

Automatic Parameter Learning for Multiple Network Alignment

Jason Flannick¹, Antal Novak¹, Chuong B. Do¹, Balaji S. Srinivasan², and Serafim Batzoglou¹

¹ Department of Computer Science, Stanford University, Stanford, CA 94305, USA

² Department of Statistics, Stanford University, Stanford, CA 94305, USA

Abstract

We developed Græmlin 2.0, a new multiple network aligner with (1) a novel scoring function; (2) an algorithm that automatically learns the scoring function's parameters; and (3) an algorithm that uses the scoring function to globally align multiple networks. Existing alignment tools use heuristic scoring functions, which must be hand-tuned to a given set of networks and do not apply to multiple network alignment.

Our scoring function can use arbitrary features of a multiple network alignment, such as protein deletions, protein duplications, protein mutations, and interaction losses. Our parameter learning algorithm uses a training set of known network alignments to learn parameters for our scoring function and thereby automatically adapts it to any set of networks. Our global alignment algorithm finds approximate multiple network alignments in linear time.

We tested Græmlin 2.0's accuracy on protein interaction networks from IntAct, DIP, and the Stanford Network Database. We show that, on each of these datasets, Græmlin 2.0 has higher sensitivity and specificity than existing network aligners. Græmlin 2.0 is available under the GNU public license at <http://graemlin.stanford.edu>.

1 Introduction

This paper describes Græmlin 2.0, a multiple network aligner with 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 by automatically adapting alignment algorithms to any network dataset.

Network alignment compares interaction networks of different species [1]. An interaction network contains nodes, which represent genes, proteins, or other molecules, as well as edges between nodes, which represent interactions. By comparing networks, network alignment finds conserved biological modules or pathways [2, 3]. Because conserved modules are usually functionally important, network alignment research growth [1] has paralleled interaction network dataset growth [4, 5].

Network alignment algorithms use a scoring function and a search algorithm. The scoring function assigns a numerical value to network alignments—high values indicate conservation. The search algorithm searches the set of possible network alignments for the highest scoring network alignment.

Most network alignment research has focused on pairwise network alignment search algorithms. PathBLAST uses a randomized dynamic programming algorithm to find conserved pathways [6] and uses a greedy algorithm to find conserved protein complexes [7]. MaWISH formulates network alignment as a maximum weight induced subgraph problem [8]. MetaPathwayHunter uses a graph matching algorithm to find inexact matches to a query pathway in a network database [9], and QNet exactly aligns query networks with bounded tree width [10]. Other network alignment algorithms use ideas behind Google’s PageRank algorithm [11] or cast network alignment as an Integer Quadratic Programming problem [12].

Two network aligners can perform multiple network alignment. NetworkBLAST extends PathBLAST to align three species simultaneously [13]. Græmlin 1.0 can align more than 10 species at once [14].

Scoring function research has focused on models of network evolution. Koyuturk et. al. [8] model protein evolution with a duplication-divergence model. Hirsh et. al. [15] model protein complex evolution with interaction gains and losses as well as protein duplications. Berg et. al. [16] model network evolution with interaction gains and losses as well as protein sequence divergences.

Despite these advances, scoring functions still have several limitations. First, existing 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.

Second, existing scoring functions use only sequence similarity, interaction conservation, and protein duplications to compute scores. As scoring functions use additional features such as protein deletions and paralog interaction conservation, 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 with no evolutionary basis (Græmlin 1.0).

In this paper, we first present a scoring function that addresses these limitations. We next present an algorithm that uses a training set of known alignments to automatically learn parameters for our scoring function. We then present an algorithm that uses our scoring function to perform approximate global network alignment in linear time. Finally, we present benchmarks of Græmlin 2.0, a new multiple network aligner that includes these three pieces, against existing network aligners.

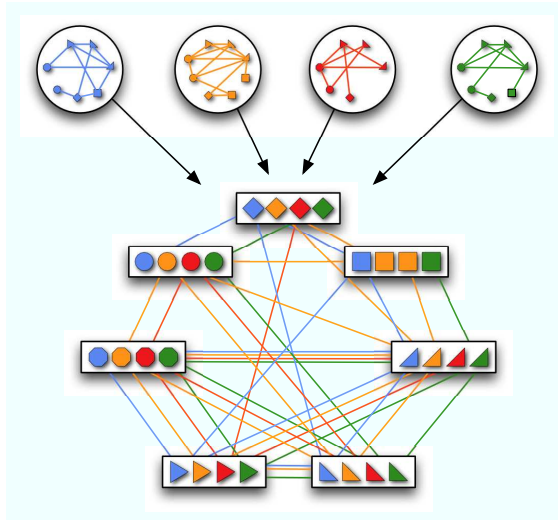


Figure 1: **A network alignment is an equivalence relation.** In this example, four protein interaction networks are input to multiple alignment. A network alignment partitions proteins into equivalence classes (indicated by boxes).

2 Methods

2.1 Network alignment formulation

The input to multiple network alignment is n networks G_1, \dots, G_n . Each network represents a different species and contains a set of nodes V_i and a set of edges E_i linking pairs of nodes. One common type of network is a protein interaction network, in which nodes represent proteins and edges represent interactions, either direct or indirect, between proteins.

A *multiple network alignment* is an equivalence relation a over the nodes $V = V_1 \cup \dots \cup V_n$. An equivalence relation is transitive and partitions V into a set of disjoint equivalence classes. A *local alignment* is a relation over a subset of the nodes in V ; a *global alignment* [11] is a relation over all nodes in V . Figure 1 shows an example of an alignment of four protein interaction networks.

Network alignments have a biological interpretation. Nodes in the same equivalence class are functionally orthologous [19]. The subset of nodes in a local alignment represents a conserved module [2] or pathway.

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, \dots, G_n . The *global network alignment problem* is to find the maximum score global network alignment. The *local network alignment problem* is to find a set of maximal score local network alignments.

In this paper, we restrict attention to global network alignment. Many ideas that apply to global network alignment also apply to local alignment. In addition, a local alignment algorithm can use global network alignment as a first step and then segment the global alignment into a set of local alignments [6, 7].

2.2 Scoring function

General definition Our scoring function computes “features” [20, 21] of a network alignment. Formally, we define a vector-valued *feature function* $\mathbf{f} : \mathcal{A} \rightarrow \mathbb{R}^n$, which maps a global alignment

to a numerical *feature vector*. More specifically, we define a node feature function \mathbf{f}^N that maps equivalence classes to a feature vector and an edge feature function \mathbf{f}^E that maps pairs of equivalence classes to a feature vector. We then define

$$\mathbf{f}(a) = \left[\begin{array}{c} \sum_{[x] \in a} \mathbf{f}^N([x]) \\ \sum_{\substack{[x],[y] \in a \\ [x] \neq [y]}} \mathbf{f}^E([x],[y]) \end{array} \right] \quad (1)$$

with the first sum over all equivalence classes in the alignment a and the second sum over all pairs of equivalence classes in a .

Given a numerical *parameter vector* \mathbf{w} , the score of an alignment a is $s(a) = \mathbf{w} \cdot \mathbf{f}(a)$. The *parameter learning problem* is to determine \mathbf{w} . We discuss our parameter learning algorithm below.

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

Implementation for protein interaction networks We implemented a feature function for protein interaction network alignment that computes evolutionary events. We first describe our feature function for the special case of pairwise network alignment (the alignment of two networks), and we then generalize our feature function to multiple network alignment.

Our pairwise node feature function computes the following seven evolutionary events:

- *Protein present* is the maintenance of proteins in both species.
- *Protein count* is the maintenance of more than one protein in both species.
- *Protein deletion* is the loss of a protein in one of the two species.
- *Protein duplication* is the duplication of a protein in one of the two species.
- *Protein mutation* is the divergence in sequence of two proteins in different species.
- *Paralog mutation* is the divergence in sequence of two proteins in the same species.

Our pairwise edge feature function computes the following two evolutionary events:

- *Edge deletion* is the loss of an interaction between two pairs of proteins in different species.
- *Paralog edge deletion* is the loss of an interaction between two pairs of proteins in the same species.

The values of the events are the entries in the feature vector.

We take two steps to generalize these pairwise feature functions to multiple network alignment. First, we use a phylogenetic tree to relate species and then sum pairwise feature functions over pairs of species adjacent in the tree, including ancestral species. Second, we modify the feature functions to include evolutionary distance.

Our pairwise feature functions generalize to ancestral species pairs. We first compute species weight vectors [22] for each ancestral species. Each species weight vector contains numerical weights that represent the similarity of each extant species to the ancestral species. We use these species

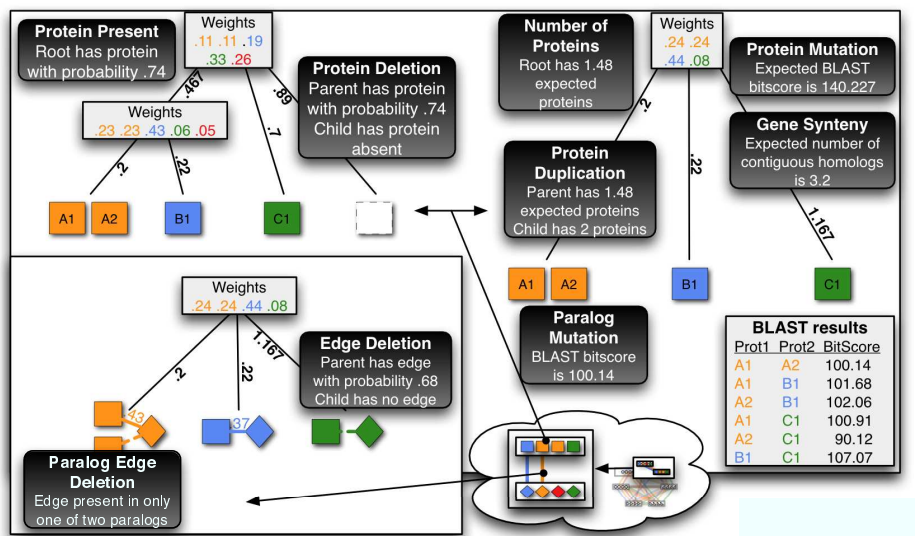


Figure 2: **Alignment feature functions compute evolutionary events.** This figure shows the set of evolutionary events that our node and edge feature functions compute. We use a phylogenetic tree with branch lengths to determine the events. We first construct species weight vectors (shown as gray boxes) at each internal node of the tree; the weight vector represents the similarity of each extant species to the internal node. We use these weight vectors to compute the likely evolutionary events (shown as black boxes) that occur. The appendix gives precise definitions of the evolutionary events.

weight vectors, together with the proteins in the equivalence class, to approximate the ancestral proteins in the equivalence class. We then compute pairwise feature functions between the approximate ancestral proteins. The appendix describes the exact procedure.

In addition, our pairwise feature functions generalize to include evolutionary distance. We replace each f_i with a set $\{f_i, f_i \times b, \dots, f_i \times b^k\}$, where b is the distance between the species pair. We used $k = 1$ for most events.

Figure 2 shows the evolutionary events our feature function computes. The appendix contains precise definitions of our feature function, as well as precise definitions of all evolutionary events.

2.3 Parameter learning algorithm

Inputs Our algorithm to determine \mathbf{w} requires a *training set* of known alignments. The training set is a collection of d *training samples*; each training sample specifies a set of networks $G^{(i)} = G_1^{(i)}, \dots, G_n^{(i)}$ and their correct alignment $a^{(i)}$.

Our learning algorithm requires a *loss function* $\mathcal{L} : \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{R}^+$. By definition, $\mathcal{L}(a^{(i)}, a)$ must be 0 when $a = a^{(i)}$ and positive when $a \neq a^{(i)}$ [23]. Intuitively, $\mathcal{L}(a^{(i)}, a)$ measures the distance of an alignment a from the training alignment $a^{(i)}$; the learned parameter vector should therefore assign higher scores to alignments with smaller loss function values.

To train parameters for our feature function, we used a training set of KEGG Ortholog (KO) groups [18]. Each training sample contained the networks from a set of species, with nodes removed that did not have a KO group. The correct alignment contained an equivalence class for each KO group.

We also defined a loss function that grows as alignments diverge from the correct alignment $a^{(i)}$. More specifically, let $[x]_{a^{(i)}}$ denote the equivalence class of $x \in V^{(i)} = \bigcup_j V_j^{(i)}$ in $a^{(i)}$ and $[x]_a$ denote the equivalence class of x in a . We define $\mathcal{L}(a^{(i)}, a) = \sum_{x \in V^{(i)}} |[x]_a \setminus [x]_{a^{(i)}}|$, where $A \setminus B$ denotes the set difference between A and B . This loss function is the number of nodes aligned in a that are not aligned in the correct alignment $a^{(i)}$.

Theory We pose parameter learning as maximum margin structured learning problem. Given a training set and loss function, the learned \mathbf{w} should score each training alignment $a^{(i)}$ higher than all other alignments a by at least $\mathcal{L}(a^{(i)}, a)$. Mathematically, this constraint is

$$\forall i, a \in \mathcal{A}^{(i)}, \mathbf{w} \cdot \mathbf{f}(a^{(i)}) \geq \mathbf{w} \cdot \mathbf{f}(a) + \mathcal{L}(a^{(i)}, a)$$

where $\mathcal{A}^{(i)}$ is the set of possible alignments of $G^{(i)}$.

We can express this constraint [23] as the following convex program [24]:

$$\begin{aligned} \min_{\mathbf{w}, \zeta^{(i)}} \quad & \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{d} \sum_{i=1}^d \zeta^{(i)} \\ \text{s.t.} \quad & \forall i, a \in \mathcal{A}^{(i)}, \mathbf{w} \cdot \mathbf{f}(a^{(i)}) + \zeta_i \geq \mathbf{w} \cdot \mathbf{f}(a) + \mathcal{L}(a^{(i)}, a) \end{aligned}$$

where λ is a regularization term that prevents overfitting. We set $\lambda = 0$; its effect is a future research topic.

We can rearrange the convex program [23] to show that the optimal \mathbf{w} minimizes

$$c(\mathbf{w}) = \frac{1}{d} \sum_{i=1}^d r^{(i)}(\mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \quad (2)$$

where $r^{(i)}(\mathbf{w}) = \max_{a \in \mathcal{A}^{(i)}} (\mathbf{w} \cdot \mathbf{f}(a) + \mathcal{L}(a^{(i)}, a)) - \mathbf{w} \cdot \mathbf{f}(a^{(i)})$.

This objective function is convex but nondifferentiable [23]. We can therefore minimize it with subgradient descent [25], an extension of gradient descent to nondifferentiable objective functions.

The subgradient of equation (2) is [23]

$$\lambda \mathbf{w} + \frac{1}{d} \sum_{i=1}^d (\mathbf{f}(a_*^{(i)}) - \mathbf{f}(a^{(i)}))$$

where $a_*^{(i)} = \arg \max_{a \in \mathcal{A}^{(i)}} \mathbf{w} \cdot \mathbf{f}(a) + \mathcal{L}(a^{(i)}, a)$ is the optimal alignment, determined by the loss function $\mathcal{L}(a^{(i)}, a)$ and current \mathbf{w} , of $G^{(i)}$.

Algorithm Based on these ideas, our learning algorithm performs subgradient descent. It starts with $\mathbf{w} = 0$. Then, it iteratively computes the subgradient \mathbf{g} of equation (2) at the current parameter vector \mathbf{w} and updates $\mathbf{w} \leftarrow \mathbf{w} - \alpha \mathbf{g}$, where α is the *learning rate*. The algorithm stops when it performs 100 iterations that do not reduce the objective function. We set the learning rate to a small constant ($\alpha = 0.05$).

The algorithm for finding $\arg \max_{a \in \mathcal{A}^{(i)}} \mathbf{w} \cdot \mathbf{f}(a) + \mathcal{L}(a^{(i)}, a)$ is the inference algorithm. It is a global alignment algorithm with a scoring function augmented by \mathcal{L} . Below we present an efficient approximate global alignment algorithm that doubles as our inference algorithm.

Our learning algorithm has an intuitive interpretation. At each iteration it uses the loss function \mathcal{L} and the current \mathbf{w} to compute the optimal alignment. It then decreases the score of features

```

LEARN( $\{G_1^{(i)}, \dots, G_n^{(i)}, a^{(i)}\}_{i=1}^d$  : training set ,  $\alpha$  : learning rate ,  $\lambda$  : regularization )
1  var  $\mathbf{w} \leftarrow \mathbf{0}$  //the current parameter vector
2  var  $m_* \leftarrow \infty$  //a measure of progress
3  var  $\mathbf{w}_* \leftarrow \mathbf{w}$  //the best parameter vector so far
4  while  $m_*$  updated in last 100 iterations
5  do
6    var  $\mathbf{g} \leftarrow \mathbf{0}$  //the current subgradient
7    var  $m = 0$  //the current margin
8    for  $i = 1 : d$ 
9    do //sum over all training samples
10     var  $a_*^{(i)} = \text{ALIGN}(G_1^{(i)}, \dots, G_n^{(i)}, \mathbf{w}, \mathcal{L})$ 
11      $\mathbf{g} \leftarrow \mathbf{g} + \mathbf{f}(a_*^{(i)}) - \mathbf{f}(a^{(i)})$  //update the subgradient
12      $m \leftarrow m + \mathbf{w} \cdot \mathbf{f}(a_*^{(i)}) + \mathcal{L}(a^{(i)}, a_*^{(i)}) - \mathbf{w} \cdot \mathbf{f}(a^{(i)})$  //update the margin
13      $\mathbf{g} \leftarrow \frac{1}{d}\mathbf{g} - \lambda\mathbf{w}; m \leftarrow \frac{1}{d}m + \frac{\lambda}{2}\|\mathbf{w}\|^2$  //add in regularization
14     if  $m < m_*$ 
15     then
16          $m_* \leftarrow m; \mathbf{w}_* = \mathbf{w}$  //update the best parameter vector
17      $\mathbf{w} \leftarrow \mathbf{w} - \alpha\mathbf{g}$  //update parameter vector
18 return  $\mathbf{w}_*$ 

```

Figure 3: **Our parameter learning algorithm**

with higher values in the optimal alignment than in the training sample and increases the score of features with lower values in the optimal alignment than in the training sample. Figure 3 shows our learning algorithm.

Our learning algorithm also has bounded performance. If the inference algorithm is exact, and if the learning rate is constant, our learning algorithm converges at a linear rate to a small region surrounding the optimal \mathbf{w} [26, 23]. A bounds on convergence with an approximate inference algorithm is a topic for further research.

2.4 Global alignment Algorithm

Our global alignment algorithm serves two roles. It finds the highest scoring global alignment once the optimal parameter vector has been learned, and it performs inference as part of our learning algorithm.

We implemented a local hillclimbing algorithm for global alignment [28]. Our alignment algorithm is approximate but efficient in practice. It requires that the alignment feature function decomposes into node and edge feature functions as in equation (1).

Our alignment algorithm iteratively performs updates of a current alignment. The initial alignment contains every node in a separate equivalence class. Our 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.

```

ALIGN( $G_1, \dots, G_n$  : set of networks ,  $\mathbf{w}$  : parameter vector ,  $\mathcal{L}$  : optional loss function )
1  var  $a \leftarrow$  an alignment with one equivalence class per node
2  while true
3  do
4      var  $\Delta_t = 0$  //the total change in score of this iteration
5      for each node  $n \in \bigcup_i G_i$ 
6          do
7              var  $\Delta^* \leftarrow 0$  //best score
8              var  $m^* \leftarrow$  undef //best move
9              for each move  $m$ 
10                 do
11                     var  $a_t \leftarrow m(a)$  //alignment after move  $m$ 
12                      $\Delta \leftarrow \mathbf{w} \cdot \mathbf{f}(a_t) + \mathcal{L}(a_t) - (\mathbf{w} \cdot \mathbf{f}(a) + \mathcal{L}(a))$  //change in score after move  $m$ 
13                     if  $\Delta > \Delta^*$ 
14                         then
15                              $\Delta^* = \Delta; m^* = m$  //new best move
16                              $a \leftarrow m^*(a)$  //do best move on alignment
17                              $\Delta_t \leftarrow \Delta_t + \Delta^*$  //update total change in score of this iteration
18             if  $\Delta_t = 0$ 
19                 then break
20 return  $\mathbf{w}$ 

```

Figure 4: Our global alignment algorithm

- Merge the entire equivalence class of the node with another equivalence class.

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

Our alignment algorithm performs inference as part of our learning algorithm. It can use any scoring function that decomposes as in equation (1). Therefore, to perform inference, we need only augment the scoring function with a loss function $\mathcal{L}^{(i)}$ that also decomposes into node and edge feature functions. The loss function presented above has this property.

Figure 4 shows our alignment algorithm.

Our alignment algorithm depends on the set of candidate equivalence classes to which processed nodes can move. As a heuristic, our algorithm considers as candidates only equivalence classes with a node that has homology (BLAST [29] e-value $< 10^{-5}$) to the processed node.

Our alignment algorithm also depends on the order in which it processes nodes. As a heuristic, it uses node scores—the scoring function with the edge feature function set to zero—to order nodes. For each node, our algorithm computes the node score change when it moves the node to each candidate equivalence class. Our algorithm saves the maximum node score change for each node and then considers nodes in order of decreasing maximum node score change.

In practice, our alignment algorithm runs in linear time. To align networks with n total nodes and m total edges, our algorithm has b iterations that each process n nodes. For each node our algorithm computes the change in score when it moves the node to, on average, C candidate classes. Because the feature function decomposes as in equation (1), to perform each score computation our algorithm need only examine the candidate class, the node’s old class, and the two classes’

neighbors. Our algorithm’s running time is therefore $O(bC(n + m))$. Empirically, b is usually a small constant (less than 10). While C can be large, our algorithm runs faster if it only considers candidate classes with high homology to the processed node (BLAST e-value $\ll 10^{-5}$.)

3 Results

Experimental setup We tested our aligner on three different network datasets: IntAct [30], DIP [31], and the Stanford Network Database [32] (SNDB). We ran pairwise alignments of the human and mouse IntAct networks, yeast and fly DIP networks, *Escherichia coli* K12 and *Salmonella typhimurium* LT2 SNDB networks, and *E. coli* and *Caulobacter crescentus* SNDB networks. We also ran a three-way alignment of the yeast, worm, and fly DIP networks, and a six-way alignment of *E. coli*, *S. typhimurium*, *Vibrio cholerae*, *Campylobacter jejuni* NCTC 11168, *Helicobacter pylori* 26695, and *C. crescentus* SNDB networks.

We used KO groups [18] for our alignment comparison metrics. To compute each metric, we first removed all nodes in the alignment without a KO group and we then removed all equivalence classes with only one node. We then defined an equivalence class as *correct* if every node 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 nodes that were in correct equivalence classes (C_{node})

To measure sensitivity, we computed two metrics:

1. the total number of nodes in correct equivalence classes (Cor)
2. the total number of equivalence classes with k species, for $k = 2, \dots, n$

We used cross validation to test Græmlin 2.0. For each set of networks, we partitioned the KO groups into ten equal sized test sets. For each test set, we trained Græmlin 2.0 on the KO groups not in the test set as described in the Methods section. 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 we used cross validation to test all aligners. For aligners other than Græmlin 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.

We compared Græmlin 2.0 to the local aligners NetworkBLAST [13], MaWISH [8], and Græmlin 1.0 [14], as well as the global aligner ISORANK [11] and a global aligner (Græmlin-global) that used our new alignment algorithm with Græmlin 1.0’s scoring function.

While we simultaneously compared Græmlin 2.0 to ISORANK and Græmlin-global, we compared Græmlin 2.0 to each local aligner separately. Local aligners may have lower sensitivity than global aligners simply because local aligners only consider nodes in conserved modules while global aligners consider all nodes. Therefore, for each comparison, we removed all equivalence classes in Græmlin 2.0’s output that did not contain a node in the local aligner’s output.

Average consistency equivalence class consistency

	SNDB						IntAct		DIP			
	eco/stm		eco/cce		6-way		hsa/mmu		sce/dme		3-way	
	C_{eq}	C_{node}	C_{eq}	C_{node}	C_{eq}	C_{node}	C_{eq}	C_{node}	C_{eq}	C_{node}	C_{eq}	C_{node}
Local aligner comparisons												
NB	0.77	0.45	0.78	0.50	–	–	0.33	0.06	0.39	0.14	–	–
Gr2.0	0.95	0.94	0.79	0.78	–	–	0.83	0.81	0.58	0.58	–	–
MW	0.84	0.64	0.77	0.54	–	–	0.59	0.36	0.45	0.37	–	–
Gr2.0	0.97	0.96	0.77	0.76	–	–	0.88	0.86	0.90	0.91	–	–
Gr	0.80	0.77	0.69	0.64	0.76	0.67	0.59	0.53	0.33	0.29	0.23	0.15
Gr2.0	0.96	0.95	0.82	0.81	0.86	0.85	0.86	0.84	0.61	0.61	0.57	0.57
Global aligner comparisons												
GrG	0.86	0.86	0.72	0.72	0.80	0.81	0.64	0.64	0.68	0.68	0.71	0.71
ISO	0.91	0.91	0.65	0.65	–	–	0.62	0.62	0.63	0.63	–	–
Gr2.0	0.96	0.96	0.78	0.78	0.87	0.87	0.81	0.80	0.73	0.73	0.76	0.76

Table 1: **Græmlin 2.0 has 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 1.0 (Gr), ISORANK (ISO), and a global aligner that used our new alignment algorithm with Græmlin 1.0’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 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 = human; mmu = mouse; sce = yeast; dme = fly.

Number of species per equivalence class

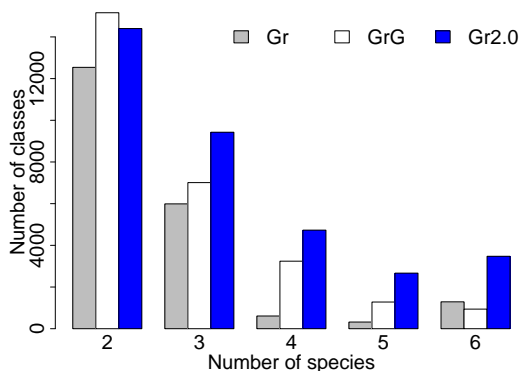


Figure 5: **Græmlin 2.0 finds more cross-species conservation.** We counted the number of equivalence classes that contained k species for $k = 2, 3, 4, 5, 6$ as described in the text. We compared Græmlin 2.0 (Gr2.0) to Græmlin 1.0 (Gr) and a global aligner (GrG) that used our new alignment algorithm with Græmlin 1.0’s scoring function. We ran the six-way alignment described in the text.

Performance comparisons Table 1 shows that Græmlin 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 2 shows that 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

Number of nodes in correct equivalence classes

	SNDB						IntAct		DIP			
	eco/stm		eco/cce		6-way		hsa/mmu		sce/dme		3-way	
	Cor	Tot	Cor	Tot	Cor	Tot	Cor	Tot	Cor	Tot	Cor	Tot
Local aligner comparisons												
NB	457	1016	346	697	–	–	65	1010	43	306	–	–
Gr2.0	627		447		–	–	258		155		–	–
MW	1309	2050	458	841	–	–	87	241	10	27	–	–
Gr2.0	1611		553		–	–	181		20		–	–
Gr	985	1286	546	847	1524	2287	108	203	35	122	27	180
Gr2.0	1157		608		2216		151		75		86	
Global aligner comparisons												
GrG	1496		720		2388		268		384		564	
Iso	2026	–	1014	–	–	–	306	–	534	–	–	–
Gr2.0	2024		1012		3578		350		637		827	

Table 2: **Græmlin 2.0 has 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 1.

classes. In the other tests, Græmlin 2.0 produces the most number of nodes in correct equivalence classes.

Figure 5 shows that Græmlin 2.0 also finds more cross-species conservation than Græmlin 1.0 and Græmlin-global. Relative to Græmlin 1.0 and Græmlin-global, Græmlin 2.0 produces two to five times as many equivalence classes with four, five, and six species.

4 Discussion

In this paper we presented Græmlin 2.0, a multiple network aligner with a new feature-based scoring function, an algorithm that automatically learns the scoring function’s parameters, and an algorithm that uses the scoring function to approximately align multiple networks in linear time. We implemented Græmlin 2.0 for protein interaction network alignment, with a feature function that computes evolutionary events. 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. Our learning algorithm automatically learns parameters specific to any set of networks. In contrast, existing alignment algorithms require manual recalibration to adjust parameters different datasets.

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, such as regulatory networks with directed edges and metabolic networks with chemical compounds [33]. With redefined feature functions, our scoring function and parameter learning algorithm apply to these kinds of networks.

Future research can analyze our learning algorithm. In particular, Græmlin 2.0 might yield better results with a different learning rate or more robust convergence criteria.

Future research can also extend our approach to local alignment. One option is to segment a global alignment into a set of local alignments. With an appropriate feature function and an appropriate inference algorithm, our learning algorithm can learn a scoring function for segmentation.

5 Acknowledgments

JF was supported in part by a Stanford Graduate Fellowship. AN was funded by NSF grant EF-0312459, NIH grant UO1-HG003162, the NSF CAREER Award, and the Alfred P. Sloan Fellowship. BSS was funded by an NSF VIGRE postdoctoral fellowship (NSF grant EMSW21-VIGRE 0502385).

References

- [1] Sharan, R., Ideker, T.: Modeling cellular machinery through biological network comparison. *Nat Biotechnol* **24** (2006) 427–433
- [2] Hartwell, L.H., Hopfield, J.J., Leibler, S., Murray, A.W.: From molecular to modular cell biology. *Nature* **402** (1999) 47–52
- [3] Pereira-Leal, J.B., Levy, E.D., Teichmann, S.A.: The origins and evolution of functional modules: lessons from protein complexes. *Philos Trans R Soc Lond B Biol Sci* **361** (2006) 507–517
- [4] Uetz, P., Finley, R.L.J.: From protein networks to biological systems. *FEBS Lett* **579** (2005) 1821–1827
- [5] Cusick, M.E., Klitgord, N., Vidal, M., Hill, D.E.: Interactome: gateway into systems biology. *Hum Mol Genet* **14 Spec No. 2** (2005) 171–181
- [6] Kelley, B.P., Sharan, R., Karp, R.M., Sittler, T., Root, D.E., Stockwell, B.R., Ideker, T.: Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proc Natl Acad Sci U S A* **100** (2003) 11394–11399
- [7] Sharan, R., Ideker, T., Kelley, B., Shamir, R., Karp, R.M.: Identification of protein complexes by comparative analysis of yeast and bacterial protein interaction data. *J Comput Biol* **12** (2005) 835–846
- [8] Koyuturk, M., Kim, Y., Topkara, U., Subramaniam, S., Szpankowski, W., Grama, A.: Pairwise alignment of protein interaction networks. *J Comput Biol* **13** (2006) 182–199
- [9] Pinter, R.Y., Rokhlenko, O., Yeger-Lotem, E., Ziv-Ukelson, M.: Alignment of metabolic pathways. *Bioinformatics* **21** (2005) 3401–3408
- [10] Dost, B., Shlomi, T., Omer, N.G., Ruppin, E., Bafna, V., Sharan, R.: Qnet: A tool for querying protein interaction networks. In: *Proceedings of the 11th Annual International Conference on Computational Molecular Biology (RECOMB 2007)*. (2007) 1–15
- [11] Singh, R., Xu, J., Berger, B.: Pairwise global alignment of protein interaction networks by matching neighborhood topology. In: *Proceedings of the 11th Annual International Conference on Computational Molecular Biology (RECOMB 2007)*. (2007) 16–31
- [12] Zhenping, L., Zhang, S., Wang, Y., Zhang, X.S., Chen, L.: Alignment of molecular networks by integer quadratic programming. *Bioinformatics* **23** (2007) 1631–1639
- [13] Sharan, R., Suthram, S., Kelley, R.M., Kuhn, T., McCuine, S., Uetz, P., Sittler, T., Karp, R.M., Ideker, T.: Conserved patterns of protein interaction in multiple species. *Proc Natl Acad Sci U S A* **102** (2005) 1974–1979

- [14] Flannick, J., Novak, A., Srinivasan, B.S., Batzoglou, S., McAdams, H.H.: Graemlin: General and Robust Alignment of Multiple Large Interaction Networks. *Genome Res* **16** (2006)
- [15] Hirsh, E., Sharan, R.: Identification of conserved protein complexes based on a model of protein network evolution. *Bioinformatics* **23** (2007) 170–176
- [16] Berg, J., Lassig, M.: Cross-species analysis of biological networks by Bayesian alignment. *Proc Natl Acad Sci U S A* **103** (2006) 10967–10972
- [17] Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., Cherry, J.M., Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T., Harris, M.A., Hill, D.P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J.C., Richardson, J.E., Ringwald, M., Rubin, G.M., Sherlock, G.: Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet* **25** (2000) 25–29
- [18] Kanehisa, M., Goto, S.: KEGG: kyoto encyclopedia of genes and genomes. *Nucleic Acids Res* **28** (2000) 27–30
- [19] Remm, M., Storm, C.E., Sonnhammer, E.L.: Automatic clustering of orthologs and in-paralogs from pairwise species comparisons. *J Mol Biol* **314** (2001) 1041–1052
- [20] Do, C.B., Gross, S.S., Batzoglou, S.: Contralign: Discriminative training for protein sequence alignment. In: *Proceedings of the 10th Annual International Conference on Computational Molecular Biology (RECOMB 2006)*. (2006) 160–174
- [21] Do, C.B., Woods, D.A., Batzoglou, S.: CONTRAfold: RNA secondary structure prediction without physics-based models. *Bioinformatics* **22** (2006) 90–98
- [22] Felsenstein, J.: Maximum-likelihood estimation of evolutionary trees from continuous characters. *Am J Hum Genet* **25** (1973) 471–492
- [23] Ratliff, N., Bagnell, J., Zinkevich, M.: (online) subgradient methods for structured prediction. In: *Eleventh International Conference on Artificial Intelligence and Statistics (AISTats)*. (2007)
- [24] Boyd, S., Vandenberghe, L.: *Convex Optimization*. Cambridge University Press, New York, NY, USA (2004)
- [25] Shor, N.Z., Kiwiel, K.C., Ruszcayński, A.: *Minimization methods for non-differentiable functions*. Springer-Verlag New York, Inc., New York, NY, USA (1985)
- [26] Nedic, A., Bertsekas, D.: *Convergence rate of incremental subgradient algorithms* (2000)
- [27] Shmoys, D.B., Swamy, C.: An approximation scheme for stochastic linear programming and its application to stochastic integer programs. *J. ACM* **53** (2006) 978–1012
- [28] Russell, S., Norvig, P.: *Artificial Intelligence: A Modern Approach*. 2nd edition edn. Prentice-Hall, Englewood Cliffs, NJ (2003)
- [29] Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res* **25** (1997) 3389–3402

- [30] Kerrien, S., Alam-Faruque, Y., Aranda, B., Bancarz, I., Bridge, A., Derow, C., Dimmer, E., Feuermann, M., Friedrichsen, A., Huntley, R., Kohler, C., Khadake, J., Leroy, C., Liban, A., Lieftink, C., Montecchi-Palazzi, L., Orchard, S., Risse, J., Robbe, K., Roechert, B., Thorneycroft, D., Zhang, Y., Apweiler, R., Hermjakob, H.: IntAct—open source resource for molecular interaction data. *Nucleic Acids Res* **35** (2007) 561–565
- [31] Xenarios, I., Salwinski, L., Duan, X.J., Higney, P., Kim, S.M., Eisenberg, D.: DIP, the Database of Interacting Proteins: a research tool for studying cellular networks of protein interactions. *Nucleic Acids Res* **30** (2002) 303–305
- [32] Srinivasan, B.S., Novak, A.F., Flannick, J.A., Batzoglou, S., McAdams, H.H.: Integrated protein interaction networks for 11 microbes. In: Proceedings of the 10th Annual International Conference on Computational Molecular Biology (RECOMB 2006). (2006) 1–14
- [33] Srinivasan, B., Shah, N., Flannick, J., Abeliuk, E., Novak, A., Batzoglou, S.: Current progress in network research: toward reference networks for key model organisms. *Brief Bioinform* (2007)
- [34] Altschul, S.F., Carroll, R.J., Lipman, D.J.: Weights for data related by a tree. *J Mol Biol* **207** (1989) 647–653

A Definition of feature functions

This section presents precise definitions of our feature function and the evolutionary events that our feature function computes.

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

Each species $i \in [1 : n+m]$ is represented by a species weight vector $\mathbf{s}^i \in \mathbb{R}^n$, where $\sum_{j=1}^n s_j^i = 1$ and s_j^i represents the similarity of species $j \in [1 : n]$ to species i . We can use a phylogenetic tree to compute the weight vectors efficiently [22, 34]. Extant species $j \in [1 : n]$ has a species weight vector $[s_1^j = 0, \dots, s_{j-1}^j = 0, s_j^j = 1, s_{j+1}^j = 0, \dots, s_n^j = 0]$.

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 .

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

Protein present We define $\mathbf{p} \in \mathbb{R}^n$ as $p_i = 1$ if species i has a protein in $[x]$ and 0 otherwise.

- $f_1^N = \mathbf{s}^r \cdot \mathbf{p}$ is the probability that species r has a protein in $[x]$.
- $f_2^N = 1 - \mathbf{s}^r \cdot \mathbf{p}$ is the probability that species r does not have a protein in $[x]$.

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

- $f_3^N = \frac{\mathbf{s}^r \cdot \mathbf{c}}{\mathbf{s}^r \cdot \mathbf{p}}$ is the expected number of proteins species r has in $[x]$, given that r has a protein.
- $f_4^N = (f_3^N)^2$

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 $p(k) = \mathbf{s}^k \cdot \mathbf{p}$ for $k = i, j$ as the probabilities that species i and j have a protein in $[x]$.

- $f_5^N(i, j) = p(i) * (1 - p(j)) + (1 - p(i)) * p(j)$ is the probability a protein deletion occurs between species i and j .
- $f_6^N(i, j) = p(i) * p(j)$ is the probability a protein deletion does not occur between species i and j .

Protein duplication We define $c(k) = \frac{\mathbf{s}^k \cdot \mathbf{c}}{\mathbf{s}^k \cdot \mathbf{p}}$ for $k = i, j$ as the expected numbers of proteins in species i and j .

- $f_7^N(i, j) = |c(i) - c(j)|$ is the expected number of proteins gained between species i and j .

Protein mutation We define a species pair weight matrix $\mathbf{S}^{ij} \in \mathbb{R}^{n \times n}$ as $S_{kl}^{ij} = s_k^i s_l^j$. We define $\mathbf{B} \in \mathbb{R}^{n \times n}$ as

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

where $b(p, q)$ is the BLAST bitscore [29] 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 $[x]$.

- $f_8^N(i, j) = \text{tr}(\mathbf{S}^{ijT} \mathbf{B})$, the sum of entry-wise products, is the expected bitscore between the proteins in species i and j .
- $f_9^N(i, j) = (f_8^N)^2$
- $f_{10}^N(i, j) = (f_8^N)^{-1}$
- $f_{11}^N(i, j) = (f_8^N)^{-2}$

Finally, we compute events for all extant species $i \in [1 : n]$.

Paralog mutation

- $f_{12}^N(i) = \mathbf{B}_{ii}$ is the expected average bitscore between a protein in species i and its paralogs.
- $f_{13}^N(i, j) = (f_{12}^N)^2$
- $f_{14}^N(i, j) = (f_{12}^N)^{-1}$
- $f_{14}^N(i, j) = (f_{12}^N)^{-2}$

We compute the edge feature function \mathbf{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 We define $e(k, p, q) = 1$, for $k \in [1 : n], p \in \Pi_k^{[x]}, q \in \Pi_k^{[y]}$ as 1 if there is an edge between p and q and 0 otherwise. We then define $\mathbf{e} \in \mathbb{R}^n$ as

$$e_k = \frac{1}{|\Pi_k^{[x]}| |\Pi_k^{[y]}|} \sum_{p \in \Pi_k^{[x]}} \sum_{q \in \Pi_k^{[y]}} e(k, p, q)$$

which represents the average probability that species k has an edge. We define e_k as NULL if $\Pi_k^{[x]}$ or $\Pi_k^{[y]}$ is empty. We define

$$e(l) = \left(\frac{1}{\sum_{k: e_k \neq \text{NULL}} e_k} \right) \sum_{k: e_k \neq \text{NULL}} e_k s_k^i \quad l = i, j$$

which represent the probabilities that species i and j have edges.

- $f_1^E(i, j) = e(i) * (1 - e(j)) + (1 - e(i)) * e(j)$ is the probability that an edge is lost between species i and j .
- $f_2^E(i, j) = e(i) * e(j)$ is the probability that an edge 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 $k \in [1 : n], p \in \Pi_k^{[x]}, q \in \Pi_k^{[y]}$ as

$$\tilde{e}(k, p, q) = \frac{1}{|\Pi_k^{[x]}| |\Pi_k^{[y]}|} \sum_{\substack{p' \in \Pi_k^{[x]} \\ q' \in \Pi_k^{[y]} \\ (p', q') \neq (p, q)}} e(k, p', q')$$

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

- $f_3^E(i) = \sum_{p \in \Pi_k^{[x]}} \sum_{q \in \Pi_k^{[y]}} \left(e(i, p, q) * (1 - \tilde{e}(i, p, q)) + (1 - e(i, p, q)) * \tilde{e}(i, p, q) \right)$ is the average probability an edge is lost between a pair of proteins in species i and all other pairs of proteins in species i .
- $f_4^E(i) = \sum_{p \in \Pi_k^{[x]}} \sum_{q \in \Pi_k^{[y]}} e(i, p, q) * \tilde{e}(i, p, q)$ is the average probability an edge 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

$$\mathbf{f}^N([x]) = [f_1^N, f_2^N, f_3^N, f_4^N, f_5^N(s, t), f_6^N(s, t), f_7^N(s, t), f_8^N(s, t), f_9^N(s, t), f_{10}^N(s, t), f_{11}^N(s, t), f_{12}^N(s) + f_{12}^N(t), f_{13}^N(s) + f_{13}^N(t), f_{14}^N(s) + f_{14}^N(t), f_{15}^N(s) + f_{15}^N(t)]$$

and the final edge feature function is

$$\mathbf{f}^E([x], [y]) = [f_1^E(s, t), f_2^E(s, t), f_3^E(s) + f_3^E(t), f_4^E(s) + f_4^E(t),]$$

For multiple alignment, the final node feature function is

$$\begin{aligned}
\mathbf{f}^N([x]) = & \\
& \left[f_1^N, f_2^N, f_3^N, f_4^N, \sum_{(i,j)} f_5^N(i,j), \sum_{(i,j)} f_5^N(i,j) \times b, \right. \\
& \sum_{(i,j)} f_6^N(i,j), \sum_{(i,j)} f_6^N(i,j) \times b, \sum_{(i,j)} f_7^N(i,j), \sum_{(i,j)} f_7^N(i,j) \times b, \\
& \sum_{(i,j)} f_8^N(i,j), \sum_{(i,j)} f_8^N(i,j) \times b, \sum_{(i,j)} f_8^N(i,j) \times b^2, \sum_{(i,j)} f_8^N(i,j) \times b^3, \\
& \sum_{(i,j)} f_9^N(i,j), \sum_{(i,j)} f_9^N(i,j) \times b, \sum_{(i,j)} f_9^N(i,j) \times b^2, \sum_{(i,j)} f_9^N(i,j) \times b^3, \\
& \sum_{(i,j)} f_{10}^N(i,j), \sum_{(i,j)} f_{10}^N(i,j) \times b, \sum_{(i,j)} f_{10}^N(i,j) \times b^2, \sum_{(i,j)} f_{10}^N(i,j) \times b^3, \\
& \sum_{(i,j)} f_{11}^N(i,j), \sum_{(i,j)} f_{11}^N(i,j) \times b, \sum_{(i,j)} f_{11}^N(i,j) \times b^2, \sum_{(i,j)} f_{11}^N(i,j) \times b^3, \\
& \left. \sum_{i=1}^n f_{12}^N(i), \sum_{i=1}^n f_{13}^N(i), \sum_{i=1}^n f_{14}^N(i), \sum_{i=1}^n f_{15}^N(i) \right]
\end{aligned}$$

and the final edge feature function is

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

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.