ALGORITHMS FOR BIOLOGICAL NETWORK ALIGNMENT

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Jason Flannick
August 2009
I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Serafim Batzoglou) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Harley McAdams)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(David Dill)

Approved for the University Committee on Graduate Studies.
Abstract

A major goal in the post-genomic era is to understand how genes and proteins organize to ultimately cause complex traits. There are multiple levels of biological organization, such as low-level interactions between pairs of molecules, higher-level metabolic pathways and molecular complexes, and ultimately high-level function of an organism.

Interaction networks summarize interactions between pairs of molecules, which are the building blocks for higher levels of molecular organization. As databases of interaction networks continue to grow in size and complexity, new computational tools are needed to search them for these higher levels of organization.

Network alignment algorithms search interaction networks from different species to identify conserved functional modules—groups of molecules that cooperate to perform a common biological function. The increasing belief that functional modules are a critical level of molecular organization has led to an upsurge of network alignment research.

This dissertation focuses on three network alignment algorithms we developed: (1) Graemlin, the first network aligner to align multiple large interaction networks efficiently; (2) Graemlin 1.1, a network aligner that uses a training set of network alignments to improve Graemlin’s accuracy; and (3) Graemlin 2.0, a network aligner that uses machine learning techniques to find the most accurate network alignments to date. Even though Graemlin 2.0 supersedes Graemlin and Graemlin 1.1, I discuss all three algorithms in detail to provide a complete account of our work.

To provide a context for our algorithms, I also discuss the intuition behind network alignment and review past research in the field. In addition, I review methods we developed to build the Stanford Network Database, a database of microbial interaction networks that we have analyzed extensively with our alignment algorithms.

Finally, I discuss several specific examples of network alignments. The examples illustrate the advantages of our alignment algorithms over other algorithms. I conclude with illustrations of applications we have developed for network alignment, including its use to predict protein function and characterize functional modules.
Acknowledgements

Much of the content in Chapter 2 paraphrases a previously submitted article [1]. In that work, Balaji Srinivasan designed the methodology behind the Stanford Network Database, wrote the article, and created Figures 2.1-2.5. I helped with the design and validation of the networks in the Stanford Network Database as well as with the design of the Stanford Network Browser.

Chapter 4 contains content from a previously published article [2]. In that work, I designed the Græmlin algorithm, wrote the article, and implemented and tested Græmlin with Tony Novak. Content from that article, which Balaji Srinivasan helped write, also appears in Section 8.1.

Chapter 6 also contains content from a previously published article [3]. In that work, I designed, implemented, and tested Græmlin 2.0, and I also wrote the article.

Section 8.3 paraphrases a submitted article [4]. In that work, I performed the network alignment and wrote the article text that describes the alignment and its analysis.

Throughout the work described in this dissertation I was supported in part by a Stanford Graduate Fellowship. Much of the work also was funded by NSF grant EF-0312459, NIH grant U01-HG003162, the NSF CAREER Award, and the Alfred P. Sloan Fellowship.

I would like to thank all of the people at Stanford who have helped me during my doctoral studies. Foremost, my advisor Serafim Batzoglou has given me invaluable support and the freedom to pursue my own research direction. Also, Balaji Srinivasan and Tony Novak have helped a great deal with the work in this dissertation. Balaji introduced me to the network alignment problem, and without him this dissertation would cover a different topic. Tony helped with much of the design behind our network alignment algorithms, and without him this dissertation would only cover Græmlin 0.9. Finally, my colleagues in Serafim's lab have helped immeasurably along the way and have helped me through many sticking points. In particular, Tom Do helped formulate the key idea of Græmlin 2.0’s scoring function, George Asimenos resolve computer and cluster issues on numerous occasions, and Eugene Fratkin helped me iron out the details behind many ideas.

Finally, I would like to thank my friends and family for all of their support. My parents have given me unquestioned love and support for which I am eternally grateful. And without my wife Katie, this work would not have been possible. The knowledge that she was there for me has helped me through the difficult times in research that come along too often.
# Contents

Abstract v

Acknowledgements vi

1 Introduction 1

1.1 Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.2 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2 The Stanford Network Database 6

2.1 Background and motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.2 Network integration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   2.2.1 Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
   2.2.2 Statistical validation and comparison . . . . . . . . . . . . . . . . . . . . . . . 9
2.3 The Stanford Network Browser . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
   2.3.1 Single protein analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
   2.3.2 Multiple protein analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
2.4 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

3 Overview of Network Alignment 17

3.1 Background and motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17
3.2 A sample network alignment problem . . . . . . . . . . . . . . . . . . . . . . . . . 19
   3.2.1 A simple formulation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
   3.2.2 Limitations of the simple formulation . . . . . . . . . . . . . . . . . . . . . . . 22
3.3 Network alignment vs. the graph matching problem . . . . . . . . . . . . . . . . . . 23
3.4 Previous work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25
3.5 Equivalence class formulation of network alignment . . . . . . . . . . . . . . . . . . 27
   3.5.1 Intuition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
   3.5.2 Formal problem definition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
3.6 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
6.4 Discussion ............................................. 97
6.A Global alignment feature function definition ........................................ 98
   6.A.1 Node feature function ............................................. 98
   6.A.2 Edge feature function ............................................. 100

7 Alignment of Known Modules ............................................. 102
   7.1 Alignments by Græmlin and Græmlin 1.1 ........................................... 102
   7.1.1 “Gaps” are important in network alignment ........................................ 103
   7.1.2 Noisy networks make network alignment more difficult ......................... 105
   7.1.3 A more lenient edge model can uncover novel connections between modules . 105
   7.1.4 Strong BLAST hits do not always imply functional orthology .................. 105
   7.1.5 Multiple alignments of many species can become unstable ....................... 106
   7.2 Alignments by Græmlin 2.0 ............................................. 107
   7.3 Discussion ............................................. 109

8 Applications of Network Alignment ............................................. 111
   8.1 Analysis of proteins and modules ............................................. 111
   8.1.1 Functional annotation ............................................. 112
   8.1.2 Module identification ............................................. 115
   8.2 Discovery of cryptic modules ............................................. 115
   8.3 Analysis of inositol metabolism ............................................. 119
   8.3.1 Background and motivation ............................................. 119
   8.3.2 The myo-inositol module in C. crescentus ........................................ 119
   8.3.3 Alignment of the myo-inositol module to other α-proteobacteria .................. 121
   8.3.4 Prediction of myo-inositol transporter in S. meliloti .................................. 122
   8.4 Discussion ............................................. 122

9 Conclusion ............................................. 124
   9.1 Open problems ............................................. 125
   9.2 Future directions ............................................. 126
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>We quantified the performance of Græmlin on 10 networks in the Stanford Network Database.</td>
<td>50</td>
</tr>
<tr>
<td>4.2</td>
<td>The Stanford Network Database contains more alignable KEGG pathways than the eukaryotic networks.</td>
<td>51</td>
</tr>
<tr>
<td>4.3</td>
<td>Græmlin performs network-to-network alignment of networks thresholded at 0.5 with high specificity, sensitivity, and efficiency.</td>
<td>53</td>
</tr>
<tr>
<td>4.4</td>
<td>Græmlin performs network-to-network alignment of networks thresholded at 0.25 with high specificity, sensitivity, and efficiency.</td>
<td>54</td>
</tr>
<tr>
<td>4.5</td>
<td>Græmlin efficiently and accurately performs query-to-network alignment of networks thresholded at 0.5.</td>
<td>55</td>
</tr>
<tr>
<td>4.6</td>
<td>Græmlin efficiently and accurately performs query-to-network alignment of networks thresholded at 0.25.</td>
<td>56</td>
</tr>
<tr>
<td>4.7</td>
<td>Græmlin efficiently and accurately performs multiple network alignment.</td>
<td>57</td>
</tr>
<tr>
<td>5.1</td>
<td>Græmlin 1.1 is more accurate than Græmlin and MaWISh.</td>
<td>76</td>
</tr>
<tr>
<td>5.2</td>
<td>Græmlin 1.1 is more specific than Græmlin.</td>
<td>77</td>
</tr>
<tr>
<td>6.1</td>
<td>Græmlin 2.0 aligns nodes with higher specificity.</td>
<td>94</td>
</tr>
<tr>
<td>6.2</td>
<td>Græmlin 2.0 aligns nodes with higher sensitivity.</td>
<td>95</td>
</tr>
<tr>
<td>6.3</td>
<td>Græmlin 2.0 groups proteins into modules with higher accuracy.</td>
<td>97</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Our network integration method uses probabilities to normalize different predictors of protein association. .................................................. 8
2.2 Our network integration method integrates multiple predictors of protein association. 10
2.3 The networks in the Stanford Network Database are more accurate than the networks in STRING. .................................................................. 12
2.4 Users can analyze individual proteins with the Stanford Network Browser. ........ 14
2.5 Users can analyze functional modules with the Stanford Network Browser. ....... 15

3.1 A toy problem illustrates many network alignment concepts. ......................... 19
3.2 A simple scoring function uses a table that specifies scores for matched nodes and edges. 20
3.3 The networks in Figure 3.1 have many possible global alignments. ................. 21
3.4 Many-to-many mappings do not always correspond to clear biological hypotheses. 22
3.5 Multiple alignments are harder to interpret than pairwise alignments. .......... 23
3.6 Local alignments are harder to score than global alignments. ...................... 24
3.7 Scoring functions that reward conserved missing edges are problematic. ....... 24
3.8 The equivalence class formulation addresses the weaknesses of the many-to-many formulation. .................................................................. 27
3.9 A network alignment is an equivalence relation. ......................................... 28

4.1 Græmlin’s scoring function measures the likelihood that an alignment represents a conserved functional module. ........................................ 32
4.2 Græmlin uses two probabilistic models to score an edge. .............................. 36
4.3 Græmlin refines a Module ESM during multiple query-to-network alignment. .... 38
4.4 Græmlin uses a progressive strategy for multiple network alignment. .......... 41
4.5 We can compute the sensitivity of Græmlin for values of \( d \) and \( T \) .......... 44
4.6 We can compute the expected running time of Græmlin for values of \( d \) and \( T \). 45
4.7 Græmlin uses a greedy algorithm to extend a seed. .................................... 46
4.8 Græmlin performs network-to-network alignment accurately. ..................... 52
4.9 Græmlin offers a speed-sensitivity trade-off and aligns multiple networks efficiently. 56
Chapter 1

Introduction

A major goal in biology today is to study function at the level of entire cells and organisms [5]. Successes in molecular biology have helped to understand how genes, proteins, and other metabolites function in isolation, but no single molecule or small group of molecules can fully explain the dynamics and function of biological systems [6]. Even simple processes such as glucose metabolism involve many enzymes and small molecules [7], many common diseases can be caused by interactions between multiple genetic variants and environmental factors [8], and complex phenomena such as epigenetic inheritance are due to interactions between many pathways and molecular complexes [9].

Nonetheless, databases of molecular information are an important first step toward the analysis of complex biological function. Although the human genome is the most famous example [10, 11], databases today contain high throughput measurements of human variation [12], gene [13] and chemical compound [14] expression signatures, epigenetic modifications [15, 16], and many other kinds of molecular information [17].

These databases contain information about the molecular building blocks of complex biological processes. Systems biology aims to understand how these building blocks organize structurally and dynamically to ultimately enable cells and organisms to function [5].

The simplest level of molecular organization, beyond that of individual molecules, is an interaction between a pair of molecules. Many types of interaction exist within a cell, such as metabolic interactions between two different enzymes, signalling interactions between transcription factors and genes, and binding interactions between proteins [18].

These different types of interactions define multiple networks within a cell. These networks combine to influence the structure and function of a cell and ultimately an entire organism [18]. Despite their different molecular mechanisms, these interaction networks share many common properties. They connect pairs of molecules through paths that contain only a few intermediate molecules, consist of a small number of highly connected “hub” molecules, and hierarchically organize into topological modules [18]. It is therefore clear that these interaction networks contain information
CHAPTER 1. INTRODUCTION

about high-level structures of molecular organization.

Functional modules [19] form one important level of molecular organization. A functional module is a collection of molecules that performs a common biological function—this function is due to the entire collection of molecules rather than any individual molecule. Functional modules have only an imprecise definition but have been the subject of substantial research, much of which focuses on the analysis of modules within interaction networks [20, 21, 22, 23].

The problem of functional module analysis within interaction networks is analogous to the problem of functional element analysis within genomic or protein sequences [24]. Functional element analyses often use a dominant biological premise: that evolutionary conservation implies functional relevance [25, 26, 27]. This premise has helped make sequence alignment, a computational method that identifies conserved elements within biological sequences, perhaps the biggest success story of computational biology [28, 29].

Network alignment is the systems biological analog of sequence alignment. Through cross-species comparison of different interaction networks, network alignment algorithms identify evolutionary conservation of various subnetworks. By extension of the premise used by sequence alignment algorithms, these conserved subnetworks represent putative functional modules.

Spurred in large part by a dramatic growth in interaction network data, network alignment has emerged as an active research area [30]. Despite the rapidly growing literature about it, network alignment is still in its infancy. Thus, much research is needed to formally define the problem of network alignment, develop algorithms to solve it accurately, and design applications of network alignment to produce biologically meaningful results.

In this dissertation I describe network alignment algorithms that we developed. I focus on the alignment of protein interaction networks—a special kind of interaction network that summarizes interactions between pairs of proteins. Within these networks, functional modules are groups of proteins that interact to perform a common biological function.

I first discuss computational methods we developed to construct and visualize protein interaction networks for any sequenced microbe. I then describe network alignment algorithms we developed that can scale to large protein interaction network datasets and produce the most accurate network alignments to date. I conclude with a description of some applications of network alignment that help analyze functional modules.

1.1 Outline

Chapter 2 describes methods we developed to build and visualize protein interaction networks. It describes a method that infers interactions between all pairs of proteins within the genome of any sequenced microbe. It also discusses the Stanford Network Database, a collection of 305 microbial protein interaction networks that we constructed with our method, and the Stanford Network
1.2. CONTRIBUTIONS

Browser, a web-based interface that allows a user to visualize and interact with the networks in the Stanford Network Database.

Chapter 3 introduces network alignment. It formally defines the network alignment problem and discusses the intuition behind network alignment algorithms. It also discusses existing approaches to network alignment.

Chapters 4-6 describe three network alignment algorithms we developed: Græmlin, Græmlin 1.1, and Græmlin 2.0. Although Græmlin 2.0 is a more advanced aligner than Græmlin 1.1, which in turn is more advanced than Græmlin, I discuss all three algorithms to give a complete history of our work and to motivate Græmlin 2.0, our current state-of-the-art aligner.

Specifically, Chapter 4 describes Græmlin, a network alignment algorithm capable of large-scale multiple network alignment. Græmlin borrows many heuristics from well-known sequence alignment algorithms, which enable it to efficiently align the large networks in the Stanford Network Database. Chapter 5 then discusses Græmlin 1.1, a more accurate version of Græmlin that learns its parameters from a training set of known network alignments. Finally, Chapter 6 introduces Græmlin 2.0, the most accurate network aligner available today. Græmlin 2.0 uses a novel approach to network alignment and automatically adapts to any network dataset.

Chapter 7 presents alignments of known functional modules to illustrate the differences between Græmlin, Græmlin 1.1, and Græmlin 2.0. It discusses some of the obstacles to accurate network alignment and describes how the approaches in Græmlin 1.1 and Græmlin 2.0 confront them.

Finally, Chapter 8 discusses some applications of Græmlin and Græmlin 2.0. We have used these algorithms to annotate proteins and modules of unknown function and to help analyze functional modules in depth.

1.2 Contributions

Every network alignment algorithm must address three subproblems. First, it must define a mathematical formulation of network alignment. Second, it must use a scoring function to assign numerical scores to network alignments—high scoring network alignments should encode valid biological hypotheses. Third, it must use a search algorithm to find high scoring network alignments.

Most network alignment research focuses on the search algorithm subproblem. This is likely due to the similarity of the network alignment problem to the general graph matching problem, on which there is a substantial literature [31]. In contrast, this dissertation argues that the mathematical formulation and scoring function subproblems are more important than the search algorithm subproblem. Specifically, it discusses a network alignment algorithm with a sophisticated scoring function and simple search algorithm that finds more accurate alignments than network alignment algorithms with sophisticated search algorithms but heuristic scoring functions.

While its main focus is the first two of the three network alignment subproblems, this dissertation
CHAPTER 1. INTRODUCTION

presents contributions to all three network alignment subproblems. It also discusses an algorithm that constructs high-quality protein interaction networks, the major dataset that existing network alignment algorithms analyze.

Specifically, this dissertation presents six contributions to the field of network alignment:

1. **A novel mathematical formulation of network alignment.** Chapter 3 introduces an equivalence class formulation for network alignment that we developed. In contrast with existing methods, alignments under the equivalence class formulation have a concrete biological interpretation and represent clear biological hypotheses about the conservation of proteins and modules. The equivalence class formulation also provides a general foundation for future network alignment research.

2. **The first large-scale multiple network aligner.** Chapter 4 describes Græmlin, the first network aligner that can align more than three interaction networks at once. Græmlin also efficiently aligns networks with many nodes and edges, in contrast to most other methods that efficiently align only small networks. Græmlin is the only network aligner that can find multiple alignments of a module to a database with hundreds of networks, a problem that gains relevance as the amount of interaction data increases.

3. **The first numerical benchmarks that quantify network alignment performance.** Chapters 4-6 present numerical benchmarks that assess the quality of network alignments—these benchmarks provide a quantitative comparison between our algorithms and other network alignment algorithms. In contrast, other network alignment research does not compare different alignment algorithms.

4. **The first network alignment scoring function trained on a collection of known network alignments.** Chapters 5-6 discuss network alignment algorithms that learn parameters for their scoring functions given a collection of known network alignments. In particular, Chapter 6 formally defines the parameter learning problem in the language of machine learning—this opens the door to statistical approaches to network alignment that complement the combinatorial approaches dominant in the literature [30].

5. **The most accurate network aligner to date.** Chapter 6 discusses Græmlin 2.0, which, according to our benchmarks, aligns networks significantly more accurately than other network alignment algorithms. Græmlin 2.0 is more accurate across multiple network datasets of different sizes, species, and mechanisms of construction. Græmlin 2.0 is accurate because it uses a sophisticated scoring function as well as a novel approach to local network alignment.

6. **Applications of network alignment to protein function prediction and module analysis.** Chapter 8 presents several novel applications of network alignment. We used our network
1.2. CONTRIBUTIONS

alignment algorithms to predict the function of unannotated proteins as well as to define hypo-
thesis modules. We also used network alignment to analyze a module for myo-inositol metabolism in Caulobacter crescentus—this analysis supported an experimental model of in-
ositol regulation, and it also suggested an inositol transporter in Sinorhizobium meliloti that sequence-similarity alone did not.

The major result of these contributions is a more accurate and principled network aligner. Fu-
ture research will undoubtedly yield still more accurate network alignment algorithms, but any such research should focus as well on the application of network alignment to novel datasets and biological problems. Chapter 9 will discuss open problems and profitable future directions for network alignment research.
Chapter 2

The Stanford Network Database

Network alignment algorithms compare interaction networks. This dissertation focuses on protein interaction networks, in which every interaction is between a pair of proteins.

Any method that constructs computational representations of protein interaction networks must predict, for all possible pairs of proteins, whether each pair interacts within a cell. This chapter discusses a computational method we developed to build protein interaction networks for all sequenced microbes.

2.1 Background and motivation

There are a wide variety of methods to predict protein interactions [32, 33, 34, 35, 36]. Some of these methods predict direct physical interactions [37], while others use correlations between protein properties to predict (possibly indirect) associations [38, 33]. A protein interaction network is a representation of these predictions of functional linkage. In these networks, nodes correspond to proteins and edges correspond to interactions between proteins.

This dissertation focuses on protein association networks—protein interaction networks with edges that correspond to protein associations. Edges therefore do not necessarily imply direct interactions between a pair of proteins but rather functional associations [1]. For example, two proteins that are a part of the same metabolic pathway but which do not directly interact are joined by an edge in a protein association network.

With many different methods that produce predictors of protein association, recent research has focused on algorithms that integrate these different predictors into a single probability of association [39, 40]. Early methods used the simple union of intersection of different predictors [41], while more recent methods used statistical approaches [42, 43, 44, 45, 46, 47]. Most of these approaches, however, focused on Saccharomyces cerevisiae. In contrast, few association networks exist for microbes, despite the large and growing number of microbial genome sequences available.
To address the limited amount of microbial network data, we developed a computational method to build protein association networks for all sequenced microbes [40, 1]. Our method uses an arbitrary number of predictors of protein association to compute the probability that two proteins are functionally linked.

We applied our method to build the Stanford Network Database, a database of association networks for 305 microbes. Our benchmarks showed that the associations in the Stanford Network Database are more accurate than those in the STRING database [36], the largest available collection of association networks. The Stanford Network Database is available through the Stanford Network Browser (http://networks.stanford.edu), a web-based interface that allows users to browse and query each of the 305 association networks.

2.2 Network integration

This section describes our method to build protein association networks. In addition, it presents numerical benchmarks that compare the Stanford Network Database to STRING.

2.2.1 Algorithm

Our method uses a training set of protein pairs. Each protein pair is a training example \((L, \vec{E})\): \(L = 1\) if the pair is functionally linked and \(L = 0\) if the pair is not functionally linked, while \(\vec{E}\) is an evidence vector. The evidence vector contains predictors of protein association (Figure 2.1A).

We used protein annotations in the KEGG database [48] to create a training set (Figure 2.1C). For each protein pair, we recorded whether both proteins had overlapping annotations (red, \(L = 1\)), whether both proteins have entirely non-overlapping annotations (blue, \(L = 0\)), or whether either protein lacked an annotation (gray, \(L = ?\)). For all microbes, we built evidence vectors from three genomic predictors of association: coinheritance [38], collocation [32], and coevolution [33]. Where available, we created additional predictors from coexpression experiments [13].

Our method uses standard machine learning techniques [40] to derive a classifier function \(\Pr(L = 1|\vec{E})\) from the training set. The classifier function maps an arbitrary evidence vector for a pair of proteins to a probability of association. Our method applies this function to all possible pairs of proteins. It uses the resulting probabilities to build an association network: edges in the network have numerical weights that indicate the probability that a pair of proteins are functionally linked.

Our method builds an association network in three steps. It first normalizes the different predictors to have a common scale. It then derives the classifier function from the normalized predictors. It finally applies the classifier function to all protein pairs.

In the first step, our method normalizes the predictors. This normalization is necessary because the raw values of the predictors have different levels of noise and scale (Figure 2.1B). Our method uses as a common scale the ability of each predictor to recapitulate the known associations in the
Figure 2.1: Our network integration method uses probabilities to normalize different predictors of protein association. A. Different predictors of association exist for each pair of proteins in a given microbe. B. Different predictors have different levels of noise and scale. Shown are probability densities for four predictors, based on measurements for all protein pairs in Helicobacter pylori 26695. C. To build our training set of labelled protein pairs, we used protein annotations in the KEGG database [50, 48]. We labeled pairs with the same annotation as linked (red, \( L = 1 \)), pairs with different annotations as unlinked (blue, \( L = 0 \)), and pairs with one or two unannotated proteins as uncertain (gray, \( L = ? \)). For clarity only a subset of edges between protein pairs are displayed. D. Our method uses the training set to compute two probability densities for each predictor: the first based on linked protein pairs (red, \( L = 1 \)) and the second based on unlinked protein pairs (blue, \( L = 0 \)). These density estimates show that each association predictor separates linked from unlinked protein pairs. Our method use Bayes’ rule to convert these density estimates to probabilities of protein association, which it then uses to discard inaccurate predictors.

training set. It first uses kernel density estimation [49] to learn probability density functions from the evidence vectors in the training set. For each predictor \( E_i \), our method learns two probability density functions: \( \Pr(E_i|L = 1) \), the probability of a predictor value for functionally linked protein pairs, and \( \Pr(E_i|L = 0) \), the probability of a predictor value for functionally unlinked protein pairs (Figure 2.1D). It then uses Bayes’ rule to infer the probability that a pair of proteins are linked for a given predictor value:

\[
\Pr(L = 1|E_i) = \frac{\Pr(E_i|L = 1) \Pr(L = 1)}{\Pr(E_i|L = 1) \Pr(L = 1) + \Pr(E_i|L = 0) \Pr(L = 0)}.
\]

Once it converts raw predictor values into probabilities, our method discards inaccurate predictors. If the probability function \( \Pr(L = 1|E_i) \) does not accurately classify the training examples, our method does not use the predictor \( E_i \) in any subsequent computations.

In the second step, our method derives the classifier function. As shown in Figure 2.2A, it first uses the training set to separate protein pairs into three groups: linked, unlinked, and uncertain.
Figure 2.2B shows that protein pairs in different groups have, on average, different predictor values. As shown by Figures 2.1B and 2.2B, different groups are distinguished more by the joint values of three predictors than they are by any single predictor.

Our method uses the predictor values for protein pairs in the linked and unlinked groups to learn the classifier function $\Pr(L = 1|\vec{E})$ (Figure 2.2C). Via kernel density estimation, it uses predictor values for linked protein pairs to obtain $\Pr(\vec{E}|L = 1)$ and predictor values for unlinked protein pairs to obtain $\Pr(\vec{E}|L = 0)$. Then, our method uses Bayes' rule to compute

$$\Pr(L = 1|\vec{E}) = \frac{\Pr(\vec{E}|L = 1)\Pr(L = 1)}{\Pr(\vec{E}|L = 1)\Pr(L = 1) + \Pr(\vec{E}|L = 0)\Pr(L = 0)}.$$ 

In the final step, our method computes the probability that uncertain protein pairs are linked. For each uncertain pair, it first computes all predictor values (Figure 2.2D). Then, it uses the classifier function to compute the probability that the pair is functionally linked (Figure 2.2E). The collection of all probabilities constitutes the association network (Figure 2.2F).

Our method generalizes to an arbitrary number of predictors. In higher dimensions, however, it uses more sophisticated procedures to account for sparse data [40].

### 2.2.2 Statistical validation and comparison

We used our method to build the Stanford Network Database, which contains protein association networks for 305 microbes. We compared the accuracy of the Stanford Network Database to the STRING database [36], the largest available collection of protein association networks.

**The STRING integration algorithm**

The STRING integration method [36] uses seven predictors of protein association: neighborhood, fusion, cooccurrence, coexpression, experimental, database, and text-mining evidence. Unlike our method, it assumes that all predictors are independent. It therefore uses the classifier function

$$\Pr(L = 1|E_1, ..., E_n) \approx 1 - \prod_{i=1}^{n}(1 - \Pr(L = 1|E_i))$$

To compute a probability of association from an evidence vector.

In reality, predictors are not independent of one another. The accuracy of STRING relative to our method depends on the degree to which the independence approximation holds.

**Comparison of the Stanford Network Database to STRING**

We compared the networks in the Stanford Network Database to the networks in STRING. We first tested the quality of the predictors used in the databases, and we then tested the accuracy of our
Figure 2.2: **Our network integration method integrates multiple predictors of protein association.** This figure shows the derivation of the classifier function for three predictors. A. Our method first divides the training set into linked (red), unlinked (blue), and uncertain (gray) pairs. B. Linked and unlinked protein pairs have, on average, different predictor values. The joint predictor values distinguish the training set groups more than any individual predictor value. C. Our method uses Bayes’ rule to calculate the classifier function, which computes the probability that a pair of proteins are functionally linked from the evidence vector. The three-dimensional surface plot shows level sets of the classifier function. As individual predictors become stronger, the posterior probability of association increases. D. Our method uses the classifier function to calculate association probabilities for uncertain protein pairs (gray). E. To build a protein association network, our method applies the classifier function to every protein pair. F. The edges in the association network have weights that correspond to the probabilities computed by the classifier function.
2.2. NETWORK INTEGRATION

integration method relative to the STRING integration method.

For all tests, we measured the ability of each predictor or method to recapitulate known protein associations in the KEGG database. We measured how many linked pairs of proteins in our training set each method predicted as linked, and we measured how many unlinked pairs of proteins in our training set each method predicted as unlinked. We used cross-validation [51] to avoid biases in favor of our method.

In detail, we measured accuracy with the AuROC (area-under-ROC) statistic. The AuROC statistic measures the area under a classifier’s ROC curve [52]. A ROC curve measures the trade-off between the true-positive rate and false-positive rate of a classifier.

We can construct a ROC curve for any association network. For any threshold \( t \), we regard protein pairs as linked if their probability of association is greater than \( t \) and unlinked if their probability of association is less than \( t \). We then compute the true positive rate

\[
TPR = \frac{TP}{P}
\]

and the false positive rate

\[
FPR = \frac{FP}{N}
\]

where \( TP \) is the number of true positives, or the number of predicted linked pairs that are linked in the training set; \( TN \) is the number of true negatives, or the number of predicted unlinked pairs that are unlinked in the training set; \( P \) is the number of positives, or the number of linked pairs in the training set; and \( N \) is the number of negatives, or the number of unlinked pairs in the training set. Higher thresholds have lower false positive rates but also have lower true positive rates.

A ROC curve for an association network plots the \( TPR \) versus the \( FPR \) for various thresholds \( t \) (Figure 2.3A). Good classifiers have high \( TPR \) for low \( FPR \), which yields AuROC values near 1.0. Bad classifiers have AuROC values near 0.5, the value that a random classifier achieves. Therefore, the AuROC value of a protein association network quantifies its accuracy.

Our first test compared the accuracy of three predictors shared by the Stanford Network Database and STRING. Figures 2.3B-D show the AuROC values for the coexpression, collocation, and coinheritance predictors. For any given organism, each predictor in the Stanford Network Database is more likely to produce networks with a higher AuROC value than the corresponding predictor in STRING.

Our second test compared the accuracy of our integration method to the STRING integration method. For each organism, we compared a network built with our method to two networks built by the STRING integration method: “assay only”, built without manually curated predictors, and “full”, built from all predictors. Figures 2.3E-F show that, with or without manually curated predictors, the STRING integration method produces less accurate networks than our method.
AuROC scatterplots show that SNB has higher accuracy

Figure 2.3: The networks in the Stanford Network Database are more accurate than the networks in STRING. A. The receiver operating characteristic (ROC) measures the accuracy of a classifier. A good classifier (blue curve) has a high true-positive rate and a low false-positive rate, in contrast to a poor classifier (red curve) or a random classifier (dashed line). The area-under-ROC (AuROC) statistic, shown here as the shaded areas, summarizes a ROC curve. A random classifier has an AuROC of 0.5 while good classifiers approach the optimal AuROC of 1.0. B,C,D. This figure shows AuROC values for association networks built from individual predictors; each dot represents a network derived for one microbe. In all three cases, the Stanford Network Database’s individual predictors provide generally higher accuracy than STRING’s individual predictors. The coexpression plot includes only 15 species because expression data is not available for all organisms. E,F. This figure shows AuROC values for the the Stanford Network Database and STRING integration methods. We compared two sets of networks built with the STRING integration method: “assay only” (without manually curated predictors) and “full” (with all predictors).
Together, these tests argue that the networks in the Stanford Network Database are more accurate than the networks in STRING. Not only are the normalized predictors used in the Stanford Network Database more accurate than the predictors used in STRING, but our integration method is also more accurate than the STRING integration method.

2.3 The Stanford Network Browser

To provide easy access to the Stanford Network Database, we developed the Stanford Network Browser (http://networks.stanford.edu). The Stanford Network Browser is a web-based interface that allows users to browse all of the networks in the Stanford Network Database. Users can see proteins predicted to associate with a given protein, see proteins predicted to associate with putative functional modules [1], and perform network alignments [2] of a putative functional modules to any network in the database.

2.3.1 Single protein analysis

Users can browse the Stanford Network Database to see information about a specific protein and its predicted associations. Users first select a species and then search for a protein, either by its name or via a free-text search of its description.

Figure 2.4A shows a sample search in Caulobacter crescentus for the CtrA (CC3035) protein, known to be a master regulator of the bacterial cell cycle [53]. Figure 2.4B shows a graph of CtrA’s predicted associations. Among the top associations are RcdA (CC3295), a degradation factor that forms an in-vivo complex with CtrA [54], and ChpT (CC3470), a histidine phosphotransferase that phosphorylates CtrA [55].

2.3.2 Multiple protein analysis

Users can also use the Stanford Network Browser to analyze a set of proteins or a putative functional module. Users can find proteins predicted to associate with the entire set of proteins and can also use network alignment to see if the set of proteins is conserved across multiple species.

Users first select a species and a set of proteins. The Stanford Network Browser supports free-text search as a method to find proteins that match a description. Figure 2.5A shows a sample search for proteins with descriptions that contain the terms “spore” or “sporulation” [56] in Bacillus subtilis.

To find new association partners for the set of proteins, users click the “protein recommender” [1] button in the main browser window (Figure 2.5B). The Stanford Network Browser then displays a ranked list of proteins predicted to associate with the entire set of proteins. For example, of the
top ten proteins predicted to associate with the set of sporulation proteins, all mentioned in at least one publication are known to be involved in sporulation or spore-coat biogenesis.

The protein recommender can suggest novel experimental targets. For example, of the top ten proteins predicted to associate with the set of sporulation proteins, at least five are annotated as “hypothetical proteins”. Based on their association with proteins known to play a role in sporulation, it is likely that these hypothetical proteins also play a role in sporulation.

To see if the set of proteins is conserved in other species, users click the “network alignment” button in the main browser window. This displays a new window in which users select a set of organisms and then run the Graemlin network alignment algorithm (Chapter 4). The Stanford Network Browser then displays the species to which the set of proteins aligns, as well as the resulting alignments.

2.4 Discussion

Protein association prediction is an important but difficult biological problem. Existing methods focus mostly on S. cerevisiae and do not handle multiple predictors of protein association in a statistically rigorous manner. Our network integration algorithm, outlined in this chapter, has several benefits.

First, our method can compute three of its predictors (collocation, coinheritance, and coevolution) given only a genome sequence. Our method therefore can build a protein association network
2.4. DISCUSSION

Figure 2.5: Users can analyze functional modules with the Stanford Network Browser. A. Users select “build a query” and then specify a species and a search query to select a set of proteins. B. In the main window, users click the protein recommender button to build a ranked list of proteins predicted to associate with the set of proteins. The table on the right displays the ranked list, while the window on the left displays the network context. Proteins present in the original query are colored red and novel candidates are colored blue.

for any sequenced microbe.

Second, our method can integrate different predictors measured on different scales. Because it benchmarks predictors in terms of their ability to predict known protein associations, our method can normalize different predictors and ignore inaccurate predictors.

Third, our method accounts for dependencies among different predictors. Because it directly learns a joint probability function \( \Pr(L = 1|E_1, .., E_n) \), it does not need to make approximations of independence like those made by the STRING integration method [36]. The result is more accurate networks (Figure 2.3).

Finally, our method generalizes from a small set of known protein associations to predict associations for every pair of proteins in a cell. The example in Figure 2.5 shows that our method can accurately predict associations between proteins involved in processes such as sporulation that do not occur in our training set.

We developed the Stanford Network Browser to enable users to browse the Stanford Network Database, a database of 305 microbial networks that we constructed with our method. The Stanford Network Browser is integrated with tools we developed to analyze sets of proteins: the protein recommender [1] and Graemlin, a network aligner [2]. These tools aid protein discovery because users can search for proteins that are predicted to associate with well known proteins or sets of proteins.

In the future, our method can extend to include additional predictors of protein association, such as regulatory motifs and structural information. Our method can also adapt to predict more details
about the molecular biology of protein association, such as causal associations [57, 58] or triplet dependencies between proteins [59].

The remaining chapters of this dissertation describe network alignment algorithms, which align protein association networks. While network alignment algorithms can align any set of interaction or association networks, this dissertation focuses on alignments of the networks in the Stanford Network Database.
Chapter 3

Overview of Network Alignment

The Stanford Network Database contains high-quality protein association data for 305 microbes. It exemplifies a general trend—experimental and computational methods now generate large amounts of interaction network data [60]. Just as large genomic sequence databases spurred improvement in methods for computational sequence analysis, large network databases demand new methods for computational network analysis.

Network alignment borrows many ideas, both conceptual and practical, from sequence alignment, perhaps the most important problem in computational biology [61]. Just as sequence alignment algorithms find subsequences conserved in DNA or protein sequences from multiple species, network alignment algorithms find subnetworks conserved in interaction networks from multiple species. Both classes of algorithms premise that evolutionarily conserved substructures are likely functional.

This chapter discusses the main concepts behind network alignment. In addition, it motivates and describes the mathematical formulation of network alignment that the algorithms discussed in Chapters 4-6 use.

3.1 Background and motivation

Interaction networks can help address many important biological problems. The main focus of this dissertation is on one problem in particular: the analysis of functional modules. A functional module is a collection of molecules that performs a common biological function [19]. Special cases of functional modules are metabolic pathways, or chains of enzymes that convert a biological substrate into a product, and protein complexes, or groups of proteins that stably associate.

Interaction networks encode information about functional modules. A key challenge is the design of computational methods that extract this information. For example, to find functional modules, some methods analyze a network for overrepresented motifs [20, 21] or dense subnetworks [22, 23].

Network alignment algorithms, or network aligners, assume that functional modules are conserved
by evolution. Specifically, if two species share a trait controlled by a functional module, then the molecules that constitute the module, as well as the interactions between them, must be conserved in both species.

Formally, network alignment is a methodology that maps proteins and interactions in one organism with their counterparts in another organism. Network alignment algorithms therefore measure the degree of network conservation across multiple species. Network alignments can help to address several biological questions [30]:

- Which sets of molecules in a cell are organized into functional modules?
- Which species contain instances of a specific functional module?
- Which proteins, interactions, and groups of interactions are likely to have equivalent functions across several species or tissues?
- How do proteins, modules, networks, and whole species evolve?
- Can information from multiple species improve network accuracy?

Many of these questions require alignments of complex networks with multiple types of molecules and interactions. For simplicity, this dissertation discusses only algorithms that align undirected protein interaction networks. However, the ideas in these algorithms provide a foundation for the alignment of more complex networks.

This dissertation distinguishes four types of network alignment:

Global network-to-network alignment is a comparison of two networks. The goal is to map every node in one network to a node in the other network. Nodes that do not have counterparts in the other network are present in the alignment, but without a matched partner. Global network alignments measure overall network conservation.

Local network-to-network alignment is a search for conserved subnetworks within two networks. The goal is to find a set of high scoring local alignments: each local alignment contains a subnetwork from each of the two networks, as well as a mapping among the nodes in the subnetworks. Local alignments usually represent conserved functional modules.

Query-to-network alignment is a search of a database of modules or networks for a match to a known or hypothetical module. The goal is to ascertain the presence or absence of the module in other species.

Multiple network alignment is the simultaneous comparison of more than two networks. Multiple network alignments can be global or local as well as network-to-network or query-to-network.
3.2 A SAMPLE NETWORK ALIGNMENT PROBLEM

Each type of network alignment corresponds to a problem in sequence alignment. This analogy allows network alignment algorithms to borrow ideas from established sequence alignment algorithms.

Regardless of its type, every network alignment algorithm must address three subproblems:

1. **Mathematical formulation.** All algorithms must represent network alignments mathematically. Any mathematical formulation must specify the biological hypotheses made by each network alignment but must also allow for a tractable algorithm to find network alignments.

2. **Scoring function.** All algorithms must assign numerical scores to network alignments. High scoring alignments should make correct biological hypotheses.

3. **Search algorithm.** All algorithms must search for high scoring network alignments. Search algorithms should find the alignment (or set of local alignments) with the highest possible score.

### 3.2 A sample network alignment problem

A simple toy problem illustrates many of the concepts in network alignment. Figure 3.1 shows two simple networks used in the toy problem.

Figure 3.1: A toy problem illustrates many network alignment concepts. This figure shows two networks used in the toy problem.
CHAPTER 3. OVERVIEW OF NETWORK ALIGNMENT

A simple scoring function uses a table that specifies scores for matched nodes and edges. This table gives scores for alignments of the networks in Figure 3.1. Aligned nodes receive the score specified in the table. Conserved interactions receive a reward of +1 and non-conserved interactions receive a penalty of −1.

### 3.2.1 A simple formulation

A simple mathematical formulation of global pairwise network alignment is a one-to-one mapping between the nodes in two networks. A node mapped to a special “null” node has no counterpart in the other network.

Figure 3.2 shows a simple scoring function. It consists of a table that specifies the score for potential mapped pairs of nodes; mapped pairs of nodes that do not appear in the table score −∞. In addition, the scoring function includes a penalty of −1 for every interaction that is not conserved and a reward of +1 for every interaction that is conserved. All nodes mapped to the “null” node receive score 0, as do all edges incident to them.

Figure 3.3 shows two global alignments of the networks in Figure 3.1. The alignment in Figure 3.3A maps each node to its highest scoring counterpart. We can compute the score with the scoring function in Figure 3.2:

- We ignore any nodes mapped to the null label, such as A6, and any edges incident to them, such as (A6, A3).

- We use Figure 3.2 to score the seven aligned node pairs: their score is 3+3+3+3+2+3 = 20.

- We examine all pairs of aligned node pairs to compute the edge score. For example, network A but not network B has an edge between A3/B3 and A4/B4—the edge score for the pair is

<table>
<thead>
<tr>
<th>A1</th>
<th>B1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A6</td>
<td>B1</td>
<td>2</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>3</td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
<td>3</td>
</tr>
<tr>
<td>A4</td>
<td>B4</td>
<td>3</td>
</tr>
<tr>
<td>A5</td>
<td>B5</td>
<td>3</td>
</tr>
<tr>
<td>A8</td>
<td>B8</td>
<td>2</td>
</tr>
<tr>
<td>A8</td>
<td>B11</td>
<td>2</td>
</tr>
<tr>
<td>B8</td>
<td>B11</td>
<td>1</td>
</tr>
<tr>
<td>A9</td>
<td>B9</td>
<td>3</td>
</tr>
<tr>
<td>A10</td>
<td>B10</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 3.2: A simple scoring function uses a table that specifies scores for matched nodes and edges. This table gives scores for alignments of the networks in Figure 3.1. Aligned nodes receive the score specified in the table. Conserved interactions receive a reward of +1 and non-conserved interactions receive a penalty of −1.
### 3.2. A SAMPLE NETWORK ALIGNMENT PROBLEM

![Network Diagram](image)

Figure 3.3: The networks in Figure 3.1 have many possible global alignments. Shown are two sample alignments. **A.** The first alignment maps every node to its highest scoring counterpart. **B.** The second alignment has the highest possible score.

therefore $-1$. On the other hand, both networks have an edge between $A3/B3$ and $A5/B5$—the edge score for the pair is therefore $+1$. In this manner, the total edge score is $-4 - 1 + 1 - 1 + 1 + 1 = -2$.

- The total alignment score is $20 + (-2) = 18$.

The alignment in Figure 3.3B is the optimal alignment of the networks in Figure 3.1. It maps $B1$ to $A6$ instead of $A1$. This results in a change in node score of $-1$ but a change in edge score of $+2 - (-4) = 6$; the score of the optimal alignment is therefore 23.

One biological interpretation of a one-to-one node mapping is that matched nodes are *functional orthologs* [62]—proteins that perform analogous functions in two different organisms. In the alignment in Figure 3.3B, node $B1$ is mapped to node $A6$ even though it has a higher node score when mapped to node $A1$. This illustrates a general phenomenon: often the best sequence match to a protein is not its true functional ortholog [63]. Network alignment can identify functional orthologs that share significant interaction conservation even if they share sub-optimal sequence conservation.
CHAPTER 3. OVERVIEW OF NETWORK ALIGNMENT

3.2.2 Limitations of the simple formulation

One-to-one mappings cannot capture all biological relationships among proteins. For example, within many conserved pathways in the KEGG database [48], some proteins have multiple functional orthologs in a single species. One-to-one mappings can only specify one functional ortholog for each protein.

In contrast, many-to-many mappings can specify multiple functional orthologs for each protein. However, many-to-many mappings do not always correspond to clear biological hypotheses. For example, the alignment in Figure 3.4 maps node B1 to nodes A1 and A6. It therefore hypothesizes that proteins B1 and A1 are functional orthologs and also that proteins B1 and A6 are functional orthologs. Intuitively, therefore, proteins A1 and A6 must be functional orthologs. However, the alignment does not satisfy this intuition because it does not map node A1 to node A6.

Many-to-many mappings of more than two networks are harder to interpret. As an alignment includes more networks, more nodes map to one another and the mappings become increasingly difficult to reconcile. For example, in the alignment in Figure 3.5, it is unclear if proteins A7 and B12 are functional orthologs or if protein B8 is functionally orthologous to protein C8 or protein B11.

The simple network alignment formulation in Section 3.2.1 has two other limitations.

First, it applies only to networks with binary edges. In reality, many networks, such as those in the Stanford Network Database (Chapter 2), have edges with weights that quantify the confidence
3.3. NETWORK ALIGNMENT VS. THE GRAPH MATCHING PROBLEM

Figure 3.5: **Multiple alignments are harder to interpret than pairwise alignments.** It is unclear if proteins $A7$ and $B12$ are functional orthologs or if protein $B8$ is functionally orthologous to protein $C8$ or protein $B11$.

that two proteins interact.

Second, the simple formulation applies only to global network alignment and is not easy to extend to local network alignment. For example, the two local alignments in Figure 3.6 have equally high conservation. However, the alignment in Figure 3.6A has an extra conserved edge and therefore a higher score than the alignment in Figure 3.6B. This suggests that the simple scoring function will preferentially reward local alignments with many edges and implicitly penalize local alignments with few edges. For example, alignments of well-conserved pathways will have lower scores than alignments of well-conserved protein complexes.

A modified scoring function that rewards not only conserved edges but also conserved missing edges assigns the same score to both alignments in Figure 3.6. However, such a scoring function will also assign the same score to the alignment in Figure 3.7, even though the proteins in that alignment have no interactions between them.

Section 3.5 discusses a mathematical formulation of network alignment that addresses many of the weaknesses in the simple network alignment formulation. However, even state-of-the-art network alignment scoring functions still have many limitations.

### 3.3 Network alignment vs. the graph matching problem

The network alignment problem is closely related to the general graph matching problem [31]. For example, the subgraph isomorphism problem, to determine if a graph is isomorphic to a subgraph of another graph, is conceptually similar to query-to-network alignment, and the maximum common subgraph isomorphism problem, to determine the largest subgraph of one graph isomorphic to a
Figure 3.6: **Local alignments are harder to score than global alignments.** The simple scoring function in Section 3.2.1 preferentially rewards alignments with many edges. **A.** A completely conserved alignment with three edges receives an edge score of $+3$. **B.** A completely conserved alignment with two edges receives an edge score of $+2$. It therefore scores less than the alignment with three edges even though it is equally conserved.

Figure 3.7: **Scoring functions that reward conserved missing edges are problematic.** An alignment with no edges scores highly when conserved missing edges receive a reward.
3.4. PREVIOUS WORK

A subgraph of another graph, is conceptually similar to local network alignment.

Nonetheless, the problems are different. For example, because they represent interaction between proteins within an organism, interaction networks obey an evolutionary relationship. This strongly constrains alignments of networks—for example, it is very unlikely that a correct alignment will match proteins with no sequence similarity. In contrast, graph matching algorithms such as those that process images [31] treat nodes as indistinguishable.

In addition, interaction networks are often noisy. Most interaction networks are undirected graphs with a low graph diameter [18] and a high degree of topological uncertainty. As an extreme example of noisy graph structure, interaction networks based primarily on yeast two-hybrid data may not even be alignable, as several studies have questioned the reliability of the yeast two-hybrid assay [64, 65, 66].

Because of the substantial graph matching literature [31], general graph matching algorithms have influenced network alignment algorithms. Most network alignment algorithms that show this influence restrict their network alignment formulation in order to apply established graph matching techniques [67, 68]. Often, however, the restricted formulations cannot represent many biological relationships.

3.4 Previous work

Before the development of network alignment algorithms, researchers used cross-species network comparisons to analyze known metabolic pathways. For example, one study [69] used biochemical data and comparative genomics to align the glycolytic pathway. Notably, the study considered the conservation of the entire biochemical capacity of the pathway, rather than conservation of individual genes. Another study used hierarchical relationships among enzymes to compare multiple metabolic pathways [70].

Other studies used gene sequence conservation to infer interaction conservation. “Interologs”—pairs of interacting proteins with corresponding orthologs in another species—helped transfer interactions from one species to another [71]. “Regulogs” extend the concept of interologs to consider conserved regulatory relationships [72]; both interologs and regulogs, however, do not use interaction conservation to match genes to one another. The same concepts behind interologs helped investigate the conservation of gene coexpression networks across different species [73].

Two studies took steps toward general network alignment algorithms. The first [74] mined conserved clusters of genes within multiple genomes. The study represented both pathways and genomes as graphs and searched for correlated clusters within the resulting graphs. It considered both gene and interaction conservation and searched for clusters in the complete genomes of 10 microbes. The second study [75] defined a general distance measure between two pathways—a simple network alignment scoring function.
The creation of PathBLAST, the first general purpose network aligner [63], brought these developments to a head. PathBLAST uses a general scoring scheme for network alignment, which considers both protein and interaction conservation, and uses a randomized algorithm to find conserved pathways. It is also available through a web interface [76]. PathBLAST’s initial application to the alignment of *Helicobacter pylori* 26695 and *Saccharomyces cerevisiae* networks revealed several insights—in particular, that the true functional ortholog of a protein is not always its best sequence match. This insight motivated general alignment tools that consider edge conservation, in contrast to earlier studies that considered only protein conservation.

Since its introduction, PathBLAST has undergone two improvements. The first improvement extended it to identify conserved protein complexes as well as conserved pathways [77]. This work also introduced a general probabilistic framework to score alignments of general modules as well as an alignment technique that paralleled the seed-extension paradigm of BLAST. The second PathBLAST improvement resulted in a new algorithm, NetworkBLAST [78], which can align three species simultaneously. Studies have used NetworkBLAST to compare *S. cerevisiae*, *Caenorhabditis elegans*, and *Drosophila melanogaster* networks—the resulting alignments helped annotate proteins and infer protein interactions. NetworkBLAST has also helped interpret genetic interactions [79] and compare the *Plasmodium falciparum* network to other eukaryotic networks [80].

Research into scoring functions has focused on evolutionary models. MaWISh [81] pioneered this effort via its introduction of a scoring function based on the duplication-divergence model of network evolution. A recent study [82] introduced a stochastic model for sequence and interaction evolution and used Bayesian analysis to infer both optimal alignment parameters as well as high scoring alignments. Another study modeled protein complex evolution with interaction gains and losses as well as protein duplications [83].

Most network alignment research, however, has focused on search algorithms. MaWISh searches for conserved dense subgraphs within two networks via a greedy algorithm similar to that used by NetworkBLAST. MetaPathwayHunter [67] uses a graph matching algorithm to find inexact matches to a query pathway in a network database, and QNet [68] exactly aligns query networks with bounded tree width. Other network alignment algorithms use ideas behind Google’s PageRank algorithm [84] or cast network alignment as an Integer Quadratic Programming problem [85].

Chapters 4-6 present alignment algorithms with several further contributions. Chapter 4 discusses Graemlin, the first network aligner capable of large-scale multiple network alignment, and Chapter 5 presents an improvement to Graemlin’s scoring function. Chapter 6 discusses Graemlin 2.0, the first alignment algorithm that automatically learns a scoring function from a database of known network alignments.
3.5. EQUIVALENCE CLASS FORMULATION OF NETWORK ALIGNMENT

This section discusses the mathematical formulation of network alignment used in the remaining chapters. Both Græmlin and Græmlin 2.0 define network alignments as equivalence relations.

3.5.1 Intuition

The many-to-many formulation of network alignment does not enforce the intuition that functional orthology is a transitive relationship. The equivalence class formulation of network alignment addresses this problem.

The equivalence class formulation defines a network alignment as an equivalence relation over the nodes in the aligned networks. An equivalence relation is reflexive, symmetric, and transitive, and partitions the proteins into a set of equivalence classes. Biologically, each equivalence class is a group of functional orthologs with a common ancestor.

Under this formulation, the alignment in Figure 3.5 is not a legal alignment; on the other hand, its transitive closure, shown in Figure 3.8, is a legal alignment. The equivalence relation represents a conserved pathway: it hypothesizes that the ancestral pathway had three proteins and has descendents in all three extant species. The middle ancestral protein has two descendents in species B.

3.5.2 Formal problem definition

Formally, the input to multiple network alignment is $d$ networks $G_1, \ldots, G_d$. Each network $G_i$ represents a different species and contains a set of nodes $V_i$ and a set of edges $E_i$ that link pairs of

Figure 3.8: The equivalence class formulation addresses the weaknesses of the many-to-many formulation. The equivalence class formulation indicates that the alignment in Figure 3.5 represents a conserved three-node pathway.
A network alignment is an equivalence relation. In this example, four protein interaction networks are inputs to multiple network alignment. A network alignment partitions nodes into equivalence classes (indicated by boxes).

Network alignments have a biological interpretation. Nodes in the same equivalence class represent functionally orthologous proteins. Within a species, the subset of nodes in a local alignment represents a conserved functional module.

A scoring function for network alignment is a map \( s : \mathcal{A} \rightarrow \mathbb{R} \), where \( \mathcal{A} \) is the set of potential network alignments of \( G_1, \ldots, G_d \). The global network alignment problem is to find the highest-scoring global network alignment. The traditional local network alignment problem is to find a set of maximally scoring local network alignments. The alignment algorithm discussed in Chapter 6 solves a different local network alignment problem: to find the highest-scoring overall set of local network alignments.

3.6 Discussion

Network alignment is a more difficult problem than sequence alignment. While the sequence alignment problem is solvable in polynomial time, it is computationally hard to design search algorithms for simple network alignment formulations [30]. Furthermore, few network alignment formulations
represent concrete but general biological hypotheses. The many different formulations in the literature attest to this.

The equivalence class formulation addresses many weaknesses of existing network alignment formulations. It is more general than the one-to-one formulation but has a more concrete biological interpretation than the many-to-many formulation. It applies equally well to pairwise and multiple alignment, and, as discussed in Chapters 5-6, it permits a principled scoring function.
Chapter 4

Græmlin

This chapter discusses Græmlin [2], the first large-scale multiple network aligner. We designed Græmlin to align the networks in the Stanford Network Database. These networks differ from other network databases such as DIP [86] and IntAct [87] because they contain many more species and have many more edges.

Græmlin is the first network aligner that can efficiently align multiple networks as large as those in the Stanford Network Database. To do this, Græmlin translates many established heuristics from sequence alignment to network alignment.

This chapter describes the initial version of Græmlin. Chapter 5 discusses subsequent improvements we have made to its scoring function.

4.1 Background and motivation

As discussed in Chapter 3, many network alignment algorithms have been developed. They include early algorithms whose main novelty is the introduction of the network alignment problem [63, 78], algorithms that include concepts of network evolution [81, 82, 83], and sophisticated algorithms that specially formulate the network alignment problem in order to reduce it to well-studied mathematical problems [67, 68, 84, 85].

However, few network alignment algorithms focus on the practical alignment of many large interaction networks. We designed Græmlin with this focus: Græmlin is fast, scalable, and is the first program capable of multiple alignment of an arbitrary number of networks.

Græmlin can generate an exhaustive list of conserved modules in a set of networks (multiple local network-to-network alignment) and can find matches to a particular module within a database of interaction networks (query-to-network alignment). Because it can align many networks at once, Græmlin enables comparative analyses that focus on functional modules present within all microbes or within a particular clade of microbes.
4.2. SCORING FUNCTION

Græmlin uses several strategies common in sequence alignment [61]. First, its variant of progressive alignment [88] scales linearly with the number of networks aligned. Second, Græmlin uses a modification of the seed extension method popularized by BLAST [89] to search for pairwise alignments. Finally, Græmlin offers an explicit speed-sensitivity trade-off through the control of a parameter analogous to the BLAST k-mer size [90].

To assess Græmlin’s ability to find conserved functional modules, we performed the first quantitative comparison of network alignment algorithms. Our benchmarks show that Græmlin accurately aligns known biological modules [50, 48] and performs fast and scalable multiple network alignment.

As for all network alignment algorithms, Græmlin must address the three network alignment subproblems: it must define a mathematical formulation, a scoring function, and a search algorithm. Græmlin uses the equivalence class formulation of network alignment discussed in Chapter 3. The sections below describe its scoring function and search algorithm.

4.2 Scoring function

Under the equivalence class formulation, a network alignment has two biological interpretations. First, proteins in the same equivalence class are functionally orthologous. Second, within a species, the subset of proteins in a local alignment represents a functional module. Taken together, these interpretations imply that a local alignment represents a conserved functional module—each equivalence class corresponds to a protein in the ancestral module.

This interpretation suggests a natural local alignment scoring function. Namely, high scoring local alignments should align subnetworks that are extant instances of an ancestral functional module. Low scoring local alignments should align subnetworks that either have no evolutionary relationship or are not true functional modules.

Græmlin’s scoring function uses two models that assign probabilities to the evolutionary events leading from the hypothesized ancestral module to the modules in the extant species. The alignment model \( \mathcal{M} \) posits that each equivalence class groups proteins descended from a common ancestral protein and that the ancestral proteins together form an ancestral functional module. The random model \( \mathcal{R} \) posits that the nodes in the alignment have no relationship. The score of the alignment is the log-ratio of the probabilities assigned by the two models—many sequence alignment scoring functions use analogous log-ratios [29]. Figure 4.1 shows a sample alignment, together with an overview of Græmlin’s scoring function.

Græmlin independently scores each equivalence class (node scoring) and each edge (edge scoring) within an alignment. The score of an alignment is the sum of all equivalence class and edge scores.
Figure 4.1: Græmlin’s scoring function measures the likelihood that an alignment represents a conserved functional module. A. Græmlin takes multiple networks as input and finds local multiple alignments within them. A local multiple alignment partitions a subset of nodes into disjoint equivalence classes (numbered 1 through 4). B. Græmlin first scores each equivalence class separately. It reconstructs the most parsimonious ancestral history of the proteins in the equivalence class and then assesses penalties for sequence mutations and protein insertions, deletions, duplications, and divergences. Græmlin scores sequence mutations with weighted sum-of-pairs scoring, with pairwise scores obtained from BLAST results, and scores all other events using heuristic constants. C. Græmlin scores each edge using an Edge Scoring Matrix (ESM) as described in the text. The figure shows three alternative ESMs.
4.2. SCORING FUNCTION

4.2.1 Node scoring

Graemlin’s node scoring function scores each equivalence class in the alignment. It bases scores on three types of evolutionary events: protein sequence mutations, protein insertions and deletions, and protein duplications and divergences. Protein insertions and deletions, as well as duplications and divergences, are reciprocal events.

Graemlin first assigns a score to an equivalence class based on protein sequence mutations. It then adds penalties for each of the other events to obtain the final score of an equivalence class.

Scores for protein sequence mutations

Graemlin scores protein sequence mutations with weighted sum-of-pairs scoring. First, for each pair of nodes $x_1, x_2$ in an equivalence class, it assigns a score

$$ S_N(x_1, x_2) = \log \frac{Pr_M(x_1, x_2)}{Pr_R(x_1, x_2)} $$

where $Pr_M$ is the probability assigned to the pair by $M$, and $Pr_R$ is the probability assigned to the pair by $R$. Given a complete set of pairwise scores, Graemlin uses the weighted sum-of-pairs score [91], taken over all pairs of nodes in the equivalence class, as the score of all mutations in the equivalence class.

Both models $M$ and $R$ assign probabilities to a pair of nodes based on the BLAST alignment [90] of the proteins they represent. Each model uses a continuous probability distribution ($f_M$ for $M$ or $f_R$ for $R$) over all possible BLAST bitscores. For example, if the bitscore of two proteins $x_1$ and $x_2$ is $S$, the probability assigned by the alignment model is $Pr_M(x_1, x_2) = \int_{S-\Delta}^{S+\Delta} f_M(t) dt$, where $\Delta$ is a small constant.

We used sample protein pairs to determine the continuous probability distributions. To determine $f_M$, we sampled random pairs of proteins from within the same COG [92]. To determine $f_R$, we sampled completely random pairs of proteins. We smoothed both models using a monotone regression function [63].

Scores for other events

Graemlin scores protein insertions, deletions, duplications, and divergences based on the most parsimonious evolutionary history of an equivalence class. First, Graemlin uses a dynamic programming algorithm to compute the most likely number of proteins at each ancestral species in the phylogenetic tree. Next, it assigns protein insertion, deletion, duplication, and divergence events to branches in the tree: a protein insertion occurs when the number of proteins increases from 0 to 1 along a branch, a protein deletion occurs when the number of proteins decreases from 1 to 0 along a branch, a protein duplication occurs when the number of proteins increases from a non-zero number along a branch, and a protein divergence occurs when the number of proteins decreases to a non-zero
number along a branch. Finally, Græmlin assigns penalties to each such event $e$ using the function $s(e) = \log \frac{\Pr_M(e)}{\Pr_R(e)}$.

In practice, it is difficult to obtain estimates of the probabilities $\Pr_M(e)$ and $\Pr_R(e)$. Therefore, Græmlin uses heuristic choices for $s(e)$. It uses the same penalty for protein deletions and insertion and the same penalty for protein duplications and divergences.

Græmlin’s scoring function therefore needs only two heuristic parameters: deletion and duplication penalties. These penalties are analogous to the gap penalty in sequence alignment [29] and control the relative importance of the insertion and duplication scores. Just as Græmlin estimates parameters for protein sequence mutation scores from biological data but heuristically sets deletion and duplication penalties, sequence alignment algorithms usually estimate substitution matrices from biological data [93, 94] but heuristically set gap penalties [95, 96, 97, 98].

4.2.2 Edge scoring

After it scores each equivalence class, Græmlin scores each edge in the alignment. It bases the score of an edge on its weight and its incident nodes.

Both models $M$ and $R$ use continuous probability distributions ($g_M$ for $M$ or $g_R$ for $R$) over all possible edge weights between 0 and 1. For each pair of nodes $x, y$ in the alignment, where $x$ and $y$ belong to the same species but different equivalence classes, Græmlin assigns an edge score of

$$S_E(x, y) = \log \frac{\int_{w-\Delta}^{w+\Delta} g_M(u)du}{\int_{w-\Delta}^{w+\Delta} g_R(u)du}$$

where $w$ is the weight of the $(x, y)$ edge, and $\Delta$ is a small constant.

Sometimes, there is no edge between $x$ and $y$. For example, although the Stanford Network Database assigns a weight to each pair of proteins, in practice many of these weights are close to zero. Thus, to avoid unnecessary computation, Græmlin ignores all edges below a threshold $t$. If there is no $(x, y)$ edge in the thresholded network, Græmlin can only assume that the edge between the $x$ and $y$ has weight less than $t$; it therefore assigns to $(x, y)$ an edge score of

$$S_E(x, y) = \log \frac{\int_0^t g_M(u)du}{\int_0^t g_R(u)du}$$

The edge weight distribution function $g_R$ for the random model depends on the nodes to which an edge is incident. This captures the intuitive notion that in any graph, two nodes of high degree are more likely to interact by chance than two nodes of low degree. The function $g_R$ is an exponential distribution parameterized by the product of the relative expected degrees of the nodes incident to an edge: the relative expected degree of a node is the sum of all incident edge weights divided by the total number of nodes in the network. To determine $g_R$, we divided the range of relative expected
4.2. SCORING FUNCTION

degrees into bins, sampled for each bin random pairs of nodes with relative expected degree product within the bin, and fit for each bin an exponential distribution to the resulting sample data points.

nodes from an un-thresholded network for various node degrees. We then mapped the parameters to make the distribution roughly linear and fit a curve using linear regression.

The edge weight distribution function $g_M$ for the alignment model is more complex. As discussed in Chapter 3, Graemlin cannot always assume that an edge existed between every pair of proteins in the ancestral module—scores based on this assumption reward highly connected modules more than equally conserved but weakly connected modules.

To define $g_M$, Graemlin therefore uses an Edge Scoring Matrix (ESM)—a novel scoring scheme that allows a user to specify the desired ancestral topology. This generalizes previous edge-scoring approaches [78] and permits searches for arbitrary module structures, including as special cases protein complexes and pathways. An ESM encapsulates the desired module structure within a symmetric matrix.

Figure 4.2 shows the methodology that both the random and alignment models use to score a single edge.

ESMs

An ESM has a set of labels that index its rows and columns. Each cell $i, j$ in the matrix contains a probability distribution $g_{ij}$ over edge weights. Given a labeling of the alignment—an assignment to each equivalence class one of the labels from the ESM—Graemlin determines the score for an edge $e = (x, y)$ from the labels of the equivalence classes to which $x$ and $y$ belong. These labels index a distribution $g_{ij}$, which Graemlin uses for $g_M$.

To score all of the edges in an alignment, Graemlin uses the labeling that results in the highest combined score. For a general ESM, Graemlin must therefore examine an exponential number of labelings. On the other hand, for certain ESMs, Graemlin can score edges efficiently. Graemlin therefore only supports three special but important ESMs: a Complex ESM, a Pathway ESM, and a Module ESM.

A Complex ESM rewards highly connected alignments, such as those that represent protein complexes [81, 77]. It has only one label and therefore one distribution $g_M$, which is biased toward high edge weights. Graemlin can easily score edges with a Complex ESM because it must assign all equivalence classes the same label.

Graemlin’s Complex ESM defines $g_M \sim N(0.5, 0.8^2)$ (a Gaussian distribution with mean of 0.5 and standard deviation of 0.8). We obtained the parameters for the distribution empirically. The wide standard deviation assigns a moderate probability to high weight edges and assigns a small but significant probability to missing edges—this accounts for noisy networks, which often lack edges between proteins that interact. In contrast, a narrow standard deviation assigns a very low probability to missing edges and will cause alignments with any missing edges to have very low
Figure 4.2: *Graemlin uses two probabilistic models to score an edge.* A. To score an edge of moderate weight, Graemlin integrates both the random model distribution function $g_R$ and the alignment model distribution function $g_M$ over a small area centered at the edge weight. The random model bases its distribution on the degrees of the nodes connected by the edge, while the alignment model bases its distribution on the labeling of the nodes and the ESM. B. Graemlin scores edges with weight below the minimum edge weight threshold $t$ in a similar manner, but it now integrates both distributions from 0 to $t$. 
4.3 Search algorithm

Græmlin uses a progressive alignment strategy to perform multiple network alignment. At each step, it selects the two closest networks, according to a phylogenetic tree, and performs a pairwise alignment. From the pairwise alignment, Græmlin generates a new network that it uses in subsequent
Figure 4.3: **Graemlin refines a Module ESM during multiple query-to-network alignment.** As Graemlin successively adds species to the multiple alignment, the distributions in the ESM cells change to reflect the new edges. In the figure, at each step, the cell with a modified distribution is highlighted together with the edge that caused the change.
steps. Graemlin continues until it has aligned all networks.

During pairwise alignment, Graemlin first finds a set of seeds between two networks and then extends each seed into an alignment. Due to the progressive alignment technique, at any given step the nodes in the networks may correspond to equivalence classes rather than proteins.

Figure 4.4 shows an outline of the Graemlin’s search algorithm, including the methodology it uses for pairwise and multiple alignment.

### 4.3.1 Pairwise alignment

To search for high scoring alignments between a pair of networks, Graemlin first generates a set of seeds. The seeds restrict the size of the search space. Graemlin uses a greedy algorithm to transform each seed into a high scoring alignment.

**d-clusters**

Graemlin uses d-clusters to generate seeds. d-clusters are conceptually similar to k-mers, which the BLAST algorithm uses to generate seeds for sequence alignment.

**Methodology** A d-cluster is a set of d nodes from a network. Graemlin creates one d-cluster for each node x—the d-cluster contains x as well as the d−1 nearest neighbors of x in the network. The k^th nearest neighbor of x is the node with the k^th shortest distance from x, the distance between two nodes is the sum of the edge lengths of the shortest path between them, and the length of an edge is the negative logarithm of its weight.

Graemlin uses a scoring function \( S_d(D_1, D_2) \) to compare two d-clusters \( D_1 \) and \( D_2 \). To compute \( S_d \), it first maps a subset of nodes in \( D_1 \) to a subset of nodes in \( D_2 \). The score of the mapping is the sum, over all pairs of nodes mapped to one another, of the pairwise node scores determined by Graemlin’s scoring function. \( S_d \) is equal to the score of the highest-scoring mapping.

High scoring d-clusters are pairs of d-clusters, one from each network, that score higher than a threshold T. Graemlin uses each high scoring pair of d-clusters as a seed. Figure 4.4B shows a sample set of d-clusters created from two networks, as well as a pair of high scoring d-clusters.

Graemlin’s use of d-clusters to generate seeds has three benefits. First, high scoring alignments will usually contain a high scoring d-cluster pair, since high scoring alignments usually have a high node score and contain nodes that are close together in the network. Second, Graemlin can compare d-clusters rapidly, since a comparison ignores edges. Finally, the parameters d and T offer a speed-sensitivity trade-off. For example, a lower value of T will generate more seeds that Graemlin must extend, which results in a longer running time but an increased likelihood that a high scoring alignment will contain a high scoring d-cluster pair.
4.3. SEARCH ALGORITHM

Implementation heuristics  Graemlin uses several heuristics to efficiently create $d$-clusters and generate seeds. These heuristics avoid $d$-cluster redundancies, efficiently find mappings between $d$-clusters, efficiently find high scoring $d$-cluster pairs, and efficiently align many queries to a common database of networks.

Graemlin’s first $d$-cluster heuristic avoids $d$-cluster redundancies. If Graemlin generates a $d$-cluster from each node and its closest neighbors, nodes incident to many edges will be present in many $d$-clusters. To avoid this, each time Graemlin selects a node as part of a $d$-cluster, it increases the distance of the node to all of its neighbors. Graemlin is therefore less likely to include the node in future $d$-clusters and creates more independent $d$-clusters, which, much like spaced seeds for genomic database searches [99, 100], can increase search sensitivity.

An extension of this idea enables Graemlin to create more than one $d$-cluster for each node. If it creates $d$-clusters from nodes in a random order, successive random orders create different sets of $d$-clusters. This ability to create multiple $d$-clusters per node is analogous to extensions of spaced seed techniques for genomic database searches, which allow for multiple seeds at each position in a sequence [101, 100].

Graemlin’s second $d$-cluster heuristic improves the efficiency of $d$-cluster mappings. In general, to see if $d$-clusters $D_1$ and $D_2$ are a high scoring pair of $d$-clusters, Graemlin must examine all possible mappings of nodes to see if any result in a score larger than $T$. Because there are an exponential number of possible mappings, Graemlin examines only greedy mappings. To obtain a greedy mapping, Graemlin successively adds to the mapping the pair of nodes with highest possible score.

Graemlin’s third $d$-cluster heuristic improves the efficiency of the search for high scoring $d$-cluster pairs. In general, Graemlin must examine all potential pairs of $d$-clusters to find all high scoring mappings, which makes large database searches expensive. To avoid this, Graemlin builds an index from the larger network as a pre-processing step. The index contains two tables: one table is keyed from the larger network as a pre-processing step. The index contains two tables: one table is keyed

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure4_4.png}
\caption[Græmlin uses a progressive strategy for multiple network alignment.]{Græmlin uses a progressive strategy for multiple network alignment. \textbf{A.} Graemlin receives as input a set of networks together with their phylogenetic relationship. \textbf{B.} Graemlin first performs a pairwise alignment of the two closest species. As described in the text, it generates a set of $d$-clusters from each network—one node and its $d-1$ closest neighbors constitute a $d$-cluster. To score all pairs of $d$-clusters, Graemlin finds for each pair the highest scoring mapping among nodes and selects the pairs that score greater than a user-specified threshold $T$; the figure highlights one particular high scoring pair together with the node mapping responsible for its score. Graemlin uses all high scoring pairs as seeds. \textbf{C.} Graemlin extends each seed using a greedy algorithm; each extension step is shown in a gray box. At each step, Graemlin adds to the frontier all nodes connected to a node currently in the alignment; the frontier is shown in the upper half of each box. Graemlin examines the frontier and adds to the alignment the pair of nodes or single node that most increases the alignment score. The extension phase stops when no nodes from the frontier can increase the alignment score. \textbf{D.} Graemlin transforms the alignment, together with the unaligned nodes from the two original networks, into three generalized networks for use in subsequent progressive alignment steps. \textbf{E.} In the next step of progressive alignment, Graemlin performs three pairwise alignments, one for each of the newly created generalized networks. Its running time will not scale by a factor of three, however, as the total number of nodes in all networks remains roughly the same.}
\end{figure}
by node and has values of all \( d \)-clusters to which the node belongs, and the other is keyed by \( d \)-cluster and has values of all nodes that belong to the \( d \)-cluster.

Græmlin uses the index to find all high scoring matches to a \( d \)-cluster in the smaller network. To help find matches, Græmlin uses a table that, for each node in the smaller network, records all sequence-similar nodes—Græmlin builds this table from BLAST results as a one-time expense. To high scoring matches to a \( d \)-cluster, Græmlin first uses the table to find for each node in the \( d \)-cluster all sequence-similar nodes in the larger network. It then uses the index to find all \( d \)-clusters that contain those sequence-similar nodes and examines only those \( d \)-clusters to find high scoring matches. Because most nodes are sequence-similar to a relatively small number of nodes, this heuristic dramatically reduces the number of \( d \)-cluster pairs that Græmlin must examine.

Græmlin’s fourth \( d \)-cluster heuristic improves the efficiency of query-to-network alignment. When it aligns multiple query networks to a single large network, Græmlin can build an index from the larger network and use it for each query network. Because it amortizes the index-building step over all query networks, Græmlin’s overall running time decreases.

**Theory** We can quantify the effects of the parameters \( d \) and \( T \) on the speed-sensitivity trade-off offered by \( d \)-clusters. We calculate \( p(d, T, S) \): the probability that no \( d \)-cluster pair with score greater than \( T \) occurs within an alignment of total score greater than \( S \). Because Græmlin must find a high scoring \( d \)-cluster pair within an alignment to find the alignment, lower values of \( p(d, T, S) \) lead to higher sensitivity.

To calculate \( p(d, T, S) \), we use a simplified model of a high scoring local alignment. We assume that each equivalence class is the same size and model the equivalence classes in an alignment as a set of node random variables. Each node random variable has value equal to the node score of the corresponding equivalence class. Similarly, we model each set of edges between two equivalence classes as an edge random variable. This model is analogous to un-gapped homologies, used in studies of spaced seeds [99]. Conditioned on the event that the alignment scores greater than \( S \), we assume that the node random variables are independent and identically distributed (i.i.d.), as are the edge random variables. We denote the score of each node random variable \( n_i \) as \( S(n_i) \), the set of \( N \) nodes as \( M \), the set of \( d \)-clusters as \( D \), the sum of the node scores as \( S_N \), and the sum of the edge scores as \( S_E \). Since the alignment is the sum of the node scores and edge scores, the event that the alignment scores greater than \( S \) implies that \( S_E + S_N > S \).

Under this model, the probability that a \( d \)-cluster match does not occur within an alignment is

\[
p(d, T, S) = \Pr \left( \exists c \in D \text{ s.t. } \sum_{n_i \in c} S(n_i) > T | S_N + S_E > S \right)
\]

We use two further simplifications of our model. First, we condition only on the node score \( S_N \), rather than the alignment score \( S \). To ensure that most alignments with total score greater than \( S \)
have node scores greater than $S_N$, we can use a conservative value for $S_N$. Second, we assume that an alignment of size $N$ has $\frac{N}{d}$ disjoint $d$-clusters within it. This assumes that $d$-clusters are disjoint and completely cover each module and usually overestimates the true value of $p(d, T, S)$.

These two simplifications specify $p(d, T, S)$ as the probability of the intersection of $\frac{N}{d}$ independent miss events, each with probability

$$
\Pr \left( \sum_{i=1}^{d} S(n_i) < T \mid \sum_{n_i \in M} S(n_i) > S_N \right).
$$

We denote $Y_i$ as a random variable that corresponds to the sum of $i$ node random variables. The probability of each miss event is then

$$
\Pr \left( \sum_{i=1}^{d} S(n_i) < T \mid \sum_{n_i \in M} S(n_i) > S_N \right) = \frac{\Pr(Y_d < T \mid Y_N > S_N)}{\Pr(Y_N > S_N)}
= \frac{\int_0^T \Pr(Y_d = x \wedge Y_N > S_N - x)dx}{\Pr(Y_N > S_N)}
= \frac{\int_0^T \Pr(Y_d = x) \Pr(Y_N - d > S_N - x)dx}{\Pr(Y_N > S_N)}.
$$

(4.1)

To estimate a value for equation (4.1), we use normal distributions for $Y_i$. Because non-orthologous proteins rarely align in a high scoring alignment, we estimate parameters for the normal distribution from known orthologous pairs of proteins.

We can then estimate $p(d, T, S)$ as

$$
p(d, T, S) \approx \left( \frac{\int_0^T \Pr(Y_d = x) \Pr(Y_N - d > S_N - x)dx}{\Pr(Y_N > S_N)} \right)^{\frac{N}{d}}.
$$

(4.2)

Given a user-specified $p_{\text{miss}}$ parameter, we use equation (4.2) to determine whether a given pair of values $d$ and $T$ has $p(d, T, S)$ no greater than $p_{\text{miss}}$.

We can also quantify how a pair of values $d$ and $T$ affects Graemlin’s running time. Each $d$-cluster pair that Graemlin compares takes $O(d^2)$ time, and each high scoring $d$-cluster pair initiates an extension. If we denote the relative cost of these two operations as $\lambda$, Graemlin’s expected running time due to specific values of $d$ and $T$ is proportional to

$$
N_1 N_2 \times \left( d^2 + \lambda \Pr(Z_d > T) \right),
$$

(4.3)
where \( N_1 \) and \( N_2 \) are the number of \( d \)-clusters in each network and \( Z_d \) is the sum of \( d \) random variables, each of value equal to the node score of two random proteins. We use \( Z_d \) rather than \( Y_d \) to assess arbitrary pairs of \( d \)-clusters rather than \( d \)-clusters part of a high scoring alignment.

Figure 4.5 shows a plot of equation (4.2) for various values of \( d \) and \( T \), and Figure 4.6 shows a plot of equation (4.3).

We can use equations (4.3) and (4.2) to estimate values of \( d \) and \( T \) that present the best speed/sensitivity trade-off. We optimize equation (4.3) subject to the constraint that \( p(d, T, S) < p_{\text{miss}} \), with \( p(d, T, S) \) given by equation (4.2). With a value for \( p_{\text{miss}} \) of .05, we obtained values of \( d = 4, T = 7 \), which Græmlin uses as default values.

**Seed extension**

**Algorithm**  Græmlin tries to transform each high scoring \( d \)-cluster pair into a high scoring local alignment. Like other network alignment algorithms [81, 78], Græmlin uses a greedy extension algorithm with successive iterations. Figure 4.4C illustrates Græmlin’s extension algorithm.

In the first iteration of the extension phase, Græmlin aligns two nodes, one from each \( d \)-cluster in the seed. It aligns the pair of nodes with the highest possible score and then places all nodes that neighbor either of these two nodes, as well as the nodes themselves, into the frontier. The frontier contains candidates Græmlin can add to the alignment at each iteration.
4.3. SEARCH ALGORITHM

Figure 4.6: We can compute the expected running time of Græmlin for values of $d$ and $T$. The objective function shows the expected computational cost of extending all high scoring $d$-clusters for various values of $d$ and $T$. $N = 20$ and $S_N = 30$.

At each successive iteration, Græmlin examines all pairs of nodes on the frontier. For each pair, it computes the change in score that will result if it aligns the pair and adds it to the alignment. If one or both nodes are present in the alignment, Græmlin must augment an equivalence class or merge two equivalence classes; if neither node is present in the alignment, Græmlin must create a new equivalence class. Græmlin also examines all individual nodes on the frontier and, for each node, computes the change in score that will result if it adds the node by itself to the alignment.

If any pair of nodes (or individual node) on the frontier will increase the score of the alignment, Græmlin updates the alignment. It then adds to the frontier any nodes that neighbor the newly aligned nodes.

Græmlin performs iterations until no changes can increase in the score of the alignment. As a final post-processing step, Græmlin examines each equivalence class in the alignment. If the removal of an equivalence class will increase the score of the alignment, Græmlin removes the equivalence class.

Figure 4.7 shows pseudocode for the basic extension algorithm.

**Implementation heuristics** Græmlin uses several heuristics to manage the frontier when it extends an alignment. These heuristics limit the possible pairings of nodes in an alignment, efficiently search the frontier for nodes to align, and limit the degree of overlap between different high scoring
ExtendSeed\((x_1, x_2)\) pair of nodes

\begin{verbatim}
var a ← ∅ //the alignment
var F ← ∅ //the frontier
[x'] ← x_1 ∪ x_2
a ← [x'] //the alignment begins with a single equivalence class that contains x_1 and x_2
while true
  do for nodes y in a
     Ensure y and all neighbors of y are in F
     \((\Delta_p, a'_p) ← \max_{y_1, y_2 ∈ F} \text{DELTA}\text{PAIR} (y_1, y_2, a)\)
     \((\Delta_s, a'_s) ← \max_{y ∈ F} \text{DELTA}\text{SINGLETON} (y, a)\)
  if \(\Delta_p > 0\) or \(\Delta_s > 0\)
     then if \(\Delta_p > \Delta_s\)
         then a ← a'_p
         else a ← a'_s
     else break
  return A
\end{verbatim}

DELTA\text{PAIR} \((y_1, y_2):\) pair of nodes, a: alignment

\begin{verbatim}
var \Delta //the change in score due to the alignment of y_1 and y_2 in a
var a' //the alignment that results from the alignment of y_1 and y_2
switch
  case y_1 ∈ a and y_2 ∈ a :
     \([y'] ← [y_1] \cup [y_2]\)
     \(a' ← a - [y_1] - [y_2] + [y']\)
  case y_1 ∈ a and y_2 ∉ a :
     \([y'] ← [y_1] \cup y_2\)
     \(a' ← a - [y_1] + [y']\)
  case y_1 ∉ a and y_2 ∈ a :
     \([y'] ← [y_2] \cup y_1\)
     \(a' ← a - [y_2] + [y']\)
  case y_1 ∉ a and y_2 ∉ a :
     \([y'] ← y_2 \cup y_1\)
     \(a' ← a + [y']\)
\Delta ← \text{SCORE}(a') - \text{SCORE}(a)
return (\Delta, a')
\end{verbatim}

DELTA\text{SINGLETON} \((y):\) node, a: alignment

\begin{verbatim}
var \Delta //the change in score due to the addition of y to a
var a' //the alignment that results from the addition of y to a
a' ← a + [y]
\Delta ← \text{SCORE}(a') - \text{SCORE}(a)
return (\Delta, a')
\end{verbatim}

Figure 4.7: \textbf{Græmlin uses a greedy algorithm to extend a seed.} The input to the algorithm is an aligned pair of nodes. The procedure uses two utility functions: one computes the change in score if Græmlin adds a pair of nodes to the alignment, and the other computes the change in score if Græmlin adds a single node to the alignment.
4.3. SEARCH ALGORITHM

alignments.

Graemlin’s first extension heuristic limits potential node pairings. When it considers pairs of
nodes to add to the alignment, Graemlin only considers pairs that have a positive node score. This
dramatically reduces the search space size but does not allow Graemlin to align proteins that have
no sequence-similarity.

Graemlin’s second extension heuristic efficiently finds the best pair of nodes to align from the
frontier. For each pair of nodes on the frontier, Graemlin caches the change in score due to the
alignment of the pair. Graemlin must update these values each time it updates the alignment, but
it does not update values for all pairs of nodes on the frontier. Instead, it stores pairs of nodes in
two different echelons, an upper echelon and a lower echelon, and updates only those pairs in the
upper echelon. The upper echelon contains pairs of nodes that change the alignment score by a value
greater than some threshold. When the change in score due to the alignment of a pair of nodes falls
below the threshold, Graemlin moves the pair of nodes into the lower echelon. It only updates nodes
in the lower echelon if no pairs in the upper echelon can increase the alignment score.

Graemlin’s third extension heuristic controls the overlap of high scoring alignments. Graemlin
accepts a parameter that controls the maximum allowable pairwise overlap of alignments and makes
alignments disjoint after all seeds have been extended. To make a set of alignments disjoint, Graemlin
examines all pairs of alignments that overlap by more than the maximum amount and successively
removes equivalence classes from one alignment; at each step it removes the equivalence class that
results in the minimum decrease in score of either alignment.

Because extensions of overlapping alignments waste time, Graemlin also controls for alignment
overlap during the extension phase. When it extends an alignment, it checks at each iteration if the
alignment overlaps any other alignment by more than the maximum amount. If it does, it removes all
nodes from the frontier present in the other alignment. For example, suppose high scoring alignment
A contains 10 nodes and the overlap threshold is 25%. When Graemlin extends alignment B, as
soon as B contains 3 nodes from A, Graemlin removes all nodes in alignment A from the frontier.

4.3.2 Multiple alignment

Graemlin’s progressive alignment algorithm performs sequential pairwise alignments. Graemlin first
performs a pairwise alignment of the two closest species A and B, according to a given phylogenetic
tree. At the parent node of A and B in the phylogenetic tree, Graemlin constructs three networks:

1. Network AB contains all high scoring alignments of A and B. In AB, there is one node for each
equivalence class in a high scoring alignment. In addition, AB contains all unaligned nodes in
A or B connected to at least f nodes in a high scoring alignment; f is a parameter that
defaults to 3. Graemlin includes unaligned nodes so that it can grow the pairwise alignments
as it adds more species to the multiple alignment. Within network AB, species A and B are
present.
2. Networks $A'$ and $B'$ contain all unaligned nodes within networks $A$ and $B$. These unaligned nodes include nodes appended to $AB$. Therefore, a small number of nodes appear in $AB$ as well as $A'$ or $B'$. Within network $A'$, only species $A$ is present, and within network $B'$, only species $B$ is present.

For example, in Figure 4.4D, Græmlin constructs three networks from the original two that it aligns. After the first pairwise alignment, Græmlin progress up the tree. At each tree node it aligns the networks at the child nodes. When it reaches a node in the tree that has internal nodes as children, Græmlin aligns all pairs of networks present at the children. For example, if one internal node contains networks $\{A', B', AB\}$ and another contains $\{C', D', CD\}$, Græmlin performs 9 pairwise alignments. While the number of networks at an internal node increases exponentially, the total size of those networks in practice does not increase because few proteins appear in more than one network.

At each pairwise alignment step, Græmlin’s scoring function considers only species that are present in the networks. For example, when it aligns networks $A'$ and $CD$, Græmlin does not assign a protein deletion penalty to equivalence classes without proteins from species $B$. Therefore, Græmlin can uncover conserved modules whose evolutionary history is not consistent with the phylogenetic tree. This is particularly important for microbes, for which prevalent horizontal gene transfer of entire operons can move modules among phylogenetically distant species [102, 103, 104, 105].

### 4.4 Benchmarks

We assessed the ability of Græmlin to align known functional modules. We compared Græmlin to two network alignment algorithms, NetworkBLAST [78] and MaWISh [81]. We performed tests on a set of 10 interaction networks from the Stanford Network Database [40].

#### 4.4.1 Test setup

**Metrics**

To assess the sensitivity of each algorithm, we counted the number of KEGG pathways it aligned between two species [48]. We identified a protein as *correctly aligned* if an algorithm aligned it to its KEGG ortholog (KO) in the other species. We defined a KEGG pathway as a *hit* if an algorithm aligned at least three proteins correctly; changes to the number of correctly aligned proteins required for a hit pathway did not affect the relative performance of the aligners. We defined the *coverage* of a pathway as the fraction of proteins within the pathway correctly aligned.

To assess the specificity of each algorithm, we counted the number of *enriched* alignments it produced. We first assigned to each protein all of its annotations from level eight or deeper in the GO hierarchy [50]. Then, to compute the enrichment of an alignment, we discarded unannotated
proteins and used the GO TermFinder [106]. We considered an alignment as enriched if the $p$-value of its enrichment was less than 0.01.

To further assess the specificity of each algorithm, we counted the fraction of proteins it aligned to proteins other than their KOs. We ignored proteins without a KO annotation.

We also assessed multiple alignment algorithms with these metrics. We computed metrics for multiple alignments just as we computed metrics for pairwise alignments, but we considered a protein as correctly aligned if an algorithm aligned it to its KOs in all species in the alignment.

**Implementation details**

We considered only *alignable* KEGG pathways when we computed our metrics. For a given set of networks, an alignable KEGG pathway must have a connected component of size at least three in each network.

We ran tests on two sets of interaction networks from the Stanford Network Database: one with an edge threshold of 0.25 and another with an edge threshold of 0.5. Table 4.1 lists statistics about these two sets of networks. For comparison, the table also shows the same statistics for the eukaryotic networks aligned in previous studies [86, 107, 108, 109]. Table 4.2 lists, for each set of networks we aligned, the number of alignable KEGG pathways.

We did not run tests on the eukaryotic networks because our sensitivity metric does not apply to them. As Table 4.1 shows, they contain few alignable KEGG pathways. In contrast, the networks in the Stanford Network Database contain enough alignable KEGG pathways to provide a meaningful comparison of different alignment algorithms. In addition, the networks in the Stanford Network Database are much larger than the eukaryotic networks and consequently better test the efficiency of an alignment algorithm.

For all tests and all alignment algorithms, we considered alignments with $p$-values less than 0.01 as high scoring. For each test case, we calculated $p$-values through runs on random datasets. We used techniques similar to those used in previous alignment studies [63, 81, 67, 77] to construct random datasets: we redistributed the edges of a real network but maintained the original node degree distribution.

Unless noted otherwise, we ran all aligners with their default parameters. We performed all tests on a 2.8 GHz Intel Xeon processor with 2 GB of RAM running the Linux operating system.

### 4.4.2 Network-to-network alignment

Our first set of tests assessed the ability of each algorithm to perform network-to-network alignment. This is the focus of both MaWISH, which searches for conserved heavy subgraphs, and NetworkBLAST, which searches for conserved protein complexes and pathways.

We tested the algorithms through alignments of *Escherichia coli* K12 and *Caulobacter crescentus*; *E. coli* and *Mycoplasma tuberculosis* H37Rv; *E. coli* and *Vibrio cholerae*; and *E. coli* and
<table>
<thead>
<tr>
<th>Species</th>
<th># Nodes</th>
<th>Edge Threshold</th>
<th># Edges</th>
<th># Edges per Node</th>
<th># Alignable KEGGs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Campylobacter jejuni NCTC 11168</td>
<td>1629</td>
<td>0.25</td>
<td>1629</td>
<td>1629</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>6171</td>
<td>3.79</td>
<td>29</td>
</tr>
<tr>
<td>Caulobacter crescentus</td>
<td>3737</td>
<td>0.25</td>
<td>40568</td>
<td>10.86</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>6018</td>
<td>1.61</td>
<td>55</td>
</tr>
<tr>
<td>Escherichia coli K12</td>
<td>4242</td>
<td>0.25</td>
<td>216426</td>
<td>51.02</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>35132</td>
<td>8.28</td>
<td>70</td>
</tr>
<tr>
<td>Helicobacter pylori 26695</td>
<td>1576</td>
<td>0.25</td>
<td>12960</td>
<td>8.22</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>3723</td>
<td>2.36</td>
<td>26</td>
</tr>
<tr>
<td>Mycoplasma tuberculosis H37Rv</td>
<td>3991</td>
<td>0.25</td>
<td>129183</td>
<td>32.37</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>17380</td>
<td>4.35</td>
<td>61</td>
</tr>
<tr>
<td>Salmonella typhimurium LT2</td>
<td>4527</td>
<td>0.25</td>
<td>94609</td>
<td>20.90</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>18149</td>
<td>4.01</td>
<td>55</td>
</tr>
<tr>
<td>Streptococcus pneumoniae TIGRA4</td>
<td>2094</td>
<td>0.25</td>
<td>25732</td>
<td>12.29</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>4607</td>
<td>2.20</td>
<td>23</td>
</tr>
<tr>
<td>Streptomyces coelicolor</td>
<td>8154</td>
<td>0.25</td>
<td>230467</td>
<td>28.26</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>60852</td>
<td>7.46</td>
<td>54</td>
</tr>
<tr>
<td>Synechocystis PCC 6803</td>
<td>3166</td>
<td>0.25</td>
<td>69439</td>
<td>21.93</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>13963</td>
<td>4.41</td>
<td>32</td>
</tr>
<tr>
<td>Vibrio cholerae</td>
<td>3835</td>
<td>0.25</td>
<td>36087</td>
<td>9.41</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5</td>
<td>7886</td>
<td>2.06</td>
<td>45</td>
</tr>
<tr>
<td>Saccharomyces cerevisiae</td>
<td>4766</td>
<td>N/A</td>
<td>15200</td>
<td>3.19</td>
<td>22</td>
</tr>
<tr>
<td>Caenorhabditis elegans</td>
<td>2629</td>
<td>N/A</td>
<td>3950</td>
<td>1.50</td>
<td>0</td>
</tr>
<tr>
<td>Drosophila melanogaster</td>
<td>7067</td>
<td>N/A</td>
<td>21822</td>
<td>3.09</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4.1: We quantified the performance of Græmlin on 10 networks in the Stanford Network Database. This table shows various statistics about the interaction networks we aligned, as well as statistics for three eukaryotic networks. For each species, we used two interaction networks: one with edge weights below 0.25 removed and one with edge weights below 0.5 removed. For each network, the columns present the total number of nodes, the total number of edges, the average number of edges per node, and the number of KEGG pathways containing a connected component of size at least three.
Table 4.2: The Stanford Network Database contains more alignable KEGG pathways than the eukaryotic networks. This table shows the number of alignable KEGG pathways in various subsets of species, as described in the text.

<table>
<thead>
<tr>
<th>Species Set</th>
<th>Threshold</th>
<th># Alignable KEGGs</th>
</tr>
</thead>
<tbody>
<tr>
<td>E. coli, C. crescentus</td>
<td>0.25</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>44</td>
</tr>
<tr>
<td>E. coli, M. tuberculosis</td>
<td>0.25</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>54</td>
</tr>
<tr>
<td>E. coli, V. cholerae</td>
<td>0.25</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>39</td>
</tr>
<tr>
<td>E. coli, S. coelicolor</td>
<td>0.25</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>43</td>
</tr>
<tr>
<td>E. coli, C. crescentus, V. cholerae</td>
<td>0.25</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>27</td>
</tr>
<tr>
<td>C. jejuni, E. coli, H. pylori</td>
<td>0.25</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>15</td>
</tr>
</tbody>
</table>

Streptomyces coelicolor. We report results for NetworkBLAST only on networks thresholded at 0.5 because NetworkBLAST did not terminate when we used it to align networks thresholded at 0.25. We also do not report results for MaWISh on S. coelicolor because S. coelicolor lacks the COG data [92] MaWISh requires. Figure 4.8 summarizes sensitivity for three test cases, and Tables 4.3 and 4.4 show more detailed results.

These results show that alignments of networks with a lower edge threshold find more conserved modules. When they align networks with a higher threshold, both MaWISh and Græmlin have lower specificity without higher specificity. Consequently, the inability of NetworkBLAST to efficiently align networks with a lower threshold limits its sensitivity.

When it searches for highly connected components, or protein complexes, Græmlin is significantly more sensitive than the other two algorithms. It hits more KEGGs, aligns more proteins, and finds alignments with a higher KEGG coverage. It achieves these sensitivity gains without a sacrifice of specificity.

When it searches for pathways, Græmlin is more sensitive than NetworkBLAST, although both algorithms are less sensitive than those that search for protein complexes. This is likely because pathways are highly connected in the networks in the Stanford Network Database, since the Stanford Network Database contains protein association networks rather than protein interaction networks. Therefore, searches for protein complexes will find pathways in the Stanford Network Database, but searches for pathways will ignore interactions between non-adjacent proteins in the pathway. Nonetheless, pathway searches may be useful in interaction networks with different properties.

Only MaWISh and Græmlin can efficiently align the large networks in the Stanford Network Database. While MaWISh is the faster of the two algorithms, the running time of Græmlin is comparable.
Figure 4.8: **Græmlin performs network-to-network alignment accurately.** For three pairwise alignments of *E. coli*, shown are the number of KEGGs hit by each aligner. For Græmlin and MaWISh, this graph includes results on networks thresholded at both 0.25 and 0.5. For NetworkBLAST, the graph includes only results on networks thresholded at 0.5 because NetworkBLAST did not terminate when we used it to align networks thresholded at 0.25.
### 4.4. BENCHMARKS

<table>
<thead>
<tr>
<th>KEGGs Hit</th>
<th>KEGG Coverage</th>
<th>Alignments Enriched</th>
<th>Nodes Aligned</th>
<th>Alignments</th>
<th>Nodes Misaligned</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MW</td>
<td>9 (20%)</td>
<td>32 %</td>
<td>72 %</td>
<td>346</td>
<td>67</td>
<td>35 %</td>
</tr>
<tr>
<td>NB</td>
<td>Pathway</td>
<td>6 (14%)</td>
<td>28 %</td>
<td>61 %</td>
<td>233</td>
<td>33 %</td>
</tr>
<tr>
<td>Complex</td>
<td>12 (27%)</td>
<td>49 %</td>
<td>72 %</td>
<td>466</td>
<td>23</td>
<td>52 %</td>
</tr>
<tr>
<td>Gr</td>
<td>Pathway</td>
<td>15 (34%)</td>
<td>47 %</td>
<td>68 %</td>
<td>594</td>
<td>56 %</td>
</tr>
<tr>
<td>Complex</td>
<td>17 (39%)</td>
<td>45 %</td>
<td>67 %</td>
<td>677</td>
<td>58</td>
<td>27 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E. coli vs. M. tuberculosis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MW</td>
<td>7 (13%)</td>
<td>20 %</td>
<td>85 %</td>
<td>202</td>
<td>33</td>
<td>28 %</td>
</tr>
<tr>
<td>NB</td>
<td>Pathway</td>
<td>7 (13%)</td>
<td>24 %</td>
<td>88 %</td>
<td>220</td>
<td>33 %</td>
</tr>
<tr>
<td>Complex</td>
<td>7 (13%)</td>
<td>32 %</td>
<td>88 %</td>
<td>262</td>
<td>15</td>
<td>40 %</td>
</tr>
<tr>
<td>Gr</td>
<td>Pathway</td>
<td>8 (15%)</td>
<td>36 %</td>
<td>89 %</td>
<td>454</td>
<td>58 %</td>
</tr>
<tr>
<td>Complex</td>
<td>8 (15%)</td>
<td>39 %</td>
<td>89 %</td>
<td>518</td>
<td>65</td>
<td>23 %</td>
</tr>
<tr>
<td>E. coli vs. V. cholerae</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MW</td>
<td>12 (31%)</td>
<td>35 %</td>
<td>64 %</td>
<td>819</td>
<td>131</td>
<td>15 %</td>
</tr>
<tr>
<td>NB</td>
<td>Pathway</td>
<td>10 (26%)</td>
<td>35 %</td>
<td>58 %</td>
<td>774</td>
<td>123</td>
</tr>
<tr>
<td>Complex</td>
<td>11 (28%)</td>
<td>41 %</td>
<td>64 %</td>
<td>1044</td>
<td>48</td>
<td>16 %</td>
</tr>
<tr>
<td>Gr</td>
<td>Pathway</td>
<td>19 (48%)</td>
<td>48 %</td>
<td>75 %</td>
<td>1414</td>
<td>115</td>
</tr>
<tr>
<td>Complex</td>
<td>15 (38%)</td>
<td>55 %</td>
<td>74 %</td>
<td>1414</td>
<td>100</td>
<td>11 %</td>
</tr>
<tr>
<td>E. coli vs. S. coelicolor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MW</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>NB</td>
<td>Pathway</td>
<td>6 (14%)</td>
<td>23 %</td>
<td>46 %</td>
<td>381</td>
<td>2041</td>
</tr>
<tr>
<td>Complex</td>
<td>10 (23%)</td>
<td>67 %</td>
<td>95 %</td>
<td>1637</td>
<td>2627</td>
<td>82 %</td>
</tr>
<tr>
<td>Gr</td>
<td>Pathway</td>
<td>8 (19%)</td>
<td>58 %</td>
<td>88 %</td>
<td>947</td>
<td>85</td>
</tr>
<tr>
<td>Complex</td>
<td>9 (21%)</td>
<td>59 %</td>
<td>85 %</td>
<td>1006</td>
<td>78</td>
<td>45 %</td>
</tr>
</tbody>
</table>

Table 4.3: Gremlin performs network-to-network alignment of networks thresholded at 0.5 with high specificity, sensitivity, and efficiency. For each pair of species, we used MaWISh (MW), NetworkBLAST (NB), and Gremlin (Gr) to perform complete network-to-network alignment. We thresholded all networks at 0.5. For each algorithm, shown, from left, is the total number of KEGG pathways hit, the fraction of KEGG pathways hit, the average coverage of a KEGG pathway, the percentage of enriched alignments, the total number of aligned proteins, the total number of significant alignments, the fraction of aligned proteins misaligned, and the total running time. We calculated the average coverage of KEGGs with respect to only those KEGGs that an aligner hit, and measured running time in CPU-seconds.
Table 4.4: Græmlin performs network-to-network alignment of networks thresholded at 0.25 with high specificity, sensitivity, and efficiency. For each pair of species, we used MaWISh (MW) and Græmlin (Gr) to perform complete network-to-network alignment. We thresholded all networks at 0.25. The columns in this table are analogous to those in Table 4.3.

Græmlin and NetworkBLAST take the longest to align E. coli and S. coelicolor, primarily due to the size of the S. coelicolor network and the large number of sequence-similar proteins between E. coli and S. coelicolor. To increase speed, Græmlin can sacrifice sensitivity through adjustments to its $d$-cluster parameters. Figure 4.9A demonstrates the impact of $T$ on running time and sensitivity. Running with its default parameters ($d = 4, T = 7$) on networks thresholded at 0.25, Græmlin finds 25 KEGG pathways in 1224 seconds, but with a slight increase in $T$ it finds 21 KEGGs in only 339 seconds.

### 4.4.3 Query-to-network alignment

Our second set of tests assessed the ability of each algorithm to perform query-to-network alignment. We tested the algorithms through queries of E. coli against C. crescentus, C. crescentus against E. coli, E. coli against M. tuberculosis, and M. tuberculosis against E. coli. Tables 4.5 and 4.6 show the results of these tests.

Græmlin is faster than MaWISh on all test cases because it amortizes its indexing step over all queries. The improvement is most dramatic when the database network is large. Therefore, Græmlin is an effective database search tool. For these tests, sensitivity and specificity results are similar to those for network-to-network alignment.

Græmlin performs comparably when it uses either a Pathway ESM or a Complex ESM. The Module ESM does not offer dramatic improvements over the other two ESMs, but it gives slightly higher...
### 4.4. BENCHMARKS

<table>
<thead>
<tr>
<th></th>
<th>KEGG Hit</th>
<th>KEGG Coverage</th>
<th>Nodes Misaligned</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E. coli vs. C. crescentus</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MaWISh</td>
<td>15 (34%)</td>
<td>31 %</td>
<td>29 %</td>
<td>37 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetworkBLAST</td>
<td>Pathway</td>
<td>8 (18%)</td>
<td>32 %</td>
<td>23 %</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>10 (23%)</td>
<td><strong>49 %</strong></td>
<td>36 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Græmlin</td>
<td>Pathway</td>
<td><strong>20 (45%)</strong></td>
<td>45 %</td>
<td><strong>16 %</strong></td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>20 (45%)</td>
<td>47 %</td>
<td>25 %</td>
</tr>
<tr>
<td></td>
<td>Module</td>
<td><strong>20 (45%)</strong></td>
<td>48 %</td>
<td>23 %</td>
</tr>
<tr>
<td><strong>C. crescentus vs. E. coli</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MaWISh</td>
<td>9 (20%)</td>
<td>32 %</td>
<td><strong>15 %</strong></td>
<td>130 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetworkBLAST</td>
<td>Pathway</td>
<td>10 (23%)</td>
<td>37 %</td>
<td>27 %</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>10 (23%)</td>
<td>41 %</td>
<td>26 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Græmlin</td>
<td>Pathway</td>
<td><strong>15 (34%)</strong></td>
<td>39 %</td>
<td>23 %</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td><strong>15 (34%)</strong></td>
<td><strong>42 %</strong></td>
<td>19 %</td>
</tr>
<tr>
<td></td>
<td>Module</td>
<td><strong>15 (34%)</strong></td>
<td><strong>42 %</strong></td>
<td>21 %</td>
</tr>
<tr>
<td><strong>E. coli vs. M. tuberculosis</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MaWISh</td>
<td>10 (19%)</td>
<td>19 %</td>
<td>26 %</td>
<td>93 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetworkBLAST</td>
<td>Pathway</td>
<td>12 (22%)</td>
<td>23 %</td>
<td><strong>15 %</strong></td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>12 (22%)</td>
<td>29 %</td>
<td>18 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Græmlin</td>
<td>Pathway</td>
<td><strong>17 (31%)</strong></td>
<td>31 %</td>
<td>27 %</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td><strong>17 (31%)</strong></td>
<td><strong>35 %</strong></td>
<td>25 %</td>
</tr>
<tr>
<td></td>
<td>Module</td>
<td><strong>17 (31%)</strong></td>
<td><strong>35 %</strong></td>
<td>24 %</td>
</tr>
<tr>
<td><strong>M. tuberculosis vs. E. coli</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MaWISh</td>
<td>6 (11%)</td>
<td>12 %</td>
<td><strong>11 %</strong></td>
<td>138 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetworkBLAST</td>
<td>Pathway</td>
<td>10 (19%)</td>
<td>19 %</td>
<td>29 %</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td>7 (13%)</td>
<td>22 %</td>
<td>29 %</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Græmlin</td>
<td>Pathway</td>
<td>13 (24%)</td>
<td>25 %</td>
<td>13 %</td>
</tr>
<tr>
<td></td>
<td>Complex</td>
<td><strong>14 (26%)</strong></td>
<td><strong>26 %</strong></td>
<td>13 %</td>
</tr>
<tr>
<td></td>
<td>Module</td>
<td><strong>14 (26%)</strong></td>
<td><strong>27 %</strong></td>
<td>15 %</td>
</tr>
</tbody>
</table>

Table 4.5: **Græmlin efficiently and accurately performs query-to-network alignment of networks thresholded at 0.5.** For each pair of species, we used MaWISh, NetworkBLAST, and Græmlin to successively align each KEGG pathway in the query species to the complete network of the database species. We thresholded all networks at 0.5. For each algorithm, shown, from left, is the total number of KEGG pathways with a database hit, the fraction of KEGG pathways with a database hit, the average coverage of a KEGG pathway, the average over queries of the fraction of aligned proteins misaligned, and the total running time.
Figure 4.9: Græmlin offers a speed-sensitivity trade-off and aligns multiple networks efficiently. A. Each point in the graph represents a run of Græmlin with \( d = 4 \). For each set of parameters, the x-axis plots the running time and the y-axis plots the fraction of KEGGs hit. B. We added species of increasing evolutionary distance to the multiple alignment. The pairwise running time is comparatively high because the two species aligned, *E. coli* and *S. typhimurium*, are the two most similar species and have many high scoring alignments. As it aligns more species, the average scaling of Græmlin is roughly linear.

<table>
<thead>
<tr>
<th></th>
<th>KEGG Hit</th>
<th>KEGG Coverage</th>
<th>Nodes Misaligned</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>E. coli</em> vs. <em>C. crescentus</em> MaWISH</td>
<td>20 (36%)</td>
<td>33 %</td>
<td>34 %</td>
<td>59 s</td>
</tr>
<tr>
<td>Græmlin Pathway</td>
<td>29 (53%)</td>
<td>63 %</td>
<td>25 %</td>
<td>50 s</td>
</tr>
<tr>
<td>Græmlin Complex</td>
<td>34 (62%)</td>
<td>63 %</td>
<td>34 %</td>
<td>64 s</td>
</tr>
<tr>
<td>Græmlin Module</td>
<td>34 (62%)</td>
<td>62 %</td>
<td>29 %</td>
<td>137 s</td>
</tr>
<tr>
<td><em>C. crescentus</em> vs. <em>E. coli</em> MaWISH</td>
<td>18 (33%)</td>
<td>40 %</td>
<td>20 %</td>
<td>554 s</td>
</tr>
<tr>
<td>Græmlin Pathway</td>
<td>30 (55%)</td>
<td>45 %</td>
<td>20 %</td>
<td>26 s</td>
</tr>
<tr>
<td>Græmlin Complex</td>
<td>36 (65%)</td>
<td>52 %</td>
<td>27 %</td>
<td>24 s</td>
</tr>
<tr>
<td>Græmlin Module</td>
<td>35 (64%)</td>
<td>56 %</td>
<td>26 %</td>
<td>74 s</td>
</tr>
<tr>
<td><em>E. coli</em> vs. <em>M. tuberculosis</em> MaWISH</td>
<td>22 (37%)</td>
<td>31 %</td>
<td>30 %</td>
<td>273 s</td>
</tr>
<tr>
<td>Græmlin Pathway</td>
<td>31 (52%)</td>
<td>48 %</td>
<td>29 %</td>
<td>17 s</td>
</tr>
<tr>
<td>Græmlin Complex</td>
<td>31 (52%)</td>
<td>47 %</td>
<td>29 %</td>
<td>17 s</td>
</tr>
<tr>
<td>Græmlin Module</td>
<td>32 (53%)</td>
<td>58 %</td>
<td>27 %</td>
<td>63 s</td>
</tr>
<tr>
<td><em>M. tuberculosis</em> vs. <em>E. coli</em> MaWISH</td>
<td>20 (35%)</td>
<td>24 %</td>
<td>28 %</td>
<td>628 s</td>
</tr>
<tr>
<td>Græmlin Pathway</td>
<td>25 (42%)</td>
<td>30 %</td>
<td>24 %</td>
<td>26 s</td>
</tr>
<tr>
<td>Græmlin Complex</td>
<td>29 (48%)</td>
<td>40 %</td>
<td>22 %</td>
<td>24 s</td>
</tr>
<tr>
<td>Græmlin Module</td>
<td>32 (53%)</td>
<td>45 %</td>
<td>18 %</td>
<td>76 s</td>
</tr>
</tbody>
</table>

Table 4.6: Græmlin efficiently and accurately performs query-to-network alignment of networks thresholded at 0.25. For each pair of species, we used MaWISH and Græmlin to successively align each KEGG pathway in the query species to the complete network of the database species. We thresholded all networks at 0.25. The columns in this table are analogous to those in Table 4.5.
4.4. BENCHMARKS

<table>
<thead>
<tr>
<th>KEGGs</th>
<th>KEGGs</th>
<th>Alignments</th>
<th>Nodes</th>
<th>Nodes</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hit</td>
<td>Coverage</td>
<td>Enriched</td>
<td>Aligned</td>
<td>Misaligned</td>
<td>Time</td>
</tr>
</tbody>
</table>

0.25 Threshold

**E. coli vs. C. crescentus vs. V. cholerae**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Pathway</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graemlin</td>
<td>27 (57%)</td>
<td>29 (62%)</td>
</tr>
<tr>
<td></td>
<td>68%</td>
<td>71%</td>
</tr>
<tr>
<td></td>
<td>72%</td>
<td>79%</td>
</tr>
<tr>
<td></td>
<td>3288</td>
<td>3403</td>
</tr>
<tr>
<td></td>
<td>27%</td>
<td>30%</td>
</tr>
<tr>
<td></td>
<td>329 s</td>
<td>251 s</td>
</tr>
</tbody>
</table>

**E. coli vs. C. jejuni vs. H. pylori**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Pathway</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graemlin</td>
<td>16 (57%)</td>
<td>17 (61%)</td>
</tr>
<tr>
<td></td>
<td>57%</td>
<td>63%</td>
</tr>
<tr>
<td></td>
<td>87%</td>
<td>89%</td>
</tr>
<tr>
<td></td>
<td>1172</td>
<td>1329</td>
</tr>
<tr>
<td></td>
<td>19%</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>44 s</td>
<td>43 s</td>
</tr>
</tbody>
</table>

0.5 Threshold

**E. coli vs. C. crescentus vs. V. cholerae**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Pathway</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graemlin</td>
<td>7 (26%)</td>
<td>9 (33%)</td>
</tr>
<tr>
<td></td>
<td>67%</td>
<td>62%</td>
</tr>
<tr>
<td></td>
<td>72%</td>
<td>75%</td>
</tr>
<tr>
<td></td>
<td>2318</td>
<td>2458</td>
</tr>
<tr>
<td></td>
<td>24%</td>
<td>24%</td>
</tr>
<tr>
<td></td>
<td>63 s</td>
<td>38 s</td>
</tr>
</tbody>
</table>

**E. coli vs. C. jejuni vs. H. pylori**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Pathway</th>
<th>Complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graemlin</td>
<td>5 (33%)</td>
<td>4 (27%)</td>
</tr>
<tr>
<td></td>
<td>41%</td>
<td>38%</td>
</tr>
<tr>
<td></td>
<td>94%</td>
<td>96%</td>
</tr>
<tr>
<td></td>
<td>642</td>
<td>524</td>
</tr>
<tr>
<td></td>
<td>40%</td>
<td>36%</td>
</tr>
<tr>
<td></td>
<td>&gt;10^6 s</td>
<td>32,394 s</td>
</tr>
</tbody>
</table>

Table 4.7: **Graemlin efficiently and accurately performs multiple network alignment.** We performed three-way multiple network alignments with NetworkBLAST and Graemlin. The columns in this table are analogous to those in Table 4.3.

KEGG coverage and misaligns slightly fewer proteins. The reason for the comparable performance of the Module ESM and the Complex ESM is likely that, because all proteins in a common KEGG pathway associate, most alignable KEGGs are highly connected in the Stanford Network Database. Therefore, a Module ESM constructed for a query network will usually resemble a Complex ESM.

4.4.4 Multiple network alignment

Our final set of tests assessed the ability of Graemlin and NetworkBLAST to perform multiple network alignment. We tested each algorithm through three-way alignments of (E. coli, C. crescentus, V. cholerae) and (E. coli, C. jejuni, Helicobacter pylori 26695). Table 4.7 shows the results of these tests.

On all test cases, Graemlin is orders of magnitude faster than NetworkBLAST. NetworkBLAST did not terminate when we used it to align the networks thresholded at 0.25 or when we used it to align one set of networks thresholded at 0.5. Because Graemlin scales effectively to large network sizes, it can perform multiple alignments of networks with a low edge threshold—Table 4.7 shows that this allows it to find many more conserved modules.

On the one test case for which it terminated, NetworkBLAST hits slightly more KEGGs than Graemlin. However, Graemlin covers a much higher fraction of each KEGG and also misaligns fewer
proteins.

To further stress Græmlin’s ability to align many species at once, Figure 4.9B shows the running times of Græmlin as it includes more species in an alignment. The roughly linear relation of running time to the number of species demonstrates the benefit of Græmlin’s progressive alignment technique.

4.5 Discussion

Græmlin is the first algorithm capable or large-scale multiple network alignment. Our tests show that Græmlin accurately and efficiently aligns large networks and is the first algorithm that can perform multiple alignments of an arbitrary number of networks. Græmlin’s performance is due to its incorporation of multiple heuristics popularized by sequence alignment algorithms.

First, Græmlin uses a log-odds scoring function like that used in the BLAST algorithm for sequence alignment. This scoring function handles evolutionary events not present in other network alignment scoring functions, such as protein deletions and duplications.

Second, Græmlin’s scoring function uses an Edge Scoring Matrix (ESM), which allows its log-odds scoring function to apply to alignments of pathways, protein complexes, and general modules. The use of an ESM allows Græmlin to search for conserved pathways and protein complexes with the same efficient algorithm.

Third, Græmlin uses $d$-clusters to perform seeded local network alignment, just as BLAST uses k-mers to perform seeded local sequence alignment. $d$-clusters offer a speed-sensitivity trade-off unlike other network alignment algorithms.

Fourth, Græmlin can perform efficient query-to-network alignment because it builds an index from a database of networks. This allows it to efficiently search a database for matches to multiple query networks.

Finally, Græmlin aligns multiple networks in linear time due to its progressive alignment technique. Its running time when aligning three networks is orders of magnitude faster than Network-BLAST, which has a running time exponential in the number of aligned networks.

Nonetheless, Græmlin has several weaknesses. Foremost is the heuristic nature of its scoring function. Only the probabilities for its sequence mutation event are based on real biological data, and Græmlin’s sum-of-pairs scoring function does not scale well to equivalence classes with many species.

Chapter 5 describes a new scoring function that addresses these weaknesses in Græmlin’s original scoring function.
Chapter 5

Improved Scoring Function for Græmlin

Although Græmlin is an efficient and accurate network aligner, its scoring function is heuristic. Users must hand-choose several scoring function parameters, and the default parameter choices have no biological basis. In addition, the number of terms in Græmlin’s sum-of-pairs scoring function grows quadratically, rather than linearly, in the number of species aligned.

This chapter describes an improvement to Græmlin’s scoring function, as well as an algorithm that uses a set of known network alignments to estimate the new scoring function’s parameters. Græmlin 1.1, a version of Græmlin that uses this new scoring function, finds more accurate alignments than the original version of Græmlin.

Because it is superseded by Græmlin 2.0, Græmlin 1.1’s main contributions are the ideas it introduces. Specifically, it is the first network aligner to use of a training set of known network alignments to weight scoring function parameters relative to one another.

5.1 Background and motivation

Most research on network alignment has focused on search algorithms rather than scoring functions. Nonetheless, to produce accurate alignments, a search algorithm must use a principled scoring function.

The history of sequence alignment demonstrates the importance of principled scoring functions. The PAM [110] and BLOSUM [93] scoring matrices are widely used because their parameters are based on real amino acid substitution rates. Recent alignment algorithms have improved accuracy mainly through the incorporation of additional features, such as three-dimensional protein structure [111] and physical properties of amino acids [112], into their scoring functions. As scoring functions
include more features, the assignment of correct weights to different features becomes more critical yet more difficult to do heuristically.

Network alignment scoring functions must incorporate multiple features as well. At the least, every network aligner must weight node and edge scores relative to one another. However, most network alignment tools use heuristic scoring functions [30]. The PathBLAST [63] and NetworkBLAST [78] algorithms use log-odds scoring for nodes and edges and also model edge topology with a probabilistic model, but they do not model evolution and have no mechanism to appropriately weight node and edge scores. The MaWISh algorithm [81] incorporates a duplication/divergence evolutionary model for pairwise network alignment, but its parameters that control the relative weights of penalties for duplications and protein sequence mismatches must be set manually. As discussed in Chapter 4, Graemlin's scoring function uses heuristic choices for many of its parameters and does not model evolution of edges within a network—it scores each edge between two equivalence classes independently with an ESM.

One recent study [82] did address the importance of appropriate weights for node and edge scores. The study developed stochastic models for the dynamics of protein and interaction evolution but focused on the detection of local functional correlations between a pair of networks rather than general multiple alignment of functional modules. In addition, the study does not address the incorporation of additional features into a scoring function.

Therefore, no current multiple network alignment scoring function uses a principled algorithm to weight multiple evolutionary events relative to one another. Graemlin 1.1 uses the first scoring function that fills this vacancy.

Graemlin 1.1’s scoring function rewards alignments that represent evolutionarily conserved modules. It parameterizes a multiple alignment by a set of evolutionary events that lead from the ancestral functional module to a set of species-specific modules. It rewards events more likely to characterize functional modules than random groups of proteins. Just as substitution matrices such as BLOSUM can be trained on databases of conserved sequences such as BLOCKS, Graemlin 1.1’s scoring function can be trained on arbitrary collections of conserved functional modules.

Previous studies of functional modules and gene evolution support the principles behind Graemlin 1.1’s scoring function. For example, studies have suggested that, relative to random groups of proteins, functional modules have a slower evolutionary rate [113, 114], distinct network connectivity patterns [115, 116] and are more evolutionary “cohesive” [117]. Furthermore, past studies have used training sets to estimate rates for events such as gene fusion, gene loss, gene duplication, and horizontal gene transfer [102, 103, 104, 105]. These results suggest that a network alignment scoring function can distinguish functional modules from random groups of proteins and learn parameters from a training set of known network alignments.

We performed numerical benchmarks that measured the accuracy of Graemlin 1.1’s alignments. The results of these benchmarks suggest that Graemlin 1.1 does indeed find more accurate alignments
5.2 Overview of scoring function

Græmlin 1.1 uses the same interpretation of network alignments as Græmlin. Namely, high scoring alignments should align subnetworks that are extant instances of an ancestral functional module, while low scoring alignments should align subnetworks that either have no evolutionary relationship or are not true functional modules.

Græmlin 1.1’s scoring function redefines the two probabilistic models, $M$ and $R$, used by Græmlin. The alignment model $M$ posits that each equivalence class groups proteins descended from a common ancestral protein, and the ancestral proteins together form an ancestral functional module. The random model $R$ posits that there is no relationship among the nodes in the alignment. The score of an alignment $a$ is

$$\log \frac{Pr_M(a)}{Pr_R(a)} = \log \frac{Pr_M(\text{nodes}) Pr_M(\text{edges|nodes})}{Pr_R(\text{nodes}) Pr_R(\text{edges|nodes})}$$

Like Græmlin’s scoring function, Græmlin 1.1’s scoring function has two components. Its node scoring function assigns probabilities to a set of nodes in an arbitrary equivalence class. Its edge scoring function assigns probabilities to a set of edges between two arbitrary equivalence classes.

To score an alignment, Græmlin 1.1 applies the node scoring function to each equivalence class to obtain a set of node scores and the edge scoring function to each set of edges to obtain a set of edge scores; the score of the alignment is the sum of the node and edge scores.

5.2.1 Evolutionary events and probabilities

Græmlin 1.1 bases the score of an alignment on the evolutionary history of the module the alignment represents. It requires a phylogenetic tree, with quantitative branch lengths, that relates the aligned species. It uses this tree to determine the most likely module at each ancestral species and then the likely set of evolutionary events that occurred over the history of the module.

Græmlin 1.1 determines the set of evolutionary events and their probabilities in four steps. First, it computes weight vectors at each node in the phylogenetic tree (phylogenetic tree nodes have no relationship to network nodes). Next, it uses these weight vectors to determine numerical properties at each tree node as well as tree branches. Then, it uses the numerical properties to determine evolutionary events that occur along the branches of the phylogenetic tree. Finally, it assigns probabilities to each event with equation (5.1).

Figure 5.1 shows how Græmlin 1.1 computes evolutionary events for an equivalence class or a set of edges. It also shows the set of evolutionary events that Græmlin 1.1 considers.
CHAPTER 5. IMPROVED SCORING FUNCTION FOR GRÆMLIN

**Protein Present**
Protein is present at root with probability 0.74

**Protein Deletion**
Parent has protein present with probability 0.74
Child has protein absent

**Protein Duplication**
Parent has 1.48 expected proteins
Child has 2 proteins

**Protein Mutation**
Species Pair Weights

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>BitScore</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>101.68</td>
</tr>
<tr>
<td>A1</td>
<td>B2</td>
<td>102.06</td>
</tr>
<tr>
<td>A1</td>
<td>C1</td>
<td>107.07</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>100.91</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>90.12</td>
</tr>
<tr>
<td>B1</td>
<td>B2</td>
<td>100.14</td>
</tr>
</tbody>
</table>

**Paralog Mutation**
BLAST bitscore is 140.227

**Paralog Edge Deletion**
Edge present in only one of two paralogs

**Edge Deletion**
Parent has edge with probability 0.68
Child has no edge

**Edge Structure**
Root has edge with probability 0.68
Conditional edge weight is 0.38

**Number of Proteins**
Root has 1.48 expected proteins

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.44</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.24</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.24</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.08</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Paralog Edge Deletion**
Edge present in only one of two paralogs

**Edge Deletion**
Parent has edge with probability 0.68
Child has no edge

**Edge Structure**
Root has edge with probability 0.68
Conditional edge weight is 0.38

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.19</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.43</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.06</td>
</tr>
</tbody>
</table>

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.19</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.43</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.06</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.19</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.43</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.06</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.19</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.43</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.06</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.19</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.43</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.06</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.19</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.43</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.06</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.19</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.43</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.23</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.06</td>
</tr>
</tbody>
</table>

**Expected BLAST bitscore**
Expected BLAST bitscore is 140.227

**Weights**

<table>
<thead>
<tr>
<th>Prot1</th>
<th>Prot2</th>
<th>Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>0.19</td>
</tr>
<tr>
<td>B1</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>B2</td>
<td>C1</td>
<td>0.54</td>
</tr>
<tr>
<td>C1</td>
<td>C1</td>
<td>0.05</td>
</tr>
</tbody>
</table>
Construction of species weight vectors

Græmlin 1.1 uses a previously developed algorithm [118] to assign to each phylogenetic tree node a *species weight vector*. A species weight vector is a normalized vector of weights, with one weight for each extant species. Each weight summarizes the distance between an extant species and the ancestral species represented by the tree node, and the weights sum to 1.

Græmlin 1.1 uses the species weight vectors to compute *pairwise species weight vectors* at each tree branch, with one weight for each pair of extant species. To obtain a pairwise species weight vector for an edge between two tree nodes, each with a species weight vector, Græmlin 1.1 first multiplies each of the weights in the first species weight vector by each of the weights in the second species vector. This results in two weights for every pair, which Græmlin 1.1 adds to obtain the final weight for the pair.

Construction of properties at tree nodes and branches

As Figure 5.1 shows, the weight vectors allow Græmlin 1.1 to estimate various properties at each tree node and each tree branch. To compute its node scoring function for an equivalence class, Græmlin 1.1 computes two properties at each tree node and two properties at each tree branch. To compute its edge scoring function for a set of edges, Græmlin 1.1 computes two properties at each tree node.

The first tree node property for the node scoring function is the *protein present property*. This probability is given by the dot product of the species weight vector with a *protein present vector*. If \((s_1, \ldots, s_m)\) is the species weight vector at a tree node \(n\), the protein present vector \((p_1, \ldots, p_m)\) has \(p_i = 1\) if species \(i\) has a protein in the equivalence class and \(p_i = 0\) otherwise. The protein present property at \(n\) is \(\sum_{i=1}^{n} s_i p_i\). Biologically, the protein present property at tree node \(n\) measures the probability that there is an equivalence class ortholog in the species represented by \(n\).

The second tree node property for the node scoring function is the *number of proteins property*. To compute this property Græmlin 1.1 uses a phylogenetic tree that contains only species with at least one protein in the equivalence class. The number of proteins property is given by the dot product of the species weight vector with a *number of proteins vector*. If \((s_1, \ldots, s_m)\) is the species weight vector at a tree node \(n\), the number of proteins vector \((c_1, \ldots, c_m)\) has \(c_i\) equal to the number of proteins species \(i\) has in the equivalence class. Biologically, the expected number of
proteins property at tree node \( n \) measures the expected number of equivalence class orthologs in the species represented by \( n \).

The first tree branch property for the node scoring function is the similarity present property. To compute this property Graemlin 1.1 uses a phylogenetic tree that contains only species with at least one protein in the equivalence class. The similarity present property is given by the dot product of the pairwise species weight vector with a similarity present vector. If \((s_{1,1}, \ldots, s_{1,m}, s_{2,1}, \ldots, s_{m,m})\) is the pairwise species weight vector at a branch \( \beta = (n_1, n_2) \), the similarity present vector \((r_{1,1}, \ldots, r_{m,m})\) has \( r_{i,j} = 1 \) if there is a BLAST hit \((90)\) of e-value less than \(10^{-5}\) between the proteins that \( i \) and \( j \) have in the equivalence class. If \( i \) or \( j \) have multiple proteins in the equivalence class, \( r_{i,j} \) is the fraction of possible pairs of proteins that have a BLAST hit. Biologically, the similarity present property at branch \( \beta = (n_1, n_2) \) is the probability that there is sequence-similarity between the equivalence class ortholog in the species represented by \( n_1 \) and the equivalence class ortholog in the species represented by \( n_2 \).

The second tree branch property for the node scoring function is the similarity value property. To compute this property Graemlin 1.1 uses a phylogenetic tree that contains only species with at least one protein in the equivalence class. The similarity value property is given by the dot product of the pairwise species weight vector with a similarity value vector. If \((s_{1,1}, \ldots, s_{1,m}, s_{2,1}, \ldots, s_{m,m})\) is the pairwise species weight vector at a branch \( \beta = (n_1, n_2) \), the similarity value vector \((b_{1,1}, \ldots, b_{m,m})\) has \( b_{i,j} \) equal to the value of the BLAST hit between the proteins that species \( i \) and \( j \) have in the equivalence class. If the proteins have no BLAST hit of e-value less than \(10^{-5}\), \( b_{i,j} = 0 \), and if \( i \) and \( j \) have multiple proteins in the equivalence class \( b_{i,j} \) is the average BLAST hit value between all possible pairs of proteins. The similarity value property is

\[
\frac{\sum_{i,j} s_{i,j} b_{i,j}}{\sum_{i,j} s_{i,j} r_{i,j}}
\]

where \( r_{i,j} \) are the values of the similarity present vector. Biologically, the similarity value property at branch \( \beta = (n_1, n_2) \) measures the degree of sequence-similarity between the equivalence class ortholog in the species represented by \( n_1 \) and the equivalence class ortholog in the species represented by \( n_2 \).

The first tree node property for the edge scoring function is the edge present property. To compute this property Graemlin 1.1 uses a phylogenetic tree that contains only species with a protein present in both equivalence classes adjacent to the set of edges. The edge present property is given by the dot product of the species weight vector with an edge present vector. If \((s_1, \ldots, s_m)\) is the species weight vector at a tree node \( n \), the edge present vector \((e_1, \ldots, e_m)\) has \( e_i = 1 \) if species \( i \) has an edge in the set of edges and \( e_i = 0 \) otherwise. If species \( i \) has multiple proteins in either equivalence class adjacent to the set of edges, \( e_i \) is the fraction of possible protein pairs that have an edge. Biologically, the edge present property at tree node \( n \) is the probability that an interaction exists
between the two equivalence class orthologs in the species represented by $n$.

The second tree node property for the edge scoring function is the edge weight property. To compute this property Græmlin 1.1 uses a phylogenetic tree that contains only species with a protein present in both equivalence classes adjacent to the set of edges. The edge weight property is given by the dot product of the species weight vector with an edge weight vector. If $(s_1, \ldots, s_m)$ is the species weight vector at a tree node $n$, the edge weight vector $(w_1, \ldots, w_m)$ has $w_i$ equal to the weight of the edge that species $i$ has in the set of edges; $w_i = 0$ if species $i$ has no edge in the set of edges. If species $i$ has multiple proteins in either equivalence class adjacent to the set of edges, $w_i$ is the average weight of the edges between all possible protein pairs. The edge weight property is

$$\frac{\sum_i s_i w_i}{\sum_i s_i e_i}$$

where $e_i$ are the values of the edge present property. Biologically, the edge weight property at tree node $n$ is the strength of interaction between the two equivalence class orthologs in the species represented by $n$.

**Determination of events**

Given a collection of properties at each tree node and each tree branch, Græmlin 1.1 obtains a set of evolutionary events that occur over the history of a module. In general, for an event to be a useful component of a scoring scheme, it must be over- or underrepresented in conserved modules relative to random groups of proteins.

The node scoring function considers protein deletions, duplications, and sequence mutations, as well as the most likely number of proteins that existed at the last common ancestor of the aligned species. The edge scoring function considers edge deletions as well as the most likely edge weight that existed at the last common ancestor. While it considers a restricted set of basic evolutionary events, Græmlin 1.1’s scoring function can extend to include any event that is localized to an equivalence class or a set of edges.

**Node scoring function events**

Græmlin 1.1’s node scoring function computes five evolutionary events, each of which has an associated value, for an equivalence class. Figure 5.1 shows the events as gray boxes.

1. **Protein present** ($pp$) has value 1 when the protein present property is greater than 0.5 at the phylogenetic tree root node. It has value 0 otherwise.

2. **Protein deletion** ($pd$) has value 1 for each branch in the phylogenetic tree where exactly one tree node incident to the branch has a protein present property greater than 0.5. It has value 0 for all other branches.
3. *Number of proteins* \((np)\) has value equal to the number of proteins property at the phylogenetic tree root node.

4. *Protein duplication* \((pu)\) has value 1 for each branch in the phylogenetic tree where the two tree nodes incident to the branch have positive but different values for the number of proteins property. It has value 0 for all other branches.

5. *Protein mutation* has two values \((sp, sv)\) for each branch in the phylogenetic tree—the similarity present property \((sp)\) of the branch and the similarity value property \((sv)\) of the branch.

6. *Paralog protein mutation* has two values \((psp, psv)\) for each species \(s\) with multiple proteins \(\Pi_s = p_s^1, \ldots, p_s^n\) in the equivalence class—the fraction of possible pairs of proteins in \(\Pi_s\) that have a BLAST hit of \(e\)-value less than \(10^{-5}\) \((psp)\) and the average BLAST hit value between all possible pairs of proteins in \(\Pi_s\) \((psv)\).

**Edge scoring function events.** Græmlin 1.1’s edge scoring function computes three evolutionary events, each of which has an associated value.

1. *Edge deletion* \((ed)\) has value 1 for each branch in the phylogenetic tree where exactly one tree node incident to the branch has an edge present property greater than 0.5. It has value 0 for each branch where both tree nodes incident to the branch have an edge present property greater than 0.5.

2. *Paralog edge deletion* \((ped)\) has 1 for each leaf with an edge present property greater than 0 but less than 1. It has value 0 for all other leaves.

3. *Edge structure* has two values \((ep, ew)\)—the edge present property \((ep)\) at the phylogenetic tree root node and the edge weight property \((ew)\) at the phylogenetic tree root node.

**Computation of probabilities**

Both models \(\mathcal{M}\) and \(\mathcal{R}\) assign probabilities to each evolutionary event. As in equation (5.1), the score for an equivalence class (or set of edges) is the sum, over all events, of the log-ratios of the probabilities assigned by each model.

The models use parametric probability distributions to assign probabilities to each event. For each event, both \(\mathcal{M}\) and \(\mathcal{R}\) use the same probability distribution but with different parameters. The probability distribution depends on the event.

The distribution for the protein present event is a Bernoulli distribution. Specifically,

\[
\Pr_\mathcal{G}(pp = 1) = 1 - \Pr_\mathcal{G}(pp = 0) = p_{\mathcal{G}}^{pp}
\]

where \(\mathcal{G}\) is either \(\mathcal{M}\) or \(\mathcal{R}\). Each model requires one parameter.
5.2. OVERVIEW OF SCORING FUNCTION

The distribution for the protein deletion event is a Bernoulli distribution with a parameter that depends on the length of the phylogenetic tree branch over which the event occurs. Specifically,

\[
\Pr_G(pd = 1|t) = 1 - \Pr_G(pd = 0|t) = a_G^{pd} \exp(-b_G^{pd}t) + c_G^{pd}
\]

where \( t \) is the branch length and the probabilities are restricted to lie between 0 and 1. Each model requires three parameters.

The distribution for the number of proteins event is a general discrete distribution. Specifically,

\[
\Pr_G(np = x < 10) = np_G^x \\
\Pr_G(np >= 10) = np_G^{10}
\]

Each model requires ten parameters.

The distribution for the protein duplication event is a Bernoulli distribution with a parameter that depends on the length of the phylogenetic tree branch over which the event occurs. Specifically,

\[
\Pr_G(pu = 1|t) = 1 - \Pr_G(pu = 0|t) = a_G^{pu} + b_G^{pu}t
\]

where \( t \) is the branch length and the probabilities are restricted to lie between 0 and 1. Each model requires two parameters.

The distribution for the protein mutation event is a mixture of a Gaussian distribution and a Bernoulli distribution. Parameters for each distribution depend on the length of the phylogenetic tree branch over which the event occurs. Specifically,

\[
\Pr_G(sp, sv|t) = (1 - sp) \times (1 - p_G^{sp}(t)) + sp \times p_G^{sp}(t) \times \mathcal{N}(\mu_G(t), \sigma_G(t))
\]

where \( t \) is the branch length and

\[
p_G^{sp}(t) = \frac{a_G^{sp}}{b_G^{sp} \exp(-c_G^{sp}t) + d_G^{sp}}
\]

\[
\mu_G(t) = f_G^{sv} \exp(-g_G^{sv}t) + h_G^{sv}
\]

\[
\sigma_G(t) = k_G^{sv} \exp(-l_G^{sv}t) + m_G^{sv}
\]

Each model requires ten parameters.

The distribution for the paralog protein mutation event is the same distribution as for the protein mutation event but without any time dependence of the parameters. Each model requires three parameters: \( p_G^{psp}, \mu_G^{psv}, \) and \( \sigma_G^{psv} \).

The distribution for the edge deletion event is a Bernoulli distribution with a parameter that
depends on the length of the phylogenetic tree branch over which the event occurs. Specifically,

\[
\Pr_G(ed = 1|t) = 1 - \Pr_G(ed = 0|t) = a^ed_G + b^ed_G t
\]

where \( t \) is the branch length and the probabilities are restricted to lie between 0 and 1. Each model requires two parameters.

The distribution for the paralog edge deletion event is a Bernoulli distribution. Specifically,

\[
\Pr_G(ped = 1) = 1 - \Pr_G(ped = 0) = p^{ped}_G
\]

Each model requires one parameter.

The distribution for the edge structure event is a general continuous distribution. Specifically,

\[
\Pr_G(ep, ew) = (1 - ep) \times (1 - p^{ep}_G) + ep \times p^{ep}_G \times \Pr_G(ew)
\]

where

\[
\Pr_G(ew) = a^{ew} \exp(-b^{ew}ew) + e^{ew}
\]

The alignment model has one value for \( p^{ep}_{\mathcal{M}} \). However, the random model uses a function \( p^{ep}_R(d) \) that depends on \( d \), the product of the relative degrees of the equivalence classes incident to the set of edges: the relative degree of an equivalence class is the number of edges to which it is incident divided by the total number of edges in the network. The function \( p^{ep}_R(d) \) is stepwise constant over ten relative degree ranges. Therefore, while the alignment model requires four parameters, the random model requires thirteen parameters.

Because there is only one edge structure distribution for each probabilistic model, both models use a Complex ESM. To apply to arbitrary ESMs, the distributions must depend on the labels given to the equivalence classes incident to the set of edges.

### 5.3 Model Training

The probabilities given to the events by \( \mathcal{M} \) and \( \mathcal{R} \) depend on multiple parameters. Græmlin 1.1 uses training sets of network alignments, one training set for each model, to learn these parameters. Figure 5.2 shows an overview of the training procedure.

Each network alignment training set contains a set of sample network alignments. The alignment model training set contains alignments that represent conserved functional modules, while the random model training set contains alignments to avoid.

Because Græmlin 1.1 scores each equivalence class and set of edges independently, a training set need not contain entire network alignments. Instead, Græmlin 1.1 can estimate the parameters for its node scoring function given a training set of equivalence classes and the parameters for its edge
scoring function given a training set of equivalence class pairs and the edges between them.

5.3. Determination of events

Gramlín 1.1 uses the same procedure to train both the alignment and random models. It first selects a set of 20 species, which it chooses in a greedy manner to obtain a wide spread of pairwise evolutionary distances. To train parameters for an event, Gramlín 1.1 uses one of four different training procedures.

Procedure 1: To train the protein deletion, protein duplication, and protein mutation events, Gramlín 1.1 samples 1000 equivalence classes from the training set. For each sample, it examines each pair of species in succession and catalogues the evolutionary distance of the species pair as well as the events that occur between each pair. The result is a set of events for all pairwise evolutionary distances.

Procedure 2: To train the edge deletion event, Gramlín 1.1 uses a modification of Procedure 1. Rather than sample single equivalence classes, it samples pairs of equivalence classes.

Procedure 3: To train the protein present, number of proteins, and paralog protein mutation events, Gramlín 1.1 samples 1000 equivalence classes from the training set. For each sample, it computes the value of the events for a random species in the equivalence class.

Procedure 4: To train the edge structure and paralog edge deletion events, Gramlín 1.1 uses a modification of Procedure 3. Rather than sample single equivalence classes, it samples pairs of equivalence classes.

5.3.2 Parameter estimation

To determine probabilities for events trained by Procedures 3 and 4, Gramlín 1.1 divides the number of times the event occurs by the number of samples. For events trained by Procedures 1 and 2, it determines separate probabilities for each pairwise evolutionary distance via the same approach.

Figure 5.2 (facing page): Gramlín 1.1 learns parameters for its scoring function from a training set of network alignments. Gramlín 1.1 trains both the alignment and the random model with the same procedure—the only difference is the content of the training set. A. The random model training set contains modules with sequence-similar but possibly functionally distinct proteins. B. The alignment model training set contains KEGG pathways. C. The training procedure for both models begins with the creation of a relevant training set. D. For each event, such as the protein deletion event as shown in the figure, Gramlín 1.1 determines the training examples in which the event occurs and the evolutionary distance over which the event occurs. E. Gramlín 1.1 tabulates the number of events for each evolutionary distance in the phylogenetic tree. F. Gramlín 1.1 converts the numbers into probabilities and then interpolates them to obtain event probabilities for all evolutionary distances.
CHAPTER 5. IMPROVED SCORING FUNCTION FOR GRÆMLIN

A. Proteins with sequence similarity

B. KEGG pathways

C. Generate training set of modules

D. Determine events occurring in history of each module

E. Tabulate event frequency for each branch length

F. Interpolate values

<table>
<thead>
<tr>
<th>Branch Length</th>
<th>Deletions</th>
</tr>
</thead>
<tbody>
<tr>
<td>.2</td>
<td>100</td>
</tr>
<tr>
<td>.22</td>
<td>120</td>
</tr>
<tr>
<td>.467</td>
<td>350</td>
</tr>
<tr>
<td>.7</td>
<td>500</td>
</tr>
<tr>
<td>.89</td>
<td>600</td>
</tr>
</tbody>
</table>
5.3. MODEL TRAINING

For discrete distributions, these probabilities correspond directly to parameters in the distributions. For continuous distributions, Græmlin 1.1 fits the functional form of the distribution to the available probabilities and obtains parameters from the fit. It uses the nonlinear least-squares (NLLS) Marquardt-Levenberg algorithm to fit all functions [119].

5.3.3 Training sets

Both training sets contain equivalence classes that Græmlin 1.1 uses to train its node scoring function and equivalence class pairs that Græmlin 1.1 uses to train its edge scoring function. The alignment model training set uses samples from the KEGG database and the random model training set uses randomly generated samples.

Alignment model

Each equivalence class in the alignment model training set corresponds to a KEGG Ortholog (KO) group in the KEGG database [48]. It consists of all proteins with the corresponding KO annotation. Each pair of equivalence classes in the alignment model training set corresponds to a pair of KO groups within the same KEGG pathway. Because edge deletion events are only defined when at least one species contains an edge between a pair of equivalence classes, Græmlin 1.1 rejects all pairs of KO groups that have no edges between them when it trains the edge deletion event.

Random model

Justification For three reasons, and unlike other network aligners [77, 78], Græmlin 1.1 does not use random collections of proteins for the equivalence classes in its random model training set. First, while it is easy to generate random pairs of proteins, it is difficult to generate random equivalence classes that can contain any number of proteins from any number of species. Second, a random model trained on such a training set will assign only a small probability to sequence-similar pairs of proteins—a scoring function that uses this random model will score any alignment of sequence-similar proteins highly, even though most sequence-similar proteins are not functionally orthologous. Finally, while it is difficult to generate random equivalence classes, it is even more difficult to generate random network alignments that contain multiple equivalence classes.

Despite these difficulties, a random model is necessary for local network alignment. As in local sequence alignment, it ensures that the expected score of a random alignment will be negative, which allows the local alignment problem to be well defined. Intuitively, the random model implicitly specifies the “boundary” between good alignments and bad alignments—alignments more likely under the random model are bad while alignments more likely under the alignment model are good. Accordingly, the problem with the completely random model is that it generates alignments too far from the “boundary”—it does not generate alignments of sequence-similar but non-orthologous
Therefore, Græmlin 1.1’s random model training set contains equivalence classes with sequence-similar but functionally distinct proteins. Each network alignment in the random model training set contains groups of equivalence classes connected to one another but with possibly distinct functions. This random model is more informed than a completely random model, as it uses protein sequence similarity and network edges to guide its network alignment construction. Græmlin 1.1’s scoring function therefore rewards alignments that reflect true functional orthology but penalizes alignments that reflect only sequence similarity.

**Training set**  Græmlin 1.1 randomly generates each equivalence class in the random model training set. It first samples a random protein and a random KO group. It then iteratively adds proteins to the equivalence class and stops when the class contains the same number of proteins as the KO group. At each iteration it adds a protein randomly chosen from all proteins with a BLAST hit to at least one protein already in the equivalence class.

To generate a pair of equivalence classes in the random model training set, Græmlin 1.1 first generates a random network alignment. It first samples a random KEGG pathway and a protein within it and then generates a random equivalence class from the protein. It then iteratively grows the alignment and stops when it contains the same number of equivalence classes as the KEGG pathway. At each iteration it randomly chooses a protein from all proteins connected to at least one equivalence class in the current alignment, and it then generates a random equivalence class from the protein. Once it generates the random network alignment, Græmlin 1.1 samples a random pair of equivalence classes from the alignment.

### 5.4 Results of parameter estimation

We used the networks from the Stanford Network Database to train parameters for Græmlin 1.1’s alignment and random models. Figure 5.3 shows the event probabilities learned from training.

For all events, there is a clear difference between the probabilities assigned by the alignment and random models. Figure 5.3C shows that the models distinguish proteins that perform the same function within a KEGG pathway from groups of proteins that merely share sequence similarity. Similarly, Figure 5.3F shows that the models distinguish KEGG pathways from arbitrary groups of connected proteins. Figure 5.3E provides confirmation that functional modules are conserved across species—it shows not only high conservation of edges within KEGG pathways but also a correlation of functional edge loss with time.

Many of the results in Figure 5.3 are consistent with previous claims. For example, Figure 5.3B agrees with previous results that gene loss correlates with time [103], while, in support of the same
study, Figure 5.3D shows a much weaker dependence of gene duplication on time [103]. Recent studies have also suggested that interaction transfer based on sequence similarity is ineffective except for highly related sequences [72, 121]; the weak correlation in Figure 5.3E of interaction loss with time in the random model, which considers only sequence similarity when it groups proteins into equivalence classes, is consistent with this claim. Finally, the difference in the parameters of the alignment and random models in Figure 5.3C is consistent with recent claims of negative correlations between gene dispensability and evolutionary rate [122, 123], since it indicates that functional orthologs within a conserved module are more similar than an average pair of sequence-similar proteins.

5.5 Benchmarks

We tested Græmlin 1.1 to quantify its performance relative to Græmlin and MaWISH. We did not test NetworkBLAST due to its impractical running times on the large network datasets in the Stanford Network Database. As for the tests in Chapter 4, we sought to examine the relative ability of the aligners to correctly align known functional modules.

5.5.1 Test setup

We used KEGG pathways as sample known modules. Because we also trained parameters for Græmlin 1.1 on sample KEGG pathways, we used 3-fold cross-validation [51] to limit biases in favor of Græmlin 1.1. We partitioned the set of KEGG pathways into three test sets, computed results for each test set with a version of Græmlin 1.1 trained only on KEGG pathways outside of the test set, and then combined the results.

Our goal was to measure the departure of an alignment from the correct alignment of a KEGG pathway. A correct alignment of a KEGG pathway has one equivalence class that corresponds

---

Figure 5.3 (facing page): The random and alignment models have different parameters. We trained parameters for Græmlin 1.1’s models as described in the text. In each plot, parameters for the alignment model $\mathcal{M}$ are shown in blue while parameters for the random model $\mathcal{R}$ are shown in red. For time-dependent parameters, branch lengths are given in units of Kimura distance [120]. A. This plot shows the probabilities for the protein present and number of proteins events. Shown are $1 - p^{pp}_G$ and $np^{pp}_G$ for $x = 1, \ldots, 5$, for $\mathcal{G} = \mathcal{M}, \mathcal{R}$. B. This plot shows the probability for the protein deletion event for various evolutionary distances $t$. Shown is $Pr_G(pd = 1|t)$. C. This plot shows parameters for the distributions of the protein mutation event and the paralog protein mutation event. The left axes and upper dataset (indicated with boxes for points) show $p^{pu}_G(t)$; the points to the left of the dashed vertical line show $p^{pu}_{G}$. The right axes and lower dataset (indicated with circles for points) show parameters for the distribution of $sv$: solid lines show the mean $\mu_{G}$ and dashed lines represent one standard deviation $\sigma_{G}$ above or below $\mu_{G}$. The points to the right of the dashed vertical line show $\mu_{G}$ and $\sigma_{G}$. D. This plot shows the probability for the protein duplication event. Shown is $Pr_G(du = 1|t)$. E. This plot shows the probabilities for the edge deletion event and the paralog edge deletion event. The main dataset shows $Pr_G(ed = 1|t)$ and the points to the left of the dashed vertical line show $p^{ed}_G$. F. This plot shows the probabilities for the edge structure event. The points to the right of the dashed lines show $Pr_G(ew)$ and the points to the left show $p^{ed}_{\mathcal{M}}$ and $p^{ed}_{\mathcal{R}}(d)$. 
CHAPTER 5. IMPROVED SCORING FUNCTION FOR GRÆMLIN

A. Number of proteins

B. Protein deletion

C. BLAST bitscore

D. Protein duplication

E. Edge deletion

F. Edge structure
to each KO group in the pathway, where each equivalence class contains every protein with the corresponding KO annotation. If a KO group is not present in the alignment it is missed by the alignment. If a protein belongs to an equivalence class that corresponds to a KO group other than its own it is misaligned; a correct KO group corresponds to an equivalence class with no misaligned proteins. Less severe mistakes are incomplete equivalence classes that lack all proteins within the corresponding KO group and split KO groups that have multiple corresponding equivalence classes.

5.5.2 Results

We first computed results for the metrics in Chapter 4, shown in Table 5.1. We computed these results for several network-to-network and query-to-network alignments, both pairwise and multiple. Our tests focused on query-to-network alignment to facilitate in-depth analysis of specific KEGG pathways. The numbers for the original Graemlin algorithm vary slightly from those in Chapter 4 due to slight differences in the algorithm implementation.

The numbers in Table 5.1 only give a rough measure of each algorithm’s sensitivity and specificity. For a more detailed analysis, we computed a detailed comparison of the six-way query-to-network alignments produced by Graemlin 1.1 and Graemlin. Table 5.2 describes and presents these results.

These results show foremost that Graemlin 1.1 is more specific than Graemlin and MaWISH. Table 5.1 shows that Graemlin 1.1 misaligns significantly fewer nodes but does not hit significantly fewer KEGGs. Table 5.2 sheds more light on mechanisms for this specificity increase: while Graemlin 1.1 includes a similar number of KO groups in its alignments as Graemlin, it aligns more KO groups correctly. Therefore, the main problem with Graemlin’s scoring function is its inclusion of incorrect proteins in an equivalence class.

Aligned KO groups with incorrect proteins are undesirable because they overestimate the conservation of a module. Because it avoids incorrect KO groups to a greater extent than Graemlin, Graemlin 1.1 enables more accurate conclusions about the biological and evolutionary properties of functional modules.

5.6 Discussion

Graemlin 1.1 addresses some of the weaknesses in Graemlin’s scoring function. It incorporates ideas used by other aligners, such as log-odds scoring [63, 2], probabilistic models [77, 82], and evolutionary ideas [81, 82, 2], into a general scoring function that it trains on a set of known network alignments.

When trained on KEGG pathways, Graemlin 1.1 has higher sensitivity and specificity than MaWISH and Graemlin. The parameters for its scoring function are also consistent with other studies: their values support claims that proteins within functional modules evolve differently than do proteins that share only sequence similarity [122, 123] and that several genomic variables depend on evolutionary distance [103].
## CHAPTER 5. IMPROVED SCORING FUNCTION FOR GRAEMLIN

<table>
<thead>
<tr>
<th>Aligner</th>
<th>KEGGs hit</th>
<th>KEGG Coverage</th>
<th>Nodes Misaligned</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Network-to-network</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>E. coli vs. C. crescentus</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graemlin</td>
<td>31</td>
<td>65%</td>
<td>12%</td>
<td>25 s</td>
</tr>
<tr>
<td>Graemlin 1.1</td>
<td>33</td>
<td>64%</td>
<td>9%</td>
<td>70 s</td>
</tr>
<tr>
<td>MaWISh</td>
<td>22</td>
<td>38%</td>
<td>34%</td>
<td>8 s</td>
</tr>
<tr>
<td><em>E. coli vs. C. jejuni vs. H. pylori</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graemlin</td>
<td>18</td>
<td>69%</td>
<td>12%</td>
<td>69 s</td>
</tr>
<tr>
<td>Graemlin 1.1</td>
<td>17</td>
<td>68%</td>
<td>10%</td>
<td>20 s</td>
</tr>
<tr>
<td><strong>Query-to-network</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>E. coli vs. C. crescentus</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graemlin</td>
<td>31</td>
<td>62%</td>
<td>16%</td>
<td>4 s</td>
</tr>
<tr>
<td>Graemlin 1.1</td>
<td>30</td>
<td>58%</td>
<td>7%</td>
<td>4 s</td>
</tr>
<tr>
<td>MaWISh</td>
<td>24</td>
<td>41%</td>
<td>74%</td>
<td>35 s</td>
</tr>
<tr>
<td><em>E. coli vs. S. coelicolor</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graemlin</td>
<td>25</td>
<td>70%</td>
<td>15%</td>
<td>248 s</td>
</tr>
<tr>
<td>Graemlin 1.1</td>
<td>26</td>
<td>59%</td>
<td>7%</td>
<td>21 s</td>
</tr>
<tr>
<td>MaWISh</td>
<td>20</td>
<td>45%</td>
<td>15%</td>
<td>440 s</td>
</tr>
<tr>
<td><em>E. coli vs. M. tuberculosis</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graemlin</td>
<td>26</td>
<td>50%</td>
<td>19%</td>
<td>9 s</td>
</tr>
<tr>
<td>Graemlin 1.1</td>
<td>30</td>
<td>44%</td>
<td>11%</td>
<td>9 s</td>
</tr>
<tr>
<td>MaWISh</td>
<td>21</td>
<td>33%</td>
<td>34%</td>
<td>158 s</td>
</tr>
<tr>
<td><em>E. coli vs. C. jejuni vs. H. pylori</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graemlin</td>
<td>17</td>
<td>62%</td>
<td>16%</td>
<td>2 s</td>
</tr>
<tr>
<td>Graemlin 1.1</td>
<td>16</td>
<td>71%</td>
<td>13%</td>
<td>2 s</td>
</tr>
<tr>
<td><em>E. coli vs. V. cholerae vs. C. crescentus</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graemlin</td>
<td>28</td>
<td>69%</td>
<td>13%</td>
<td>9 s</td>
</tr>
<tr>
<td>Graemlin 1.1</td>
<td>29</td>
<td>65%</td>
<td>6%</td>
<td>8 s</td>
</tr>
<tr>
<td><em>E. coli vs. S. typhimurium, V. cholerae, C. jejuni, H. pylori, C. crescentus</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graemlin</td>
<td>11</td>
<td>84%</td>
<td>19%</td>
<td>26 s</td>
</tr>
<tr>
<td>Graemlin 1.1</td>
<td>11</td>
<td>81%</td>
<td>11%</td>
<td>24 s</td>
</tr>
</tbody>
</table>

Table 5.1: **Graemlin 1.1 is more accurate than Graemlin and MaWISh.** We aligned various subsets of *Escherichia coli* K12, *Salmonella typhimurium* LT2, *Vibrio cholerae*, *Campylobacter jejuni* NCTC 11168, *Helicobacter pylori* 26695, and *Caulobacter crescentus*. For network-to-network alignment we aligned the complete networks of two or three species; for query-to-network alignment we successively queried each *E. coli* KEGG pathway against a database of one or more species. The metrics are the same as in Chapter 4.
5.6. DISCUSSION

<table>
<thead>
<tr>
<th>Metric</th>
<th>Mean</th>
<th>Median</th>
<th>Superior</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Græmlin</td>
<td>Græmlin 1.1</td>
<td>Græmlin</td>
</tr>
<tr>
<td>Size of pathway</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total KOs</td>
<td>13.5</td>
<td>13.9</td>
<td>11</td>
</tr>
<tr>
<td>Correct KOs</td>
<td>11</td>
<td>12.3</td>
<td>7.5</td>
</tr>
<tr>
<td>Incorrect KOs</td>
<td>2.5</td>
<td>1.6</td>
<td>0</td>
</tr>
<tr>
<td>Size of correct KOs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Species per KO</td>
<td>4.27</td>
<td>4.27</td>
<td>4.22</td>
</tr>
<tr>
<td>Proteins per KO</td>
<td>4.88</td>
<td>4.90</td>
<td>4.86</td>
</tr>
</tbody>
</table>

Table 5.2: **Græmlin 1.1 is more specific than Græmlin.** We aligned each *E. coli* KEGG pathway against a database with the networks of *S. typhimurium*, *V. cholerae*, *C. jejuni*, *H. pylori*, and *C. crescentus*. The metrics shown in the table measure how close an alignment approximates the correct alignment of a KEGG pathway. Shown for each metric is the mean and median values over all KEGG pathways, as well as the fraction of KEGG pathways on which one algorithm is superior to the other with respect to that metric.

**Size of pathway.** These metrics measure the degree to which an alignment contains all KO groups from a pathway. Total KOs. The total number of KO groups in an alignment. Correct KOs. A KO group is correct if its corresponding equivalence class contains no misaligned proteins. Incorrect KOs. A KO group is incorrect if it appears in the alignment but is not a correct KO group. **Size of correct KOs.** These metrics measure the degree to which an equivalence class contains all proteins from a KO group. Number of species per KO. The number of species in an equivalence class, averaged over all equivalence classes in the alignment that correspond to correct KO groups. Number of proteins per KO. The number of proteins in an equivalence that, averaged over all equivalence classes in the alignment that correspond to correct KO groups.

Nonetheless, weaknesses remain in Græmlin 1.1’s mathematical formulation, scoring function, and search algorithm. For example, Græmlin 1.1’s local alignment formulation requires a random model, which restricts the form of its scoring function. Græmlin 1.1’s scoring function has an evolutionary basis but its the log-odds framework is ultimately heuristic. In addition, its greedy search algorithm searches for each alignment independently and ignores relationships among alignments.

The next chapter discusses the weaknesses of Græmlin 1.1, and other network alignment algorithms, in more detail. It presents Græmlin 2.0, a completely different network aligner that addresses many of these weaknesses.
This chapter describes Græmlin 2.0, a multiple network alignment algorithm with new multi-stage approach to local alignment, a novel scoring function, a fully automatic algorithm that learns the scoring function’s parameters, and an algorithm that uses the scoring function to align multiple networks in linear time. Græmlin 2.0 significantly increases accuracy when aligning protein interaction networks and aids network alignment users because it automatically adapts its scoring function to any network dataset.

6.1 Background and motivation

As described in Chapter 5, most network alignment research has focused on pairwise network alignment search algorithms, while scoring function research has focused on various models of network evolution. Despite advances in both areas, network alignment tools still have several limitations.

First, local alignment tools conflate steps that match conserved proteins with steps that group proteins into modules. This makes the local alignment problem more difficult and makes it harder to apply established techniques from related problems such as clustering.

Second, local alignment tools that look for all high scoring local alignments must use a random model in their scoring function. The random model ensures that the expected score of a local alignment is negative but is difficult to define, as Chapter 5 discusses.

Third, existing network alignment scoring functions cannot automatically adapt to multiple network datasets. Because networks have different edge densities and noise levels, which depend on the experiments or integration methods used to obtain the networks, parameters that align one set of networks accurately might align another set of networks inaccurately.

Fourth, existing scoring functions use only sequence similarity, interaction conservation, protein duplications, and protein deletions to compute scores. As scoring functions use additional features, parameters become harder to hand-tune.
Finally, existing evolutionary scoring functions do not apply to multiple network alignment. Existing multiple network aligners either have no evolutionary model (NetworkBLAST) or use heuristic parameter choices (Graemlin).

Graemlin 2.0 is a network aligner that addresses these limitations. It uses a new multi-stage approach to local network alignment, a novel network alignment scoring function, an algorithm that uses a training set of known alignments to automatically learn parameters for the scoring function, and a search algorithm that uses the scoring function to perform approximate network alignment in linear time. Our benchmarks show that Graemlin 2.0 has higher sensitivity and specificity than existing network alignment algorithms.

6.2 Algorithm

Graemlin 2.0 uses a novel formulation of the local network alignment problem. Unlike other network alignment algorithms, it does not search for a set of maximally scoring local network alignments. Instead, it searches for the set of local network alignments of maximum total score. Graemlin 2.0 searches for local alignments in three stages:

**Stage 1: Global alignment** Graemlin 2.0 globally aligns the input networks—this groups nodes into equivalence classes. Graemlin 2.0’s global alignment scoring function computes evolutionary events and is trained on a set of known global network alignments.

**Stage 2: Disjoint local alignment** Graemlin 2.0 segments the global alignment into a set of disjoint local alignments—this groups equivalence classes into local alignments. Graemlin 2.0’s local alignment scoring function measures local alignment connectivity and is trained on a set of known local network alignments.

**Stage 3: Probabilistic assignment** Graemlin 2.0 assigns each node a probabilistic membership in each local alignment—this allows equivalence classes to belong to multiple local alignments. Graemlin 2.0 casts this stage as a supervised learning problem and uses a non-parametric Bayesian classifier.

Graemlin 2.0’s approach, illustrated in Figure 6.1, addresses several weaknesses of current local alignment algorithms.

First, Graemlin 2.0 performs each alignment stage separately. Each stage is simpler to solve than the entire local alignment problem at once.

Second, the first two stages of Graemlin 2.0 each optimize a global objective function. Graemlin 2.0 therefore does not need a random model because it does not locally optimize an objective function.

Third, Graemlin 2.0 uses feature-based scoring functions that can use arbitrary features of a network alignment. The global alignment scoring function computes evolutionary features, and
Figure 6.1: **Graemlin 2.0 performs local network alignment in three stages.** In stage 1, Graemlin 2.0 globally aligns the input set of networks. In stage 2, Graemlin 2.0 segments the global alignment into a set of disjoint local alignments. In stage 3, Graemlin 2.0 assigns each node a probabilistic membership in each local alignment. Graemlin 2.0 learns scoring functions for the global alignment phase and the disjoint local alignment phase from a training set of known alignments.

Both the global alignment and the local alignment scoring functions apply to multiple alignment as well as pairwise alignment.

Finally, Graemlin 2.0 learns parameters for its scoring functions from a set of known network alignments. These parameters produce more accurate alignments than manually tuned scoring functions common in current algorithms. Graemlin 2.0 can automatically learn parameters to align any set of networks when given a suitable training set.

### 6.2.1 Stage 1: Global alignment

The input to Graemlin 2.0’s global alignment stage is $d$ networks. The output is a global alignment $a^*_g = \arg \max_{a_g \in \mathcal{A}_g} s_g(a_g)$, where $\mathcal{A}_g$ is the set of potential global alignments of the networks and $s_g$ is a scoring function for global network alignment. The global alignment stage therefore groups nodes into equivalence classes.

Graemlin 2.0 hypothesizes that a node will not align to different nodes in different local alignments. It can therefore determine the grouping of nodes into equivalence classes (Stage 1) before it determines the grouping of equivalence classes into local alignments (Stage 2).

### Scoring function

Graemlin 2.0’s global alignment scoring function computes “features” [112, 124] of a global network alignment. Formally, we define a vector-valued global alignment *feature function* $f_g : \mathcal{A}_g \rightarrow \mathbb{R}^n$, which maps a global alignment to a numerical *feature vector*. More specifically, we define a node feature function $f^N$ that maps equivalence classes to a feature vector and an edge feature function
6.2. ALGORITHM

\( f^E \) that maps pairs of equivalence classes to a feature vector. We then define

\[
f_g(a_g) = \left( \sum_{[x] \in a_g} f^N([x]) \right) \sum_{[x],[y] \in a_g, [x] \neq [y]} f^E([x],[y])
\] (6.1)

with the first sum over all equivalence classes in the alignment \( a_g \) and the second sum over all pairs of equivalence classes in \( a_g \).

Given a numerical parameter vector \( w_g \), the score of a global alignment \( a_g \) is \( s_g(a_g) = w_g \cdot f_g(a_g) \). The global alignment parameter learning problem is to find \( w_g \). Græmlin 2.0’s parameter learning algorithm is discussed below.

The feature function isolates the biological meaning of network alignment. Græmlin 2.0’s learning and alignment algorithms make no further biological assumptions. Furthermore, one can define a feature function for any kind of network. Græmlin 2.0’s scoring function therefore applies to any set of networks, regardless of the meaning of nodes and edges.

**Implementation for protein interaction networks** Græmlin 2.0 uses a global alignment feature function that computes evolutionary events. The feature function is simplest for the special case of pairwise global network alignment (the alignment of two networks) but generalizes to multiple global network alignment. Figure 6.2 illustrates the evolutionary events that Græmlin 2.0’s feature function computes.

Græmlin 2.0’s pairwise node feature function computes the occurrence of four evolutionary events between the species in an equivalence class:

1. **Protein deletion**: the loss of a protein in one of the two species
2. **Protein duplication**: the duplication of a protein in one of the two species
3. **Protein mutation**: the divergence in sequence of two proteins in different species
4. **Paralog mutation**: the divergence in sequence of two proteins in the same species

Græmlin 2.0’s pairwise edge feature function computes the occurrence of two evolutionary events between the species in a pair of equivalence classes:

1. **Edge deletion**: the loss of an interaction between two pairs of proteins in different species
2. **Paralog edge deletion**: the loss of an interaction between two pairs of proteins in the same species

The value of each event is one if the event occurs and zero if it does not. The entries in the feature vector are the values of the events.
CHAPTER 6. GRÆMLIN 2.0

The pairwise feature functions require two changes to generalize to multiple network alignment. First, a multiple alignment feature function is the sum of pairwise feature functions, each computed between pairs of adjacent (and possibly ancestral) species in a phylogenetic tree. Second, a multiple network alignment feature function includes features that depend on evolutionary distance.

To compute a feature function for a pair of ancestral species, Græmlin 2.0 reconstructs approximate ancestral equivalence classes. It first computes species weight vectors \([118]\) for each ancestral species. Each species weight vector contains numerical weights that represent the similarity of each extant species to the ancestral species. Græmlin 2.0 uses these species weight vectors, together with the proteins in the equivalence class, to approximate the proteins in the ancestral equivalence class. It then computes pairwise feature functions between the approximate ancestral proteins. Section 6.4 describes the exact procedure.

To include dependencies on evolutionary distance in a feature function, Græmlin 2.0 uses an augmented feature vector. It includes a new feature \(f_i \times b\), where \(b\) is the distance between the species pair, for each original feature \(f_i\). Effectively, this transformation allows features to have linear dependencies on \(b\). Additional terms such as \(f_i \times b^2, f_i \times b^3, \ldots\) have more complex dependencies on \(b\).

Section 6.4 contains precise definitions of Græmlin 2.0’s global alignment feature function as well as precise definitions of all evolutionary events.

**Parameter learning algorithm**

**Inputs** Græmlin 2.0’s algorithm to find \(w_g\) requires a training set of known global alignments. The training set is a collection of \(m\) training examples; each training example specifies a set of networks \(\{G^{(i)} = G_1^{(i)}, \ldots, G_d^{(i)}\}\) and their correct global alignment \(a_g^{(i)}\).

Græmlin 2.0’s parameter learning algorithm also requires a loss function \(\Delta : A_g \times A_g \rightarrow \mathbb{R}^+\). By definition, \(\Delta(a_g^{(i)}, a_g)\) must be 0 when \(a_g^{(i)} = a_g\) and positive when \(a_g^{(i)} \neq a_g\) [125]. Intuitively, \(\Delta(a_g^{(i)}, a_g)\) measures the distance of an alignment \(a_g\) from the training alignment \(a_g^{(i)}\); the learned parameter vector should therefore assign higher scores to alignments with smaller loss function values.

To train parameters for Græmlin 2.0’s global alignment feature function, we used a training set of KEGG Ortholog (KO) groups [48]. Each training example contained the networks from a set of species, with nodes removed that did not have a KO group. The correct global alignment contained an equivalence class for each KO group.

We also defined a loss function that grows as alignments diverge from the correct alignment \(a_g^{(i)}\). More specifically, we let \([x]_{a_g^{(i)}}\) denote the equivalence class of \(x \in V^{(i)} = \bigcup_j V_j^{(i)}\) in \(a_g^{(i)}\), and we let \([x]_{a_g}\) denote the equivalence class of \(x\) in \(a_g\). We define \(\Delta(a_g^{(i)}, a_g) = \sum_{x \in V^{(i)}} [x]_{a_g} \setminus [x]_{a_g^{(i)}}\), where \(A \setminus B\) denotes the set difference between \(A\) and \(B\). This loss function is proportional to the number of nodes aligned in \(a_g\) but not aligned in the correct alignment \(a_g^{(i)}\).
Figure 6.2: Graemlin 2.0's global alignment feature function computes evolutionary events. This figure shows the set of evolutionary events that Graemlin 2.0's node and edge feature functions compute. Graemlin 2.0 uses a phylogenetic tree with branch lengths to determine the events. It first constructs species weight vectors at each internal tree node; the weight vector represents the similarity of each extant species to the internal tree node. It then uses these weight vectors to compute the likely evolutionary events (shown as boxes) that occur; the appendix gives precise definitions of these events. Graemlin 2.0 combines the values of the events into a feature vector, and the score of the global alignment is the dot product of a numeric weight vector with the feature vector.
We experimented with the natural opposite of this loss function—the number of nodes aligned in the correct global alignment \(a_g\) but not aligned in \(a_g\). As expected, this alternate loss function resulted in a scoring function that aligned more nodes. We found empirically, however, that the original loss function was more accurate.

**Theory** We pose parameter learning as a maximum margin structured learning problem. We find a parameter vector that solves the following convex program [125]:

\[
\begin{align*}
\min_{\mathbf{w}_g, \xi_1, \ldots, \xi_m} & \quad \frac{\lambda}{2} \| \mathbf{w}_g \|^2 + \frac{1}{m} \sum_{i=1}^{m} \xi_i \\
\text{s.t.} & \quad \forall i, a_g \in \mathcal{A}_g^{(i)}, \mathbf{w}_g \cdot \mathbf{f}_g(a_g) + \xi_i \geq \mathbf{w}_g \cdot \mathbf{f}_g(a_g) + \Delta(a_g^{(i)}, a_g).
\end{align*}
\]

The constraints in this convex program encourage the learned \(\mathbf{w}_g\) to satisfy a set of conditions: each training alignment \(a_g^{(i)}\) should score higher than all other global alignments \(a_g\) by at least \(\Delta(a_g^{(i)}, a_g)\). The slack variables \(\xi_i\) are penalties for each unsatisfied condition. The objective function is the sum of the penalties with a regularization term that prevents overfitting. Given the low risk of overfitting the few free parameters in our model, we set \(\lambda = 0\) for convenience. In more complex models with richer feature sets, overfitting can be substantially more severe when the amount of training data is limited; the use of effective regularization techniques in such cases is a topic for future research.

We can show [125] that this constrained convex program is equivalent to the unconstrained minimization problem

\[
c(\mathbf{w}_g) = \frac{1}{m} \sum_{i=1}^{m} r^{(i)}(\mathbf{w}_g) + \frac{\lambda}{2} \| \mathbf{w}_g \|^2,
\]

where \(r^{(i)}(\mathbf{w}_g) = \max_{a_g \in \mathcal{A}_g^{(i)}} (\mathbf{w}_g \cdot \mathbf{f}_g(a_g) + \Delta(a_g^{(i)}, a_g)) - \mathbf{w}_g \cdot \mathbf{f}_g(a_g^{(i)}).
\]

This objective function is convex but nondifferentiable [125]. We can therefore minimize it with subgradient descent [126], an extension of gradient descent to nondifferentiable objective functions.

A subgradient of equation (6.2) is [125]

\[
\lambda \mathbf{w}_g + \frac{1}{m} \sum_{i=1}^{m} (\mathbf{f}_g(a_g) - \mathbf{f}_g(a_g^{(i)})),
\]

where \(a_g^{(i)} = \arg \max_{a_g \in \mathcal{A}_g^{(i)}} \mathbf{w}_g \cdot \mathbf{f}_g(a_g) + \Delta(a_g^{(i)}, a_g)\) is the optimal global alignment, determined by the loss function \(\Delta(a_g^{(i)}, a_g)\) and current \(\mathbf{w}_g\), of \(G^{(i)}\).

**Algorithm** Based on these ideas, Græmlin 2.0’s parameter learning algorithm performs subgradient descent. It starts with \(\mathbf{w}_g = 0\). Then, it iteratively computes the subgradient \(\mathbf{g}\) of equation (6.2) at the current parameter vector \(\mathbf{w}_g\) and updates \(\mathbf{w}_g \leftarrow \mathbf{w}_g - \alpha \mathbf{g}\), where \(\alpha\) is the learning rate.
LEARN($\{G_1^{(i)}, \ldots, G_d^{(i)}, a_g^{(i)}\}_{i=1}^m$ : training set, $\alpha$ : learning rate, $\lambda$ : regularization)

1. var $w_g \leftarrow 0$
2. var $c_* \leftarrow \infty$
3. var $w_* \leftarrow w_g$
4. while $c_*$ updated in last 100 iterations do
   5. var $g \leftarrow 0$
   6. var $c = 0$
   7. for $i = 1 : m$
      8. do
         9. var $a_*^{(i)} = \text{ALIGN}(G_1^{(i)}, \ldots, G_d^{(i)}, w_g, \Delta)$
         10. $g \leftarrow g + f_g(a_*^{(i)}) - f_g(a_g^{(i)})$
         11. $c \leftarrow c + w_g \cdot f_g(a_*^{(i)}) + \Delta(a_g^{(i)}, a_*^{(i)}) - w_g \cdot f_g(a_g^{(i)})$
         12. $g \leftarrow \frac{1}{m}g + \lambda w_g; c \leftarrow \frac{1}{m}c + \frac{\lambda}{2} \| w_g \|^2$
         13. if $c < c_*$
            14. then
               15. $c_* \leftarrow c; w_* = w_g$
               16. $w_g \leftarrow w_g - \alpha g$
         17. return $w_*$

Figure 6.3: Græmlin 2.0’s parameter learning algorithm performs subgradient descent. The parameter learning algorithm iteratively updates the parameter vector. At each iteration, it decreases the weight of features with higher values in the optimal alignment than in the training example and increases the weight of features with lower values in the optimal alignment than in the training example.

The algorithm stops when it performs 100 iterations that do not reduce the objective function. Græmlin 2.0 uses a small constant ($\alpha = 0.05$) for the learning rate.

The algorithm for finding $\arg\max_{a_g \in A_g^{(i)}} w_g \cdot f_g(a_g) + \Delta(a_g^{(i)}, a_g)$ is the global alignment inference algorithm. It is a global alignment search algorithm with a scoring function augmented by $\Delta$. The next section describes an efficient approximate global alignment search algorithm that Græmlin 2.0 uses as an approximate inference algorithm.

Græmlin 2.0’s parameter learning algorithm has an intuitive interpretation. At each iteration it uses the loss function $\Delta$ and the current $w_g$ to compute the optimal global alignment. It then decreases the weight of features with higher values in the optimal alignment than in the training example and increases the weight of features with lower values in the optimal alignment than in the training example. Figure 6.3 shows Græmlin 2.0’s parameter learning algorithm.

Græmlin 2.0’s parameter learning algorithm also has performance guarantees. If the inference algorithm is exact, and if the learning rate is constant, the learning algorithm converges at a linear rate to a small region surrounding the optimal $w_g$ [127, 125]. A bound on convergence with an approximate inference algorithm is a topic for further research.
Global alignment search algorithm

Græmlin 2.0’s global alignment search algorithm produces a global alignment \( a_g \) of a set of input networks. Its goal is to maximize \( s_g(a_g) \).

The search algorithm serves two roles. It finds the highest scoring global alignment given an optimal learned parameter vector, and it performs inference as part of Græmlin 2.0’s global alignment parameter learning algorithm.

Græmlin 2.0’s search algorithm (Figure 6.4) is a local hillclimbing algorithm [128]. The algorithm is approximate but efficient in practice. It requires that the global alignment feature function decompose into node and edge feature functions as in equation (6.1).

The search algorithm iteratively updates a current global alignment. The initial alignment contains every node in a separate equivalence class. The algorithm then proceeds in a series of iterations. During each iteration, it processes each node and evaluates a series of moves for each node:

- Leave the node alone.
- Create a new equivalence class with only the node.
- Move the node to another equivalence class.
- Merge the entire equivalence class of the node with another equivalence class.

For each move, Græmlin 2.0 computes the alignment score before and after the move and performs the move that increases the score the most. Once it has processed each node, the algorithm begins a new iteration. It stops when an iteration does not increase the alignment score.

Græmlin 2.0’s global alignment search algorithm also performs inference for Græmlin 2.0’s global alignment parameter learning algorithm. The search algorithm can use any scoring function that decomposes as in equation (6.1). Therefore, Græmlin 2.0 can use its search algorithm as an inference algorithm if it augments its scoring function with the loss function \( \Delta \), provided that the loss function also decomposes into node and edge scores. The loss function presented above has this property.

The performance of Græmlin 2.0’s search algorithm depends on the set of candidate equivalence classes to which processed nodes can move. As a heuristic, it considers as candidates only equivalence classes with a node that has homology (BLAST [89] e-value \(< 10^{-5}\)) to the processed node.

As a heuristic, it uses node scores—the scoring function with the edge feature function set to zero—to order nodes. For each node, Græmlin 2.0 computes the node score change when it moves the node to each candidate equivalence class. It saves the maximum node score change for each node and then considers nodes in order of decreasing maximum node score change.

In practice, Græmlin 2.0’s global alignment search algorithm runs in linear time. To align networks with \( n \) total nodes and \( m \) total edges, Græmlin 2.0 performs \( b_1 \) iterations that each
6.2. ALGORITHM

\texttt{ALIGN}(G_1, \ldots, G_d : \text{set of networks}, w_g : \text{parameter vector}, \Delta : \text{optional loss function })

1 \texttt{var } a_g \leftarrow \text{an alignment with one equivalence class per node}
2 \texttt{while } \texttt{true}
3 \texttt{do}
4 \texttt{\hspace{1em}var } \delta_t = 0
5 \texttt{\hspace{1em}for each node } p \in \bigcup_i G_i
6 \texttt{\hspace{2em}do}
7 \texttt{\hspace{3em}var } \delta^* = 0
8 \texttt{\hspace{3em}var } o^* \leftarrow \text{undef}
9 \texttt{\hspace{3em}for each move } o \text{ of node } p
10 \texttt{\hspace{4em}do}
11 \texttt{\hspace{5em}var } a_t \leftarrow o(a_g)
12 \texttt{\hspace{5em}\delta \leftarrow w_g \cdot f_g(a_t) + \Delta(a_t) - (w_g \cdot f_g(a_g) + \Delta(a_g))}
13 \texttt{\hspace{5em}if } \delta > \delta^* \texttt{ then}
14 \texttt{\hspace{6em}\delta^* = \delta; o^* = o}
15 \texttt{\hspace{5em}a_g \leftarrow o^*(a_g)}
16 \texttt{\hspace{5em}\delta_t \leftarrow \delta_t + \delta^*}
17 \texttt{\hspace{3em}if } \delta_t = 0 \texttt{ then break}
18 \texttt{return } a_g

\textbf{Figure 6.4:} \texttt{Græmlin 2.0's global alignment search algorithm is a local hillclimbing algorithm}

The search algorithm iteratively updates the global alignment. At each iteration, it evaluates a series of moves and performs the move that increases the score of the alignment the most.
process a nodes. For each node Græmlin 2.0 computes the change in score when it moves the node to, on average, C candidate classes. Because the global alignment feature function decomposes as in equation (6.1), to perform each score computation Græmlin 2.0 needs only to examine the candidate class, the node’s old class, and the two classes’ neighbors. Its running time is therefore $O(b_1C(n + m)))$. Empirically, $b_1$ is usually a small constant (less than 10). While $C$ can be large, Græmlin 2.0 runs faster if it only considers candidate classes with high homology to the processed node (BLAST e-value $\ll 10^{-5}$.)

### 6.2.2 Stage 2: Disjoint local alignment

The input to Græmlin 2.0’s disjoint local alignment stage is a global alignment $a^*_g$. The output is a set of disjoint local alignments $A^*_\ell = \arg \max_{A_\ell \in A_{\ell}(a^*_g)} S_\ell(A_\ell)$; here, $A_{\ell}(a^*_g)$ is the set of possible partitions of $a^*_g$’s equivalence classes into disjoint local alignments, and $S_\ell$ is a scoring function for a set of local alignments. The disjoint local alignment stage therefore groups the equivalence classes, determined in the global alignment stage, into local alignments.

Græmlin 2.0 searches for the set of local alignments of maximum total score. This search contrasts with the traditional search for a set of maximally scoring alignments, or $A^*_\ell = \{a_\ell : s_\ell(a_\ell) \text{ is maximal}\}$, where $s_\ell$ is a scoring function for a single local alignment.

The disjoint local alignment stage is similar to the traditional clustering problem [51]. Græmlin 2.0 can in fact use any clustering algorithm in its disjoint local alignment stage. However, while clustering algorithms can use simple distance metrics for pairwise alignment of networks with single node and edge types, it becomes hard to define robust distance metrics for complex networks with multiple node or edge types [129, 130, 131]. Below, we present an algorithm that can use arbitrary features of a set of local alignments and generalizes to align complex networks.

#### Scoring function

Græmlin 2.0’s local alignment scoring function uses the same principles as its global alignment scoring function. We therefore outline only the main differences between the global and local alignment scoring functions.

A local alignment feature function $f_\ell$ maps a local alignment to a numerical feature vector. Given a numerical parameter vector $w_\ell$, the score of a local alignment $a_\ell$ is $s_\ell(a_\ell) = w_\ell \cdot f_\ell(a_\ell)$.

The score of a set of local alignments $A_\ell$ is then $S_\ell(A_\ell) = \sum_{a_\ell \in A_\ell} s_\ell(a_\ell) = \sum_{a_\ell \in A_\ell} w_\ell \cdot f_\ell(a_\ell) = w_\ell \cdot \sum_{a_\ell \in A_\ell} f_\ell(a_\ell) \equiv w_\ell \cdot F_\ell(A_\ell)$, where $F_\ell(A_\ell)$ is the sum of the feature vectors of the local alignments in $A_\ell$.

#### Implementation for protein interaction networks

Græmlin 2.0 uses a local alignment feature function that computes the degree of connectivity between equivalence classes in a local alignment. It computes six features, illustrated in Figure 6.5:
6.2. ALGORITHM

Figure 6.5: Graemlin 2.0’s local alignment feature function computes features of a local alignment. This figure shows the features that Graemlin 2.0’s local alignment feature function computes as well as values of the features for a sample alignment. Graemlin 2.0 combines the values of the features into a feature vector, and the score of the local alignment is the dot product of a numeric weight vector with the feature vector.

1. **Edge present**: the number of edges between nodes
2. **Edge absent**: the number of missing edges between nodes
3. **Edge weight sum**: the sum of the weights of edges between nodes
4. **Edge weight squared sum**: the sum of the squared weights of edges between nodes
5. **Closest neighbor**: the number of nodes in the alignment with a nearest neighbor (the neighbor in the network of maximum edge weight) also in the alignment
6. **Species non-overlap**: the number of equivalence class pairs that do not have a species in common

These features measure three types of local alignment connectivity. The first four features measure the average edge weight between nodes in the alignment. The fifth feature allows the scoring function to tolerate weakly connected alignments if many nodes are paired with their nearest neighbors. The final feature accounts for equivalence class pairs that the first five features ignore—equivalence class pairs with disjoint sets of species lack the potential to interact and are distinct from pairs that have the potential to interact but do not.
We experimented with other features that did not depend on network edges, such as the size of the alignment and amount of synteny present in the alignment. However, we found that the edge-based features played by far the largest role in the scoring function. As networks incorporate multiple data types and become more accurate, other features will likely become more important.

**Parameter learning algorithm**

The parameter learning algorithm in Figure 6.3 applies to both Græmlin 2.0’s global and local alignment scoring functions. The only differences between the global and local alignment parameter learning algorithms are the form of the training set, the definition of the loss function, and the definition of the inference algorithm.

To train parameters for Græmlin 2.0’s local alignment feature function, we used a training set of KEGG pathways. Each training example contained the networks from a set of species and a correct global alignment with an equivalence class for each KO group. The correct set of local alignments contained a local alignment for each KEGG pathway.

In addition, we set the loss function to a constant value. We experimented with a loss function analogous to that used in the global alignment parameter learning algorithm, but we found such a loss function computationally prohibitive.

The local alignment inference algorithm is a local alignment search algorithm with a scoring function augmented by the loss function. The next section discusses an approximate local alignment search algorithm that Græmlin 2.0 uses as an approximate inference algorithm.

**Disjoint local alignment search algorithm**

Græmlin 2.0’s disjoint local alignment search algorithm segments the equivalence classes in a global alignment $a_g$ into a disjoint set of local alignments $A_L$. Its goal is to maximize the total score $S_L(A_L)$.

Like its global alignment search algorithm, Græmlin 2.0’s local alignment search algorithm performs local hillclimbing. The algorithm begins with an initial set of alignments in which every equivalence class is in a separate local alignment. Then, it performs a series of iterations, each of which processes each equivalence class in turn. Græmlin 2.0 computes the change in score resulting from moving the equivalence class to any other local alignment, and performs the move that increases the score the most. It stops when it performs a iteration that does not increase the score.

In practice, Græmlin 2.0’s local alignment search algorithm runs in linear time. To segment a global alignment of networks with $n$ total nodes and $m$ total edges, Græmlin 2.0 performs $b_2$ iterations, each of which processes at most $n$ equivalence classes. With suitable caching (technical details omitted), Græmlin 2.0 must only examine the edges incident to an equivalence class to compute the change in score resulting from moving the equivalence class to each alignment. The total running time is therefore $O(b_2 \times (n + m))$. Because $b_2$ is usually a small constant (less than 10), the algorithm is efficient in practice.
6.2. Algorithm

## 6.2.3 Stage 3: Probabilistic assignment

The input to Græmlin 2.0’s probabilistic assignment stage is a set of disjoint local alignments \( A^*_\ell \). The output, for each local alignment \( a_\ell \) and each equivalence class \( [x] \) in the global alignment, is the probability that \( [x] \) belongs to \( a_\ell \).

Græmlin 2.0 uses these probabilities to obtain the final set of local alignments. To each local alignment \( a_\ell \) in \( A^*_\ell \), it adds all equivalence classes that belong \( a_\ell \) with probability greater than a user-specified threshold. Lower thresholds yield larger but less accurate local alignments.

Græmlin 2.0 uses a supervised learning algorithm to compute probabilities, with the set of disjoint local alignments as a training set. It bases the probability that an equivalence class \([x]\) belongs to a local alignment \( a_\ell \) on \( \delta_{a_\ell}([x]) = s_\ell(a_\ell \cup [x]) - s_\ell(a_\ell \setminus [x]) \), the difference between the score of \( a_\ell \) with \([x]\) and the score of \( a_\ell \) without \([x]\). Figure 6.6 illustrates the idea.

In detail, Græmlin 2.0 first builds a separate Bayesian classifier for each disjoint local alignment. It computes three statistics for each equivalence class \([x]\) and each local alignment \( a_\ell \):

- \( \Pr([x] \in a_\ell) \), the prior probability that \([x]\) is in \( a_\ell \)
- \( \Pr(\delta_{a_\ell}([x]) \mid [x] \in a_\ell) \), the conditional distribution of \( \delta_{a_\ell} \) given that \([x]\) is in \( a_\ell \)

Figure 6.6: **Græmlin 2.0 assigns probabilistic membership in local alignments.** For each equivalence class \([x]\) and each local alignment \( a_\ell \), Græmlin 2.0 computes the change in score that results when it adds \([x]\) to \( a_\ell \). Intuitively, the probability that \([x]\) belongs to \( a_\ell \) is high if the change in score when Græmlin 2.0 adds \([x]\) to \( a_\ell \) is high relative to the change in score when Græmlin 2.0 adds other nodes to \( a_\ell \).

Græmlin 2.0’s local alignment algorithm is relatively simple compared to other existing local alignment algorithms. However, Section 6.3 shows that Græmlin 2.0 produces accurate local alignments, mainly because it uses an accurate scoring function.
Græmlin 2.0 then uses Bayes’s rule to assign \([x]\) to \(a_\ell\) with probability

\[
\Pr ([x] \in a_\ell \mid \delta_{a_\ell}([x])) = \frac{\Pr (\delta_{a_\ell}([x]) \mid [x] \in a_\ell) \Pr ([x] \in a_\ell)}{\Pr (\delta_{a_\ell}([x]))}
\]

where

\[
\Pr (\delta_{a_\ell}([x])) = \Pr (\delta_{a_\ell}([x]) \mid [x] \in a_\ell) \Pr ([x] \in a_\ell) + \Pr (\delta_{a_\ell}([x]) \mid [x] \notin a_\ell) \Pr ([x] \notin a_\ell)
\]

Græmlin 2.0 estimates \(\Pr([x] \in a_\ell)\) as the ratio of the number of equivalence classes in \(a_\ell\) to the total number of equivalence classes in all local alignments.

Græmlin 2.0 estimates the two conditional distributions using kernel density estimation [49]. For samples from \(\Pr (\delta_{a_\ell}([x]) \mid [x] \in a_\ell)\), it uses the values for the equivalence classes already in \(a_\ell\). For samples from \(\Pr (\delta_{a_\ell}([x]) \mid [x] \notin a_\ell)\), it uses the values for the equivalence classes not already in \(a_\ell\). Græmlin 2.0 uses a Gaussian kernel, with bandwidth chosen by Silverman’s “rule of thumb” [132].

### 6.3 Benchmarks

We performed two sets of benchmarks. The first measured the accuracy of the equivalence class groupings found in Græmlin 2.0’s global alignment stage. The second measured the overall accuracy of the local alignments found by Græmlin 2.0.

#### 6.3.1 Equivalence class accuracy comparisons

**Test setup**

We tested equivalence class accuracy on three different network datasets: IntAct [87], DIP [86], and the Stanford Network Database (SNDB). We ran pairwise alignments of the *Homo sapiens* and *Mus musculus* IntAct networks, *Saccharomyces cerevisiae* and *Drosophila melanogaster* DIP networks, *Escherichia coli* K12 and *Salmonella typhimurium* LT2 SNDB networks, and *E. coli* and *Caulobacter crescentus* SNDB networks. We also ran two multiple alignments: a three-way alignment of the *S. cerevisiae*, *Caenorhabditis elegans*, and *D. melanogaster* DIP networks as well as a six-way alignment of the *E. coli*, *S. typhimurium*, *Vibrio cholerae*, *Campylobacter jejuni* NCTC 11168, *Helicobacter pylori* 26695, and *C. crescentus* SNDB networks.

We used KO groups [48] for our equivalence class comparison metrics. To compute each metric, we first removed all proteins in the alignment without a KO group and we then removed all equivalence
classes with only one protein. We then defined an equivalence class as correct if every protein in it had the same KO group.

To measure specificity, we computed two metrics:

1. the fraction of equivalence classes that were correct ($C_{eq}$)
2. the fraction of proteins that were in correct equivalence classes ($C_{node}$)

To measure sensitivity, we computed two metrics:

1. the total number of proteins that were in correct equivalence classes (Cor)
2. the total number of equivalence classes that contained $k$ species, for $k = 2, \ldots, n$

We used cross validation to test Graemlin 2.0. For each set of networks, we partitioned the KO groups into ten equal sized test sets. For each test set, we trained Graemlin 2.0 on the KO groups outside of the test set. We then aligned the networks and computed our metrics on only the KO groups in the test set. Our final numbers for a set of networks were the average of our metrics over the ten test sets.

To limit biases from the averaging process we used cross validation to test all aligners. For aligners other than Graemlin 2.0 we aligned the networks only one time. However, we did not compute our metrics on all KO groups at once; instead, we computed our metrics separately for each test set and then averaged the numbers.

As a final check that our test and training sets were independent, we computed similar metrics using Gene Ontology (GO) categories [50, 78] instead of KO groups. We do not report the results of these tests because they are similar to the results of our tests on KO groups.

We compared Graemlin 2.0 to the local aligners NetworkBLAST [78], MaWISh [81], and Graemlin 1.1, as well as the global aligner IsoRank [84] and a global aligner (Graemlin-global) that used Graemlin 2.0’s global alignment search algorithm with Graemlin 1.1’s scoring function.

While we simultaneously compared Graemlin 2.0 to IsoRank and Graemlin-global, we compared Graemlin 2.0 to each local aligner separately. According to our definitions, local aligners may have lower sensitivity than global aligners simply because local aligners only align proteins that belong to conserved modules while global aligners align all proteins. Therefore, for each comparison to a local aligner, we removed all equivalence classes in Graemlin 2.0’s output that did not contain a protein in the local aligner’s output.

**Performance comparisons**

Table 6.1 shows that, with respect to the alignment of nodes, Graemlin 2.0 is the most specific aligner. Across all datasets, it produces both the highest fraction of correct equivalence classes as well as the highest fraction of nodes in correct equivalence classes.
Table 6.1: **Græmlin 2.0 aligns nodes with higher specificity.** We measured the fraction of correct equivalence classes \( (C_{eq}) \) and the fraction of nodes in correct equivalence classes \( (C_{node}) \), as described in the text. We compared Græmlin 2.0 (Gr2.0) to NetworkBLAST (NB), MaWISh (MW), Græmlin 1.1 (Gr1.1), IsoRank (Iso), and a global aligner that used Græmlin 2.0’s alignment search algorithm with Græmlin 1.1’s scoring function (GrG). As described in the text, we ran four pairwise alignments, a three-way alignment, and a six-way alignment. For each comparison between Græmlin 2.0 and a local aligner, we removed equivalence classes from Græmlin 2.0’s output that did not contain a node in the local aligner’s output; Table 6.2 shows the number of remaining nodes for each aligner. MaWISh and IsoRank are not multiple aligners; NetworkBLAST can align only up to three species and aborted on the three-way alignment. eco = *E. coli*; stm = *S. typhimurium*; cce = *C. crescentus*; hsa = *Homo sapiens*; mmu = *Mus musculus*; sce = *S. cerevisiae*; dme = *D. melanogaster*.

Table 6.2 shows that, with respect to the alignment of nodes, Græmlin 2.0 is also the most sensitive aligner. In the SNDB pairwise alignments, Græmlin 2.0 and IsoRank produce the most number of nodes in correct equivalence classes. In the other tests, Græmlin 2.0 produces the most number of nodes in correct equivalence classes.

Figure 6.7 shows that Græmlin 2.0 also finds more cross-species conservation than Græmlin 1.1 and Græmlin-global. Relative to Græmlin 1.1 and Græmlin-global, Græmlin 2.0 produces two to five times as many equivalence classes with four, five, and six species.

These results suggest that a network aligner’s scoring function is more important than its search algorithm. Græmlin 2.0 performs better than existing aligners, despite its simple search algorithm, because of its accurate scoring function.

We performed our tests on a 2.2GHz machine with 4GB of RAM. For each pairwise alignment, Græmlin 2.0, MaWISh, Græmlin 1.1, and Græmlin-global ran in less than a minute while IsoRank and NetworkBLAST each ran for over an hour. For each pairwise alignment training run, Græmlin 2.0 ran for under ten minutes. On the six-way alignment, Græmlin 2.0, Græmlin 1.1, and Græmlin-global each ran for under three minutes, and Græmlin 2.0 trained in under forty-five minutes.
6.3. BENCHMARKS

<table>
<thead>
<tr>
<th></th>
<th>SNDB</th>
<th>IntAct</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eco/stm</td>
<td>eco/cce</td>
<td>6-way</td>
</tr>
<tr>
<td></td>
<td>Cor</td>
<td>Tot</td>
<td>Cor</td>
</tr>
<tr>
<td>Local aligner comparisons</td>
<td>NB</td>
<td>Gr2.0</td>
<td>457</td>
</tr>
<tr>
<td></td>
<td>MW</td>
<td>Gr2.0</td>
<td>1309</td>
</tr>
<tr>
<td></td>
<td>Gr1.1</td>
<td>Gr2.0</td>
<td>985</td>
</tr>
</tbody>
</table>

Global aligner comparisons

|               | GrG | –   | 1496 | 720  | 2388 | –   | –   | 268  | 384  | 564 |
|               | Iso | Gr2.0 | 2026 | –   | 1014 | –   | –   | 306  | 534  | –   |
|               |     |      | 2024 | 1012 | 3578 | 350 | 637 | 827  |      |      |

Table 6.2: Gramlin 2.0 aligns nodes with higher sensitivity. We measured the number of nodes in correct equivalence classes (Cor), as described in the text. To show the number of nodes considered in each local aligner comparison, we also measured the number of nodes aligned by each local aligner (Tot). Methodology and abbreviations are the same as in Table 6.1.

6.3.2 Local alignment accuracy comparisons

Test setup

We tested local alignment accuracy on DIP and SNDB (the IntAct networks produced local alignments too small for meaningful comparisons). We ran pairwise alignments of the S. cerevisiae and D. melanogaster DIP networks and the E. coli and C. crescentus SNDB networks, and we ran a six-way alignment of the E. coli, S. typhimurium, V. cholerae, C. jejuni, H. pylori, and C. crescentus SNDB networks.

We used KEGG pathways [48] for our local alignment comparison metrics. To compute each metric, we first removed all proteins in the alignment not assigned to a KEGG pathway and we then removed all local alignments with only one equivalence class. For each local alignment, we defined the closest KEGG as the KEGG pathway that overlapped the local alignment the most. For each KEGG pathway, we defined the closest alignment as the local alignment that overlapped the KEGG pathway the most.

To measure specificity and sensitivity, we computed two metrics:

1. the average, over all local alignments, of the fraction of proteins in the alignment that were also in the closest KEGG (Spec)
2. the average, over all KEGG pathways, of the fraction of proteins in the KEGG pathway that were also in the closest alignment (Sens)

Intuitively, the closest KEGG and closest alignment concepts attempt to find the best match between KEGG pathways and local alignments. Our specificity metric (Spec) measures the degree
Figure 6.7: **Graemlin 2.0 finds more cross-species conservation.** We counted the number of equivalence classes that contained $k$ species for $k = 2, 3, 4, 5, 6$ as described in the text. We compared Graemlin 2.0 (Gr2.0) to Graemlin 1.1 (Gr1.1) and a global aligner (GrG) that used Graemlin 2.0’s alignment search algorithm with Graemlin 1.1’s scoring function. We ran the six-way alignment described in the text.

to which each local alignment contains only proteins that belong to a single KEGG pathway, and our sensitivity metric (Sens) measures the degree to which each KEGG pathway appears in a single local alignment.

We also computed the benchmarks for local alignment used in Chapter 4, which count the number of KEGG pathways overlapped by a local alignment. However, Graemlin 2.0 by nature will produce alignments that overlap more KEGG pathways because it includes every protein in its set of local alignments while the other local aligners include only some proteins in their sets of local alignments. We found that our new sensitivity metric was less biased in favor of Graemlin 2.0.

As we did with the tests of equivalence class accuracy, we used cross validation to test Graemlin 2.0. For each set of networks, we partitioned the KEGG pathways into ten equal sized test sets. For each test set, we trained Graemlin 2.0 on the KEGG pathways outside of test set. We then aligned the networks and computed our metrics on only the KEGG pathways in the test set. Our final numbers for a set of networks were the average of our metrics over the ten test sets. We applied this procedure to all tested aligners to limit biases that might arise from the averaging process.

We also ran tests using various levels of the GO hierarchy in place of KEGG pathways. We omit the results for brevity because they were very similar to those we obtained with KEGG pathways.

We compared Graemlin 2.0 to NetworkBLAST [78], MaWISh [81], and Graemlin 1.1. We used a threshold of 0.9 to obtain Graemlin 2.0’s final set of local alignments from the results of its probabilistic assignment stage. Lower thresholds yielded a sensitivity/specificity trade-off as expected, but we found the loss in specificity to outweigh the increase in sensitivity.
6.4. DISCUSSION

<table>
<thead>
<tr>
<th></th>
<th>SNDB</th>
<th>DIP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>eco/cc</td>
<td>6-way</td>
</tr>
<tr>
<td></td>
<td>Sens</td>
<td>Spec</td>
</tr>
<tr>
<td>NetworkBLAST</td>
<td>.33</td>
<td>.64</td>
</tr>
<tr>
<td>MaWISH</td>
<td>.22</td>
<td>.80</td>
</tr>
<tr>
<td>Græmlin 1.1</td>
<td>.26</td>
<td>.79</td>
</tr>
<tr>
<td>Græmlin 2.0</td>
<td>.33</td>
<td>.78</td>
</tr>
</tbody>
</table>

Table 6.3: **Græmlin 2.0 groups proteins into modules with higher accuracy.** As described in the text, we measured the sensitivity (Sens) and specificity (Spec) of local alignments produced by Græmlin 2.0, NetworkBLAST, MaWISH, and Græmlin 1.1 on the DIP and the Stanford Network Database network datasets. Abbreviations are the same as in Table 6.1.

**Performance comparisons**

Table 6.3 shows that, with respect to the grouping of proteins into modules, Græmlin 2.0 is significantly more sensitive than existing aligners and still maintains high specificity. Græmlin 2.0’s sensitivity increase is due in part to its ability to find weakly conserved and sparsely connected modules, two of which Chapter 7 discusses.

All local aligners ran in under ten minutes, except for NetworkBLAST, which ran in a few hours. Training runs for Græmlin 2.0 took less than five hours, although parameters nearly reached their final values within the first hour.

6.4 Discussion

Græmlin 2.0 is a multiple network aligner with a multi-stage approach to local network alignment, a new feature-based scoring functions for global and local network alignment, an algorithm that automatically learns parameters for the scoring functions, and algorithms that use the scoring functions to approximately align multiple networks in linear time. Græmlin 2.0 has higher accuracy than existing network alignment algorithms across multiple network datasets.

Græmlin 2.0 allows users to easily apply network alignment to their network dataset. In contrast to existing alignment algorithms that require manual recalibration to adjust parameters to different datasets, Græmlin 2.0’s learning algorithm automatically learns parameters specific to any set of networks.

Græmlin 2.0 also extends in principle beyond protein interaction network alignment. As more experimental data gathers and network integration algorithms improve, network datasets with multiple data types will appear [129], such as expression networks with boolean edges [131] and metabolic networks with chemical compounds [130]. With future work to redefine Græmlin 2.0’s feature functions, Græmlin 2.0’s scoring function and parameter learning algorithm will apply to these kinds of networks.
Several future research directions might further improve the performance of Græmlin 2.0. While Græmlin 2.0’s scoring function for global alignment models protein and interaction evolution [3], its scoring function for disjoint local alignment does not model module evolution. Features such as module “cohesion” [117] may improve the local alignment scoring function. In addition, future work can make Græmlin 2.0’s alignment algorithm more sophisticated or can make its parameter learning algorithm explicitly handle an inexact inference algorithm.

6.A Global alignment feature function definition

This section presents precise definitions of Græmlin 2.0’s global alignment feature function and the evolutionary events that the feature function computes. The feature function for local alignment uses the same principles.

We define evolutionary events for possibly ancestral species. We assume that we have \( n \) extant species \( 1, \ldots, n \) and \( m \) ancestral species \( n + 1, \ldots, n + m \), all related by a phylogenetic tree.

Each species \( i \in (1 : n + m) \) is represented by a species weight vector \( s^i \in \mathbb{R}^n \), where \( \sum_{j=1}^{n} s^i_j = 1 \) and \( s^i_j \) represents the similarity of species \( j \in (1 : n) \) to species \( i \). We can use a phylogenetic tree to compute the weight vectors efficiently [118, 91]. Each extant species \( j \in (1 : n) \) has a species weight vector \( (s^j_1 = 0, \ldots, s^j_{j-1} = 0, s^j_j = 1, s^j_{j+1} = 0, \ldots, s^j_n = 0) \).

We denote an equivalence class \([x]\) as a set of proteins \( \bigcup_{i=1}^{n} \Pi_i^{[x]} \), where \( \Pi_i^{[x]} \) is the projection of \([x]\) to species \( i \).

6.A.1 Node feature function

We compute the node feature function \( f^N \) for an equivalence class \([x]\) as follows. First, we compute events for species \( r \) at the phylogenetic tree root node.

**Protein present** We define \( p \in \mathbb{R}^n \) as \( p_i = 1 \) if \( \Pi_i^{[x]} \neq \emptyset \) and 0 otherwise.

- \( f^N_1 = s^r \cdot p \) is the probability that species \( r \) has a protein in \([x]\).
- \( f^N_2 = 1 - s^r \cdot p \) is the probability that species \( r \) does not have a protein in \([x]\).

**Protein count** We define \( c \in \mathbb{R}^n \) as \( c_i = |\Pi_i^{[x]}| \), the number of proteins that species \( i \) has in \([x]\).

- \( f^N_3 = \frac{s^r \cdot c}{s^r \cdot p} \) is the expected number of proteins species \( r \) has in \([x]\), given that \( r \) has a protein.
- \( f^N_4 = (f^N_3)^2 \)

---

\(^1\)In the appendix, the symbols \( n \) and \( m \) have different meanings than in the main text.
The protein present and protein count features describe the most recent common ancestor of the extant species in the equivalence class.

Next, we compute events for all pairs of species \(i, j \in (1: n + m), i \neq j\) adjacent in the tree.

**Protein deletion** We define \(p(k) = \mathbf{s}^k \cdot \mathbf{p}\) as the probability that species \(k\) has a protein in \([x]\).

- \(f_N^5(i, j) = p(i) \times (1 - p(j)) + (1 - p(i)) \times p(j)\) is the probability a protein deletion occurs between species \(i\) and \(j\).

- \(f_N^6(i, j) = p(i) \times p(j)\) is the probability a protein deletion does not occur between species \(i\) and \(j\).

**Protein duplication** We define \(c(k) = \mathbf{s}^k \cdot c\) as the expected numbers of proteins that species \(k\) has in \([x]\).

- \(f_N^7(i, j) = |c(i) - c(j)|\) is the expected number of proteins gained between species \(i\) and \(j\).

**Protein mutation** We define a species pair weight matrix \(S^{ij} \in \mathbb{R}^{n \times n}\) as \(S^{ij}_{kl} = s^i_k s^j_l\). We define \(B \in \mathbb{R}^{n \times n}\) as

\[
B_{kl} = \frac{1}{|\Pi_k||\Pi_l|} \sum_{p \in \Pi_k} \sum_{q \in \Pi_l} b(p, q)
\]

where \(b(p, q)\) is the BLAST bitscore \([89]\) of proteins \(p\) and \(q\). \(B_{kl}\) is the average bitscore among the proteins in species \(k\) and \(l\). \(B_{kl}\) equals 0 if either species \(k\) or \(l\) has no proteins in \([x]\).

- \(f_N^8(i, j) = \text{tr}(S^{ij}B)\), the sum of entry-wise products, is the expected bitscore between the proteins in species \(i\) and \(j\).

- \(f_N^9(i, j) = (f_N^8)^2\)

- \(f_N^{10}(i, j) = (f_N^8)^{-1}\)

- \(f_N^{11}(i, j) = (f_N^8)^{-2}\)

Features \(f_N^9\) through \(f_N^{11}\) allow Græmlin 2.0’s scoring function to include nonlinear dependencies on the BLAST bitscore of the proteins.

Finally, we compute events for all extant species \(i \in (1: n)\).

**Paralog mutation**

- \(f_N^{12}(i) = B_{ii}\) is the expected average bitscore between a protein in species \(i\) and its paralogs.

- \(f_N^{13}(i, j) = (f_N^{12})^2\)

- \(f_N^{14}(i, j) = (f_N^{12})^{-1}\)

- \(f_N^{15}(i, j) = (f_N^{12})^{-2}\)
6.A.2 Edge feature function

We compute the edge feature function $f^E$ for equivalence classes $[x]$ and $[y]$ as follows. First, we compute events for all pairs of species $i, j \in (1 : n + m), i \neq j$ adjacent in the tree.

**Edge deletion** For $k \in (1 : n), p \in \Pi^x_k, q \in \Pi^y_k$, we define $e(k, p, q) = 1$ if there is an interaction between $p$ and $q$ and 0 otherwise. We then define $e \in \mathbb{R}^n$ as

$$e_k = \frac{1}{|\Pi^x_k| |\Pi^y_k|} \sum_{p \in \Pi^x_k} \sum_{q \in \Pi^y_k} e(k, p, q)$$

which represents the average probability that species $k$ has an interaction. We define $e_k$ as NULL if $\Pi^x_k$ or $\Pi^y_k$ is empty. We define

$$e(l) = \left( \frac{1}{\sum_{k : e_k \neq \text{NULL}} e_k} \right) \sum_{k : e_k \neq \text{NULL}} e_k s^l_k$$

which represent the probabilities that species $i$ and $j$ have interactions.

- $f^E_1(i, j) = e(i) \times (1 - e(j)) + (1 - e(i)) \times e(j)$ is the probability that an interaction is lost between species $i$ and $j$.

- $f^E_2(i, j) = e(i) \star e(j)$ is the probability that an interaction is not lost between $i$ and $j$.

Next, we compute events for all extant species $i \in (1 : n)$.

**Paralog edge deletion** We define $\tilde{e}(k, p, q) = 1$, for $k \in (1 : n), p \in \Pi^x_k, q \in \Pi^y_k$ as

$$\tilde{e}(k, p, q) = \frac{1}{|\Pi^x_k| |\Pi^y_k|} \sum_{p' \in \Pi^x_k} \sum_{q' \in \Pi^y_k} e(k, p', q')$$

which represents the probability, ignoring $p$ and $q$, that species $k$ has an interaction.

- $f^E_3(i) = \sum_{p \in \Pi^x_k} \sum_{q \in \Pi^y_k} \left( e(i, p, q) \times (1 - e(i, p, q)) + (1 - e(i, p, q)) \times \tilde{e}(i, p, q) \right)$ is the average probability an interaction is lost between a pair of proteins in species $i$ and all other pairs of proteins in species $i$.

- $f^E_4(i) = \sum_{p \in \Pi^x_k} \sum_{q \in \Pi^y_k} e(i, p, q) \times \tilde{e}(i, p, q)$ is the average probability an interaction is not lost between a pair of proteins in species $i$ and all other pairs of proteins in species $i$. 
For pairwise alignment of two species $s$ and $t$, the final node feature function is

$$f^N([x]) = (f_1^N, f_2^N, f_3^N, f_4^N, \sum_{(i,j)} f_5^N(i,j), \sum_{(i,j)} f_6^N(i,j) \times b, \sum_{(i,j)} f_7^N(i,j), \sum_{(i,j)} f_8^N(i,j) \times b, \sum_{(i,j)} f_9^N(i,j) \times b^2, \sum_{(i,j)} f_{10}^N(i,j) \times b^3, \sum_{(i,j)} f_{11}^N(i,j) \times b^2, \sum_{(i,j)} f_{12}^N(i,j) \times b^3, \sum_{(i,j)} f_{13}^N(i,j) \times b^2, \sum_{(i,j)} f_{14}^N(i,j) \times b^3, \sum_{(i,j)} f_{15}^N(i,j) \times b^3)$$

and the final edge feature function is

$$f^E([x], [y]) = (f_{11}^E(s, t), f_{12}^E(s, t), f_{13}^E(s) + f_{14}^E(t), f_{14}^E(s) + f_{15}^E(t))$$

For multiple alignment, the final node feature function is

$$f^N([x]) = \left(f_1^N, f_2^N, f_3^N, f_4^N, \sum_{(i,j)} f_5^N(i,j), \sum_{(i,j)} f_6^N(i,j) \times b, \sum_{(i,j)} f_7^N(i,j), \sum_{(i,j)} f_8^N(i,j) \times b, \sum_{(i,j)} f_9^N(i,j) \times b^2, \sum_{(i,j)} f_{10}^N(i,j) \times b^3, \sum_{(i,j)} f_{11}^N(i,j) \times b^2, \sum_{(i,j)} f_{12}^N(i,j) \times b^3, \sum_{(i,j)} f_{13}^N(i,j) \times b^2, \sum_{(i,j)} f_{14}^N(i,j) \times b^3, \sum_{(i,j)} f_{15}^N(i,j) \times b^3\right)$$

and the final edge feature function is

$$f^E([x], [y]) = \left(\sum_{(i,j)} f_1^E(i,j), \sum_{(i,j)} f_2^E(i,j) \times b, \sum_{(i,j)} f_3^E(i,j), \sum_{(i,j)} f_4^E(i,j) \times b, \sum_{i=1}^n f_5^E(i), \sum_{i=1}^n f_6^E(i)\right)$$

where the sums over $(i, j)$ are taken over branches of the phylogenetic tree and the sums $i$ are taken over the leaves of the tree.
Chapter 7

Alignment of Known Modules

The goal of network alignment is to analyze functional modules through cross-species comparisons. The benchmarks in the previous chapters suggest that Græmlin and Græmlin 2.0 accurately align known functional modules, but benchmarks only summarize properties of alignments.

To analyze strengths and weaknesses of Græmlin and Græmlin 2.0, this chapter presents alignments of specific functional modules. The alignments illustrate the differences between the aligners discussed in Chapters 4-6. The also highlight several of the challenges faced by network alignment algorithms.

7.1 Alignments by Græmlin and Græmlin 1.1

Two properties make modules difficult to align. First, some modules have missing interactions between proteins in the module due to network noise. Second, some modules contain proteins sequence-similar to numerous proteins other than their functional orthologs. As a result of the first property, the alignment of the module excludes proteins weakly connected to the rest of the module, even if those proteins are a part of the module. As a result of the second property, the alignment of the module aligns proteins that are not functionally orthologous.

This section presents specific examples of Græmlin and Græmlin 1.1 alignments with these two properties. The alignments illustrate the advantage of Græmlin 1.1 over Græmlin and suggest the mechanisms for Græmlin 1.1’s improved accuracy over other network alignment algorithms.

Figure 7.1 shows the alignments discussed in this section. All alignments are of networks in the Stanford Network Database.
### 7.1. ALIGNMENTS BY GRÆMLIN AND GRÆMLIN 1.1

#### 7.1.1 “Gaps” are important in network alignment

Figure 7.1A shows a six-way alignment of KEGG pathway 00540, which represents lipopolysaccharide biosynthesis, in *Escherichia coli* K12, *Salmonella typhimurium* LT2, *Vibrio cholerae*, *Campylobacter jejuni* NCTC 11168, *Helicobacter pylori* 26695, and *Caulobacter crescentus*. The proteins in the alignment belong to three branches of the pathway: *rfaE* and *rfaD* participate in the biosynthesis pathway of ADP-l-glycero-β-d-manno-Heptose [133], *kdtA*, *htrB*, and *msbB* take part in the synthesis of KDO2-lipid A [134], and the remaining proteins are involved in synthesis of the hexose region of the lipopolysaccharide core [135].

While the pathway is present in all six species, not all equivalence classes contain proteins from each species. In some cases equivalence classes lack proteins due to alignment errors, indicated by diagonally shaded place-holders for proteins that should be present in the alignment. In other case, however, the missing proteins represent “pathway holes” [136], which occur when a protein in one species has no clear functional ortholog in another species. For example, no functional ortholog of *rfaE* exists in *V. cholerae*, and no functional ortholog of *msbB* exists in *C. jejuni*, *H. pylori*, and *C. crescentus*.

To find alignments with “pathway holes”, aligners must tolerate protein deletions—the analog of “gaps” in sequence alignment. Both Graemlin and Graemlin 1.1 penalize protein deletions, but only Graemlin 1.1 assigns an accurate weight to the protein deletion penalty relative to its other scoring function parameters. This accurate weight becomes more important as alignments include more species—in alignments of many species, Graemlin’s fixed heuristic deletion penalty over-penalizes alignments with many protein deletions.

In the alignment in Figure 7.1A, Graemlin 1.1 calculates that strong sequence conservation and
CHAPTER 7. ALIGNMENT OF KNOWN MODULES

C. jejuni  E. coli  M. tuberculosis  S. pneumoniae  Synechocystis
C. crescentus  H. pylori  S. typhimurium  S. coelicolor  V. Cholerae
moderate connectivity of the module outweigh the protein deletions. It therefore assigns the alignment a high score; in contrast, Græmlin assigns the alignment a negative score.

7.1.2 Noisy networks make network alignment more difficult

Searches for conserved pathways [63, 78, 2] or conserved protein complexes [77, 78, 81, 2] miss many conserved modules in noisy networks. Even though most modules are highly connected in ideal protein association networks, in practice networks have many missing edges. Therefore, searches for protein complexes will assign alignments overly low scores. Because missing network edges do not follow any set pattern, searches for conserved pathways will also miss many modules.

Because Græmlin 1.1’s training set includes noisy modules, it assigns missing interactions a relatively low penalty. This results in high scoring alignments of even weakly-connected modules, provided that the interactions are stronger than those expected by chance. Figures 7.1A and 7.1B (discussed below) provide two examples of alignments that are only weakly connected but that Græmlin 1.1 nonetheless considers high scoring.

7.1.3 A more lenient edge model can uncover novel connections between modules

Græmlin 1.1’s relatively low missing edge penalty allows it to uncover module relationships missed by other network aligners. Figure 7.1B shows an example of a three-way alignment of *E. coli*, *C. jejuni*, and *H. pylori* in which the proteins involved belong to two broad groups. The *atp* proteins belong to ATP synthase, the well known molecular complex that synthesizes ATP [7], and most of the remaining proteins are involved in cell growth: *gidA* and *gidB* are involved in cell division, *gyrA* and *gyrB* in DNA replication, and *rplO*, *rplB*, *rpoC*, *fusA*, and *tufB* in translation.

It is not surprising that ATP synthase is connected to growth processes. In the Stanford Network Database, however, the linkages between the two broad groups of proteins are rather weak—in fact, Græmlin finds an alignment of the ATP synthase complex but without any connection to any other modules. Because Græmlin 1.1’s scoring function is robust to weak interactions in light of strong conservation of other alignment features, it uncovers the expanded alignment shown in Figure 7.1B.

7.1.4 Strong BLAST hits do not always imply functional orthology

The best sequence match to a protein is not always its functional ortholog [63, 78]. In addition, many proteins have multiple strong BLAST hits [90]—in this case, alignment algorithms must determine which of the strong BLAST hits are functionally orthologous.

The alignment in Figure 7.1C, of the pathway for flagellar assembly, contains many proteins with multiple strong BLAST hits. This alignment, obtained by Græmlin, contains misaligned proteins in almost every equivalence class. The culprits are proteins in *S. typhimurium* involved in the Type III
secretion pathway, which shares significant structural and morphological similarities to the flagellar complex [137] but nonetheless serves a distinct function.

The misalignments occur because the proteins in the Type III system share significant sequence similarity with the proteins in the flagellum—in most cases they have BLAST e-values less than $10^{-50}$. Existing network alignment tools that base scores on e-values [78, 81, 2] therefore assign high positive scores to both the correct and the incorrect alignments.

In contrast, Græmlin 1.1’s strict random model allows it to recognize proteins that share significant similarity but nonetheless lack functionally orthology. For example, while the true $flhA$ orthologs have a BLAST bitscore of 983.02 in *E. coli* and *S. typhimurium*, $flhA$ and $ssaV$ have a bitscore of only 186.04. Because Græmlin 1.1’s random model assigns a high probability to the bitscore of 186.04, Græmlin 1.1 does not align $flhA$ and $ssaV$ even though they have an e-value of $7.6 \times 10^{-51}$. A similar pattern holds for each equivalence class in the alignment, which allows Græmlin 1.1 to omit the incorrect proteins from almost every equivalence class; this is shown by the alignment in Figure 7.1D.

7.1.5 Multiple alignments of many species can become unstable

Proteins with multiple strong BLAST hits case misalignments in multiple alignments to an even greater degree than in pairwise alignments. For example, Græmlin may assign only a small positive score to two proteins with weak sequence-similarity and therefore not align them in a pairwise alignment. In a five-way alignment, however, Græmlin may align five proteins with weak sequence-similarity due to the combined sum of their small positive scores.

Figure 7.1E shows a 10-way alignment by Græmlin with numerous misalignments. The alignment contains proteins from a portion of the fatty acid biosynthesis pathway, and only the equivalence class of $fabH$ has no misalignments. The other equivalence classes contain numerous incorrect proteins, chiefly from *Mycobacterium tuberculosis* H37Rv and *Streptomyces coelicolor*. Most of these misaligned proteins are involved in the synthesis of polyketides, secondary metabolites with a wide range of functions [138]. Polyketide proteins have many similarities to but are distinct from the acyl carrier proteins in the correct alignment, which participate in fatty acid chain elongation [139, 140].

Because Græmlin 1.1 is more specific than Græmlin, it avoids many of the misalignments by Græmlin. As shown by the alignment in Figure 7.1F, Græmlin 1.1 produces a much more accurate alignment of the fatty acid biosynthesis pathway than Græmlin—Græmlin 1.1’s alignment has only one equivalence class with a misaligned protein. The discrepancy between Græmlin and Græmlin 1.1 will only grow as they align more species.

Græmlin 1.1’s alignment of the fatty acid biosynthesis pathway also provides another example of Græmlin 1.1’s ability to find connections between different modules. In addition to the acyl carrier proteins $fabA,fabB,fabD,fabF,fabG$, and $fabH$, the alignment includes several carboxyl transferases ($accA,accB,accC$, and $accD$) that link fatty acid biosynthesis to pyruvate metabolism. The carboxyl
7.2. **Alignments by Græmlin 2.0**

Græmlin 2.0 further improves upon Græmlin 1.1. This section presents examples of conserved modules that Græmlin 2.0 finds but existing aligners, including Græmlin 1.1, miss. The alignments also are examples of the multiple alignment memberships that Græmlin 2.0’s probabilistic assignment stage produces.

Græmlin 2.0’s training set teaches it to tolerate weakly conserved alignments. For example, Figure 7.2A shows an alignment of part of the module for glutathione metabolism in *E. coli* and *C. crescentus*. The alignment is weakly connected and interactions are weakly conserved, which causes aligners like MaWISh and Græmlin that search for only very highly conserved protein complexes to miss it. Græmlin 2.0 finds this alignment because it learns from examples in its training set that many modules are weakly conserved and weakly connected. Furthermore, because it searches for the entire set of local alignments at once, Græmlin 2.0 adds the weakly connected proteins to the alignment when it determines that they belong to no other high scoring alignments.

**Figure 7.2:** **Græmlin 2.0 finds more complete functional modules.** This figure shows two local alignments that Græmlin 2.0 finds. Græmlin 2.0 is more sensitive than existing network aligners, in part because it learns parameters on a training set with weakly conserved and sparsely connected modules. **A.** Græmlin 2.0 tolerates weak connectivity in a local alignment of proteins involved in glutathione metabolism. Networks are from the Stanford Network Database (blue: *E. coli*; green: *C. crescentus*). **B.** Græmlin 2.0 finds a weakly conserved but highly connected local alignment of part of the RNA polymerase complex. Networks are from DIP (red: *S. cerevisiae*; gold: *D. melanogaster*). Transferases have fairly weak linkages to the acyl carrier proteins in the Stanford Network Database, but Græmlin 1.1 is sensitive enough to uncover the connection between them.
Figure 7.3: **Probabilistic assignment expands and clarifies functional modules.** This figure shows two alignments augmented in Græmlin 2.0’s probabilistic assignment stage. Proteins with lower membership probabilities are shaded with fainter colors; black lines show the presence of one or more interactions between a pair of equivalence classes. **A.** Græmlin 2.0 identifies proteins in the chemotaxis module, which are strongly connected to the flagellum module, as members of both the chemotaxis and flagellum modules during the probabilistic assignment stage. **B.** Græmlin 2.0 identifies two connected but distinct pathways for aminoacyl-tRNA biosynthesis and fatty acid biosynthesis during its probabilistic assignment stage.

In noisy and incomplete networks, Græmlin 2.0’s tolerance of weakly conserved modules is more critical. For example, Figure 7.2B shows an alignment of a portion of RNA polymerase in *S. cerevisiae* and *D. melanogaster*. While the alignment has stronger connectivity than the alignment in Figure 7.2A, interaction conservation is still weak—interactions are present predominantly in *S. cerevisiae* and are all but absent in *D. melanogaster*. In addition, several *D. melanogaster* proteins are absent from the alignment (orthologs of P34087 and P41896) or from the network entirely (orthologs of Q06834, P27999, and P20434). Græmlin 2.0 recognizes the alignment as weakly conserved but highly connected because it uses separate scoring functions for global and local alignment.

Græmlin 2.0’s probabilistic assignment algorithm reveals additional members of conserved modules. For example, Figure 7.3A shows an alignment of part of the chemotaxis module in *C. crescentus* and *E. coli*. In the set of disjoint alignments, half of the module is included in an alignment of the flagellum module instead. When Græmlin 2.0 assigns each protein multiple probabilistic memberships, it places the remaining chemotaxis proteins into the alignment of the chemotaxis module. However, the chemotaxis proteins originally included in the flagellum alignment receive a relatively low probability of membership in the chemotaxis alignment, which indicates their relatively strong connection to the flagellum module.

In addition, Græmlin 2.0’s probabilistic assignment algorithm finds connections between different
7.3 DISCUSSION

modules. For example, Figure 7.3B shows parts of the modules for aminoacyl-tRNA biosynthesis and fatty acid biosynthesis. In the original set of disjoint alignments, Græmlin 2.0 places proteins from the two modules into separate alignments. However, the probabilistic assignment stage combines the two modules into one alignment. The proteins in the aminoacyl-tRNA biosynthesis module have a relatively low probability of membership in the fatty acid biosynthesis module, which indicates that the modules are distinct.

7.3 Discussion

Some modules, such as the ribosomal complex shown in Figure 7.4, are well-conserved and highly connected. Most alignment algorithms can easily align such modules with high accuracy. In contrast, many modules are weakly conserved, weakly connected, and contain proteins sequence-similar to many proteins other than their functional orthologs. The main contribution of Græmlin 1.1 and Græmlin 2.0 is the improved ability to align such modules. The alignments in this chapter provide a qualitative analysis of the benefits of Græmlin 1.1 and Græmlin 2.0, to coincide with the quantitative analysis in Chapters 4-6.

Because Græmlin 1.1 is trained on a set of known network alignments, it learns that conserved functional modules can have weak sequence conservation and weak edge connectivity. As a result, it finds high scoring alignments of modules that Græmlin and other network alignment algorithms miss.

Because Græmlin 2.0 is also trained on a set of known network alignments, it has all the advantages of Græmlin 1.1. It is even more sensitive than Græmlin 1.1 for several reasons. First, its scoring function is more principled than Græmlin 1.1’s and does not use a random model, which leads to more accurate alignments. Second, it finds the highest scoring set of local alignments, rather than a set of high scoring local alignments like other algorithms, which leads to an overall more accurate set of local alignments. Finally, it uses a probabilistic assignment algorithm that reveals weak connections between different modules.

While alignments of known modules help to compare and analyze the performance of network alignment algorithms, they do not yield novel biological insights. The next chapter discusses some applications of network alignment that contribute to biological discoveries.
CHAPTER 7. ALIGNMENT OF KNOWN MODULES

Figure 7.4: Well-conserved and highly connected modules are easy to align. Most aligners accurately align the ribosomal complex. Shown is the alignment found by Graemlin 2.0.
Chapter 8

Applications of Network Alignment

Network alignment algorithms must be coupled with network alignment applications. Thus far, however, research into improved network alignment algorithms has outpaced research into novel biological applications of them.

Most applications of network alignment have used the PathBLAST [63] and NetworkBLAST [78] network aligners. The first applications used alignments to find conserved pathways and transfer interaction predictions across species [63, 78]. In addition, alignments have helped to categorize genetic interactions [79], predict protein function [141, 142], and study the transcriptional regulation of protein complexes [143]. An alignment of the human and \textit{Plasmodium falciparum} interaction networks revealed little conservation between the two species despite a significant number of highly-connected modules specific to \textit{P. falciparum} [80].

This chapter describes some biological applications we developed for Graemlin and Graemlin 2.0. All applications in this chapter use alignments of networks in the Stanford Network Database.

8.1 Analysis of proteins and modules

As discussed in Chapters 3 and 4, network alignments have two biological interpretations. First, nodes in the same equivalence class represent functionally orthologous proteins. Second, within a species, the subset of nodes in a local alignment represents a conserved functional module.

We applied these interpretations to alignments produced by Graemlin to generate hypotheses about protein and module function. We first used Graemlin to perform network-to-network alignment of ten species from the Stanford Network Database: \textit{E. coli}, \textit{S. typhimurium}, \textit{V. cholerae}, \textit{C. crescentus}, \textit{C. jejuni}, \textit{H. pylori}, \textit{Synechocystis}, \textit{S. coelicolor}, \textit{M. tuberculosis}, and \textit{S. pneumoniae}. This produced roughly 2000 significant local alignments, each of which contained all or a subset of the 10 species. We selected a handful of alignments that led to obvious biological hypotheses. A complete examination of the set of alignments is a direction for future research.
8.1.1 Functional annotation

Network alignment can help assign roles to proteins of unknown function in two ways. First, annotation transfer assigns to a protein of unknown function the annotation of a protein to which it aligns. This procedure is similar to the traditional method of annotation transfer based on sequence alignment—however, annotation transfer based on network alignment uses conserved interactions as well as conserved sequence. Second, landmark extension assigns to a protein of unknown function the annotation of other “landmark” proteins in a common local network alignment [144]. More highly connected and highly conserved alignments strengthen the hypothesis that the unknown protein shares functionality with the landmark protein.

Figure 8.1 shows an example of functional annotation suggested by pairwise and multiple network alignments. The alignments include the components of the primosome (\textit{dnaB}, \textit{dnaA}, \textit{gyrA}, \textit{gyrB}), the subunits of topoisomerase IV (\textit{parE}, \textit{parC}), and the \(\beta\) subunit of DNA polymerase III (\textit{dnaN}). These protein families are all known to be involved in DNA replication. Therefore, it is likely that the other proteins in the alignment also play a role in DNA replication.

Namely, the alignment hypothesizes that \textit{recF}, \textit{gidA}, \textit{gidB}, and \textit{trmE} are involved in DNA replication. Even though none of these proteins has DNA replication as a primary annotation, past studies support the hypotheses made by the alignment. First, a recent study found a link between \textit{recF} and DNA replication [145]. In addition, transcription of \textit{gidA} affects DNA replication, both \textit{gidA} and \textit{trmE} are involved in tRNA modification, and \textit{trmE} has been implicated in cell cycle control [146]. These past studies, together with the alignment, suggest that both the \textit{gid} proteins and \textit{trmE} play a role in the cell cycle regulated control of DNA replication.

The nine-way multiple alignment in Figure 8.1B strengthens the hypotheses of the pairwise alignment and suggests additional protein annotations. In particular, the presence of the \textit{trmE} protein in all nine species provides a compelling argument in favor of its role in DNA replication. In addition, the 60 kD inner membrane protein \textit{yidC} is present in all nine species and is highly connected to the other proteins in the alignment. While \textit{yidC} is involved in protein secretion, the multiple alignment hypothesizes that it is also linked to DNA replication.

The 10-way alignment in Figure 8.2 suggests annotations for several more proteins. The alignment includes \textit{ftsZ}, \textit{ftsW}, and \textit{ftsI}, well-known proteins involved in cell division, along with a number of other proteins from the \textit{mur} and \textit{mra} families that play a role in peptidoglycan biogenesis. Many of the \textit{mur} and \textit{mra} proteins are in contiguous operons in some species [147] but are scattered over the genome in species such as \textit{C. jejuni} and \textit{H. pylori}, which makes it difficult to determine their function. The alignment, however, implicates them in cell division by association with the landmark proteins \textit{ftsZ}, \textit{ftsW}, and \textit{ftsI}. 
8.1. ANALYSIS OF PROTEINS AND MODULES

Figure 8.1: A network alignment predicts proteins involved in DNA replication. In this figure, the boxes and colors have the same meaning as in Figure 7.1. To avoid clutter, individual proteins are not labeled—as an example of the set of proteins aligned in an equivalence class, the detailed inset shows the specific proteins aligned to gyrB. In this figure, equivalence classes in the multiple alignment are highlighted the same color as the pairwise equivalence classes that they subsume. A. A pairwise alignment between E. coli and C. crescentus includes several proteins involved in DNA replication. B. A multiple alignment extends the pairwise alignment to include S. typhimurium, V. cholerae, C. jejuni, H. pylori, M. tuberculosis, S. pneumoniae, and Synechocystis.
Figure 8.2: **A network alignment predicts proteins involved in cell division.** The presence of proteins involved in cell division suggests that the other proteins in the alignment also play a role in cell division.
8.2 DISCOVERY OF CRYPTIC MODULES

8.1.2 Module identification

Local network alignments also suggest putative functional modules. Alignments that include proteins of related but different functions suggest that the proteins cooperate in a higher-level biological process.

For example, the alignment in Figure 8.3 suggests part of a putative functional module for DNA uptake [148]. The alignment includes six species (E. coli, S. typhimurium, V. cholerae, C. jejuni, H. pylori, and C. crescentus), and hypothesizes that several proteins from the exb/tol family of biopolymer transporters interact with a set of proteins involved in DNA recombination and integration. While these proteins only weakly interact in any one species, their combined interactions in six distinct species suggest a significant relationship among them. In particular, the interactions suggest that the module may transport DNA through the tol channels and use the recombination proteins to integrate it into the chromosome.

Previous biological experiments strengthen the hypothesis that the alignment in Figure 8.3 represents part of a module for DNA uptake. In particular, disruption of the exbB protein in Pseudomonas stutzeri reduced transformation efficiency to one fifth of its previous level [149]. While this study in P. stutzeri identified the exb genes due to their location immediately downstream of two competence related proteins, in species such as C. jejuni and H. pylori this chromosomal contiguity is not evident. Network alignment identifies the module due to conserved interactions.

8.2 Discovery of cryptic modules

Many of the alignments found by Graemlin and Graemlin 2.0 do not correspond to modules with obvious function. These “cryptic” modules often contain many proteins of unknown function or proteins with very distinct functions. While there are many proteins of unknown function within a given species, a network alignment that groups a collection of them and indicates conservation of the entire collection suggests potentially profitable targets for experimental studies.

For example, the alignment of six α-proteobacteria in Figure 8.4A suggests several experimental targets in C. crescentus (shown in green). In particular, while dnaJ, gst4, pheT, and dnaN have putative functions, the remaining equivalence classes contain only hypothetical proteins. The high degree of connectivity between the six hypothetical C. crescentus proteins in Figure 8.4A, as well as their conservation in five other organisms, suggests that the set as a whole may be of particular interest. Indeed, as Figure 8.4B shows, the alignment has a high degree of conservation in twenty α-proteobacteria.

The alignments in Figure 8.5 contain another collection of well-conserved and highly connected hypothetical proteins. The alignments are not as strongly conserved as those in Figure 8.4 but are still notable targets for experimental studies.
Figure 8.3: **Network alignments suggest putative functional modules.** In this alignment, proteins involved in biopolymer transport interact with proteins involved in DNA recombination. The sum total of these interactions in six species suggests that the proteins may be a part of a conserved functional module responsible for transformation.
8.2. DISCOVERY OF CRYPTIC MODULES

Figure 8.4: **Cryptic modules are potential targets for experimental studies.** A set of six equivalence classes that contain only hypothetical proteins constitute the majority of this alignment. Although most microbes contain many hypothetical proteins, this set’s high conservation and high connectivity suggest that it may be a profitable target for experimental studies. **A.** The six-way alignment includes six α-proteobacteria. **B.** The module is conserved in many more species, as indicated by a twenty-way alignment. Each species is represented by a different color (list of species and corresponding colors not shown).
Figure 8.5: Graemlin finds many cryptic modules. This alignment is another example of a highly conserved alignment produced by Graemlin that contains multiple proteins of unknown function.
8.3 Analysis of inositol metabolism

Network alignment can also aid studies of known functional modules. This section describes an alignment of a module for myo-inositol metabolism in *Caulobacter crescentus*. In particular, the alignment offers insights into the module’s regulation and evolutionary conservation.

8.3.1 Background and motivation

myo-inositol, or cis-1,2,3,5-trans-4,6-cyclohexanediol, is a cyclic polyol that can serve as a carbon and energy source for many bacterial species [150, 151, 152, 153, 154]. While the *C. crescentus* genome contains proteins for the metabolism of inositol to acetyl-CoA and CO$_2$, standard sequence-based homology searches do not identify transporter and regulatory proteins involved in the process.

Forward and reverse genetic experiments in *C. crescentus* suggested genes responsible for myo-inositol degradation, transport, and regulation [4]. The collection of these genes, as well as the proteins previously known to be involved in myo-inositol metabolism, constitute a putative functional module. The module is responsible for the transport and metabolism of myo-inositol, as well as the regulation of the entire process.

To further analyze the module for myo-inositol metabolism in *C. crescentus*, we aligned it to five other α-proteobacteria. Our alignment showed that the module had high conservation, strengthened the hypothesis that the module is under common regulatory control, and predicted a transporter protein in *Sinorhizobium meliloti*. The function of this *S. meliloti* protein as an inositol-specific transporter does not follow from sequence comparison to the *C. crescentus* transporter operon alone—there are multiple proteins in the *S. meliloti* genome with nearly identical BLAST scores to the *C. crescentus* myo-inositol transporter.

8.3.2 The myo-inositol module in *C. crescentus*

Figure 8.6A shows a representation of the myo-inositol module in *C. crescentus*. The top operon contains proteins involved in myo-inositol metabolism and a protein (*iolR*) that controls expression of the entire module. The bottom operon contains proteins responsible for myo-inositol transport.

As shown in Figure 8.6B, the *C. crescentus* network in the Stanford Network Database indicates strong associations among proteins within each operon. In addition, it indicates strong association between the two operons, even though the operons are far apart on the *C. crescentus* genome. Therefore, the Stanford Network Database supports the hypothesis that the two operons in Figure 8.6A function as part of a single module.

Because experiments suggested that *iolR* represses transcription of the module, we looked for regulatory motifs in the promoter regions of the two operons. A MEME search [155] suggested a consensus palindromic motif (Figure 8.6C) upstream of *ibpA*, *idhA*, *iolC* and *iolA*. Subsequent experiments [4] showed that mutagenesis of the *iolC* motif repressed promoter activity, which suggested
The inositol module is composed of two disjoint loci

Loci were predicted to associate computationally

Multiple network alignment reveals conservation across alpha-proteobacteria

Upstream regions of module genes contain a putative motif

Network level conservation improves motif identification

Refined motif: E-value = 4.3e-75

that the motif is an IolR regulatory site.

### 8.3.3 Alignment of the myo-inositol module to other α-proteobacteria

To identify homologs of the *C. crescentus* myo-inositol module in other species, we used Græmlin to align the module to five closely related α-proteobacteria. In several species, the alignment contained several candidate functional orthologs for each of the proteins in the transporter operon. To improve the quality of the alignment, we manually refined the alignment to keep only the best candidates in each species. First, in each species, we considered only candidates that were contiguous on the chromosome; this resulted in several candidate conserved operons in each species. Second, in each species, we assessed the similarity of each candidate operon to all other species in the alignment—we computed, for each protein in the candidate operon, the average BLAST score to its counterpart in all other species, and we kept in the alignment only the candidate operon with the highest sum of these averages.

Figure 8.6C shows the final alignment of the myo-inositol module. The alignment indicates significant conservation of proteins as well as interactions in each species. In particular, the prediction of association between the transporter and catabolic operons is strengthened by strong evidence of association in *Mesorhizobium loti* and *S. meliloti*.

We reasoned that the strong conservation of genes within the module should imply conservation of the mechanism for the module’s regulation. In particular, we predicted that the palindromic motif found in *C. crescentus* should be conserved in the other species in the alignment. A MEME search of the promoter regions in all six species uncovered many more examples of this same palindromic motif, which confirmed our hypothesis. As Figure 8.6D shows, the additional species significantly improve the significance score for the motif.

*Bradyrhizobium japonicum* is the only one of the six species in the multiple network alignment that contains no motif homologs. Importantly, *B. japonicum* also lacks the regulatory protein IolR in the network alignment. This suggests that myo-inositol uptake and catabolism in *B. japonicum* employs a different regulatory strategy.
8.3.4 Prediction of myo-inositol transporter in *S. meliloti*

We sought to predict the operon in *S. meliloti* responsible for myo-inositol transport. We first used BLAST to search *S. meliloti* for homologs to the IbpA protein, which experiments suggested as the inositol-binding protein in the *C. crescentus* transporter operon [4]. However, the BLAST search uncovered periplasmic binding proteins from five different ABC transporter operons in *S. meliloti* with similar scores. Thus, simple pairwise comparisons with the known myo-inositol transporter in *C. crescentus* did not distinguish the inositol transport system in *S. meliloti*.

One of the five *S. meliloti* operons with homology to the *C. crescentus* operon is Smb20712-4. This operon had previously been annotated as a putative rhizopine transporter, based on homology to the known MocB rhizopine transporter in another strain of *S. meliloti* [156]. Because rhizopine is derived from myo-inositol, this homology to MocB suggests that Smb20712-4 may be a predictor of myo-inositol transport. However, the periplasmic binding protein within the Smb20712-4 operon is not the highest scoring BLAST hit to the *C. crescentus* IbpA protein.

In contrast, our network alignment strongly predicted that the genes Smb20712-4 encoded the myo-inositol transporter in *S. meliloti*. Importantly, while other operons in *S. meliloti* showed higher overall conservation with *C. crescentus*, the Smb20712-4 operon clearly showed the highest conservation to the other 4 species.

Subsequent experiments confirmed our prediction. *Sinorhizobium meliloti* mutants with transposon insertions in the predicted periplasmic binding protein could not grow on medium with myo-inositol as a sole carbon source, even though they grew normally on medium with glucose. Therefore, cross-species comparison at a network level predicted true functional orthology, even when traditional sequence based comparisons failed to do so.

8.4 Discussion

Thus far, research into network alignment algorithms has outpaced research into network alignment applications. Nonetheless, network alignments have already aided biological studies of protein and module function.

One application of network alignment is the analysis of proteins that lack functional annotation. Network alignment can use conserved interactions as well as sequence to align a protein of unknown function in one species to a protein of known function in another species. Alternatively, even if a protein has no functional ortholog of known function, its occurrence as part of an alignment near well-known “landmark” proteins permits inferences about its function [144].

In addition, network alignment can identify groups of proteins that interact with one another and are conserved across multiple species. In many cases this suggests novel functional modules that may play a role in important biological processes.

As network alignment algorithms, and the interaction networks they analyze, proliferate, they
will increasingly help to direct biological experiments. The alignment of the *Caulobacter crescentus*
myo-inositol module is one such example of this. Our alignment of this module suggests that, in
many cases, network level conservation provides more information than does sequence conservation
alone.
Chapter 9

Conclusion

Network alignment holds much promise as a field of study. It has the potential to aid many studies of high-level biological processes, a key goal of the emergent field of systems biology [5]. Important processes are likely to exhibit conservation not only at the level of individual molecules but also at the network level. As network alignment algorithms generalize to analyze complex networks beyond those that represent only proteins, the role of network comparisons in systems biology may approach the role of sequence comparisons in genomics.

Despite this promise, the field of network alignment is still in its infancy and needs further research. This thesis has described several contributions I have made to the field of network alignment.

1. **A novel mathematical formulation of network alignment.** Our equivalence class formulation of network alignment assigns concrete biological hypotheses to each network alignment. Other network alignment algorithms either represent network alignments as many-to-many mappings, which result in complex alignments that are hard to interpret, or one-to-one mappings, which do not model the reality of multiple proteins within an organism that perform the same function. In addition, the equivalence class formulation allows databases of functional modules, such as KEGG [48] and GO [50], to be encoded as network alignments—a requirement for network alignment algorithms to use these databases as training data.

2. **The first large-scale multiple network aligner.** Græmlin uses several heuristics inspired by sequence alignment algorithms to efficiently and accurately align multiple large protein interaction networks. Its use of $d$-clusters allows for a speed-sensitivity trade-off and facilitates searches of large network databases, analogous BLAST’s use of k-mers [90]. Its use of progressive multiple alignment allows for efficient alignments of many networks, in contrast to other multiple network aligners that scale exponentially in the number of species aligned [78].

3. **The first numerical benchmarks that quantify network alignment performance.** We conducted detailed benchmarks that compare Græmlin, Græmlin 1.1, and Græmlin 2.0 to
other network alignment algorithms. Prior to this work, no comparison of different algorithms existed. Quantitative comparisons are necessary to evaluate if novel algorithms truly improve over existing algorithms and also to suggest how to improve network alignment algorithms in the future.

4. **The first network alignment scoring function trained on a collection of known network alignments.** Graemlin 1.1 introduces the first network aligner to incorporate training data of known functional modules. This focus is distinct from that of most network alignment research, which is the design of ever more sophisticated search algorithms. Our results in Chapter 6 argue that this focus is profitable: the combination of a simple search algorithm with a sophisticated scoring function trained on real biological data results in network alignments more accurate than those found by other network alignment algorithms.

5. **The most accurate network aligner to date.** Graemlin 2.0 introduces several ideas that enable it to find accurate network alignments. Its multi-stage formulation of local alignment avoids many problems encountered by other local aligners, such as the need for a random model and the conflation of the search for conservation with the search for functional modules. Graemlin 2.0 automatically learns parameters specific to any network dataset, which allows it to achieve high accuracy on datasets with different noise levels and methods for interaction determination.

6. **Applications of network alignment to protein function prediction and module analysis.** We have used Graemlin to suggest functions for proteins and entire modules, some of which the available literature partially confirms. In addition, we have used network alignment to help analyze the module for myo-inositol in *Caulobacter crescentus*. Our alignment of the module increased the statistical evidence for a regulatory motif, suggested an alternate regulatory strategy in *Bradyrhizobium japonicum*, and predicted the true ortholog of the myo-inositol binding transporter protein in *Sinorhizobium meliloti* despite the presence of multiple sequence-similar potential transporters.

### 9.1 Open problems

There is no shortage of network alignment research topics. Current network aligners are designed to analyze protein interaction networks, which, while more complex than biosequences, are still relatively simple. As datasets grow that contain more complex relationships between biological processes, network alignment algorithms must improve as well.

One key challenge will be to adapt network alignment algorithms to compare networks with multiple node and edge types. Some networks already contain edges that represent different boolean relationships between proteins [131] or nodes that represent chemical compounds as well as proteins...
[130]. Græmlin 2.0’s scoring function provides a foundation for the alignment of such networks: with additional features in its feature function that summarize the extra information contained in rich-featured networks, Græmlin 2.0’s parameter learning and search algorithms can still be used.

In the future, it will also be important to search for complex functional modules. Existing aligners only search for linear pathways or highly connected protein complexes. Future algorithms should search for modules that have complex topologies or fit partially defined schemas [157], which specify properties that certain proteins in the module should have. A key challenge will be the design of search algorithms to efficiently find such complex modules.

In addition, while query-to-network alignments directly help to study specific functional modules, network-to-network alignment typically produces thousands of high scoring alignments. Methods to filter through these alignments to identify those with novel and testable biological hypotheses are needed. Otherwise, network alignment tools will not help to make sense of the growing collection of complex high-throughput biological data but will instead generate more of it.

### 9.2 Future directions

In the future, network alignment will play an increasing role in the organization, annotation, and study of functional modules. Once databases of known modules become commonplace, network alignment algorithms will help to annotate hypothetical modules through matches to a database, as BLAST [89] does for proteins. In addition, multiple network alignment offers the potential to study evolutionary properties of modules. Further algorithmic development may lead to data-motivated population genetic models for network evolution [158, 81], where conserved interactions and conserved proteins play the role that conserved residues play in models of protein evolution. It is possible that a SCOP-like hierarchy [159] for module families is on the horizon.

However, with increasingly complex networks on the horizon [129], future applications of network alignment may be far removed from the traditional problem of functional module comparison. For example, networks specific to individual eukaryotic cell types, rather than individual species, offer the chance to investigate network level properties of disease states and specific tissues—in this case, differences, rather than similarities, between the networks will aid experimental studies.

Much of the information behind complex biological processes can naturally be expressed in networks. For example, histone modifications [16], DNA methylation [15], and non-coding RNAs [160] all play a role in epigenetic inheritance [9]. Analysis of networks that express relationships between these different phenomena may reveal cross-talk between different pathways that is only apparent at the network level.

Much work remains to realize applications such as these. Improved algorithms will advance the field of network alignment only so far. Ultimately, its impact will depend on the development of novel and important biological applications. Only then will network alignment fulfill its promise.
Bibliography


