrence as well. Here, $\alpha\delta(\cdot,\cdot)$ is a function that assigns a cost to a pair of nodes $\{u,v\}$ that is proportional to the distance between the existence intervals of these nodes (and is 0 if they overlap). The constant, α, is provided as input to the algorithm and can be interpreted as the factor by which interactions are penalized between nodes which do not overlap in time according to the inferred phylogenetic branch lengths. At $\alpha = \infty$, branch lengths become hard constraints, and proteins between which the existence intervals do not overlap are not allowed to interact; this α also prohibits the formation of blocking loops. However, results tend to be better (higher F1-score) when one allows some constraints from branch lengths to be violated. This approach allows our algorithm to take phylogenetic branch lengths into account in a way that incorporates the information they encode without suffering from the potential issues that occur when considering these lengths as hard constraints.

Accounting for Weighted Inputs

Finally, we also modify the recurrence to handle weighted input networks. That is, rather than requiring the input to be simply the topology of the experimentally measured inter-

action network, we assume that each input edge can take on a weight that signifies the level "confidence" of the interaction it represents. Such weighted networks are becoming increasingly available through such database services as HitPredict [101], MINT [81] and STRING [120]. To take advantage of this information, we modify the base case of our recurrence to account for potential weights on input interactions. First, we allow $f(u, v)$ to take on values in $[0, 1]$ rather than simply 0 or 1. Then, the base case for the undirected recurrence becomes

$$A(u, v, 1) = c_{\text{loss}}(1.0 - f(u, v))$$
$$A(u, v, 0) = c_{\text{add}} f(u, v)$$

The base case for the directed recurrence can similarly be modified, denoting by w_{fwd} and w_{rev} the weights of edges (u, v) and (v, u) respectively, as

$$A(u, v, 1, 1) = c_{\text{loss}}(1.0 - w_{\text{fwd}}) + c_{\text{loss}}(1.0 - w_{\text{rev}})$$
$$A(u, v, 1, 0) = c_{\text{loss}}(1.0 - w_{\text{fwd}}) + c_{\text{add}} w_{\text{rev}}$$
$$A(u, v, 0, 1) = c_{\text{add}} w_{\text{fwd}} + c_{\text{loss}}(1.0 - w_{\text{rev}})$$
$$A(u, v, 0, 0) = c_{\text{add}} w_{\text{fwd}} + c_{\text{add}} w_{\text{rev}}$$

The modified recurrences ensure that, in the base case, we only pay the cost for the creation or deletion of an edge proportional to our confidence in that edge's existence.

4.4 Results and Discussion

We analyze the performance of our parsimony-based approach to ancestral network reconstruction on both simulated and real biological data. To generate simulated data, we consider a number of plausible models of network evolution and show that the parsimony approach is able to reconstruct ancestral networks reasonably well over a wide range of model parameters. Further, following the experiment of Pinney et al. [105], we evaluate

the performance of our approach on reconstructing the state of several ancestral network states of the bZIP family of proteins. We observe that our parsimony-based approach obtains high precision and recall, even on fairly distant ancestral networks.

Generating plausible simulated histories

We use a *degree-dependent model* (DDM) to simulate the evolutionary path from a putative ancestral network to its extant state. The model simulates node duplication, node deletion, independent interaction gain, and independent interaction loss with given probabilities P_{ndup}, $\mathrm{P}_{\text{nloss}}$, $\mathrm{P}_{\text{egain}}$ and $\mathrm{P}_{\text{eloss}}$, respectively. The nodes or edges involved in a modification are chosen probabilistically based on their degrees (as in [117]) according to the following expressions:

$$\mathrm{P}(u \mid \text{node duplication}) \propto 1/k_u \qquad \mathrm{P}(u \mid \text{node loss}) \propto 1/k_u \tag{4.11}$$

$$\mathrm{P}((u,v) \mid \text{interaction gain}) \propto k_u^o \qquad \mathrm{P}((u,v) \mid \text{interaction loss}) \propto 1/k_u^o, \tag{4.12}$$

where k_u^o is the out-degree of a node u, and k_u is the total degree. At each time step, the distribution of possible modifications to the graph is calculated as $\mathrm{P}(\text{modification}) = \mathrm{P}_{\text{operation}}\mathrm{P}(\text{object} \mid \text{operation})$. Nodes with out-degree of 0 are removed. Varying parameters P_{ndup}, $\mathrm{P}_{\text{nloss}}$, $\mathrm{P}_{\text{egain}}$ and $\mathrm{P}_{\text{eloss}}$ can produce a wide variety of densities and sizes. We also consider a *degree-independent model* (DIM) in which the four conditional probabilities in Eqns. (4.11) and (4.12) are all equal.

The DDM model is theoretically capable of producing evolutionary trajectories between any two networks while incorporating preferential attachment to the source node and random uniform choice of the target node. Furthermore, choosing a node for duplication or loss in inverse proportion to its degree favors an event in inverse relation to its expected disruption of the network.

We also consider a model of regulatory network evolution by Foster et al. [48], which is based on gene duplication, with incoming and outgoing interactions kept after duplication

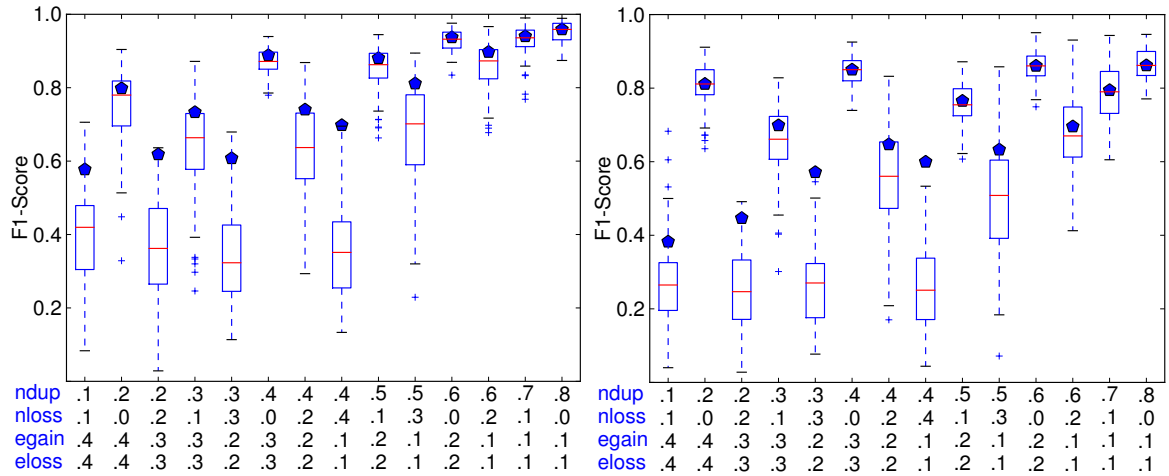
as in other models (P_{inkeep} and $P_{outkeep}$ probabilities respectively). New edges are added with probability $P_{innovation}$.

In all of the network evolution models, we started with a random connected seed graph that has 10 nodes and 25 interactions. We evolved it to X by 200 operations after which we introduce a speciation event, and then both G_1 and G_2 evolve from X by an additional 200 operations each. To generate more biologically plausible ancestral graphs, instances were kept only if the ancestral graph X had an in-degree that fit an exponential distribution with parameter between 1.0 and 1.2 or an out-degree that was scale-free with parameter between 1.8 and 2.2.

Reconstructing simulated networks

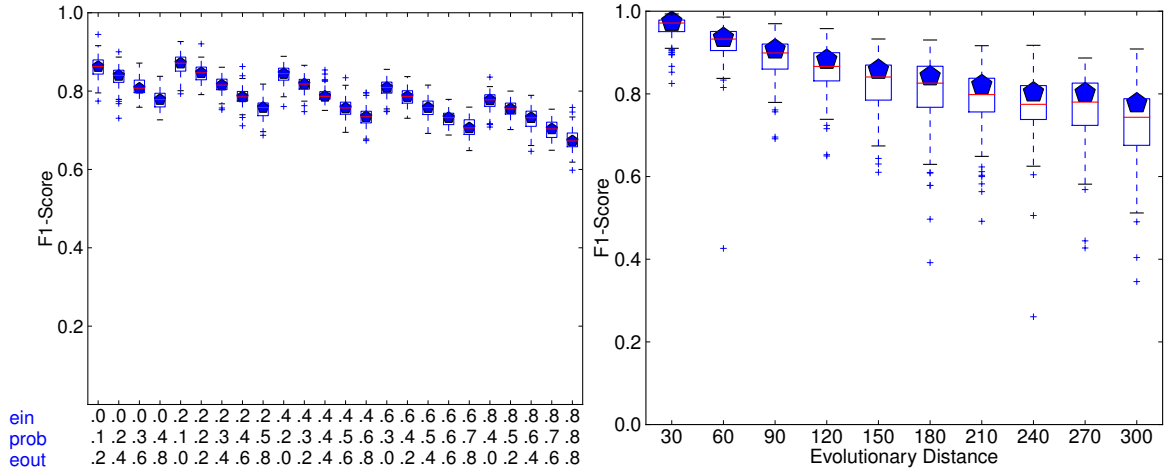
Optimality of loop breaking. The greedy procedure to break blocking loops produces histories that are very close to optimal. We generated 1400 networks using the DDM model with the range of parameters shown on the x-axis of Figure 4.3a. In the vast majority of cases (1325 out of 1400), either no loop breaking is required, or the solution discovered after greedily breaking all loops has the same cost as the original solution. In these cases, therefore, the method returned a provably maximally parsimonious set of interaction modification events. In the remaining 75 cases (5.4%), greedily removing blocking loops increased the number of interaction modifications by no more than 10 ($< 2\%$ of the initial number of interaction modification events). Since the initial solution provides a lower bound on the optimal, we can verify that the greedy procedure always found a solution within 2% of the optimal (and perhaps even better). Thus, it seems that in practice, while blocking loops occur, the greedy procedure does a good job of eliminating them without increasing the number of events significantly.

Effect of growth model and its parameters. Modeling the evolutionary dynamics of a regulatory network is still an active topic of research. We therefore experimented with three different network models. Despite their differences, high precision and recall (implied



(a) *Degree-dependent model (Sec. Generating plausible simulated histories)*

(b) *Degree-independent model*



(c) *Foster et al. [48] model*

(d) *Divergence of G_1, G_2 from ancestor*

Figure 4.3: *ANR: Synthetic Results*

(a-c) Effect of model parameters on reconstruction accuracy under three different models. "Prob" in (c) is $P_{innovation}$. (d) Effect of evolutionary distance (number of network modification operations) on the quality of the ancestral network reconstruction. In both plots, boxes show 1st and 3rd quartile over 100 networks with median indicated by a line. Pentagons show the median if interactions incident to nodes lost in both lineages are not considered.

from the F1 score) can be obtained for all of them for many choices of their parameters (Figure 4.3a-c). We measure the precision, the recall, and the F1-score defined as:

$$\text{precision} := \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$\text{recall} := \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$\text{F1-score} := \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Very good performance can be achieved under the general model presented above whether degree distributions are taken into account (Figure 4.3a) or not (Figure 4.3b) when selecting nodes and interactions to modify. In these cases, for most parameter choices, precision is close to 1.0, meaning every interaction predicted to be in the ancestor, in fact, was. Recall is often lower. The Foster et al. [48] model, with its heavy reliance on duplication events and lack of node loss events, tends to be the simplest under which to reconstruct the ancestral graph (Figure 4.3c).

The largest factor leading to poorer performance is lower recall caused by gene losses. If all descendants of a gene are lost in both extant networks, it is not possible to reconstruct interactions incident to it. If these interactions are excluded from the computation of recall, the F1 score often improves dramatically. Median F1 scores excluding these interactions are shown as pentagons in Figure 4.3.

Robustness to evolutionary divergence. Naturally, the ability to recover the ancestral network degrades as time passes and the extant networks diverge. However, the degradation is slow (Figure 4.3d, using the degree-dependent model with parameters fixed at $P_{ndup} = 0.35$, $P_{nloss} = 0.05$, $P_{egain} = 0.3$, and $P_{eloss} = 0.3$). When the distance is small (measured as the number of events separating them), we are almost always able to recover the

ancestral network well, as illustrated by the high F1-scores and small interquartile ranges in Figure 4.3d. Even when the distance between the ancestral and extant networks is large (300) compared to the average ancestral network size (55), we obtain an F1-score of 0.72 (0.77 when homology groups lost in both lineages are not considered).

Reconstructing ancestral bZIP networks

We also repeated the test performed by Pinney et al. [105] by using our method to reconstruct ancestral interactions among the bZIP family of transcription factors. The interactions between dimerizing bZIP transcription factors are strongly mediated by their coiled-coil leucine zipper domains, and the strength of these interactions can be computationally predicted with high sensitivity and specificity using sequence alone [45]. This sequence-based method was used to predict both the interaction strength between extant bZIP proteins and inferred ancestral protein sequences. These interactions were used as the ground truth [105].

The duplication history relating the bZIP proteins is built atop the extant networks of 4 relatively distant species, *D. rerio*, *T. rubripes*, *H. sapiens*, and *C. intestinalis*. From the interactions in these extant networks and the structure of the duplication history of the constituent proteins, we reconstruct 3 ancestral networks: the Teleost (ancestor of *D. rerio* and *T. rubripes*), Vertebrata (ancestor of *D. rerio*, *T. rubripes* and *H. sapiens*) and Chordate (ancestor of *D. rerio*, *T. rubripes*, *H. sapiens*, and *C. intestinalis*) networks.

Table 4.1 compares the relative performance of our parsimony-based approach and the probabilistic method described by Pinney et al. [105] Our results were generated using a ratio of $11.4 : 1$ for the cost of interaction creation to interaction deletion (the same ratio as was used in the probabilistic method). Furthermore, we choose not to penalize interactions based on phylogenetic branch length (i.e. $\alpha = 0$ in δ_α), thus allowing our algorithm to explore the entire solution space. We note that our approach outperforms the probabilistic method, particularly on the Teleost and Vertebrata networks. One explanation for the

Table 4.1: *ANR: Performance of Parsimony vs. Probabilistic Approach*

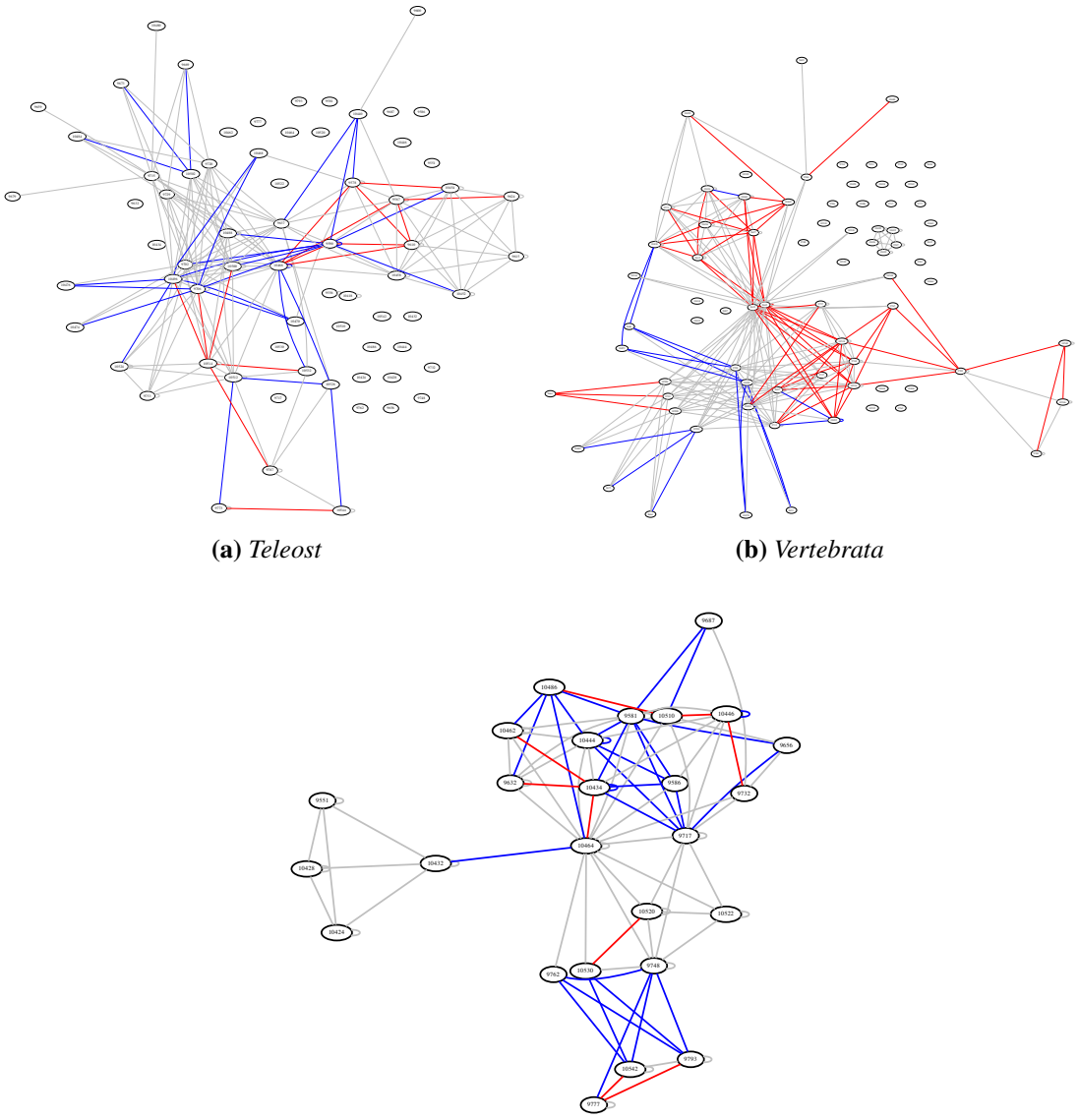
The relative performance of our parsimony approach and the probabilistic method described by Pinney et al. in reconstructing the ancestral interaction networks we consider.

Ancestor	Method	Precision	Recall	F1
Teleost	Parsimony	0.84	0.91	0.87
	Probabilistic	0.68	0.88	0.77
Vertebrata	Parsimony	0.79	0.94	0.86
	Probabilistic	0.75	0.81	0.78
Chordata	Parsimony	0.67	0.87	0.76
	Probabilistic	0.74	0.74	0.75

improved performance of our method is that it considers a larger set of ancestral interactions by not explicitly disallowing parsimonious interactions based solely on potentially misleading phylogenetic branch lengths.

We corroborated this hypothesis by measuring the reconstruction performance of our approach for increasing values of α, and noticed a very slow but steady decrease in performance as α increases. Nonetheless, at $\alpha = \infty$ (using branch lengths as hard constraints as Pinney et al. do), our method still outperforms the probabilistic method on the Teleost network (F1 score of 0.84 vs 0.77). This experiment suggests that, at least on this family of protein interactions, relying on the phylogenetic branch lengths to aid inference does not improve — and potentially harms — performance.

A visual inspection (see Figure 4.4) of the inferred ancestral networks revealed no strong patterns among the interactions predicted based on sequence versus those predicted using our parsimony approach. However, if a protein is involved in a disagreement, it is often involved in more than one.



(a) *Teleost*

(b) *Vertebrata*

(c) *Chordata*

Figure 4.4: *ANR: Ancestral Predictions*

The inferred networks of the Teleost, Vertebrata and Chordata ancestors. Edges drawn in gray were inferred by both our parsimony-based approach and by the sequence-based approach. Red edges were inferred based on sequence but not by the parsimony method, and the blue edges were inferred by the parsimony method but not based on sequence.

4.5 Conclusion

We have presented a novel framework for representing network histories involving gene duplications, gene loss, and interaction gain and loss for both directed and undirected graphs. We also provide a combinatorial characterization for valid histories. Our experiments demonstrate that a fast heuristic can recover optimal histories in a large majority of instances. We further provide evidence that, even with a probabilistic, weighted, generative model of network growth, a parsimony approach can recover accurate ancestral networks (F1 scores ≥ 0.8 for a wide range of parameters under several different models). Finally, we show that our method accurately reconstructs a number of ancestral networks for the bZIP family of proteins. Interestingly, we observe that we obtain the highest accuracy in ancestral network reconstruction when we do not impose a particular ordering on unrelated duplication events (as implied by phylogenetic branch lengths). This suggests that the ability of our approach to explore a larger space of potential solutions than previous work can provide practical benefits. In future work, it will be interesting to explore topological properties of the ancestral networks, such as modularity and degree distribution, and to analyze how these properties may have changed over time. We would also like to extend the evolutionary history framework and inference algorithm to handle *de novo* gene birth events, which are known to contribute to network growth [125].

Chapter 5

A Sum Over Parsimonious Histories Approach to Ancestral Network Reconstruction

5.1 Introduction

In the previous chapter, we introduced a combinatorial framework for representing network evolution histories and presented an approach that recovers almost parsimonious histories. In general, however, there may be a large number of optimal and near-optimal histories. A priori, we don't know how different these solutions may be or how representative of the ensemble the solution at which we arrive is.

To overcome this limitation and discover a more faithful representation of the space of solutions to the ancestral network reconstruction (ANR) problem, one might consider enumerating or sampling from the solution space. However, this approach suffers from two substantial issues. First, it is unclear, using the framework given in Chapter 4, how to efficiently sample from the solution space. Second, it is possible that, to arrive at a faithful and unbiased characterization of this space, the required number of samples will render this approach impractical.

We present an approach, based on a novel algorithm and advanced dynamic programming techniques, which addresses the problem of efficiently characterizing the relevant portion of the solution space without resorting to sampling. By formulating our dynamic program in the forward hypergraph framework [49], it becomes clear how to visit the space of solutions in a principled way. We develop an extension of the k-best parsing algorithm

of Huang and Chiang [56] that allows us to aggregate solutions of equivalent quality. As a result, rather than enumerating individual solutions, we are able to enumerate solution classes (i.e. the set of all solutions having an equivalent parsimony cost) and provide a characterization of the space of optimal and near-optimal solutions to an instance of the ANR problem. We call this method a sum over parsimonious histories (SOPH) approach to ancestral network reconstruction.

For every potential interaction — either ancestral or extant — our algorithm computes the posterior probability, summed over the most parsimonious histories, that the interaction exists. This provides a number of benefits over the single network history recovered by the algorithm presented in Chapter 4. In particular, we can now rank interactions by confidence. Since posterior probabilities are also provided for extant interactions, we are able to impute missing interactions and to quantify the consistency (in terms of evolutionary parsimony) of a given set of interactions.

In the rest of this chapter, we will refer to solutions having minimum cost as *optimal*, regardless of their inclusion of blocking loops. Thus, when we say a solution is optimal, we mean that it has the absolute minimum cost with regard to the parsimony criteria of any history generating the extant interactions, even though it may include blocking loops. Accordingly, when we say that a solution is near-optimal, we mean that it has a cost that is close to that of an optimal solution; it may or may not contain blocking loops.

5.2 The Ordered Hypergraph Framework

We will formulate the ANR problem in the ordered hypergraph framework [56], as this will make it more clear how to devise an efficient algorithm for its solution. This framework allows one to explicitly represent the space of solutions to certain classes of combinatorial problems by encoding these solutions in the topology of a directed hypergraph. The ordered hypergraph representation is used in a wide variety of different fields, including natural language processing where it is used to represent parsing problems [65, 56], op-

erations research where it is used to represent transportation and planning problems [93], and computational biology where it is used to represent the problem of pseudoknotted RNA folding [108]. In particular, problems exhibiting the optimal substructure and overlapping subproblem properties [28] — those which can be solved efficiently by dynamic programming techniques — are particularly amenable to efficient algorithms executed on an ordered hypergraph representation.

The intuition behind the ordered hypergraph representation is to explicitly encode the structure of a dynamic programming recurrence using the topology of a hypergraph. Each vertex in the hypergraph represents a term of the recurrence, and the hyperarcs encode the sub-terms (tail nodes of the arc) on which the resulting term (head node of the arc) depends. By traversing the hypergraph in topological order, subproblems can be solved so that when the solution to a larger problem needs to be computed, the solutions to the smaller subproblems on which it depends are already available. This is the basic strategy behind traditional dynamic programming approaches and the hypergraph representation simply makes the relation between the terms of the recurrence explicit by encoding them in the topological structure of the hypergraph.

5.2.1 Definitions

Before we can describe how the ancestral network reconstruction problem can be encoded using the hypergraph framework, we must introduce some preliminary definitions. We use the hypergraph definition and a number of related definitions given by Huang and Chiang [56]. Specifically, we define an ordered hypergraph as $H = (V_H, E_H, r, \mathbf{C})$, where V is the set of vertices, E is the set of ordered hyperarcs, r is a designated root node and $\mathbf{C}$ a cost function defined on the hyperarcs. For each hyperarc $\mathbf{e}$ in E, we call $h(\mathbf{e})$ the head of the hyperarc, and the vector $t(\mathbf{e})$ the tail of the hyperarc. When we say that a hyperarc is ordered, we mean that its tail consists of a vector rather than a set, so that the tail nodes can be consistently ordered and indexed. We denote by $t(\mathbf{e})_i$ the i^{th} ordered element of the tail of $\mathbf{e}$. The cost function $c : E \to \mathbb{R}$, assigns a cost to each hyperarc.

For a vertex v of the hypergraph, we call the set of incoming hyperarcs the backward star of v, and denote it by $\text{BS}(v) = \{\mathbf{e} \in E_H \mid v = h(\mathbf{e})\}$. Any vertex w which appears in the tail of some hyperarc $\mathbf{e}$ where $\mathbf{e} \in \text{BS}(v)$ is said to precede v. Additional definitions are necessary, but will be given as they are encountered.

5.2.2 Encoding the Ancestral Network Reconstruction Problem

We now reformulate the ancestral network reconstruction problem in terms of the ordered hypergraph framework. This requires encoding the recurrences from Chapter 4 in terms of a hypergraph. We will demonstrate how to encode the recurrence given in **????** as a hypergraph H, though all others can be encoded in an analogous manner.

Let T denote the set of all terms in the original recurrence. Each term in **??** is parameterized by two variables, a node u in the duplication forest and an interaction function f encoding the interactions in the subtree rooted at u. The terms in **??** are additionally parameterized by a third variable v which is another node in the duplication forest. We define a pair of nodes, say $\{u, v\}$, in the duplication forest together with a particular interaction function f, as an *interaction state*, which we denote as $(\{u, v\}, f)$. We can also define an interaction state for a single node u and interaction function f as $(\{u\}, f)$. There is an injective mapping $M : T \to I$ from the terms of the recurrence into the set I of interaction states. Note that this mapping is injective because there exist interaction states for which there is no corresponding term in the recurrence (e.g. those states where u is an ancestor of v). For each interaction state $s = (\{u, v\}, f)$ in the image of T under M, we create a corresponding vertex x in the hypergraph. We label x with $(\{u, v\}, f)$ and note that it can be unambiguously referenced using this label. The set of all such vertices constitutes the vertex set of H.

Each term in the recurrence is either a base case or depends upon some other set of terms. For example, term $S(u, f)$ depends upon the terms $S(u_L, f)$, $S(u_R, f)$ and $A(u_L, u_R, f)$. Given the mapping M we have defined above, this means that the vertex $(\{u\}, f)$ depends upon the vertices $(\{u_L\}, f)$, $(\{u_R\}, f)$ and $(\{u_L, u_R\}, f)$. We en-

code this recurrence by placing a hyperarc $\mathbf{e}$ in H where $h(\mathbf{e}) = (\{u\}, f)$ and $t(\mathbf{e}) = ((\{u_L\}, f), (\{u_R\}, f), (\{u_L, u_R\}, f))$. More generally, let t be an arbitrary term in the recurrence, and let $t^1, t^2, \ldots, t^k$ be the set of terms upon which it depends (i.e. those terms appearing on the right-hand side of an equation where t appears on the left-hand side). For each such dependency, we create a hyperarc $\mathbf{e}$ with $h(\mathbf{e}) = t$ and $t(\mathbf{e}) = (t^1, t^2, \ldots, t^k)$. The cost of each hyperarc $c(\mathbf{e})$ encodes the cost of the transition from $t(\mathbf{e})$ to $h(\mathbf{e})$ and comes directly from the recurrence for the corresponding terms.

To make this description of H more clear, we translate a single, generic term of the recurrence from **??** into the hypergraph framework, describing the involved vertices, hyperarcs and values of the cost function:

$$
\begin{aligned}
x &= (\{u, v\}, f) \\
\mathbf{e_1} &= (\{u, v\}, f), ((\{u_L, v\}, f), (\{u_R, v\}, f)) \\
\mathbf{e_2} &= (\{u, v\}, f), ((\{u, v_L\}, f), (\{u, v_L\}, f)) \\
\mathbf{e_3} &= (\{u, v\}, f), ((\{u_L, v\}, \bar{f}), (\{u_R, v\}, \bar{f})) \\
\mathbf{e_4} &= (\{u, v\}, f), ((\{u, v_L\}, \bar{f}), (\{u, v_R\}, \bar{f})) \\
\mathrm{BS}(x) &= \{\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}, \mathbf{e_4}\} \\
c(\mathbf{e_1}) &= c(\mathbf{e_2}) = 0 \\
c(\mathbf{e_3}) &= c(\mathbf{e_4}) = 1
\end{aligned}
$$

The corresponding illustration can be found in Figure 5.1. Here, the backward star of x describes the choices that can be made in the recurrence (i.e. the ways we can descend down the duplication history starting from $(\{u, v\}, f)$). We can descend into either the subtree rooted at u or the one rooted at v, and we can either flip the state of the function to $\bar{f}$, or leave it unchanged. If we traverse any hyperarc between which the head and tail vertices have a different state of the interaction function, we incur the appropriate cost (in this case 1 for hyperarcs $\mathbf{e_3}$ and $\mathbf{e_4}$). Otherwise, the descent into a subtree keeps the

interaction function unmodified and costs nothing ($\mathbf{e_1}$ and $\mathbf{e_2}$). All of the modifications to the recurrence and weight function described in chapter 4 can be encoded in the hypergraph framework, including modifications for self-loops, directed edges, asymmetric interaction gain and loss costs and weighted branch length costs. The first two modifications alter the topology of the resulting hypergraph, while the latter two simply alter the cost function (i.e. the cost for traversing a given hyperarc).

A solution to a particular term in the dynamic programming recurrence corresponds to a *derivation* of its corresponding vertex x. Borrowing the definition of Huang and Chiang, a derivation D of a vertex, the size of a derivation $|D|$ and the cost of a derivation $c\,(D)$ are all defined recursively, as follows:

$$D = \begin{cases} < \mathbf{e}, \epsilon > & \text{if } \mathbf{e} \in \text{BS}(x) \text{ and } |\mathbf{e}| = 0 \\ < \mathbf{e}, D_1 \ldots D_{|\mathbf{e}|} > & \text{otherwise} \end{cases} \tag{5.1}$$

$$|D| = \begin{cases} 1 & \text{if } \mathbf{e} \in \text{BS}(x) \text{ and } |\mathbf{e}| = 0 \\ 1 + \sum_{i=1}^{|\mathbf{e}|} |D_i| & \text{otherwise} \end{cases} \tag{5.2}$$

$$c\,(D) = \begin{cases} \text{given} & \text{if } \mathbf{e} \in \text{BS}(x) \text{ and } |\mathbf{e}| = 0 \\ c\,(\mathbf{e}) + \sum_{i=1}^{|\mathbf{e}|} c\,(D_i) & \text{otherwise} \end{cases} \tag{5.3}$$

In the above equations, D_i is a derivation of $t(\mathbf{e})_i$ — the i^{th} tail node of hyperarc $\mathbf{e}$. Note, in **??**, that for the base case (i.e. leaf nodes of the hypergraph) the cost of a derivation is given, and correspond to the base cases of the recurrence.

Denote by $D(x)^j$ the j-th best (j-th lowest cost) derivation of vertex x. We define a *derivation with back-pointers* as $\hat{D} = \langle \mathbf{e}, \mathbf{i} \rangle$ where $\mathbf{i} \in \mathbb{Z}_{\geq 0}^{|\mathbf{e}|}$. A derivation with back-pointers defines a derivation of a vertex x in terms of the derivation of its preceding vertices. The back-pointer vector $\mathbf{i}$ simply encodes, for each of the vertices in the tail of a hyperarc $\mathbf{e}$, which of the derivations should be used to derive x. The back pointers are simply a more

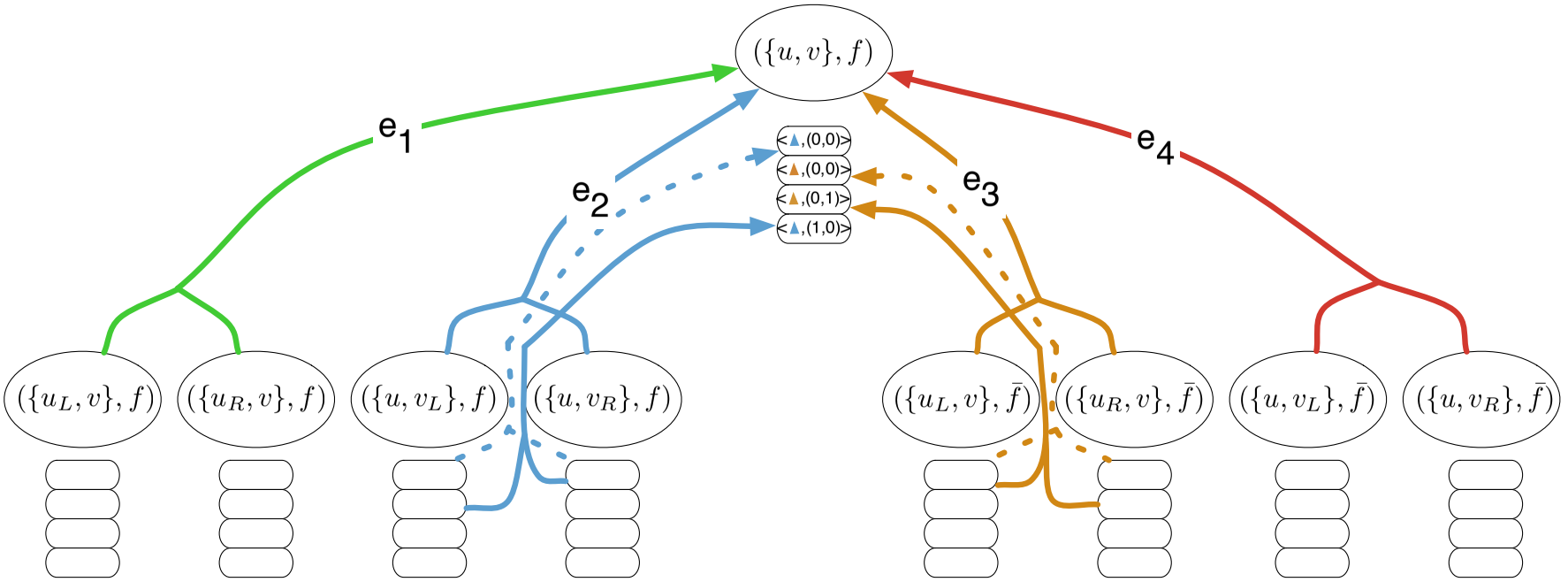


Figure 5.1: *SOPH: Mapping Recurrence to Hypergraph*

An illustration representing a particular recurrence term in the hypergraph. Each hyperarc encodes a set of subterms that must be evaluated to provide a solution to the head vertex $(\{u, v\}, f)$. The arrows denote derivations with back-pointers, and show the 1st (dashed blue), 2nd (dashed orange), 3rd (solid blue), and 4th (solid orange) best derivations of the head vertex, and which derivations of the tail vertices were used to achieve them.

efficient way of encoding a particular derivation, and there is, in fact, a bijection between derivations and derivations with back-pointers defined as follows:

$$\left\langle \mathbf{e}, (D_{j_1}, \ldots, D_{j_{|\mathbf{e}|}}) \right\rangle \rightarrowtail\!\!\!\!\!\rightarrow \left\langle \mathbf{e}, (j_1, \ldots, j_{|\mathbf{e}|}) \right\rangle = \hat{D}$$

We will use the notation $\hat{D}(x)^j$ to denote the j-th best derivation with back-pointers of vertex x.

5.3 Solving the Original Dynamic Program

Having constructed a hypergraph representing the desired recurrence, we can now describe a simple algorithm that solves the original dynamic program presented in chapter 4. In fact, obtaining an optimal solution (not accounting for blocking loops) to the recurrence now reduces to applying the Viterbi algorithm to the representative hypergraph [56].

Though algorithm 3 only yields the cost of the optimal solution, the set of flips can easily be recovered by simply marking, at each vertex x, which of the incoming hyperarcs yielded the optimal score. Since the hypergraph encodes the dependencies between terms of the recurrence, the key to algorithm 3 is to visit the nodes of H in a topological order. A topological ordering of V_H can be obtained by simply performing a postorder traversal of H, and visiting the vertices in this order ensures that the optimal solution for all of the vertices in the backward star of x have been computed before x, itself, is visited. Further, since our hypergraph is acyclic, such a topological ordering is always possible.

Algorithm 3: SOPH: Viterbi Algorithm

input : Hypergraph H, designated root vertex r
output: Optimal cost of deriving r

foreach $x \in V_H$ *in topological order* **do**
    $c_x = \infty$;
    foreach $\mathbf{e} \in BS(x)$ **do**
        $t = \sum_{y \in t(\mathbf{e})} c_y$;
        $c_x = \min(c_x, c(e) + t)$;

return c_r;

Hypergraph Viterbi

5.4 Summing Over Parsimonious Histories

The algorithm presented in Section 5.3 solves the same dynamic programming problem that was introduced in Chapter 4, and thus, suffers from the same shortcomings. In particular, we obtain a single optimal solution from a space of optimal and near-optimal solutions that is potentially enormous. To overcome this limitation, we develop an algorithm that sums the frequency of occurrence of potential interactions over all optimal and near-optimal parsimonious histories. To accomplish this, we will no longer deal with individual solutions/histories, but with cost classes of solutions. Consider a derivation with back-pointers $\langle \mathbf{e}, \mathbf{i} \rangle$, with cost $c(\langle \mathbf{e}, \mathbf{i} \rangle)$, deriving a particular vertex x. The cost class of $\langle \mathbf{e}, \mathbf{i} \rangle$ is the set of all derivations of x having the same total cost; that is

$$\left[\hat{D}(x)\right] = [\langle \mathbf{e}, \mathbf{i} \rangle] = \{\langle \mathbf{e}', \mathbf{i}' \rangle \mid c\left(\langle \mathbf{e}', \mathbf{i}' \rangle\right) = c\left(\langle \mathbf{e}, \mathbf{i} \rangle\right) \wedge h(\mathbf{e}) = h(\mathbf{e}')\}. \tag{5.4}$$

The chosen notation denotes that the cost classes define an equivalence relation on the set of derivations and derivations with back-pointers. Just as with derivations with back-pointers, we denote the j-th best cost class of a vertex as $\left[\hat{D}(x)\right]^i$. Finally, the notation $\hat{D}(x)_s$ also denotes a cost class of vertex x. Specifically, it denotes the cost class having cost s; that is $\hat{D}(x)_s = \left[\hat{D}(x)\right]^i$ where $c\left(\left[\hat{D}(x)\right]^i\right) = s$.

We now want to accumulate the top-k cost classes of the root r of the hypergraph.

In general, this constitutes many more than the top k individual solutions, because there are many ways to obtain different solutions of equivalent cost. The key to developing an efficient algorithm for this task is to realize that we can count all derivations belonging to the top-k score classes of a vertex without enumerating them.

Consider a generic vertex x in the hypergraph, and assume that, for all preceding vertices of x, we have computed their top j cost classes. It is then possible to compute the top j cost classes for x. First, note that since the cost function is monotonically increasing, the lowest cost solution that can be obtained via hyperarc $\mathbf{e}$ is the sum of the cost of traversing $\mathbf{e}$ plus the cost of the best solution classes for each of the vertices in the tail of $\mathbf{e}$.

Given a monotonically increasing cost function and a derivation with back-pointers $\langle \mathbf{e}, \mathbf{i} \rangle$, the succeeding (i.e. next-best) cost class yielded via a hyperarc $\mathbf{e}$ will always reside in the *neighborhood* of $\langle \mathbf{e}, \mathbf{i} \rangle$; this is the essential observation behind so-called cube pruning and cube growing approaches [56, 51] for enumerating k best derivations. Denote by b_ℓ the vector having a 1 in its ℓ-th position and a 0 everywhere else. Then, we define the neighborhood of $\langle \mathbf{e}, \mathbf{i} \rangle$ as $\mathcal{N}(\langle \mathbf{e}, \mathbf{i} \rangle) = \{\langle \mathbf{e}, \mathbf{i} + \mathbf{b}_\ell \rangle\}_{\ell=0}^{|\ell|-1}$.

This means that we can efficiently enumerate the top k cost classes for a vertex x by maintaining a priority queue of the potential best derivations. The queue is initially populated with $\{\langle \mathbf{e}, \mathbf{0} \rangle\}_{\mathbf{e} \in \mathrm{BS}(x)}$. When a derivation is removed from the queue, its neighbors are added to the queue, and this process continues until all derivations have been exhausted or until the top k cost classes have been enumerated. The process can be made even more efficient using the faster cube pruning approach introduced by Gesmundo and Henderson [51], which partially orders derivations to ensure that for each derivation, only a single of its potential predecessors will attempt to add it to the queue.

Let $\langle \mathbf{e}, \mathbf{i} \rangle$ be the j-th best derivation using hyperarc $\mathbf{e}$. We can define the count of this derivation as follows:

$$\#\left(\langle \mathbf{e}, \mathbf{i} \rangle\right) = \prod_{\ell=0}^{|\mathbf{i}|-1} \#\left(\left[\hat{D}(t(\mathbf{e})_\ell)\right]^{\mathbf{i}_\ell}\right). \tag{5.5}$$

That is, the number of ways we can obtain the score $c\left(\langle \mathbf{e}, \mathbf{i} \rangle\right)$ using hyperarc $\mathbf{e}$ is the product of the sizes of the cost classes used from each of the sub-derivations in $\langle \mathbf{e}, \mathbf{i} \rangle$. The cost of $\langle \mathbf{e}, \mathbf{i} \rangle$ when considering cost classes is the same as when considering individual derivations, and remains unchanged from **??**; this is because, by definition, all of the solutions belonging to the same cost class have the same cost.

Finally, to obtain the count of a cost class of derivations for a particular vertex x, we can merge the equivalent cost classes over all incoming hyperarcs. Let $\#\left(\mathbf{e}_s\right)$ be the number of ways of deriving $h(\mathbf{e})$ using hyperarc $\mathbf{e}$ at a cost of s. Then the count of the cost class of vertex x having score s is given as follows:

$$\#\left(\hat{D}(x)_s\right) = \sum_{\mathbf{e} \in \mathrm{BS}(x)} \#\left(\mathbf{e}_s\right). \tag{5.6}$$

This equation simply stipulates that whenever different incoming hyperarcs of x have cost classes of the same cost s, their counts are additively combined to obtain the cardinality of the cost class of x at this cost.

5.4.1 The Up-Down Algorithm

We develop an algorithm, called the "up-down" algorithm, to compute the frequency with which ancestral states occur in the ensemble of optimal and near-optimal solutions. The algorithm has two phases. The first (up) phase computes the top k cost classes at each vertex and across each visited hyperarc. This provides information about the costs of these classes and their cardinalities to the second (down) phase of the algorithm. The down phase computes the frequency of occurrence of each vertex and hyperarc that participates in the near-optimal ensemble of solutions. A probability mass of 1 is given for deriving the root vertex r via some considered solution, and this probability mass is divided up among the vertices in the solution ensemble by considering the weighted frequency with which they participate in optimal and near-optimal solutions.

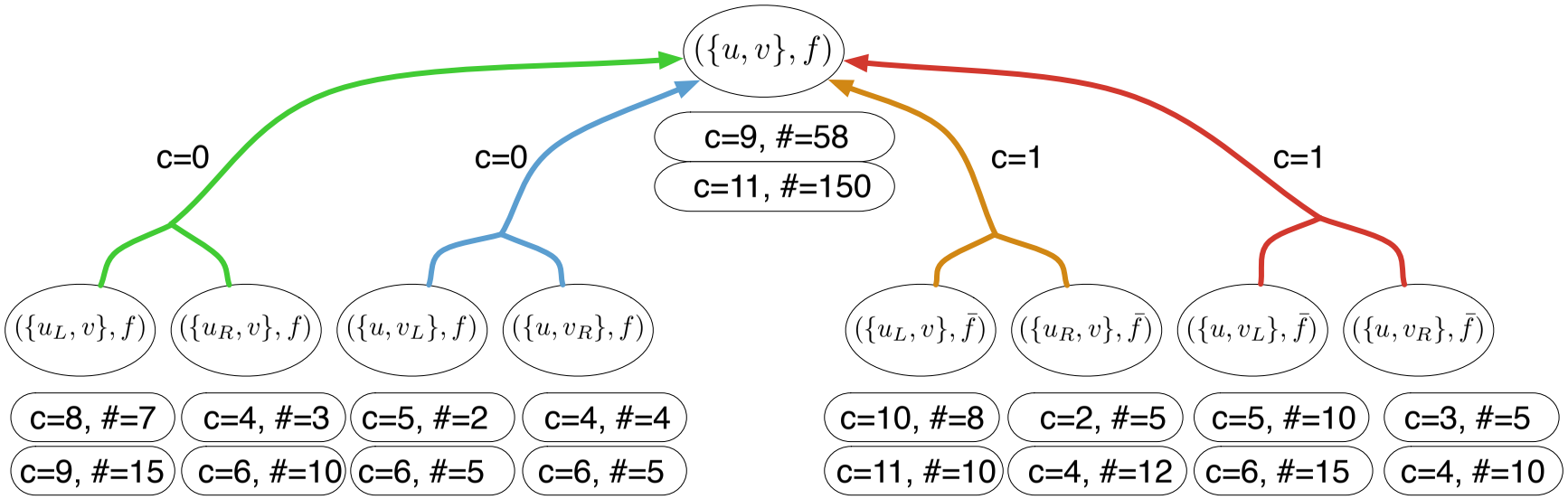


Figure 5.2: *SOPH: Up Phase Example*

An example from the up phase of the algorithm, in which the top 2 cost classes of vertex $(\{u, v\}, f)$ are computed from the top 2 cost classes of its preceding vertices. The top cost class of $(\{u, v\}, f)$ has a cost of 9, and can be derived via the blue edge (in 8 ways) and the red edge (in 50 ways). These derivations of these edges are combined when computing the top cost class for $(\{u, v\}, f)$. The second cost class has a cost of 10, and can be obtained in 100 different ways, but only via the red edge.

Algorithm 4: SOPH: Top-k Algorithm

input : x, k, Q
output: Vector $\mathbf{C}$ of the top k cost classes of x

$\mathbf{C} = \emptyset$;
while $|Q| > 0$ *and* $|\mathbf{C}| < k$ **do**
  $\langle \mathbf{e}, \mathbf{i} \rangle = Q$.pop();
  $L = |\mathbf{C}| - 1$;
  `/* If this dbp has the same cost as the last */`
  if $c(\langle \mathbf{e}, \mathbf{i} \rangle) = c(\mathbf{C}[L])$ **then**
    `/* Then merge their counts */`
    merge($\mathbf{C}[L], \langle \mathbf{e}, \mathbf{i} \rangle$);
  else
    `/* Else, create and append a new cost class */`
    append($\mathbf{C}$, CostClass($\langle \mathbf{e}, \mathbf{i} \rangle$));
  foreach $\langle \mathbf{e}, \mathbf{i}' \rangle \in \mathcal{N}(\langle \mathbf{e}, \mathbf{i} \rangle) \setminus Q$ **do**
    Q.append($\langle \mathbf{e}, \mathbf{i}' \rangle$);
return $\mathbf{C}$;
TopKCostClasses

Up Phase The up phase of the algorithm (algorithm 5) traverses H in topological order, computing the top k cost classes at each vertex. Since the cost function is monotonically increasing within each edge, we can assure that, to obtain the top k score classes at a vertex x, it will always be sufficient to have computed the top k cost classes for all of x's preceding vertices. The up phase of our algorithm is very similar to algorithm 2 from [56], except that cost classes of derivations with equivalent costs using different hyperarcs are merged. A simple example from the up phase of the algorithm is illustrated in Figure 5.2.

Down Phase The down phase of the algorithm (algorithm 6) traverses H in reverse topological order. The root vertex r is given a probability mass of 1, and the purpose of this phase of the algorithm is to determine how this probability mass should be distributed over the vertices and hyperarcs that participate in the ensemble of optimal and near-optimal solutions. For each cost class $\hat{D}(x)_s$ of cost s, it's fraction of the total contribution is given

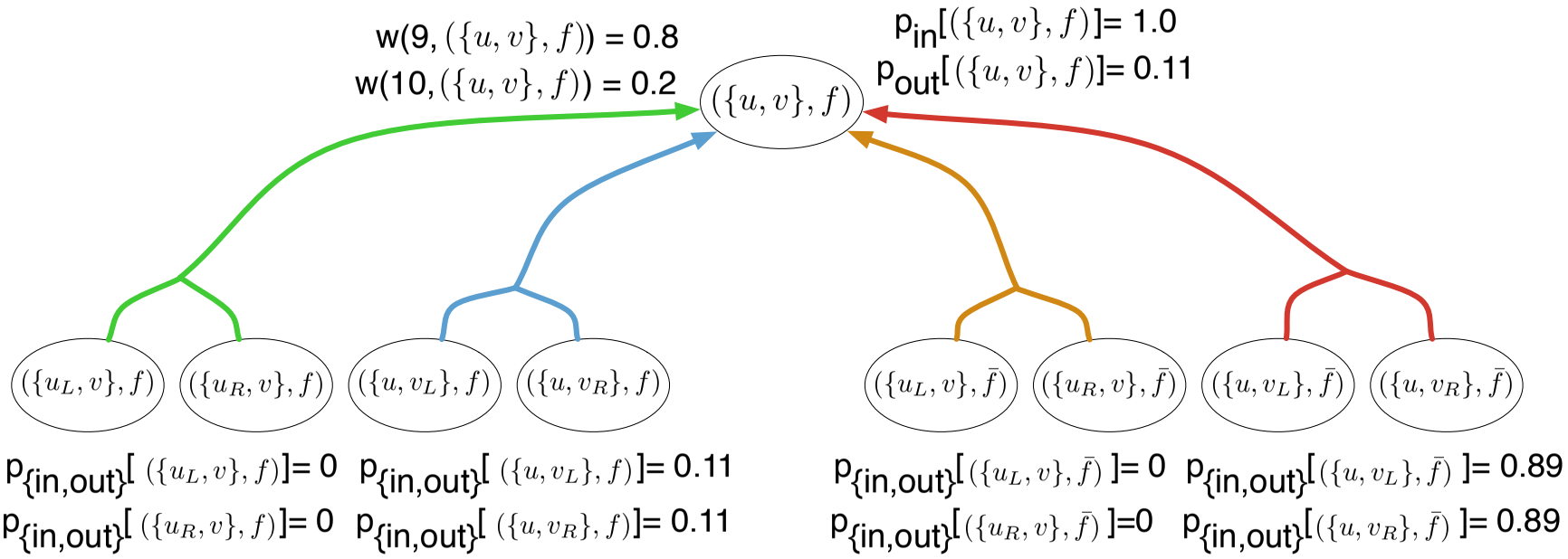


Figure 5.3: *SOPH: Down Phase Example*

An example from the down phase of the algorithm. The vertex $(\{u, v\}, f)$ is given an *a priori* "in" probability of 1. In this example, the first cost class is attributed 80% of this probability mass while the second cost class is attributed 20% of the probability mass. Given the cost classes shown in Figure 5.2, the "in" and "out" probability for all of the vertices preceding $(\{u, v\}, f)$ are shown. Note that for leaf nodes (i.e. those with no incident hyperarcs) the "out" probabilities and the "in" probabilities are equal.

Algorithm 5: SOPH: Up Phase Algorithm

input : Hypergraph H, designated root vertex r
output: Top k cost classes for all vertices in V_H

```
/* We know all potential solutions for the leaves   */
```
foreach $x \in \mathit{Leaves}(H)$ **do**
  populate the top k score classes of x.
$\mathbf{C} = \emptyset$;
foreach $x \in V_H$ *in topological order* **do**
  $Q = \emptyset$;
  foreach $\mathbf{e} \in \mathit{BS}(x)$ **do**
    Q.append($\langle \mathbf{e}, \mathbf{0} \rangle$);
  $\mathbf{C}[x] =$ `TopKCostClasses` (x, k, Q);
return $\mathbf{C}$;
Up Phase

by the weight function $w(s, x)$ described in **??**), and the probability mass assigned to all derivations of x of cost s is divided proportionally among the incoming hyperarcs. That is, the fraction of probability mass contributed to hyperarc $\mathbf{e}$ by cost class $\hat{D}(x)_s$ is simply the number of derivations of cost s using $\mathbf{e}$ divided by the total number of derivations of this cost. Each hyperarc will receive probability mass in this manner from each of the cost classes in which it participates. Finally, the probability mass assigned to a vertex is the sum over all hyperarcs for which it is contained in the tail of the probability mass of the head vertex of the hyperarc times the total probability of traversing that hyperarc. However, we do not want to simply distribute the probability over the top k cost classes for each vertex, because not all of these cost classes may contribute to solutions deriving the top k cost classes of the root. We define the *frontier* of a cost class $\hat{D}(x)_s$ and hyperarc $\mathbf{e} \in \hat{D}(x)_s$ to be the maximum index, over all derivations in the cost class, of the back pointer to each tail node of $\mathbf{e}$. More formally:

$$\textbf{frontier}(\hat{D}(x)_s, \mathbf{e}) = \left(\max_{\langle \mathbf{e}, \mathbf{i} \rangle \in \hat{D}(x)_s} (\mathbf{i}_j) \right)_{0 \leq j < |t(\mathbf{e})|} \tag{5.7}$$

As a slight abuse of notation, we will allow **frontier**$(\hat{D}(x)_s, \mathbf{e})$ to be addressed by both the index of a tail vertex in $\mathbf{e}$, as well as by the vertex y itself. By starting at the root vertex and keeping track of the appropriate frontiers, we can distribute probability mass over only those cost classes of each vertex that are used to derive a top k cost class of the root.

We must also decide how the probability mass at a particular vertex should be distributed over the solutions in each of its cost classes. For example, how much more likely is a history that belongs to the optimal cost class than one that belongs to the second best, or more generally, the i^{th} best cost class. If there is only a single cost class, all of the probability mass is assigned to the solutions from the class. Otherwise, these weights are distributed among the cost classes using following equation and are based on a user-provided parameter γ:

$$w(s, x) = \frac{\exp(\gamma \frac{x_{\min} - s}{x_{\max} - x_{\min}})}{\mathcal{Z}}, \tag{5.8}$$

where

$$\mathcal{Z} = \sum_s \exp(\gamma \frac{x_{\min} - s}{x_{\max} - x_{\min}})$$

is a normalizing constant, and $x_{\min}$ and $x_{\max}$ are shorthand for the minimum and maximum costs for the computed cost classes of vertex x.

Finally, notice that in Algorithm 6, we keep track of two distinct probabilities for each vertex, which we call the "in" and "out" probabilities. As the output of the algorithm, we are interested only in the "out" probabilities, but both must be computed. The "in" probability quantifies the relative frequency, among parsimonious histories, of entering the vertex x corresponding to a particular state (e.g. $(\{u, v\}, f)$). However, the derivations of this vertex will often traverse an incoming hyperarc that represents changing the state of the ancestral interaction between u and v. For example, if we derive $x = (\{u, v\}, f)$ via hyperarc $\mathbf{e}$, where the state of the interaction function for vertices in $t(\mathbf{e})$ is $\bar{f}$, then the probability mass for deriving x via $\mathbf{e}$ should really be given to the state $\left(\{u, v\}, \bar{f}\right)$. This is because the derivation via $\mathbf{e}$ actually implies an ancestral history in which the flip from f to

$\bar{f}$ occurred between u and v. However, as the "in" probabilities are necessary to compute the "out" probabilities, we keep track of both of them in the down phase of the algorithm. We use the **res** function to determine to which state's "out" probability a vertex / hyperarc pair should contribute. For leaf vertices (i.e. those with no incident hyperarcs), the "out" probabilities are set equal to the "in" probabilities. Let $x = (\{u, v\}, f)$ be an arbitrary vertex in the hypergraph, and $\mathbf{e}$ be a hyperarc in $\mathrm{BS}(x)$, then we define

$$\mathbf{res}(x, \mathbf{e}) = \begin{cases} (\{u, v\}, f) & \text{if the state of the interaction function is } f \text{ for } y \in t(\mathbf{e}) \\ \left(\{u, v\}, \bar{f}\right) & \text{otherwise} \end{cases}. \tag{5.9}$$

A simple example from the down phase of the algorithm is illustrated in Figure 5.3.

Algorithm 6: SOPH: Down Phase Algorithm

input : Hypergraph H, with top-k cost classes.
output: Relative frequencies, over parsimonious histories, for all ancestral states.

`/* Probabilities and frontiers start at 0 */`
$p_{\text{in}} = \mathbf{0}$;
$p_{\text{out}} = \mathbf{0}$;
maxClass $= \mathbf{0}$;
`/* The root uses k cost classes and has prob. 1 */`
maxClass$[r] = k$;
$p_{in}[r] = 1.0$;
foreach $x \in V_H$ *in reverse topological order* **do**
  $p_{\text{arc}} = \mathbf{0}$;
  foreach $0 \leq i <$ *maxClass*$[x]$ **do**
  $s = c\left(\left[\hat{D}(x)\right]^i\right)$;
  $p_s = w(s, x)$;
  foreach $\mathbf{e} \in \left[\hat{D}(x)\right]^i$ **do**
    $p_{\text{arc}}[\mathbf{e}] = p_{\text{arc}}[\mathbf{e}] + (p_s \frac{\#(\mathbf{e})_s}{\#\left(\hat{D}(x)\right)_s})$;
    $f_{\mathbf{e}} = \text{frontier}(\hat{D}(x), \mathbf{e})$;
    foreach $y \in t(\mathbf{e})$ **do**
      maxClass$[y] = \max(\text{maxClass}[y], f_{\mathbf{e}}(y) + 1)$;
  foreach $0 \leq i <$ *maxClass*$[x]$ **do**
    $s = c\left(\left[\hat{D}(x)\right]^i\right)$;
    foreach $\mathbf{e} \in \left[\hat{D}(x)\right]^i$ **do**
      $z = \textbf{res}(x, \mathbf{e})$;
      $p_{\text{out}}[z] = p_{\text{out}}[z] + p_{\text{in}}[x] * p_s^x * p_{\text{arc}}[\mathbf{e}]$;
      foreach $y \in \mathbf{e}$ **do**
        $p_{\text{in}}[y] = p_{\text{in}}[y] + p_{\text{in}}[x] * p_{\text{arc}}[\mathbf{e}]$;
Down Phase

5.5 Results

We test the performance of our approach on two different tasks. First, we again consider the reconstruction of the ancestral interaction networks for the bZIP family of proteins. However, in addition to the experiments performed in Chapter 4, we also consider the performance of our new method under the addition of simulated noise to the input data. How the method behaves under the presence of noise is particularly important, given the high rate of false-positive and false-negative interactions that occur in experimentally measured data [118]. For all experiments presented in this section, we consider the top $k = 100$ cost classes and set γ, the parameter that determines the relative weight of the different cost classes to $k/2 = 50.0$.

5.5.1 Reconstructing bZIP Networks in the Presence of Noise

The reconstruction of the ancestral network state for the bZIP family of proteins was first undertaken by Pinney et al. [105]. The bZIP transcription factors make an enticing set of data on which to test methods for ancestral network reconstruction because the interactions between these transcription factors are strongly mediated by their coiled-coil leucine zipper domains, and the strength of these interactions can be computationally predicted with high sensitivity and specificity using sequence alone [45]. This means that the interaction affinity of ancestral proteins can be estimated with reasonably high confidence by first estimating the ancestral sequence and then performing a sequence-based prediction of the interaction affinity between the ancestral protein sequences. This sequence-based method was used to predict the interaction strength between both extant and inferred ancestral bZIP proteins sequences. These interaction affinities were used to generate both the input data (i.e. the extant interactions) as well as the "ground truth" ancestral interactions [105].

We consider the ancestral network reconstruction problem on the bZIP family of transcription factors under three different sets of input data. The original data consists of interaction scores as predicted by the software of Fong et al. [45]. This software computes

Table 5.1: *SOPH: Results SOPH vs. Probabilistic Approach*

The relative performance of our sum over parsimonious histories (SOPH) approach and the probabilistic method described by Pinney et al. in reconstructing the ancestral interaction networks we consider.

Ancestor	Method	F1-Score ($\sigma = 0, 10, 20$)	BEDROC ($\sigma = 0, 10, 20$)
Teleost	SOPH	$0.84, 0.77, 0.70$	$0.90, 0.88, 0.84$
	Probabilistic	$0.79, 0.68, 0.58$	$0.82, 0.73, 0.69$
Vertebrata	SOPH	$0.88, 0.79, 0.73$	$0.92, 0.93, 0.87$
	Probabilistic	$0.82, 0.72, 0.61$	$0.92, 0.83, 0.76$
Chordata	SOPH	$0.77, 0.72, 0.68$	$0.89, 0.86, 0.75$
	Probabilistic	$0.75, 0.71, 0.62$	$0.88, 0.85, 0.60$

a score for each pair of proteins which predicts the affinity of their potential interaction. Higher scores are assigned to pairs of proteins where the model predicts a greater propensity for a strong interaction between these proteins. All data is binarized by creating an input interaction for all pairs of proteins where the interaction score is greater than or equal to 30.6 (the score for which the probability of an interaction existing given the score is 0.5) [105]. To create noisy versions of the data, Gaussian noise with mean 0 and standard deviations of 10 and 20 was added to the data and the resulting interaction scores binarized. For each of the noise levels of the input data (0, 10 and 20) we reconstruct three ancestral networks — Teleost (ancestor of *D. rerio* and *T. rubripes*), Vertebrata (ancestor of *D. rerio*, *T. rubripes* and *H. sapiens*) and Chordate (ancestor of *D. rerio*, *T. rubripes*, *H. sapiens*, and *C. intestinalis*).

To measure the quality of the ancestral network reconstruction, we consider two separate metrics, the F1-Score (the harmonic mean of the precision and recall), and the BEDROC score [127]. The BEDROC metric is an AUC metric meant to deal with the so-called early enrichment or early recognition problem. Intuitively, the BEDROC metric weights the accuracy more heavily early on in the retrieval list.

Table 5.1 demonstrates the performance of our ancestral network reconstruction procedure compared to the probabilistic model used by Pinney et al. [105]. We find that our

method outperforms the probabilistic method under both of the metrics that we consider. The truly interesting trend, however, is the growing difference in performance as the data becomes noisier. In the no noise case, the performance difference between the two methods is small, suggesting they both perform reasonably well on this dataset given high quality input. As the noise increases, so does the performance gap between the two methods. In fact, with the exception of the Chordata network, the SOPH approach is more accurate at a noise level of 20 than the probabilistic method is at a noise level of 10.

These results suggest the potential benefit of employing the sum over parsimonious histories approach to the ancestral network reconstruction problem, especially on real data, where the input may be very noisy and the false-positive and false-negative rates very high. More generally, the results demonstrate that the probabilistic method, though potentially more robust to noise than the naïve parsimony approach, is not inherently superior in this aspect to advanced (i.e. ensemble) methods based on parsimony. By exploring all optimal and near-optimal parsimonious histories, our method is able to overcome one of the main shortcomings of previous parsimony-based approaches and to provide substantially better performance, in most cases, than any of the pre-existing methods.

5.5.2 Imputing Missing Interactions

The down phase of our algorithm generates scores not only for ancestral interactions, but also for extant interactions. That is, given the structure of the duplication forest and the extant interactions, we obtain a score for each potential extant interaction, quantifying how much we expect this interaction to exist. One way to view these scores is as a parsimony weighted smoothing of the input data. This suggests that we may use the output scores of potential interactions to identify specific interactions that we would or would not expect to see given the duplication histories and the rest of the observed interactions. For all experiments discussed in this section, we computed the top $k = 100$ cost classes and set $\gamma = 1.0$.

Missing Data Cross Validation

To test the ability of the parsimony scores to predict potential extant interactions, we consider a set of leave-one-out cross validation experiments. We use the herpesviral protein interaction networks of Fossum et al. [47]. In particular, they consider the whole proteome interaction networks of 5 different herpes viruses the Epstein-Barr virus (EBV), herpes simplex virus 1 (HSV-1), murine cytomegalovirus (mCMV), Kaposi's sarcoma-associated herpesvirus (KSHV), and the varicella-zoster virus (VZV). Together, these viruses span the α, β, and γ herpesvirus subfamilies and represent a sampling of viruses which have diverged substantially since the speciation of their common ancestor about 400M years ago [83, 84]. Despite this divergence, there is still a set of core orthologs which are present in all of the species.

To generate the data for our experiments, we use the species tree representing the relationships between the 5 herpes virus species given by [83, 84]. For each of the proteins in the core orthology groups assigned by Fossum et al. [47], we obtained the sequences from the UniProt database [27]. We then constructed gene trees for each of the orthology groups using PyCogent [66]. Finally, the gene trees were rooted, reconciled with the species tree and rearranged using the Notung 2 software [35, 129] with the default parameters.

Given the reconciled gene trees for each orthology group and the high-confidence interactions reported by Fossum et al. [47], we perform the following experiment. Let O denote the set of orthology groups, and for each pair (a, b) of groups in $O \times O$, let I_{ab} denote the set of interactions between groups a and b. For each pair (a, b) of orthology groups where $|I_{ab}| > 1$, we consider each interaction i in I_{ab} in turn, and remove i while leaving the remaining interactions fixed. This yields a problem instance for our algorithm consisting of the reconciled trees T_a and T_b for orthology groups a and b, and the set of interactions $I_{ab} \setminus \{i\}$. We run our algorithm on this instance, and record the score assigned to each potential interaction. We sort the potential interactions according to their probabilities, and report the relative rank of i, the left-out interaction, among the list of extant,

non-input interactions. In other words, let L_a and L_b denote the leaf nodes of T_a and T_b (not considering nodes marked as lost by the reconciliation algorithm). Then, we consider all potential interactions $i' \in P_{ab}$, where $P_{ab} = (L_a \times L_b) \setminus (I_{ab} \setminus \{i\})$, and sort them in ascending order according to their assigned scores. We compute the relative rank of i in this list as $\text{rank}_{\text{rel}}(i) = \text{rank}(i) / |P_{ab}|$.

One consideration to note is that we look at the scores assigned to the potential interactions in ascending, not descending, order of their probabilities. This means that we are considering those potential interactions as highly ranked which have low, not high, scores. At first, this may seem counterintuitive. However, the reason we want to look for low-scoring rather than high-scoring potential interactions is because we are interested in those which are surprising in light of the structure of the duplication histories and other extant interactions. For example, if there is a pair of proteins in one species whose orthologs all interact in evolutionarily close species, but we observe no interaction between this pair, it represents a surprising and somewhat unparsimonious scenario. We expect the weighted score of this potential interaction over a sum of parsimonious histories to be low, not high. Thus, when computing the relative ranks of potential interactions, we sort them in order by their scores to look for the most surprising missing interactions.

Ideally, given the supporting evidence for the left-out interaction in terms of the structure of the gene trees and the remaining interactions, our algorithm will compute a probability for the left-out interaction that is relatively high with respect to the other potential interactions, resulting in small relative rank. The relative rank is always in the range of 0 to 1 (inclusive), and if the ranks were assigned randomly, we would expect the left-out interaction to have relative rank of 0.5 on average. We find that, across all homology groups, the relative ranks computed by our algorithm for the left-out interactions are substantially lower than we would expect by chance, with an average relative rank of 0.317.

We further characterize the benefit obtained by using our method to impute interactions

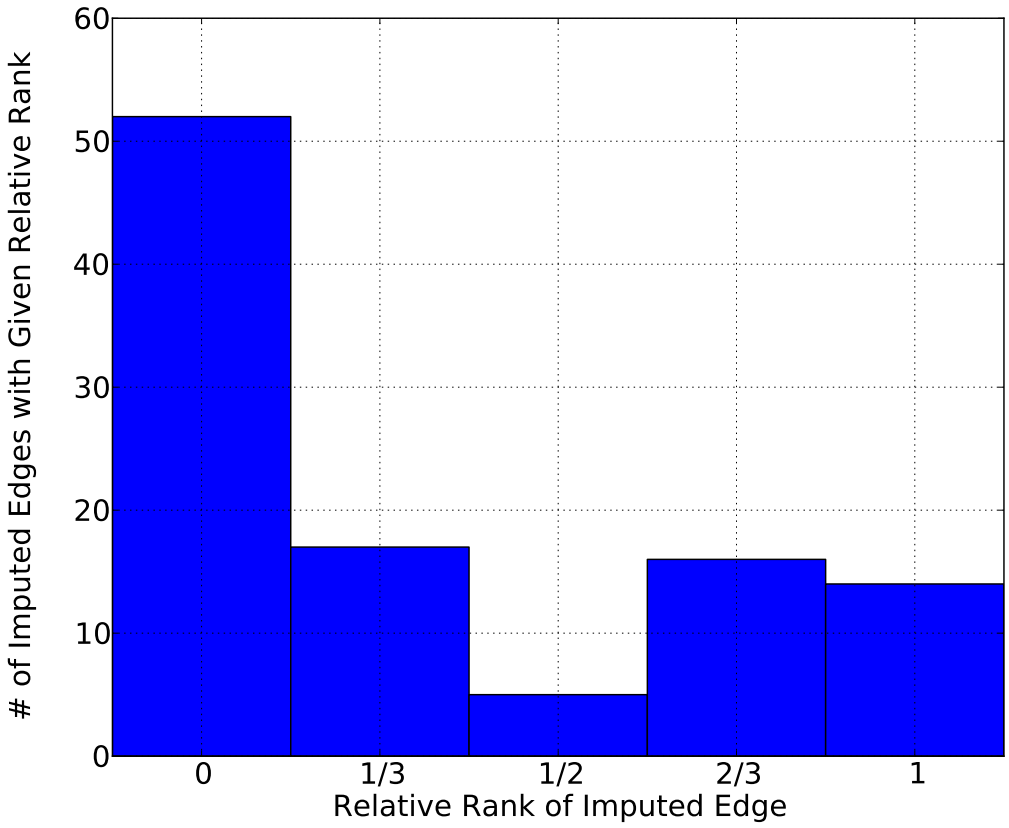


Figure 5.4: *SOPH: Imputing Missing Interactions*

A histogram of the relative ranks of the "left-out" edge in our cross-validation experiments.

in two ways. First, we compute the histogram of the relative rank of the left-out interaction over all (104) experiments. This histogram is illustrated in Figure 5.4. Observe that about half of all interactions have a relative rank of 0; meaning that they occur first in their respective lists. The distribution dips around 0.5; though this is likely due more to the data than the method, as the majority of retrieval lists have length 3, making a relative rank of 0.5 impossible. Finally, some number of interactions have relative ranks greater than 0.5. These cases are likely due to interactions which are surprising from an evolutionary perspective, or simply a result of the sparsity of the input dataset. In particular, since the experimental dataset used to perform these tests is hypothesized to have a relatively high false-negative rate itself [47], it is likely the case that the evolutionary evidence to improve

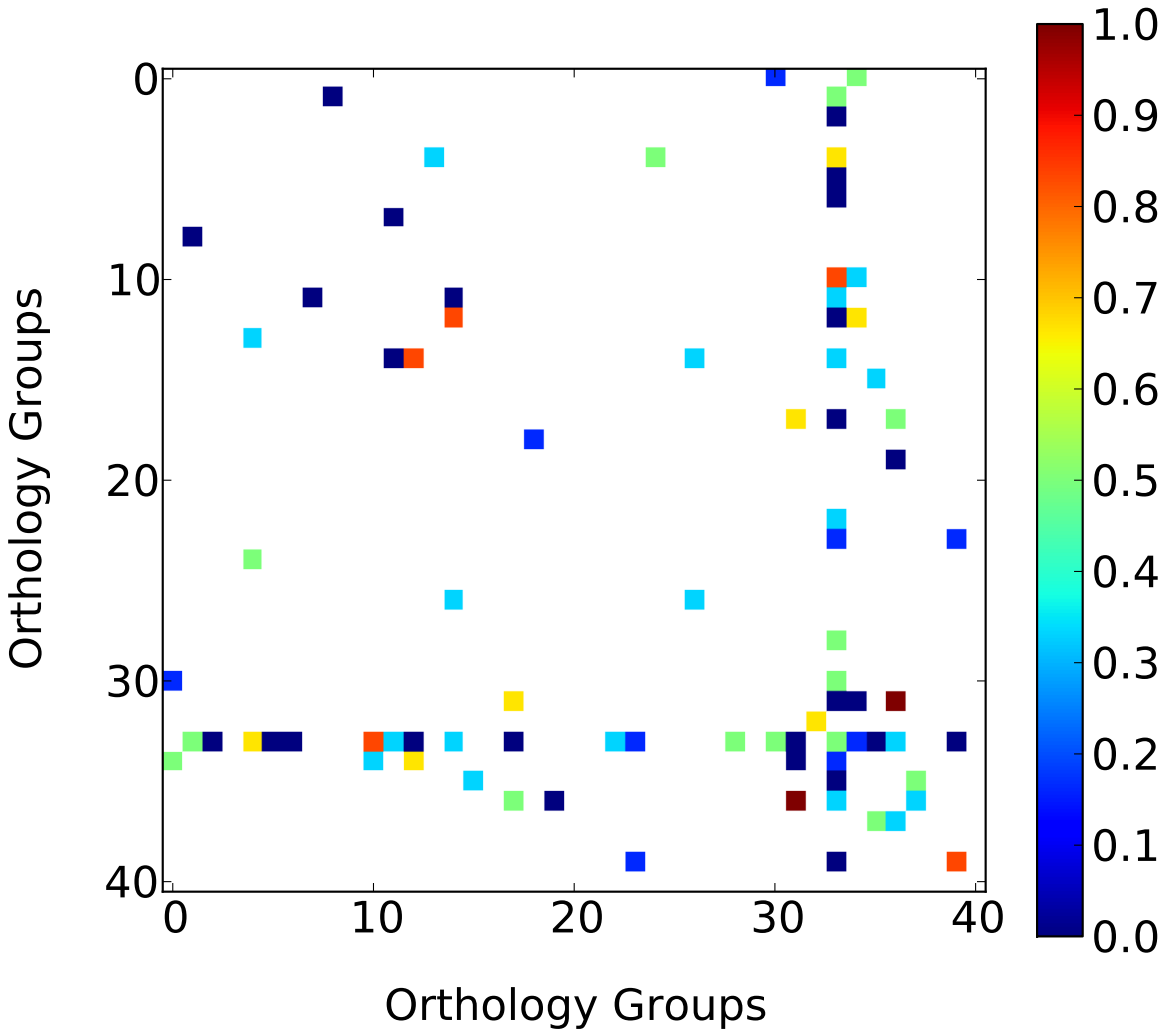


Figure 5.5: *SOPH: Imputing Missing Interactions (Per-group)*

A heatmap of the average relative ranks of the "left-out" edge between pairs of orthology groups. Note that while for most groups, the relative rank is substantially lower than one would expect by chance, certain pairs of groups (e.g. $(31, 36)$) exhibit an average relative rank higher than would be expected by chance.

the prediction of edges much more is simply missing.

In addition to the coarse grained view of the results provided by the histogram, Figure 5.5 provides a heatmap of the average relative rank of imputed interactions between pairs of homology groups. Recall that to perform an experiment, we require a pair of interaction groups to have at least two interactions between them; otherwise, removing the single interaction between the two groups would erase all evidence, making the prediction task pointless. Due to the sparsity of the initial data and the presumed low density of the true interaction networks, many pairs of homology groups contain one or zero interactions between them (they appear white in Figure 5.5) and are left out of the experiment. Among the remaining groups, we again notice a somewhat bimodal distribution of relative ranks. Between many pairs of groups, the missing interactions can be perfectly imputed (relative rank of 0), while between others the task seems incredibly difficult (e.g. between groups 31 and 36 the average relative rank of the left-out edge was 1). Again, this suggests that when there is sufficient evolutionary evidence, missing interactions can be imputed with high accuracy. Because we don't have a true gold-standard set of interactions, we cannot reliably hypothesize whether the imputed interactions with large relative ranks are due to a failure of the method (i.e. evolutionarily non-parsimonious interactions) or simply false-negatives in the input data.

5.6 Conclusion

In this chapter, we introduced a novel sum over parsimonious histories method for solving the ancestral network reconstruction problem. It addresses the shortcoming of the method presented in Chapter 4 by considering, rather than a single history, a weighted ensemble consisting of all optimal and near-optimal parsimonious histories. We show that this makes the results robust to the presence of noise in the input (Section 5.5.1), and allows our parsimony approach to outperform the probabilistic approach to ancestral network reconstruction [105] at all considered noise levels. Further, we observe that, as the noise

level increases, so does the gap in performance between our method and the probabilistic approach; suggesting the desirability of applying our method to experimental data, which is known to contain high levels of noise.

We also test the ability of our method to impute potentially missing extant interactions in a set of herpesviral protein interaction data (Section 5.5.2). We find that our method can reliably exploit evolutionary evidence for the existence of missing interactions; recalling the true missing interaction in a ranked list of potential interactions significantly sooner than would be expected by chance. Our method, therefore, may be useful in prioritizing low-throughput but high-accuracy protein interaction experiments by suggesting which interactions are more likely than others to exist given the current experimental and evolutionary evidence.

As future work, we'd like to extend the framework to handle evolutionary events for genes aside from duplication and loss. Though duplication and loss constitute the majority of events by which the genome (and hence the protein network) evolves, rarer events — specifically horizontal gene transfer and novel gene birth — can play a significant role. By extending our combinatorial encoding of interaction histories to include these rarer events, and by augmenting our algorithms to account for them, we will likely be able to improve the accuracy of our ancestral network inference even further.

Chapter 6

A Non-parametric Framework for Learning Network Growth Models

This chapter is based on the paper "The Missing Models: A Data-Driven Approach for Learning How Networks Grow", written in collaboration with G. Duggal, E. Sefer, H. Wang, D. Filippova and C. Kingsford, that will appear in the Proceedings of KDD 2012.

6.1 Introduction

In this chapter, we develop a novel framework to represent network growth models and propose a non-parametric method to learn new models of network growth by matching user-specified properties of real-world networks. While the approaches in Chapters 4 and 5 allowed us to reconstruct the growth process of specific biological networks and ask questions about the state of their ancestors, we often wish to characterize the temporal evolution of networks on a larger and more general scale. One successful approach to understanding the general principles of network growth is via the creation of idealized network models such as the forest fire model [75], the Kronecker model [76], duplication/mutation models [9, 57, 116, 121, 128], preferential attachment models [e.g. 8], and others [18, 33, 62, 73, 110]. These models provide a probabilistic and mechanistic way to describe the growth of particular classes of networks, generally in terms of combinations of simple operations such as node and edge creation and deletion, node duplication, node expansion (replacing a node by a subgraph), or influence propagation. Such models are particularly important in understanding the emergence of topological characteristics such

as shrinking diameter, assortativity or disassortativity, and modularity as networks grow. In fact, the creation of network growth models to explain the processes by which biological, social, and technological networks have evolved over time represents a substantial body of research, and has become increasingly central to gaining insight into how these networks function.

In addition to providing an idealized simulation of real-world growth, network models have a number of other uses—some of which do not even require that the models themselves are interpretable. For example, they can serve as null models for the detection of statistically surprising topological features in a graph, can be used for large-scale performance testing for time-consuming graph algorithms, can aid in reconstructing ancient networks [92], and can help with anonymization [76].

Early theoretical work on network models began in 1960 with the Erdős-Rényi model [38]. Subsequent work identified a scale-free degree distribution [8] and small-world property [130] as common features of real-world networks and produced models that generated them. Later models incorporated other network properties as objectives in various domains, such as shrinking diameter of a growing social network [75] (the forest fire model) and clustering coefficient for biological protein interaction networks [128] (the duplication, mutation, with complementarity or DMC model). Subsequent efforts [e.g. 1, 97] have attempted to manually design models that fit various additional features simultaneously in order to produce more realistic networks. Recently, there has also been work on models that attempt to match not only the topology of real networks, but also richer features such as node attributes [119, 61]. The creation of a parsimonious, plausible, and well-fitting growth model is typically a challenging task, and as more varied, large-scale networks are studied, new important properties will be identified, requiring new models to be developed. However, hand-crafted models can match desired topological properties only as well as the creativity and persistence of the model designer allow.

Here, we introduce a formal representation that can encode many of the most commonly

studied growth models, as well as many other models yet to be discovered. We also present an optimization framework that allows for the automatic discovery of new models fitting desired properties within this formal representation. These learned models can be used to generate large classes of exemplar networks that match input features well. They are also interpretable (with some amount of effort). Often, because the framework can generally find many distinct models that match the desired properties, the set of models itself can be minded for motifs that are effective at generating a particular property. Additionally, the ease with which good-fitting models can be found can be used as a measure of the ubiquity of that feature among graph growth mechanisms. Finally, in many cases, the computationally optimized models match real-world properties better than hand-crafted models.

Very little previous work has addressed the challenge of automatic design of network models. Some previous frameworks are capable of adapting models to new data by re-estimating parameters that govern the network growth process. For example, the Kronecker graph model [76] can be combined with a Markov Chain Monte Carlo method (KronFit [76]) to estimate its parameters in order to fit some properties for very large networks. Similar parameter estimation has been done for other recursive network models [1]. These approaches, however, are limited to parameter estimation only and cannot generate truly novel network growth mechanisms. Middendorf et al. [85, 86] address the model-selection problem of choosing from among a small number of existing models, for example, they found that protein-protein interaction networks were best fit by the DMC [128] model. However, their procedure neither generates new models nor fits parameters for existing models.

The framework we propose, GrowCode, addresses these deficiencies by representing basic random graph operations and other natural building blocks of models as instructions that operate in a register-based virtual machine. The intuitive motivation behind our framework is to provide a general and effective set of atomic operations or building blocks of

network growth. A sequence of such operations defines an iteration of the network growth process, and repeated iterations of this sequence of operations evolve a network over time. We show that a small set of instructions — only 4 of which have parameters — are sufficient to describe preferential attachment, a forest-fire-like model, and a duplication model (and intuitively many other models as well). Because, additionally, the instructions operate on a simple machine with only 3 registers, this formal representation limits the search space of possible programs, allowing a genetic algorithm to search the space effectively.

We show that it is possible to quickly and automatically learn network growth models that satisfy key properties of social, technological, and biological networks using the GrowCode framework. The fit of these models to the basic topological properties of degree distribution, assortativity, and clustering coefficient is often superior to hand-crafted models. In particular, we learn a model for yeast protein interaction networks [132] that generates graphs with far better agreement to the observed values of the clustering coefficient and degree distribution than the popular duplication/mutation with complementary (DMC) model often used to simulate these networks. For a recent scientific co-authorship network [17], we are able to better match assortativity and degree distribution than graphs produced by the Kronecker model with optimized parameters [75]. Finally, for an autonomous systems internet graph, we are able to find a model that is simultaneously a much better match than a Kronecker model for clustering coefficient, assortativity, and degree distribution. The models we learn in all these settings produce graphs that are at least as diverse as those produced by the competing hand-crafted models, indicating that we are producing truly random graph models.

Although the framework we present here applies to undirected and unattributed graphs, the GrowCode approach is general, and can be easily extended to other classes of graphs as well. GrowCode also points the way to new techniques for more systematic and automatic study of network growth models themselves.

6.2 The GrowCode Framework

We describe a novel framework, GrowCode, in which growth models may be expressed concisely and programmatically. We define a simple register machine and a set of 15 instructions that execute on it. Every sequence of instructions is a syntactically correct program that encodes some network growth model in a compact form. The instruction set contains instructions that represent specific, basic operations affecting the graph topology. There are also few instructions to manage registers and control the program flow. The instructions were selected because they are natural building blocks of growth models capable of representing a variety of extant and unknown models.

As the GrowCode machine executes a program, it modifies the topology of a growing graph. Each pass through a GrowCode program defines a single step of the growth procedure. To grow a network for t steps using the GrowCode program, the program is executed from start to finish t times. Thus, the model described by every GrowCode program is implicitly parameterized on t, which is related to the desired size of the output graph. Between subsequent growth steps (i.e. between subsequent invocations of a program), the registers of the GrowCode machine are populated randomly with nodes from the current graph. In addition to several randomized instructions, this helps GrowCode programs encode nondeterministic growth models, and different runs of the same program nearly always produce different graphs.

6.2.1 A Register Machine

GrowCode runs on a virtual machine with three registers `r0`, `r1`, `r2` that can store positive integers. The positive integer values in the registers usually correspond to vertex IDs in the graph, although they sometimes hold implicit parameters used by some instructions. The special value `NIL` in a register means that the register is empty.

The machine maintains a program counter, `PC`, indicating the currently executing instruction. After an instruction is executed, `PC` is incremented so that the program is exe-

Table 6.1: *GC: Instruction Set*

Complete GrowCode instruction set

Name	Description
NEW NODE	create new node
CREATE EDGE	create new edge
RANDOM NODE	pick random node
RANDOM EDGE	pick random edge
INFLUENCE NEIGHBORS(p)	label neighbors with u
ATTACH TO INFLUENCED	add edges to neighbors labeled u
DETACH FROM INFLUENCED	remove edges to neighbors labeled u
CLEAR INFLUENCED	clear all labels from `L`
REWIND(r, i)	jump back r positions i times
SKIP INSTRUCTION(p)	skip next instruction
SET(i)	copy node ID to `r2`
SAVE	copy `r0` to `r2`
LOAD	copy `r2` to `r0`
SWAP	swap `r0` and `r1`
CLEAR `r2`	set `r2` to `NIL`

cuted sequentially unless one of the instructions responsible for control flow manipulates the `PC`. Programs can be self-modifying in a very limited way to support looping (see the REWIND instruction below). Program execution terminates once the location of `PC` has exceeds the length of the program.

Additionally, the machine has a limited memory $\texttt{L} : V \to V$ that can store a single vertex ID associated with each vertex in V, the set of vertices in the growing graph. The value $\texttt{L}(v)$ on node v need not be the vertex ID of v, but rather can be the ID of some other node in V. This allows programs to mark nodes with IDs of arbitrary other nodes in the graph, which is how programs can spread the influence of a node in the graph (see section 6.2.2). If $\texttt{L}(v) = \texttt{NIL}$, then v is considered to have no label.

6.2.2 An Instruction Set

Instruction Set Design. Design of instruction sets is a difficult, problem. Operations in the GrowCode instruction set were selected so that they are representative of the basic

network operations. Individual instructions are easily interpretable and are similar to those used by the hand-created growth models. Intuitively, combinations of these instructions can produce growth models that can generate networks with the desired properties. The combination of instructions used here is but one example among many possible instruction sets. This set of instructions can be extended to include other instructions to accommodate new growth processes. A good set of instructions makes it much easier to optimize a difficult objective [14]; however, it is out of the scope of this paper to completely resolve the problem of instruction set design. Rather we provide evidence that one instruction set (Table 6.1) works well for several common classes of graphs.

GrowCode Instructions. The GrowCode instructions (Table 6.1) can be subdivided into 4 categories: (1) graph operations, (2) influence operations, (3) control flow operations, and (4) register manipulation operations. The first two sets of operations deal with modifying the topology of the growing graph while the 3rd and 4th sets of operations deal with managing the control flow of the GrowCode program and the state of the GrowCode machine. See Table 6.1 for a complete list of instructions. Below, we describe how each of these affects the state of the GrowCode machine and the graph being generated. In section 6.3, we show how several network growth models can be expressed with these instructions.

Graph Operations. The graph operations perform basic modifications of graph topology. The NEW NODE operation creates a new node in the growing graph with a unique ID that is placed in register `r0`. The CREATE EDGE operation is used to introduce a single new edge in the graph. The machine first fetches the nodes from `r0` (u) and `r1` (v) and then creates a new edge $\{u, v\}$ in the graph. The state of the registers after a CREATE EDGE operation remains unchanged, so if such an edge already exists, the operation has no effect. The RANDOM NODE operation selects a node uniformly at random from the current graph and places this node into `r0`. Finally, the RANDOM EDGE operation selects an edge $\{u, v\}$ uniformly at random from the current graph, and places u in `r0` and v in `r1`.

Influence Operations. The influence operations allow nodes to exert an influence on other nodes in the graph. We say that a node v is influenced by u if $\texttt{L}(v) = u$. This concept of influence is important to produce graphs with properties such as homophily where, to varying degrees, connected nodes share topological neighborhoods. The core influence operation, INFLUENCE NEIGHBORS(p), allows a node to influence a subset of its neighborhood. To execute the INFLUENCE NEIGHBORS(p) operation, the machine fetches the node ID from `r0`; this node, u, becomes the central, or *influential* node. It then assigns the mark u to every neighbor v of u by setting $\texttt{L}(v) = u$ independently with probability p. Each newly marked node, v, in turn marks its neighbors with the value u with probability $p^{d(u,v)}$, where $d(u, v)$ is the shortest path distance between u and v. If `r2` is not `NIL`, only nodes v such that $d(u, v) < \texttt{r2}$ can potentially be affected by the influence operation. If $\texttt{r2} = \texttt{NIL}$, the influence operation continues until the probabilistic process dies out and no more nodes are marked.

The INFLUENCE NEIGHBORS(p) operation works in conjunction with three other influence operations. ATTACH TO INFLUENCED creates edges between the node in `r0`, w, and all nodes in the graph marked with the value in $\texttt{r1} = u$. That is, it creates edges $\{w, v\}$ for all v such that $\texttt{L}(v) = u$. This provides a general mechanism to make the neighborhoods of two nodes more similar to each other. The DETACH FROM INFLUENCED operation fetches a node u from `r0` and removes all edges $\{u, v\}$ from the graph where $\texttt{L}(v) = u$. Finally, the CLEAR INFLUENCED operation erases the contents of `L` so that future operations can work with a clear memory. Figure 6.1 illustrates these influence operations.

Control Flow Operations. The control flow operations alter the order of execution of GrowCode instructions. The REWIND(r, i) instruction allows for loop-like structures in GrowCode programs. The first argument r is a natural number specifying the number of times `PC` should be decremented when the instruction is executed (i.e. how far the `PC` should jump backwards). The second argument i specifies the number of times the instruc-

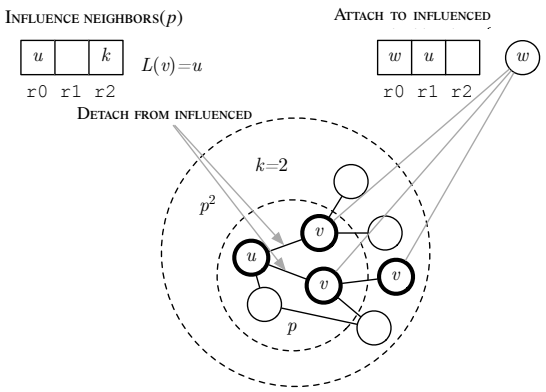


Figure 6.1: *GC: Influence Operations*

Schematic of the three influence operations. First node u influences its neighbors with probability p, then the influenced neighbors v influence their neighbors with probability p^2. If u were to detach from its influenced neighbors the two edges indicated by the gray arrows would be removed from the graph. Finally, w can attach to all nodes influenced by u.

tion should be executed. Each time the instruction is executed, the instruction is modified by decrementing the value of i by 1. When $i = 0$, the instruction will no longer be executed and the value of `PC` will not be rewound. The parameters of the rewind instruction are reset between consecutive program executions. The other instruction in this group, SKIP INSTRUCTION(p), advances the `PC` by a value of 2 with probability p. This allows the next instruction to be conditionally executed with probability $1 - p$.

Register Operations. Finally, the register operations allow one to manipulate the 3 registers directly. The SET(i) instruction assigns integer i to `r2`. The SAVE operation copies the contents of `r0` into `r2`. Conversely, LOAD places the contents of `r2` into `r0`. The SWAP operation swaps the contents of `r0` and `r1`. Finally, the CLEAR `r2` sets the contents of `r2` to `NIL`.

6.3 Representing Existing Models

To demonstrate the generality of the GrowCode instruction set, we show how it can be used to express three existing network growth models, Barabási-Albert (B-A) [8], duplication and mutation with complementarity (DMC) [128], and forest fire (FF) [75] by writing

Algorithm 7: GC: B-A Program

```
NEW NODE                                  // Create a new node, u
SAVE
RANDOM EDGE                               // Choose a random edge, e
SKIP INSTRUCTION(0.5)                     // Choose random endpoint v of e
SWAP
LOAD
CREATE EDGE                               // Create an edge between v and u
REWIND(5, i)                              // Attach it to i random nodes
Barabási-Albert
```

hand-coded GrowCode programs that match the properties of these models. These growth models match different classes of real-world networks, and they exhibit different topological qualities. For example, the DMC model (with the appropriate parameters) produces graphs with a range of clustering coefficients that match those observed in protein-protein interaction networks, while the FF model produces graphs that exhibit shrinking diameter and a densification power law property as they grow. Despite significant differences in the mechanisms they model and the graph properties they produce, there are fairly simple GrowCode programs capable of representing each of these models while using the same set of primitive instructions. The instructions are re-used in different models which indicates their overall utility in expressing different network growth behavior.

6.3.1 Barabási-Albert

In the B-A model, new nodes added to the growing network are more likely to connect to existing high-degree nodes [8]. This process reproduces the scale-free degree distribution often found in real-world networks, where there are relatively few nodes having a very high degree and a long tail of low-degree nodes.

The GrowCode program in Algorithm 7 closely simulates the B-A model. While there are subtle differences between the program and the model, the graphs generated by the program match those produced by B-A closely. The essence of the B-A model is encoded in lines 3–5. The RANDOM EDGE instruction (line 3) picks an edge that is

likely to have a high-degree node as one of its endpoints. The probability that a randomly chosen edge e contains the node u is directly proportional to the degree of u: $d(u)/|E| = 2d(u)/\sum_{v \in V} d(v)$. Once e is selected, instructions 4 and 5 choose an endpoint for this edge at random to ensure that there is no bias when selecting a node within the edge. This process selects nodes proportional to their degree as desired with the minor difference that the degrees $d(u)$ are changed as the program executes, in contrast to B-A . The newly added node is then connected to u (line 7) completing the preferential attachment of the new node. The rewind on line 8 iterates this procedure so that we connect the new node to i existing nodes.

6.3.2 Duplication and Divergence

The duplication and mutation with complementarity model [128] (abbreviated DMC) aims to reproduce the topological characteristics of protein interaction networks. Under the DMC model, the driving mechanism of network growth is the duplication of existing nodes. The model has two parameters, q_{MOD} and q_{CON}, that govern the process as follows. Each new node u chooses an anchor v and attaches to all of v's neighbors. Then, for each node w now adjacent to both u and v, an edge is randomly chosen that connects w either to u or v, and the edge is removed with probability q_{MOD}. Finally, the edge $\{u, v\}$ is added with probability q_{CON}. This mechanism of growth is motivated by the common occurrence of gene duplication, wherein genes, the precursors of proteins, are commonly copied within the genome. Initially, the copied genes are exact duplicates, and therefore the resultant proteins maintain the same set of interactions as the original protein. However, after duplication, evolutionary pressure on genes to maintain the interactions is reduced, and the interaction patterns between the original and copied genes start to diverge. Algorithm 8 gives a close approximation to the DMC model in GrowCode.

The DMC model presented in Algorithm 8 differs slightly from that introduced by Vazquez et al. [128] in that we cannot precisely mimic the procedure of selecting shared

Algorithm 8: GC: DMC Program

```
DMC RANDOM NODE                          // Put a random node v in r0
SET(1)                          // Set r2 (k-hop for influence) to 1
INFLUENCE NEIGHBORS(1.0)                 // Influence v's neighbors
SWAP                                                 // Swap r0 and r1
NEW NODE                 // Create a new node u and put it in r0
ATTACH TO INFLUENCED               // Connect u to influenced nodes
CLEAR INFLUENCED
INFLUENCE NEIGHBORS(q_MOD/2)             // Influence u's neighbors
SWAP
INFLUENCE NEIGHBORS(q_MOD/2)             // Influence v's neighbors
DETACH FROM INFLUENCED            // Actually delete edges from v
SWAP
DETACH FROM INFLUENCED                          // Do the same for u
CLEAR INFLUENCED
SKIP INSTRUCTION(1.0 - q_CON)                   // Skip adding {u, v}
CREATE EDGE                                  // with probability q_CON
```

neighbors of u and v with probability q_{CON} and then deleting the edge to one or the other. However, to achieve a similar effect, we can influence the shared neighbors of each node with probability $q_{\text{MOD}}/2$ (lines 8 and 10) after we have copied v's neighborhood to u. If the set of influenced neighbors is disjoint, then the influence instruction has exactly the same effect as the traditional DMC operation. It is possible that a neighboring node will be influenced by both u and v. Complementarity (the fact that only the edge to u or v is removed, and not both) is maintained in this case as well since the mark on the shared node will be overwritten, ensuring that only one of $\{u, w\}$ and $\{v, w\}$ can be deleted. This procedure can result in values of q_{MOD} having a slightly different effect in the GrowCode program as compared to the original DMC model. However, we have verified that graphs generated by GrowCode DMC and the original DMC have similar Zipf plots (through visual inspection) and clustering coefficients (section 6.5.3), which are the two features on which the authors of the DMC model focused. Further, Algorithm 8 produces graphs with similar Zipf plots and clustering coefficients as those observed in the yeast protein interaction network. Thus, despite the subtle differences, the GrowCode algorithm 8 maintains

Algorithm 9: GC: FF Program

```
FF
RANDOM NODE                          // Put a random node in r0
CLEAR r2          // Clear r2 to allow full graph influence
INFLUENCE NEIGHBORS(b)  // Breadth-first recursive influence
SWAP                            // Move the random node into r1
NEW NODE                                  // Create a new node, u
CREATE EDGE ATTACH TO INFLUENCED   // Connect u to influenced
nodes
```

the essential characteristics of the original growth model.

6.3.3 Forest Fire

The forest fire (FF) model [75] was first introduced to model scale-free degree distributions (of both in-degree and out-degree) as well as shrinking diameter and densification over time (under certain parameter regimes). The FF model is very intuitive and easy to explain from the perspective of network growth. We present a model that has been slightly altered to apply to undirected networks. When a new node u enters the network, it chooses an existing node v uniformly at random to act as an *ambassador*, and the edge $\{u, v\}$ is added to the network. Next, a number n is drawn from a geometric distribution with probability b of success, and n neighbors of v are chosen to be *burned*. An edge is added from u to each of these burned nodes, and the process of selecting a set of neighbors and burning them is repeated recursively.

Algorithm 9 shows a GrowCode program that encodes the forest fire model. We observe that it produces networks with the same essential characteristics as those produced via the forest fire model. In particular, the networks produced by algorithm 9 exhibit (for certain parameter ranges of b) shrinking diameter and densification power law during network growth.

6.4 Learning GrowCode Models

One of the benefits of expressing growth models as a set of instructions is that we can now formalize the problem of learning a growth model as an optimization problem over the space of GrowCode programs. Given a set of graph features, we use genetic programming techniques to learn a GrowCode program that produces graphs closely approximating these features. These graph properties are encoded into the fitness function of an individual GrowCode program. The goal of our learning procedure is not to recover previously proposed growth models, but rather to learn programs that grow graphs that are representative of a particular class of graphs as measured under specific similarity measures.

6.4.1 Constructing a Fitness Function

We define a feature collection $\mathbf{x} = [x_1, x_2, \ldots, x_m]$ to be a m-long vector of features where each property x_i may be a scalar (such as clustering coefficient) or a vector (such as a sampling of the graph's effective diameter during its growth process). The goal of the feature collection is to represent the essential graph characteristics that we want our growth model to match. We define a (possibly weighted) similarity measure between any two feature collections $s(\mathbf{x}^i, \mathbf{x}^j)$, which are of the same size, as:

$$s(\mathbf{x}^i, \mathbf{x}^j) = \sum_{\ell=1}^{m} w_\ell s_\ell(x_\ell^i, x_\ell^j), \tag{6.1}$$

where $s_\ell(\cdot, \cdot)$ is a user-defined measure of similarity between the ℓ^{th} features of the collections. This measure of similarity can be as simple as the inverse of the difference between the two features (e.g. for scalar features), or it could be as complex as a measure of the similarity of distributions (for more complex features). The only requirement on $s_\ell(x_\ell^i, x_\ell^j)$ is that it should be a monotonically non-decreasing function of similarity between the two features, and it should achieve its maximum value when $x_\ell^i = x_\ell^j$. The w_ℓ allow one to weight each feature differently, forcing the optimization procedure to prefer some features

over the others. In the experiments reported here, $w_\ell = 1$ for all ℓ.

We define the fitness of a GrowCode program based on **??**. Let $\mathbf{x}^{\mathtt{T}}$ be a target feature collection and let $\mathbf{x}^{\mathtt{P}}$ be a random variable representing the feature collection for the graph generated by the randomized program `P`. Then we can define our problem as the search for $\mathtt{P}^*$ such that:

$$\mathtt{P}^* = \arg\max_{\mathtt{P}} \mathbb{E}[s(\mathbf{x}^{\mathtt{P}}, \mathbf{x}^{\mathtt{T}})], \tag{6.2}$$

where the expectation is taken over various runs of `P`. That is, we seek the program $\mathtt{P}^*$ such that the graph generated by $\mathtt{P}^*$ are expected to have features most similar to those given by $\mathbf{x}^{\mathtt{T}}$ as measured by the similarity function $s(\cdot, \cdot)$. This optimization problem is difficult given the size of the space of potential programs. To tackle this problem effectively, we adopt genetic programming techniques which have proven effective in similarly difficult optimization scenarios.

6.4.2 Optimization with Genetic Algorithms

We use the ECJ package [82] to perform the optimization, and we use its abilities to evaluate individuals within a generation in parallel, customize the selection methods and breeding architecture for multiple sub-populations, perform NSGA-II multi-objective optimization [32], and handle arbitrary representations of fixed and variable length genomes.

Each individual encodes a program. At each generation, we evaluate the fitness for all individuals in the fixed-size population. Each program's fitness is calculated by running it for k iterations, and comparing its feature vector $\mathbf{x}^{\mathtt{P}}$ against the target set of features. This is repeated some number M times, and the results are averaged, so that the fitness of program `P` is:

$$\mathcal{F}(\mathtt{P}) = \operatorname{avg} s(\mathbf{x}^{\mathtt{P}}, \mathbf{x}^{\mathtt{T}}). \tag{6.3}$$

Alternatively, we can average each $s_\ell(x_\ell^i, x_\ell^j)$ in **??** as a separate objective, and employ a multi-objective optimization strategy (e.g. NSGA-II) [32].

When breeding individual programs, a two-point crossover operation allows the pro-

grams to mix with each other thus varying their contents and length. At the end of each generation, individuals compete in a tournament of successive comparisons of two randomly chosen individuals, where winners are chosen deterministically based on the higher fitness value. Winners of the tournament become members of the subsequent population. Individuals are drawn with replacement and can thus be replicated in the subsequent population. More fit individuals are more likely to win tournaments, making "elite" members more likely to survive into the next generation.

6.5 Applications to Synthetic and Real Networks

We demonstrate the use of our framework to learn GrowCode programs that produce networks matching specified properties of both synthetic and real networks. Unless specified otherwise, we use the following parameters in our optimization procedures. All programs in the first generation of an optimization are initialized randomly with 10 instructions. Each generation consists of 100 programs that are evaluated on the basis of a single-objective (section 6.5.1) or multi-objective (sections 6.5.2 to 6.5.4) fitness function. Individuals are chosen to advance to successive generations using tournament-selection. At the start of each generation, the population of individuals is bred from the selected individuals from the previous generation using two-point list crossover breeding. This produces new individuals, potentially with programs of different length, which are then subject to mutation (we use a mutation rate of 0.1). The optimization procedure is carried out for 15 generations, and we select the most fit individual from the final generation as the representative GrowCode program against which we compare other models.

6.5.1 Learning Scale-free Graphs

Scale-free distributions are the key network property that motivated the B-A growth model, and we show that GrowCode can learn models that produce scale-free distributions. Given the large space of models defined by the instruction set and their parameters, it is unclear at first if effectively exploring this space is even possible.

We measure the similarity of two degree distributions via a shape function. Although one straightforward option is to choose the goodness of fit to a scale-free distribution as one of our features, this approach is specific to scale-free distributions, and in general, we would like to generate graphs that match the shape of any specified degree distribution, not only scale-free distributions. We define the shape $\boldsymbol{\psi}_{\text{shape}}$ of a graph to be the cumulative distribution of node degrees where the support of the distribution (the degrees of the nodes) is normalized between 0 and 1. This normalization allows for the comparison of the degree distribution of graphs of different sizes. We define the similarity metric for the shape feature to be:

$$s_{\text{shape}}(\boldsymbol{\psi}^i_{\text{shape}}, \boldsymbol{\psi}^j_{\text{shape}}) = \frac{1}{\|\boldsymbol{\psi}^i_{\text{shape}} - \boldsymbol{\psi}^j_{\text{shape}}\|_1 + \epsilon}, \tag{6.4}$$

where ϵ is a small positive constant to assure that the fitness is defined (as a large value) when the shapes coincide exactly.

The B-A model is parameterized on i, the number of vertices to which a new vertex connects. To get the target degree distribution shape for various i, we generate graphs for $i = 3, 4, 5, 6$ and obtain maximum-likelihood exponents of $\alpha = 2.61$, 2.71, 2.73, 2.92, respectively. We then use s_{shape} in **??** to define the fitness of a program as the difference in degree-distribution shape between the graphs produced by the program and the estimated target shape.

With this fitness, GrowCode learns many non-identical programs that are scale free. Algorithm 10 shows one of the effective scale free GrowCode programs learned during the optimization process. To test for the plausibility of a scale-free distribution, we use statistical tests specific to that distribution as described in [25]. We find that even though the α parameter was not directly used in the fitness function, the networks instantiated from the learned models that pass the scale-free test have an average α value of 2.69, which is reasonably close to that of the target graphs.

In fact, we posit that it is quite easy for our optimization procedure to discover a Grow-

Algorithm 10: GC: Learned Scale-Free Program

```
Example Learned Scale-Free Model NEW NODE
RANDOM NODE
ATTACH TO INFLUENCED
CLEAR r2
SET(1)
RANDOM EDGE
DETACH FROM INFLUENCED
RANDOM NODE
CREATE EDGE
INFLUENCE NEIGHBORS(0.692)
```

Code program that produces networks with a scale-free degree distribution. Figure 6.2 shows a trace of one of the optimization runs when attempting to fit a degree-distribution generated from the B-A model with $i = 4$. Scale-free models are discovered in the first generation of the optimization procedure, even before fitness selection has had an opportunity to affect the population. The total fitness of the scale-free individuals grows quickly, and by generation 6, is already substantially greater than the total fitness of the non-scale-free individuals (figure 6.2). These observations have two implications. First, the shape function seems to correlate well with the scale-free plausibility of the graph. Second, discovering a scale-free model is not difficult.

6.5.2 Performance on a Social Network

We apply GrowCode to a recent network of co-authorship of genome-wide association studies (GWAS) [17]. We consider the social network of "repeated co-authorship" where pairs of scientists are linked if they published together more than once. This network is associated with a high assortativity (0.19), which implies that scientists who collaborate prolifically tend to connect with scientists who also collaborate with many other researchers. We simultaneously optimize for the shape distribution feature commonly studied in social networks [25] and assortativity using the multi-objective scheme of section 6.4.2.

We compare graphs generated by GrowCode programs to graphs generated by the Kro-

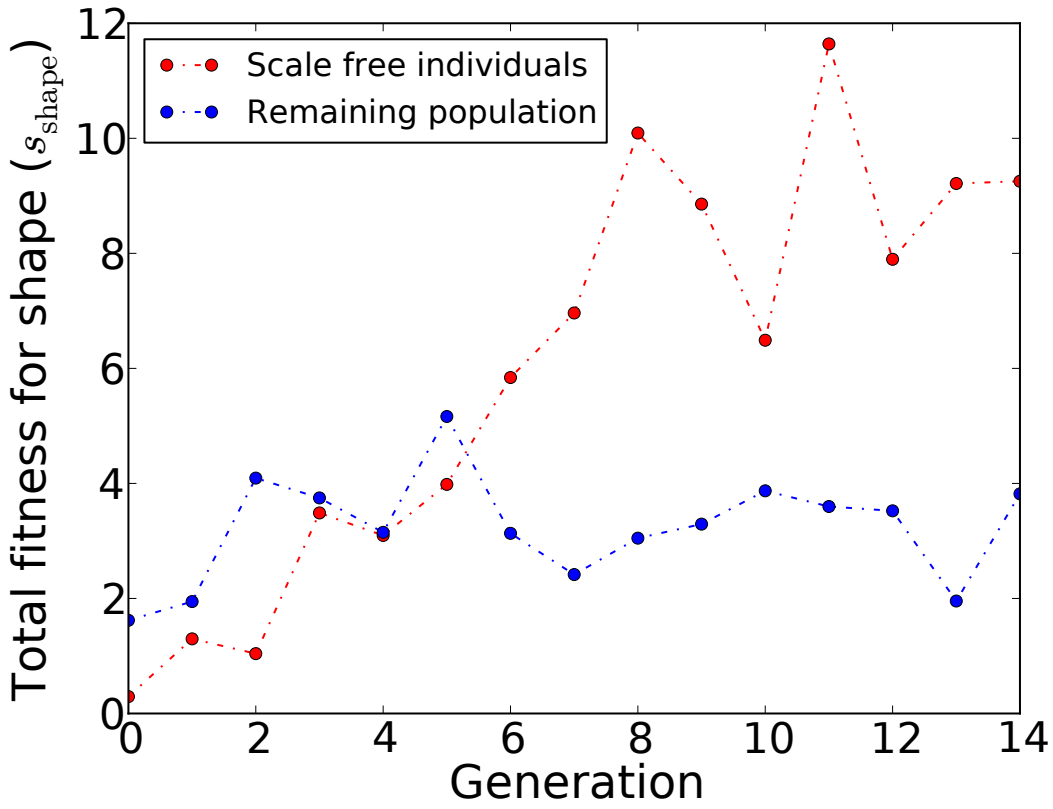


Figure 6.2: *GC: Rapid Discovery of Scale-Free Programs*

Total fitness for shape (s_{shape}) at each generation. The total fitness for those individuals that pass the scale-free plausibility test is drawn in red, while the total fitness for the rest of the individuals in the generation is drawn in blue. After just the fifth generation, the total fitness of scale-free individuals is approximately twice as large as the remaining population.

necker model [76], a fast, recursive graph generation model that has been shown to reproduce many characteristics of real-world networks. The Kronecker model requires parameters to generate random graphs with the desired properties, and we use the KronFit maximum-likelihood algorithm on the GWAS graph to estimate these parameters. We compare the features of 100 graphs generated by each model to the real GWAS network.

The learned GrowCode model better matches the assortativity of the original graph as well as the shape of the degree distribution (Figure 6.3) than the best-fit Kronecker model. The average shape difference of the GrowCode model (mean 8.47, std. dev 2.24) is closer

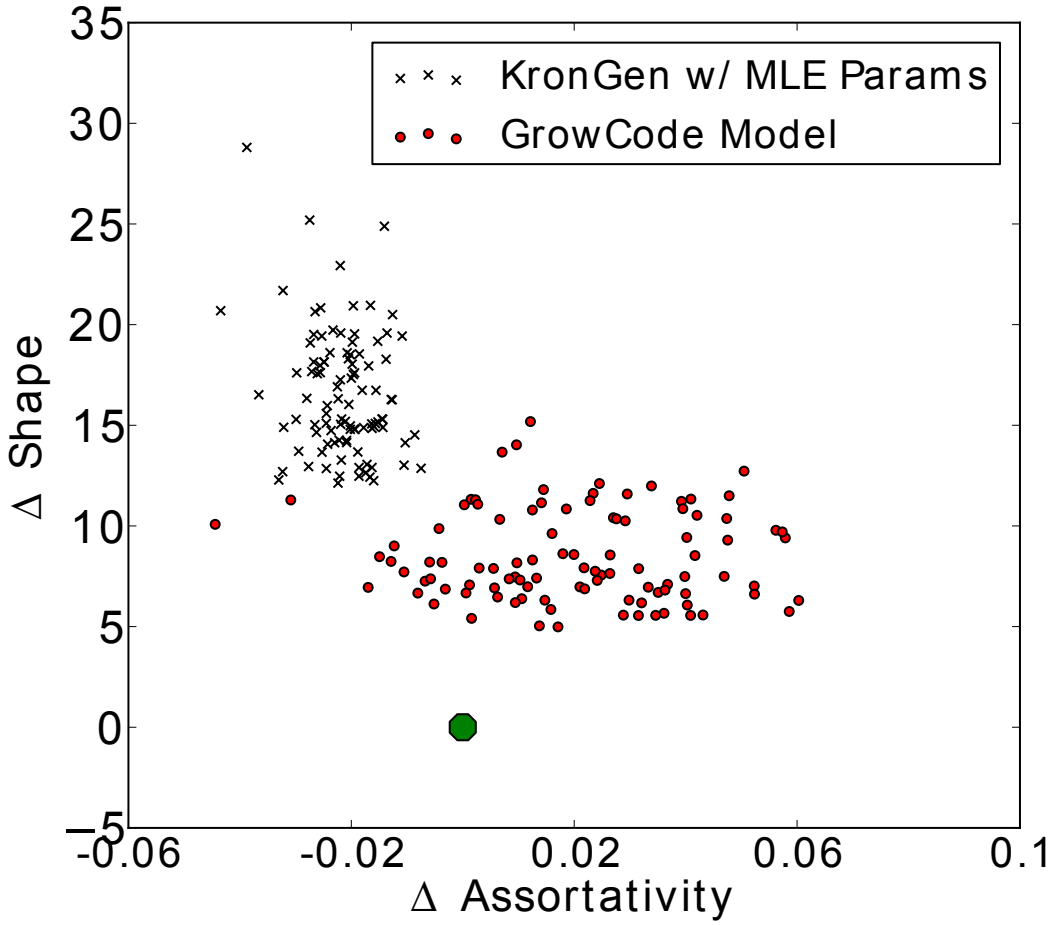


Figure 6.3: *GC: Fitting GWAS Network*

GWAS target network. Each point represents a single generated graph from a model. The x axis represents the difference between the assortativity of a graph and the target co-authorship graph. The y axis represents the difference between the shape of a learned graph and the co-authorship graph. The green dot represents a perfect match to the co-authorship graph.

to the shape of the co-authorship network than that for the Kronecker model (mean 16.6, std. dev 3.09). The mean assortativity for the GrowCode graphs is 0.206 while the mean for the Kronecker graphs is 0.165. However, in this case, different graphs produced by the GrowCode model are associated with a wider range of assortativity values (std. dev 0.0208) than the Kronecker graphs (std. dev 0.00629).

6.5.3 Performance on a Biological Network

We demonstrate the ability of GrowCode to learn a program that generates graphs similar to a high-quality and recent yeast protein interaction network compiled by Gibson et al. [53]. We optimize for both the shape distribution and the clustering coefficient, both shown to be biologically relevant in protein interaction networks [128]. We follow a similar procedure as in section 6.5.2, but instead of the Kronecker model, we use the DMC model as the baseline comparison. We determined the best parameters for DMC ($q_{\text{mod}} = 0.55$ and $q_{\text{con}} = 0.37$) via a grid search over the parameter space, and we selected the pair of parameters for which the graphs produced by the model closely match the number of edges, clustering coefficient, and diameter of the input PPI network.

The networks generated by the learned GrowCode program match the target characteristics of the real network substantially better than the networks produced by the DMC model (Figure 6.4). The mean average clustering coefficient produced by the GrowCode program is 0.091 (std. dev 0.006), which matches the average clustering coefficient (0.099) of the protein interaction network very well. Conversely, the mean average clustering coefficient produced by DMC is 0.227 (std. dev 0.013) which is quite far from the true value. The results for the shape distribution yield a similar conclusion (figure 6.4). Over the random networks generated by the GrowCode program, the average shape distribution distance is 4.58 (std. dev 1.69) while the average distance among the DMC-generated networks is 15.48 (std. dev 6.29). The GrowCode program not only produces graphs that match the target better but also that exhibit greater parametric stability (i.e. less variance) with regard to these metrics.

6.5.4 Performance on a technological network

Above, we showed that GrowCode performs well when learning models for pairs of network properties. In each case, the particular pair of properties were chosen because they

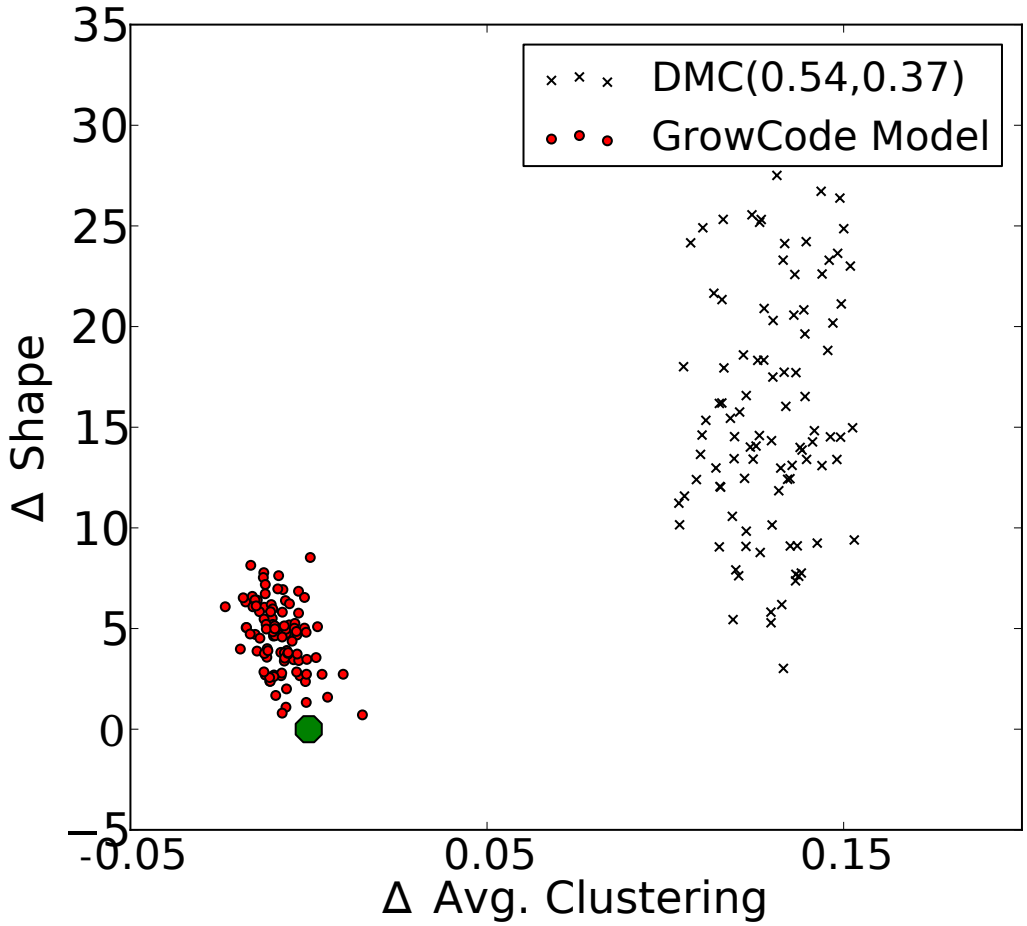


Figure 6.4: *GC: Fitting Yeast PPI*

PPI target network. Each point represents a single generated graph. The x axis gives the difference between the average clustering coefficients of a graph and the protein interaction graph. The y axis gives the difference between the shape of a learned graph and the protein interaction graph. The green dot represents the origin and the protein interaction graph.

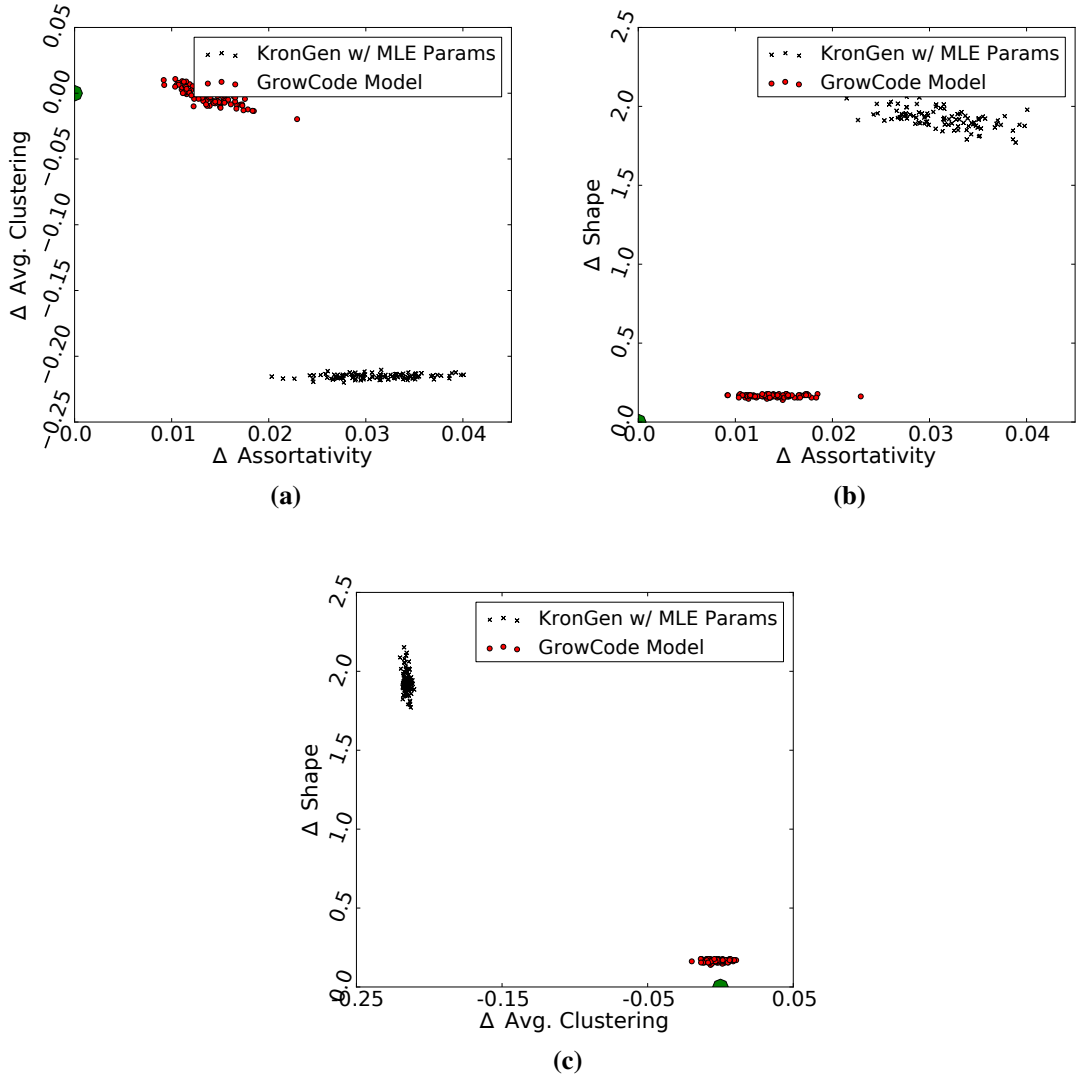


Figure 6.5: *GC: Fitting AS Network*

AS target network. Each point represents a single generated graph. The difference between the coefficients of a graph and the AS graph are plotted for three pairs of network properties: average clustering vs. assortativity, shape vs. assortativity, and shape vs. average clustering. The green dot represents the origin and the AS graph.

had been studied in the context of the corresponding class of network. The GrowCode framework is not restricted to only two target features, and here, we provide evidence that it is possible to extend the learning process to more attributes. Using the Autonomous Systems (AS) Route View graph discussed in [76] as a target, we learn a GrowCode program by simultaneously optimizing for all three of the previously considered features (shape, assortativity, and average clustering coefficient). In this case, 150 individuals were evolved for 25 generations. As in Section 6.5.2, we compare graphs generated by the learned GrowCode program to those generated by the Kronecker model.

The three plots in Figure 6.5 show that for all pairs of properties, graphs generated by GrowCode are close to the real world network and, in fact, match the AS graph better than those generated by the Kronecker model. Note that the plots show only two dimensions at a time, but a single learned GrowCode program was optimized for all three features at once. As with the biological network in Section 6.5.3, the GrowCode program produces graphs that have low variance with respect to the target topological properties.

6.5.5 GrowCode generates random models

To ensure that the diversity of the graphs produced by GrowCode matches that of those produced by hand-coded models, we computed the mean and standard deviation of the spectral distance between 100 graphs generated by both the traditional B-A model and the GrowCode program that best fit a scale-free graph. The spectral distance is a reasonable, efficiently-computable measure of graph similarity that correlates well with graph edit distance [131]. In order to estimate spectral distances between graphs, we used a discretized histogram of the normalized Laplacian eigenvalue distribution (100 bins). The spectral distance is then the Euclidean distance between such histograms.

As Table 6.2 shows, GrowCode does not produce deterministic models, and in fact the models learned by GrowCode generate an ensemble of graphs that has higher diversity than the ensemble produced by the B-A model, despite matching the target properties better. Similar diversity is also observed when this experiment is repeated to compare graphs

Table 6.2: *GC: Generated Graphs Are Random*

Mean and standard deviation ($\mu \pm \sigma$) of spectral distance between all graphs generated by the B-A model and by GrowCode programs.

i	3	4	5	6
B-A	0.0086 ± 0.0058	0.0034 ± 0.0007	0.0029 ± 0.0006	0.0026 ± 0.0005
GC	0.0141 ± 0.0115	0.0252 ± 0.0206	0.0288 ± 0.0228	0.0182 ± 0.0152

generated by the optimal model learned by GrowCode to fit the yeast PPI network and the graphs generated by DMC (data not shown).

6.6 Conclusions and Future Work

We have introduced a novel framework, GrowCode, for representing network growth models as programs composed from a short and descriptive set of instructions. We demonstrate that this representation is sufficiently general to reproduce close approximations to several existing growth models. Additionally, this formal encoding allows for an effective search procedure to find models with desired properties. In representative social, biological, and technological networks, a fairly fast optimization procedure (no run took more than 30 minutes for two objectives and no more than 4 hours for three objectives) is able to produce GrowCode programs that are competitive with recent network growth models designed to match properties of graphs in these domains.

We are also able to match scale-free graphs with several different attachment parameters i, and we are able to learn GrowCode programs that pass rigorous statistical tests for scale-free plausibility. Indeed, our optimization procedure comes across scale-free GrowCode programs quickly, and by the end yields a large number of non-identical programs that produce graphs passing the scale-free plausibility test. Additionally, we show that the graphs produced by these GrowCode programs are at least as diverse as graphs generated by the B-A model. This indicates that the scale-free property is quite ubiquitous among possible growth models.

The framework presented here applies to both undirected and unattributed networks, yet we believe it can be extended to handle directed networks and networks with node and edge attributes. Certain generalizations may be achieved by extending the instruction set, for example, by incorporating instructions that create and modify directed edges, allow node attributes to spread throughout the network, or that encode a more complex and general influence mechanism. Others may require enhancements to the machine. For example, handling edge attributes may require the addition of an edge memory akin to the label memory of the current machine. Such modifications, however, are not conceptually difficult, though their careful design is important.

Finally, although the instructions used in GrowCode programs are individually interpretable, the optimization procedure may produce programs whose overall growth mechanisms are sometimes, though not always, opaque. In the future, we plan to explore how ensembles of learned programs can be analyzed to extract from them interpretable mechanisms of growth by finding commonly occurring instruction patterns (motifs). For example, on further analysis of programs like Algorithm 10, we have noticed certain repeated patterns of instructions that are often used to match scale-free networks and to create edges to existing nodes proportional to their degrees. These patterns show up with NEW NODE and CREATE EDGE instructions as well as the influence operations. Mining GrowCode programs for repeated patterns could reveal sets of instructions that are interpretable as a unit.

Chapter 7

Conclusion

In this dissertation, we have presented effective computational methods to attack several important problems in the analysis of biological networks. Chapter 2 discusses a diverse ensemble of classifiers that can be used to effectively infer the network that represents the binding affinity relationship between epitopes and human immunoglobulin antibodies. Furthermore, we successfully combined this classifier with a *de novo* peptide generation technique, and were able to produce a diverse array of peptides which, when experimentally tested, closely matched our predictions of binding affinity.

Chapter 3 introduced a novel method, GHOST, to perform global biological network alignment. GHOST is a hybrid method that combines many different computational ingredients including a new multiscale spectral signature, a seed-and-extend alignment approach combined with an approximate solution to the quadratic assignment problem and even a local search procedure, to arrive at a high quality (in terms of both biological relevance and shared topology) network alignment. We demonstrate, through a series of experiments, that the alignments produced by GHOST represent a substantially better combination of biological and topological quality than those produced by alternative alignment procedures.

Chapters 4 and 5 both explore the ancestral network reconstruction problem. In Chapter 4, we develop a novel combinatorial encoding for interaction histories that admits an efficient dynamic programming solution to find almost parsimonious histories efficiently. We show how phylogenetic branch lengths can be incorporated into the dynamic program

as a soft constraint and demonstrate that our approach achieves good performance on both synthetic and real networks. This approach, however, like most parsimony approaches, only finds one from among a potentially huge number of solutions having an equal cost. We extend our method significantly in Chapter 5, and develop an approach that is capable of summing over all optimal and near-optimal solutions. We call this improved method a sum over parsimonious histories approach to ancestral network reconstruction, and demonstrate that it is both effective at inferring ancestral interactions and highly robust to noisy input data, a condition that is very important given current experimental error rates. Finally, we show how this new approach can be employed to impute protein interactions, which are potentially missing from experimental data, for which there is experimental evidence. The sum over parsimonious histories approach is thus useful both for inferring ancestral interactions and prioritizing potentially burdensome and time-consuming experiments.

Finally, in Chapter 6 we present a non-parametric method for learning network growth models that produce networks matching desired topological features. We represent network growth models as programs, with instructions that perform different primitive topological operations, that run on a virtual machine. We call this system GrowCode. To the best of our knowledge, it represents the first work in the area of automatically learning new network growth models, as previous related work has either focused on manually constructed models or on automatic techniques to learn parameters for existing models. We show that we are able to automatically learn GrowCode models, using linear genetic programming techniques, which closely match target topological characteristics of real-world networks. In fact, we demonstrate results on 3 classes of networks — biological, social and technological — where automatically learned GrowCode models are able to match target characteristics of real world networks more closely existing manually constructed growth models with optimized parameters.

In all of the cases above, we have demonstrated how appropriately modeling the problem and applying carefully designed algorithms can lead to effective solutions to very dif-

ficult problems involving the inference, comparison and evolution of biological networks. The success of these approaches, and the manner in which they are related, naturally leads to a few directions for future work.

7.1 Future Work

We plan to extend GHOST to handle directed and multimodal networks, which will allow us to align regulatory and metabolic network in addition to the protein interaction networks we already handle. Additionally, we plan to extend GHOST to allow for local and approximate alignments. While global alignment enforces a unique and singular mapping for each protein from the smaller of the two networks being aligned, such constraints are not part of a local alignment. The more lenient local alignment problem may better reflect the true versatility of homologous proteins which can perform multiple functions as part of different pathways in different species. Further, given that our current biological network data is known to be both incomplete and noisy — containing both false-positive and false-negative interactions — local alignment may provide a more enticing way to compare biological networks, especially among highly divergent species.

We also plan to extend our ancestral network reconstruction framework — specifically the sum over parsimonious histories approach — to incorporate a broader range of evolutionary events. In particular, we plan to modify our framework to account for lateral gene transfer and novel gene birth events, in which genes are introduced to an organism from outside the normal processes of duplication an divergence. Though these events are much less common than gene duplication and loss in determining the evolutionary history of biological networks, they may nonetheless play an important role, particularly in bacteria and archaea, where lateral gene transfer is known to be somewhat prevalent [46].

Further, we recognize that there is a deep relationship between network alignment and ancestral network reconstruction, and we believe that our approaches can be combined to improve both of these tasks. For example, having a reasonable approximation of the inter-

action network of the common ancestor of two extant species provides important information that can greatly improve potential network alignments, as it provides an intermediate network (i.e. the ancestral network), in which the shared structure has been made explicit. Knowing how protein and interactions in the ancestral network map to those of the extant network greatly constrains the alignment problem, as only alignments between extant networks that respect this ancestral mapping need to be considered.

Conversely, accurate network alignments provide valuable information that aids in the ancestral network reconstruction problem. Specifically, the proper classification of proteins into different homology groups is an essential step in the process of ancestral network reconstruction. In many instances, assigning homology based on sequence similarity is sufficient, but this is not always the case. Network alignment provides a complementary line of evidence that allows us to improve the homology mapping between species, and therefore, to improve the input to our ancestral network reconstruction procedures.

Finally, both the problem of reconstructing ancestral network state and aligning biological networks rely on our understanding of the procedure by which the networks themselves grow. For example, our combinatorial framework for encoding interaction histories is based on a generalized duplication and divergence model of network growth. While this is likely the right class of growth model to describe how regulatory and protein interaction networks evolve, the evolution of different types and groups of biological networks are probably better approximated by variations of this model. Having a more accurate and network-specific growth model may allow for a more accurate inference of the ancestral state of that network. This, in turn, may lead to more accurate network alignments. In fact, one might consider a pipeline where, given a set of extant networks, a specific network growth model is learned which accurately describes the topological properties of these networks. Then, both the reconstruction of the ancestral states of these networks and the alignments between them are iteratively improved by supplying the output of the ancestral network reconstruction procedure to the alignment procedure and vice versa. Such

an iterative approach to solving these problems may yield results substantially better than any existing algorithms, including those presented in this dissertation. Such computational approaches to these difficult problems may allow us to gain insight and understanding of the structure and evolution of biological networks that would otherwise remain hidden.

Bibliography

[1] Leman Akoglu and Christos Faloutsos. RTG: a recursive realistic graph generator using random typing. *Data Mining and Knowledge Discovery*, 19(2):194–209, 2009. ISSN 1384-5810.

[2] M. Aldana, E. Balleza, S. Kauffman, and O. Resendiz. Robustness and evolvability in genetic regulatory networks. *J. Theor. Biol.*, 245(3):433–448, 2007.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215(3):403–410, October 1990.

[4] Lars Arvestad, Ann-Charlotte Berglund, and Bengt Sennblad. Bayesian Gene/Species Tree Reconciliation and Orthology Analysis Using MCMC. *Bioinformatics*, 19(Suppl. 1):i7–i15, 2003.

[5] László Babai, D. Yu. Grigoryev, and David M. Mount. Isomorphism of graphs with bounded eigenvalue multiplicity. In *Proc. of the 14th Annual ACM Symposium on Theory of Computing*, STOC '82, pages 310–324, New York, NY, USA, 1982. ACM. ISBN 0-89791-070-2. doi: http://doi.acm.org/10.1145/800070.802206.

[6] Sourav Bandyopadhyay, Roded Sharan, and Trey Ideker. Systematic identification of functional orthologs based on protein network comparison. *Genome Res.*, 16(3): 428–435, March 2006. ISSN 1088-9051. doi: 10.1101/gr.4526006.

[7] Anirban Banerjee. Structural distance and evolutionary relationship of networks. *Biosystems*, 107(3):186 – 196, 2012.

[8] A. Barabási. Emergence of Scaling in Random Networks. *Science*, 286(5439): 509–512, 1999. ISSN 00368075.

[9] A. Bhan, D.J. Galas, and T.G. Dewey. A duplication growth model of gene expression networks. *Bioinformatics*, 18:1486–1493, 2002.

[10] N. P. Boghossian, O. Kohlbacher, and H. P. Lenhof. Rapid software prototyping in molecular modeling using the biochemical algorithms library (ball). *J. Exp. Algorithmics*, 5:16, 2000. ISSN 1084-6654. doi: http://doi.acm.org/10.1145/351827.384258.

[11] E. Borenstein and M. W. Feldman. Topological signatures of species interactions in metabolic networks. *J. Comput. Biol.*, 16(2):191–200, 2009.

[12] E. Borenstein, M. Kupiec, M. W. Feldman, and E. Ruppin. Large-scale reconstruction and phylogenetic analysis of metabolic environments. *Proc. Natl. Acad. Sci. USA*, 105(38):14482–14487, 2008.

[13] I. Bozic, G.L. Zhang, and V. Brusic. Predictive vaccinology: Optimisation of predictions using support vector machine classifiers. In *Lecture Notes in Computer Science*, volume 3578, pages 375–381, 2005.

[14] M. Brameier. *On linear genetic programming*. PhD thesis, University of Hamburg, 2004.

[15] Bobby-Joe Breitkreutz, Chris Stark, and Mike Tyers. The GRID: The general repository for interaction datasets. *Genome Biol.*, 4(3):R23, 2003. ISSN 1465-6906. doi: 10.1186/gb-2003-4-3-r23.

[16] V. Brusic, N. Petrovsky, G. Zhang, and V.B. Bajic. Prediction of promiscuous peptides that bind HLA class I molecules. *Immunology and Cell Biology*, 80(3):280–285, 2002.

[17] Brendan K. Bulik-Sullivan and Patrick F. Sullivan. The authorship network of genome-wide association studies. *Nature Genetics*, 44(2):113–113, 2012. ISSN 1061-4036.

[18] D. Callaway, J. E. Hopcroft, J. M. Kleinberg, M. E. Newman, and S. H. Strogatz. Are randomly grown graphs really random? *Phys. Rev. E*, 64:041902–041908, 2001.

[19] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[20] Kevin Chen, Dannie Durand, and Martin Farach-Colton. NOTUNG: A program for dating gene duplications and optimizing gene family trees. *Journal of Computational Biology*, 7(3-4):429–447, August 2000.

[21] Rong Chen, Li Li, and Zhiping Weng. ZDOCK: an initial-stage protein-docking algorithm. *Proteins*, 52(1):80–87, July 2003. ISSN 1097-0134. doi: 10.1002/prot.10389.

[22] Leonid Chindelevitch, Chung-Shou Liao, and Bonnie Berger. Local optimization for global alignment of protein interaction networks. *Pacific Symposium On Biocomputing*, 132:123–132, 2010.

[23] F R K Chung. *Spectral Graph Theory*, volume 92. American Mathematical Society, 1997.

[24] Fan Chung, Linyuan Lu, T Gregory Dewey, and David J Galas. Duplication models for biological networks. *J. Comp. Biol.*, 10(5):677–687, January 2003.

[25] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-Law Distributions in Empirical Data. *SIAM Review*, 51(4):661, 2009. ISSN 00361445.

[26] Sean R Collins, Patrick Kemmeren, Xue-Chu Zhao, Jack F Greenblatt, Forrest Spencer, Frank C P Holstege, Jonathan S Weissman, and Nevan J Krogan. Toward a comprehensive atlas of the physical interactome of *Saccharomyces cerevisiae*. *Molecular Cellular Proteomics*, 6(3):439–450, 2007.

[27] The UniProt Consortium. Reorganizing the protein space at the Universal Protein Resource (UniProt). *Nucleic Acids Research*, 40:D71–D75, January 2012. ISSN 1362-4962. doi: 10.1093/nar/gkr981.

[28] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844, 9780262033848.

[29] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. ISSN 0885-6125.

[30] Charles Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, London, 1859.

[31] Jesse Davis and Mark Goadrich. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 233–240, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143874.

[32] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6 (2):182–197, 2002. ISSN 1089778X.

[33] S. N. Dorogovtsev, J. F. Mendes, and A. N. Samukhin. Structure of growing networks with preferential linking. *Phys. Rev. Lett.*, 85(21), 2000.

[34] Olivier Duchenne, Francis Bach, In-So Kweon, and Jean Ponce. A tensor-based algorithm for high-order graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(12):2383–2395, 2011.

[35] Dannie Durand, Bjarni V Halldórsson, and Benjamin Vernot. A hybrid micro-macroevolutionary approach to gene tree reconstruction. *Journal of computational biology*, 13(2):320–335, 2006.

[36] Janusz Dutkowski and Jerzy Tiuryn. Identification of functional modules from conserved ancestral protein-protein interactions. *Bioinformatics*, 23(13):i149–i158, 2007. doi: 10.1093/bioinformatics/btm194.

[37] Mohammed El-Kebir, Jaap Heringa, and Gunnar W. Klau. Lagrangian relaxation applied to sparse global network alignment. In *Pattern Recognition in Bioinformatics*, pages 225–236, 2011.

[38] P Erdős and A Rényi. On the evolution of random graphs. *Evolution*, 5(1):17–61, 1960. ISSN 00029947.

[39] S. Erten, X. Li, G. Bebek, J. Li, and M. Koyuturk. Phylogenetic analysis of modularity in protein interaction networks. *BMC Bioinformatics*, 10:333, 2009.

[40] C. Espinosa-Soto, O. C. Martin, and A. Wagner. Phenotypic robustness can increase phenotypic variability after nongenetic perturbations in gene regulatory circuits. *J. Evol. Biol.*, 24(6):1284–1297, 2011.

[41] S. Fields and O. Song. A novel genetic system to detect protein-protein interactions. *Nature*, 340(6230):245–246, July 1989.

[42] J. Flannick, A. Novak, B. S. Srinivasan, H. H. McAdams, and S. Batzoglou. Graemlin: general and robust alignment of multiple large interaction networks. *Genome Res.*, 16(9):1169–1181, 2006.

[43] Jason Flannick, Antal Novak, Balaji S Srinivasan, Harley H McAdams, and Serafim Batzoglou. Graemlin: General and Robust Alignment of Multiple Large Interaction Networks. *Genome Research*, 16(9):1169–1181, 2006.

[44] Jason Flannick, Antal Novak, Chuong B Do, Balaji S Srinivasan, and Serafim Batzoglou. Automatic parameter learning for multiple local network alignment. *J. Computat. Biol.*, 16(8):1001–1022, 2009.

[45] Jessica H Fong, Amy E Keating, and Mona Singh. Predicting specificity in bzip coiled-coil protein interactions. *Genome Biology*, 5(2):R11, 2004.

[46] W. Ford Doolittle. The attempt on the life of the tree of life: science, philosophy and politics. *Biology and Philosophy*, 25:455–473, 2010. ISSN 0169-3867. 10.1007/s10539-010-9210-x.

[47] Even Fossum, Caroline C. Friedel, Seesandra V. Rajagopala, BjÃűrn Titz, Armin Baiker, Tina Schmidt, Theo Kraus, Thorsten Stellberger, Christiane Rutenberg, Silpa Suthram, Sourav Bandyopadhyay, Dietlind Rose, Albrecht von Brunn, Mareike Uhlmann, Christine Zeretzke, Yu-An Dong, Hélène Boulet, Manfred Koegl, Susanne M. Bailer, Ulrich Koszinowski, Trey Ideker, Peter Uetz, Ralf Zimmer, and Jürgen Haas. Evolutionarily conserved herpesviral protein interaction networks. *PLoS Pathog.*, 5(9):e1000570, 09 2009. doi: 10.1371/journal.ppat.1000570.

[48] D. V. Foster, S. A. Kauffman, and J. E. S. Socolar. Network growth models and genetic regulatory networks. *Phys. Rev. E*, 73(3):031912, Mar 2006. doi: 10.1103/PhysRevE.73.031912.

[49] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Appl. Math.*, 42(2-3):177–201, April 1993. ISSN 0166-218X. doi: 10.1016/0166-218X(93)90045-P.

[50] Anne-Claude Gavin, Patrick Aloy, Paola Grandi, Roland Krause, Markus Boesche, Martina Marzioch, Christina Rau, Lars Juhl Jensen, Sonja Bastuck, Birgit Dumpelfeld, Angela Edelmann, Marie-Anne Heurtier, Verena Hoffman, Christian Hoefert, Karin Klein, Manuela Hudak, Anne-Marie Michon, Malgorzata Schelder, Markus Schirle, Marita Remor, Tatjana Rudi, Sean Hooper, Andreas Bauer, Tewis Bouwmeester, Georg Casari, Gerard Drewes, Gitte Neubauer, Jens M. Rick, Bernhard Kuster, Peer Bork, Robert B. Russell, and Giulio Superti-Furga. Proteome survey reveals modularity of the yeast cell machinery. *Nature*, 440(7084):631–636, 03 2006.

[51] Andrea Gesmundo and James Henderson. Faster Cube Pruning. In Marcello Federico, Ian Lane, Michael Paul, and François Yvon, editors, *Proceedings of the seventh International Workshop on Spoken Language Translation (IWSLT)*, pages 267–274, 2010.

[52] T. A. Gibson and D. S. Goldberg. Reverse engineering the evolution of protein interaction networks. *Pac. Symp. Biocomput.*, pages 190–202, 2009.

[53] Todd A Gibson and Debra S Goldberg. Improving evolutionary models of protein interaction networks. *Bioinformatics*, 27(3):376–382, 2011.

[54] S. Henikoff. Amino Acid Substitution Matrices from Protein Blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, November 1992. ISSN 0027-8424. doi: 10.1073/pnas.89.22.10915.

[55] Lei Huang and Yang Dai. A support vector machine approach for prediction of T-cell epitopes. In Yi-Ping Phoebe Chen and Limsoon Wong, editors, *APBC*, Proceedings of the 3rd Asia-Pacific Bioinformatics Conference. Imperial College Press, London, 2005. ISBN 1-86094-477-9.

[56] Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, Parsing '05, pages 53–64, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[57] I. Ispolatov, P. L. Krapivsky, and A. Yuryev. Duplication-divergence model of protein interaction network. *Phys. Rev. E*, 71(6 Pt 1):061911, 2005.

[58] Samira Jaeger, Christine T Sers, and Ulf Leser. Combining modularity, conservation, and interactions of proteins significantly increases precision and coverage of protein function prediction. *BMC Genomics*, 11(1):717, 2010.

[59] David S. Johnson, Ali Mortazavi, Richard M. Myers, and Barbara Wold. Genome-wide mapping of in vivo protein-dna interactions. *Science*, 316(5830):1497–1502, 2007. doi: 10.1126/science.1141319.

[60] Shuichi Kawashima, Piotr Pokarowski, Maria Pokarowska, Andrzej Kolinski, Toshiaki Katayama, and Minoru Kanehisa. AAindex: amino acid index database, progress report 2008. *Nucleic Acids Research*, 36(suppl 1):D202–D205, 2008. doi: 10.1093/nar/gkm998.

[61] Myunghwan Kim and Jure Leskovec. Multiplicative attribute graph model of real-world networks. *Internet Mathematics*, 8(1-2):113–160, 2012.

[62] W. K. Kim and E. M. Marcotte. Age-dependent evolution of the yeast protein interaction network suggests a limited role of gene duplication and divergence. *PLoS Comput. Biol.*, 4(11):e1000232, 2008.

[63] J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas. On combining classifiers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(3):226 –239, mar 1998. ISSN 0162-8828. doi: 10.1109/34.667881.

[64] Gunnar W Klau. A new graph-based method for pairwise global network alignment. *BMC Bioinformatics*, 10(Suppl 1):S59, 2009.

[65] Dan Klein and Christopher D. Manning. Parsing and hypergraphs. In *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT-2001), 17-19 October 2001, Beijing, China*. Tsinghua University Press, 2001. ISBN 7-302-04925-4.

[66] Rob Knight, Peter Maxwell, Amanda Birmingham, Jason Carnes, J Gregory Caporaso, Brett C Easton, Michael Eaton, Micah Hamady, Helen Lindsay, Zongzhi Liu, et al. PyCogent: a toolkit for making sense from sequence. *Genome Biology*, 8(8): R171, 2007.

[67] A. Kreimer, E. Borenstein, U. Gophna, and E. Ruppin. The evolution of modularity in bacterial metabolic networks. *Proc. Natl. Acad. Sci. USA*, 105(19):6976–6981, 2008.

[68] O. Kuchaiev, T. Milenkovic, V. Memisevic, W. Hayes, and N. Przulj. Topological network alignment uncovers biological function and phylogeny. *J. R. Soc. Interface*, 7(50):1341–1354, 2010.

[69] Oleksii Kuchaiev and Natasa Pržulj. Integrative network alignment reveals large regions of global network similarity in yeast and human. *Bioinformatics*, 27(6):1–7, 2011.

[70] J Kuczynski and H Wozniakowski. Estimating the largest eigenvalue by the power and lanczos algorithms with a random start. *SIAM J Matrix Anal Appl*, 4(4):1094, 1992.

[71] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[72] Pavel Kuksa, Pai-Hsi Huang, and Vladimir Pavlovic. Fast and accurate multi-class protein fold recognition with spatial sample kernels. In *Computational Systems Bioinformatics: Proceedings of the CSB2008 Conference*, pages 133–143, 2008.

[73] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 611–617, 2006. doi: http://doi.acm.org/10.1145/1150402.1150476.

[74] M Leordeanu and M Hebert. A Spectral Technique for Correspondence Problems Using Pairwise Constraints. *Tenth IEEE International Conference on Computer Vision ICCV05 Volume 1*, 2:1482–1489, 2005.

[75] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time. In *Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 177–187, New York, USA, 2005.

[76] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker Graphs: An Approach to Modeling Networks. *The Journal of Machine Learning Research*, 11:985–1042, 2010. ISSN 1532-4435.

[77] Christina S. Leslie, Eleazar Eskin, and William Stafford Nobel. The spectrum kernel: a string kernel for SVM protein classification. *Pacific Symposium on Biocomputing*, pages 564–575, 2002. ISSN 1793-5091.

[78] Christina S. Leslie, Eleazar Eskin, Jason Weston, and William Stafford Noble. Mismatch string kernels for SVM protein classification. In *Neural Information Processing Systems*, pages 1417–1424, 2002.

[79] E. D. Levy and J. B. Pereira-Leal. Evolution and dynamics of protein interactions and networks. *Curr. Opin. Struct. Biol.*, 18(3):349–357, 2008.

[80] Chung-Shou Liao, Kanghao Lu, Michael Baym, Rohit Singh, and Bonnie Berger. IsoRankN: spectral methods for global alignment of multiple protein networks. *Bioinformatics*, 25(12):i253–i258, June 2009. ISSN 1460-2059.

[81] Luana Licata, Leonardo Briganti, Daniele Peluso, Livia Perfetto, Marta Iannuccelli, Eugenia Galeota, Francesca Sacco, Anita Palma, Aurelio Pio Nardozza, Elena Santonico, Luisa Castagnoli, and Gianni Cesareni. MINT, the molecular interaction database: 2012 update. *Nuc. Acids Res.*, 40(Database issue):D857–61, 2012.

[82] Sean Luke. *Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat*. PhD thesis, University of Maryland, College Park, 2000.

[83] Duncan J McGeoch and Derek Gatherer. Integrating reptilian herpesviruses into the family herpesviridae. *Journal of Virology*, 79(2):725–731, 2005.

[84] Duncan J. McGeoch, Frazer J. Rixon, and Andrew J. Davison. Topics in herpesvirus genomics and evolution. *Virus Research*, 117(1):90 – 104, 2006. ISSN 0168-1702. doi: 10.1016/j.virusres.2006.01.002.

[85] M. Middendorf, E. Ziv, C. Adams, J. Hom, R. Koytcheff, C. Levovitz, G. Woods, L. Chen, and C. Wiggins. Discriminative topological features reveal biological network mechanisms. *BMC Bioinformatics*, 5:181, 2004.

[86] M. Middendorf, E. Ziv, and C.H. Wiggins. Inferring network mechanisms: The *Drosophila melanogaster* protein interaction network. *Proc. Natl. Acad. Sci. USA*, 102:3192–3197, 2005.

[87] Tijana Milenkoviç, Weng Leong Ng, Wayne Hayes, and Nataša Pržulj. Optimal network alignment with graphlet degree vectors. *Cancer Informatics*, 9:121–137, 2010.

[88] B. G. Mirkin, T. I. Fenner, M. Y. Galperin, and E. V. Koonin. Algorithms for computing parsimonious evolutionary scenarios for genome evolution, the last universal common ancestor and dominance of horizontal gene transfer in the evolution of prokaryotes. *BMC Evol. Biol.*, 3:2, 2003.

[89] A Mithani, GM Preston, and J Hein. A stochastic model for the evolution of metabolic networks with neighbor dependence. *Bioinformatics*, 25(12):1528–1535, 2009.

[90] Loris Nanni and Alessandra Lumini. MppS: An ensemble of support vector machine based on multiple physicochemical properties of amino acids. *Neurocomputing*, 69 (13-15):1688 – 1690, 2006. ISSN 0925-2312. doi: DOI:10.1016/j.neucom.2006.04.001.

[91] Loris Nanni and Alessandra Lumini. A new encoding technique for peptide classification. *Expert Systems with Applications*, In Press, Uncorrected Proof:–, 2010. ISSN 0957-4174. doi: DOI:10.1016/j.eswa.2010.09.005.

[92] S. Navlakha and C. Kingsford. Network archaeology: Uncovering ancient networks from present-day interactions. *PLoS Comput. Biol.*, 7(4):e1001119, 2011.

[93] Lars Relund Nielsen, Kim Allan Andersen, and Daniele Pretolani. Finding the k shortest hyperpaths. *Comput. Oper. Res.*, 32(6):1477–1497, June 2005. ISSN 0305-0548. doi: 10.1016/j.cor.2003.11.014.

[94] A. Noma and R.M. Cesar. Sparse representations for efficient shape matching. In *Graphics, Patterns and Images (SIBGRAPI), 2010 23rd SIBGRAPI Conference on*, pages 186 –192, Sept. 2010.

[95] Kristian Ovaska, Marko Laakso, and Sampsa Hautaniemi. Fast gene ontology based clustering for microarray experiments. *BioData mining*, 1(1):11, 2008.

[96] L. Pachter. An introduction to reconstructing ancestral genomes. In *Proc. Symp. in Applied Mathematics*, volume 64, pages 1–20, 2007.

[97] G. Palla, L. Lovász, and T. Vicsek. Multifractal network generator. *Proc. Natl. Acad. Sci. USA*, 107:7640–7645, 2010.

[98] Victor Y. Pan and Zhao Q. Chen. The complexity of the matrix eigenproblem. In *Proc. of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 507–516, New York, NY, USA, 1999. ACM. ISBN 1-58113-067-8.

[99] Jodi R Parrish, Jingkai Yu, Guozhen Liu, Julie A Hines, Jason E Chan, Bernie A Mangiola, Huamei Zhang, Svetlana Pacifico, Farshad Fotouhi, Victor J DiRita, Trey Ideker, Phillip Andrews, and Russell L Finley. A proteome-wide protein interaction map for *Campylobacter jejuni*. *Genome Biol.*, 8(7):R130, 2007.

[100] Romualdo Pastor-Satorras, Eric Smith, and Ricard Sole. Evolving protein interaction networks from gene duplication. *J. Theor. Biol.*, 222:199–210, 2003.

[101] Ashwini Patil, Kenta Nakai, and Haruki Nakamura. HitPredict: a database of quality assessed protein-protein interactions in nine species. *Nuc. Acids Res.*, 39(Database issue):D744–D749, 2011.

[102] José M Peregrín-Alvarez, Xuejian Xiong, Chong Su, and John Parkinson. The modular organization of protein interactions in *Escherichia coli. PLoS Computat. Biol.*, 5(10):e1000523, 2009.

[103] J. B. Pereira-Leal, E. D. Levy, C. Kamp, and S. A. Teichmann. Evolution of protein complexes by duplication of homomeric interactions. *Genome Biol.*, 8(4):R51, 2007.

[104] Catia Pesquita, Daniel Faria, André O. Falcão, Phillip Lord, and Francisco M. Couto. Semantic similarity in biomedical ontologies. *PLoS Computat. Biol.*, 5(7): e1000443, 07 2009.

[105] John W. Pinney, Grigoris D. Amoutzias, Magnus Rattray, and David L. Robertson. Reconstruction of ancestral protein interaction networks for the bZIP transcription factors. *Proc. Natl. Acad. Sci. USA*, 104(51):20449–20453, 2007. doi: 10.1073/pnas.0706339104.

[106] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[107] Jay W. Ponder and David A. Case. Force fields for protein simulations. *Advances in protein chemistry*, 66:27–85, 2003. ISSN 0065-3233.

[108] Yann Ponty and Cédric Saule. A combinatorial framework for designing (pseudo-knotted) RNA algorithms. In Teresa M. Przytycka and Marie-France Sagot, editors, *WABI*, volume 6833 of *Lecture Notes in Computer Science*, pages 250–269. Springer, 2011. ISBN 978-3-642-23037-0.

[109] Victor M. Preciado and Ali Jadbabaie. From local measurements to network spectral properties: Beyond degree distributions. In *49th IEEE Conference on Decision and Control*, pages 2686–2691, 2010.

[110] Natasa Przulj, Oleksii Kuchaiev, Aleksandar Stevanovic, and Wayne Hayes. Geometric evolutionary dynamics of protein interaction networks. *Pac. Symp. Biocomput.*, 15:178–189, 2010.

[111] K. Raman and A. Wagner. Evolvability and robustness in a complex signalling circuit. *Mol. Biosyst.*, 7(4):1081–1092, 2011.

[112] Roded Sharan, Silpa Suthram, Ryan M Kelley, Tanja Kuhn, Scott McCuine, Peter Uetz, Taylor Sittler, Richard M Karp, and Trey Ideker. Conserved patterns of protein interaction in multiple species. *Proc. Natl. Acad. Sci. USA*, 102(6):1974–9, 2005. ISSN 0027-8424. doi: 10.1073/pnas.0409522102.

[113] Yoshikazu Shimoda, Sayaka Shinpo, Mitsuyo Kohara, Yasukazu Nakamura, Satoshi Tabata, and Shusei Sato. A large scale analysis of protein-protein interactions in the nitrogen-fixing bacterium *Mesorhizobium loti*. *DNA Research*, 15(1):13–23, 2008.

[114] Rohit Singh, Jinbo Xu, and Bonnie Berger. Pairwise global alignment of protein interaction networks by matching neighborhood topology. In *Proc. Intl. Conf. on Research in Computational Molecular Biology (RECOMB)*, pages 16–31, 2007.

[115] Rohit Singh, Jinbo Xu, and Bonnie Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proc. Natl. Acad. Sci. USA*, 105(35):12763–8, September 2008. ISSN 1091-6490. doi: 10.1073/pnas.0806627105.

[116] Ricard Solé, Romualdo Pastor-Satorras, Eric Smith, and Thomas B. Kepler. A model of large-scale proteome evolution. *Adv. Complex Syst.*, 5(1):43–54, 2002.

[117] A. J. Stewart, R. M. Seymour, and A. Pomiankowski. Degree dependence in rates of transcription factor evolution explains the unusual structure of transcription networks. *Proc. Biol. Sci.*, 276(1666):2493–2501, 2009.

[118] Michael P H Stumpf, William P Kelly, Thomas Thorne, and Carsten Wiuf. Evolution at the System Level: the Natural History of Protein Interaction Networks. *Trends in Ecology & Evolution*, 22(7):366–373, 2007.

[119] Yizhou Sun, Yintao Yu, and Jiawei Han. Ranking-based clustering of heterogeneous information networks with star network schema. In *Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 797–806, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9. doi: 10.1145/1557019.1557107.

[120] Damian Szklarczyk, Andrea Franceschini, Michael Kuhn, Milan Simonovic, Alexander Roth, Pablo Minguez, Tobias Doerks, Manuel Stark, Jean Muller, Peer Bork, , LJ Jensen, and C von Mering. The STRING database in 2011: functional interaction networks of proteins, globally integrated and scored. *Nuc. Acids Res.*, 39 (Database issue):D561–D568, 2011.

[121] Sarah A Teichmann and M Madan Babu. Gene regulatory network growth by duplication. *Nat. Genetics*, 36(5):492–6, May 2004.

[122] Choon Hui Teo and S. V. N. Vishwanathan. Fast and space efficient string kernels using suffix arrays. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 929–936, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: http://doi.acm.org/10.1145/1143844.1143961.

[123] The Gene Ontology Consortium, M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene Ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, May 2000. ISSN 1061-4036. doi: 10.1038/75556.

[124] Wenhong Tian and Nagiza F Samatova. Pairwise alignment of interaction networks by fast identification of maximal conserved patterns. *Pacific Symposium On Biocomputing*, pages 99–110, 2009.

[125] Macarena Toll-Riera, Nina Bosch, Nicolas Bellora, Robert Castelo, Lluis Armengol, Xavier Estivill, and M Mar Alba. Origin of primate orphan genes: A comparative genomics approach. *Molecular Biology and Evolution*, 26(3):603–612, 2009.

[126] Lorenzo Torresani, Vladimir Kolmogorov, and Carsten Rother. Feature correspondence via graph matching: Models and global optimization. In *European Conference on Computer Vision*, pages 596–609, 2008.

[127] Jean-François Truchon and Christopher I. Bayly. Evaluating Virtual Screening Methods:âĂL' Good and Bad Metrics for the "Early Recognition" Problem. *Journal of Chemical Information and Modeling*, 47(2):488–508, March 2007. ISSN 1549-9596. doi: 10.1021/ci600426e.

[128] A. Vazquez, A. Flammini, A. Maritan, and A. Vespignani. Modeling of protein interaction networks. *Complexus*, 1(38):9, 2001.

[129] Benjamin Vernot, Maureen Stolzer, Aiton Goldman, and Dannie Durand. Reconciliation with non-binary species trees. *Journal of Computational Biology*, 15(8): 981–1006, 2008.

[130] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 363:202–204, 1998.

[131] Richard C. Wilson and Ping Zhu. A study of graph spectra for comparing graphs and trees. *Pattern Recognition*, 41:2833–2841, 2008. doi: 10.1016/j.patcog.2008.03.011.

[132] Haiyuan Yu, Pascal Braun, Muhammed A Yildirim, Irma Lemmens, Kavitha Venkatesan, Julie Sahalie, Tomoko Hirozane-Kishikawa, Fana Gebreab, Na Li, Nicolas Simonis, Tong Hao, Jean-François Rual, Amélie Dricot, Alexei Vazquez, Ryan R Murray, Christophe Simon, Leah Tardivo, Stanley Tam, Nenad Svrzikapa, Changyu Fan, Anne-Sophie De Smet, Adriana Motyl, Michael E Hudson, Juyong Park, Xiaofeng Xin, Michael E Cusick, Troy Moore, Charlie Boone, Michael Snyder, Frederick P Roth, Albert-László Barabási, Jan Tavernier, David E Hill, and Marc Vidal. High-quality binary protein interaction map of the yeast interactome network. *Science*, 322(5898):104–110, 2008.

[133] Mikhail Zaslavskiy, Francis Bach, and Jean-Philippe Vert. Global alignment of protein-protein interaction networks by graph matching methods. *Bioinformatics*, 25(12):i259–1267, 2009.

[134] G.L. Zhang, A.M. Khan, K.N. Srinivasan, J.T. August, and V. Brusic. Neural models for predicting viral vaccine targets. *Journal of Bioinformatics and Computational Biology*, 3(5):1207–1225, 2005.

[135] Wen Zhang, Yi Xiong, Meng Zhao, Hua Zou, Xinghuo Ye, and Juan Liu. Prediction of conformational B-cell epitopes from 3d structures by random forest with a distance-based feature. *BMC Bioinformatics*, 12(1):341, 2011. ISSN 1471-2105. doi: 10.1186/1471-2105-12-341.

[136] Xiuwei Zhang and Bernard Moret. Refining transcriptional regulatory networks using network evolutionary models and gene histories. *Alg. Mol. Biol.*, 5(1):1, 2010. doi: 10.1186/1748-7188-5-1.

[137] Xiuwei Zhang and Bernard M. Moret. Boosting the performance of inference algorithms for transcriptional regulatory networks using a phylogenetic approach. In *Proc. Intl. Workshop on Algorithms in Bioinformatics (WABI)*, pages 245–258, 2008.

[138] Y. Zhao, C. Pinilla, D. Valmori, R. Martin, and R. Simon. Application of support vector machines for T-cell epitopes prediction. *Bioinformatics*, 19(15):1978–1984, October 2003. ISSN 1367-4803.