A Supplementary Information

A.1 Network Construction

Here we present a heavily abbreviated discussion of network integration; a lengthier description can be found in (Srinivasan et al., 2006a) and (Srinivasan et al., 2006b).

We formulate the network integration problem as a binary classifier, where the goal is to distinguish functionally linked proteins ($L = 1$) from non-interacting pairs ($L = 0$). In this formulation, a vector of interaction predictors is the input to a binary classifier function, which returns the integrated probability that two proteins are functionally linked. To calculate the mapping between raw interaction data and integrated probabilities, the classifier function is trained on a set of known interactions. Applying this classifier to predict interaction probabilities for *all* protein pairs in a genome yields a probabilistic protein interaction network.

We generated the training set of known interactions by using the KEGG (Kanehisa and Goto, 2000) annotation of individual proteins to produce an annotation of protein *pairs*. For each pair we recorded if the proteins had overlapping annotations ($L = 1$), if both were in entirely nonoverlapping categories ($L = 0$), and if either protein lacked an annotation code or was marked as unknown ($L =?$) (Figure 1(a)). We also calculated four functional genomic predictors: coexpression (Pearson correlation between expression profiles), coinheritance (Pearson correlation between phylogenetic profiles), coevolution (Pearson correlation between distance matrices, taken elementwise), and colocation (average chromosomal distance between ORFs). Each of these predictors is defined on a pair of proteins rather than an individual protein and can be arranged in a four dimensional vector:

$$\vec{E} = (E_{\text{coex}}, E_{\text{coin}}, E_{\text{coev}}, E_{\text{coloc}}) = (E_1, E_2, E_3, E_4)$$

As shown in Figure 1(b), it is an empirical fact that the distribution of functionally linked protein pairs is shifted relative to the distribution of functionally unlinked pairs. Intuitively, this means that each functional genomic evidence type is a predictor of protein interaction. We can combine these predictors to obtain the integrated probability of protein interaction via Bayes' rule. In practice, the quotient formula for the Bayesian posterior is quite sensitive to fluctuations in the denominator. To deal with this, we use bootstrap aggregation(Breiman, 1996) to smooth the posterior.

$$P(L = 1|\vec{E}) = \frac{1}{M}\sum_{i=1}^{M} \frac{P_i(\vec{E}|L = 1)P(L = 1)}{P_i(\vec{E}|L = 1)P(L = 1) + P_i(\vec{E}|L = 0)P(L = 0)}$$

Thus, for each pair of proteins, we have a value $P(L = 1|\vec{E})$ which represents the integrated probability of protein interaction over several data types. These edge weights are the weights in our aligned networks. For further details, see (Srinivasan et al., 2006a, Srinivasan et al., 2006b).

A.1.1 Comparative Genomics and Network Alignment

An important question is whether the use of comparative genomic methods such as colocation, coinheritance, and coevolution during network integration significantly impacts the process of network alignment. While expression data is truly independent from species to species, methods based on the distribution of homologs have some degree of potential redundancy. For example, the GyrA and GyrB proteins in *Caulobacter crescentus* will have similar lists of sequence homologs to the GyrA and GyrB proteins in *Escherichia coli*. Subsequent calculation of phylogenetic profile correlations (E_{coin}) will show that the GyrA and GyrB families are strongly coinherited across the set of all

microbes, and in both *C. crescentus* and *E. coli* this high coinheritance value will contribute to the integrated interaction probability of this protein pair.

Within each individual species' network this is clearly not a problem, as it is clear that coinheritance is a predictor of functional interaction(Pellegrini et al., 1999, Srinivasan et al., 2005). However, when using network alignment to detect conserved modules, the question is whether interactions partially inferred from comparative genomic methods will show spurious conservation. The main issue is that if a given pair of proteins has a strong coinheritance, colocation, or coevolution(Srinivasan et al., 2005) correlation in one species, it is likely to have a similarly strong correlation in most other species, as the columns of the phylogenetic profile, colocation, and coevolution matrices are derived from the pool of sequenced genomes. Highly coexpressed pairs in one species are not quite so guaranteed to be strongly coexpressed in closely related species, as the columns of the expression matrices come from different experiments.

In this work, we decided that omitting the comparative genomic predictors of interaction would artificially reduce our ability to find conserved modules. From a pragmatic standpoint, the observation that two protein families are globally colocated across the set of microbial genomes is a notable fact. If two species contain representatives from both of these families, it becomes more likely that the species contain a conserved module. As the purpose of network alignment is to find conserved modules, we would be remiss in omitting this comparative genomic signal. That said, in later work it may be possible to take a middle ground between discarding comparative genomic predictors of interaction entirely versus completely retaining them in unaltered fashion as inputs to each integrated network. One possibility is to partition the pool of genomes (e.g. through random sampling or phylogenetic considerations) and to calculate "independent" comparative genomic predictions for each species which we wish to align. However, such a method would be laborious in that it would require N-way partitioning (and network construction) for an N-way alignment. Another possibility is to reformulate the network alignment scoring function to include separate terms for species-specific and comparative genomic interaction predictions. These are interesting directions for further research.

A.2 Scoring Alignments

There are two components to Græmlin's scoring framework: node scores and edge scores. Both are based on the evolutionary interpretation of an alignment, in which each equivalence class represents a set of proteins descending from a common ancestor. For each component of the scoring function, we must specify two probabilistic models: the alignment model $\mathcal{M}$ and the random model $\mathcal{R}$.

A.2.1 Node scoring

To score an alignment, Græmlin first scores each equivalence class in the alignment. It uses each scoring model to assign probabilities to an equivalence class by considering three types of evolutionary events: protein sequence mutations, protein insertions and deletions, and protein duplications and divergences.

Græmlin scores protein sequence mutations using weighted sum-of-pairs scoring. First, for each pair of proteins v_1, v_2 in an equivalence class it assigns a score $S_N(v_1, v_2) = \log \frac{\Pr_{\mathcal{M}}(v_1,v_2)}{\Pr_{\mathcal{R}}(v_1,v_2)}$, where $\Pr_{\mathcal{M}}$ is the probability assigned to the pair by the alignment model, and $\Pr_{\mathcal{R}}$ is the probability assigned by the random model. Both $\Pr_{\mathcal{M}}$ and $\Pr_{\mathcal{R}}$ are probability distributions over BLAST alignments; each specifies the probability of obtaining every possible BLAST bitscore. For example, if the bitscore of two proteins is S, the probability of the pair according to the alignment model is $\Pr_{\mathcal{M}}(S-\Delta < x < S+\Delta)$. Given a complete set of pairwise scores, Græmlin uses the weighted sum-

of-pairs score, taken over all pairs of proteins in the equivalence class, as the score of all mutations in the equivalence class.

To determine the distribution of bitscores for the alignment model, we sampled random pairs of proteins from within the same COG; for the random model, we sampled completely random pairs of proteins. We smoothed both models using a monotone regression function in the same manner as has been done in the past (Kelley et al., 2003).

Græmlin considers protein insertions, deletions, duplications, and divergences explicitly, using the most parsimonious evolutionary history of an equivalence class. To construct this history, it uses a dynamic programming algorithm to compute the most likely number of proteins at each ancestral node 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, and a protein deletion occurs when the number of proteins decreases from 1 to 0 along a branch. Similarly, protein duplications occur when the number of proteins increases from a non-zero number along a branch, and divergences occur when the number of proteins decreases to a non-zero number along a branch.

After determining the number of insertions, deletions, duplications, and divergences in the history of an equivalence class, Græmlin assigns penalties to each such event e using the function $s(e) = \log \frac{\Pr_{\mathcal{M}}(e)}{\Pr_{\mathcal{R}}(e)}$. The score of all events is then $\sum_i s(e_i)$.

In practice, obtaining the actual probabilities $\Pr_{\mathcal{M}}(e)$ and $\Pr_{\mathcal{R}}(e)$ is difficult, as few suitable training sets exist. For now, therefore, we empirically derived the necessary parameters. Under the assumption that the insertion and deletion penalties are symmetric, as are the duplication and divergence penalties, we must estimate only two parameters: deletion and duplication penalties. These numbers are analogous to the gap penalty in traditional sequence alignment and control the importance assigned to the insertion or duplication scores relative to the protein similarity scores.

A.2.2 Edge scoring

Græmlin scores each edge in an alignment based on its weight and the nodes it connects. Both the alignment and the random models specify the probabilities of observing all possible edge weights between 0 and 1. For each pair of proteins v_1, v_2 in the alignment, where v_1 and v_2 belong to the same species but different equivalence classes, Græmlin assigns an edge score of $\log \frac{\Pr_{\mathcal{M}}(w-\Delta<x<w+\Delta)}{\Pr_{\mathcal{R}}(w-\Delta<x<w+\Delta)}$, where w is the weight of the (v_1, v_2) edge and Δ is a small value.

One minor technical issue that arises is that there may be no edge between v_1 and v_2. The networks generated by algorithms like SRINI are complete: they assign a weight to each pair of proteins. In practice, however, many of these weights are close to zero, because biologically each of the n proteins typically has only $k << n$ interaction partners. Thus, to avoid unnecessary computation, we threshold the input networks by removing edges with weights less than some cut-off m; for the results presented in this paper, we have used $m = 0.25$ and $m = 0.5$. If there is no (v_1, v_2) edge in the thresholded network, Græmlin can only assume that the edge between the pair has weight less than m; it therefore gives the edge a score of $\log \frac{\Pr_{\mathcal{M}}(0<x<m)}{\Pr_{\mathcal{R}}(0<x<m)}$.

Because a node with more incident edges will be more likely to have an edge to a random node, particularly if that random node also has many incident edges, the edge weight distribution function for the random model $\Pr_{\mathcal{R}}$ depends on the nodes to which an edge is incident. It is parametrized by the product of their relative expected degrees; 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 the specific function, we sampled random pairs of proteins 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 input ESM specifies the distribution function for the alignment model $\mathcal{M}$. Before scoring an edge, Græmlin must assign a label to each equivalence class, where the set of potential labels is specified in the ESM. It determines $\Pr_{\mathcal{M}}$ for an edge by using the cell in the ESM that corresponds to the labels of the nodes connected by the edge. Figure 2 shows the methodology used by both the random and alignment models for scoring a single edge.

To assign a final score to the edges in an alignment, Græmlin uses the labeling that yields the highest score. Because the number of labelings scales exponentially with the number of potential labels, we currently focus on three special but important cases of an ESM: the Complex ESM, the Pathway ESM, and the Module ESM. Each of these cases permits a tractable labeling algorithm.

A Complex ESM is the simplest ESM, consisting of only one label and therefore allowing for a trivial labeling algorithm; the distribution $\Pr_{\mathcal{M}}$ is the same for all edges and is biased toward high edge weights. Using a Complex ESM, Græmlin will score protein complexes highly, similar to past methods (Koyuturk et al., 2005, Sharan et al., 2005), but will not adequately reward conserved modules that are not highly connected.

In principle one could train this distribution on known biological modules, but for our tests we used $\mathcal{N}(0.5, 0.8^2)$ (a Gaussian distribution with mean of 0.5 and standard deviation of 0.8), obtained empirically. The mean of 0.5 is high enough to reward the presence of edges, without overly penalizing missing edges; the wide standard deviation allows the sensitivity of Græmlin to be robust with respect to noisy networks. For example, a narrow deviation restricts alignments to be essentially completely connected, as a missing edge will incur a high penalty. In contrast, a wide deviation allows an alignment with missing edges to still register as high scoring, which is important because protein complexes do not always appear as fully connected.

A Pathway ESM is slightly more complicated than a Complex ESM and allows searches for conserved linear chains among networks. Such an ESM consists of n labels, where n is the maximum pathway length for which the ESM can search; row i in the matrix corresponds to the i^{th} node in the pathway. The sub- and super-diagonal elements of the matrix contain distributions biased toward high weight edges, which enforces the requirement that adjacent proteins in the pathway are connected. As with the Complex ESM, we use an empirically derived distribution, in this case, $\mathcal{N}(0.8, 0.15^2)$; we chose a higher mean and narrower standard deviation than in the case of protein complexes because when searching for pathways we want to heavily penalize any missing edges between adjacent proteins. The other matrix cells can contain arbitrary distributions, but the default Pathway ESM eschews penalizing or rewarding edges between non-adjacent pathway nodes by using the same distribution as $\mathcal{R}$, resulting in a score of 0 for every such edge.

To avoid searching through all possible labelings when using a Pathway ESM, Græmlin uses a simple heuristic that is efficient and works well in practice. It begins by assigning label $\frac{n}{2}$ to the first equivalence class encountered. It then continues labeling nodes but at each step only considers unused labels; furthermore, it only considers labels that are adjacent to labels that have already been used. To find all pathways of length at least p using this algorithm, Græmlin must use $n = 2p$ labels.

The most specific type of ESM is a Module ESM, which captures an arbitrary topology. Græmlin uses such an ESM only when aligning a small query network to a larger network. In this case, the Module ESM contains one label for each protein in the query network, and distributions in the cells reflect the edges that are present in the query. Græmlin generates a Module ESM using kernel density estimation to create a histogram for each matrix cell, initializing each with a small number of pseudocounts.

To allow for a tractable labeling algorithm when using a Module ESM, Græmlin uses a heuristic to reduce the number of labelings that must be considered. Given an equivalence class to label, it finds all proteins in the original query network that are homologous to any protein in the equiv-

alence class, using high-scoring BLAST hits to determine homology. It then considers only labels corresponding to those homologs, greedily assigning each protein its highest scoring label.

When performing multiple alignments, Græmlin uses the high-scoring alignments to further refine a Module ESM. After each phase of progressive alignment, it rebuilds the ESM by generating one label for each equivalence class. As Græmlin adds more and more species to the alignment, it uses more and more edges to generate the histograms with kernel density estimation. In this fashion, the ESM becomes an LSSM, which should begin to approximate the underlying ancestral module structure.

A.3 Algorithm

Græmlin uses a progressive alignment strategy to perform multiple alignments. At each step, it selects the two networks that are closest evolutionarily and performs a pairwise alignment, generating a new network based upon the results. During pairwise alignment it first finds a set of seeds between two networks and then extends each seed into an alignment. At any given step, the nodes in the networks may be equivalence classes rather than proteins, but this presents no difficulties.

A.3.1 Seeded alignment using d-clusters

Methodology Græmlin uses d-clusters to find seeds between two networks; a d-cluster is a set of d nodes taken from one network. Græmlin first generates a set of d-clusters from each network and then finds all high-scoring pairs of them; two d-clusters are high-scoring if there is some way of mapping the nodes that scores higher than T. Both d and T are user-specified parameters that control the speed-sensitivity trade-off presented by the seeding algorithm.

In the first step of the seeding phase, Græmlin creates a set of d-clusters from each network. It generates one d-cluster for each node v by creating a set consisting of v as well as the $d-1$ closest nodes in the network. Græmlin interprets the length of an edge as the negative logarithm of its confidence; the distance between two nodes is then the sum of the edge lengths of the shortest path between them, and the nearest neighbor of a node is the node that is closest to it. Græmlin finds nearest neighbors using a simple breadth-first-search.

One problem with this approach is that very high degree hubs will be present in many d-clusters, resulting in substantial overlap and redundancy in seeds. Græmlin avoids this to some extent by using a heuristic when generating d-clusters: each time it selects a node as part of a d-cluster, it increases the distance of the node to all of its neighbors. Græmlin will therefore be less likely to include the node in any future d-clusters and will create d-clusters that are more independent, which, much like spaced seeds for genomic database searches (Ma et al., 2002, Sun and Buhler, 2005), often increases search sensitivity.

Also in analogy with spaced seeds, which allow for multiple seeds to be indexed at each position in a sequence (Li et al., 2003, Sun and Buhler, 2005), Græmlin can generate more than one d-cluster for each node. Using the above method of increasing edge lengths after each d-cluster is generated, successive runs of the algorithm will generate different d-clusters each time.

Once it generates a set of d-clusters for each network, Græmlin finds all high-scoring pairs of d-clusters between the networks. In theory, when comparing a pair of d-clusters, it must examine all possible mappings of nodes to see if any result in a score larger than T, but to increase speed it only examines *greedy mappings*. Græmlin obtains a greedy mapping by successively adding to the mapping the highest scoring remaining pair of nodes, stopping when no pair can increase the score. This heuristic has the advantage of being fast while missing very few high-scoring d-cluster pairs.

While using greedy mappings to compare d-clusters is fast, examining all potential pairs of d-clusters is quadratic in the number of nodes in the networks; this is prohibitive when searching large databases. Græmlin improves upon this bound by building an index from the larger network as a pre-processing step; it can use this index for many query networks, which is useful when it performs database searches. The index contains two tables: one table is keyed by node and has values of all d-clusters to which each node belongs, and the other is keyed by d-cluster and has values of all nodes belonging to the d-cluster.

When faced with finding all high-scoring matches for a d-cluster in a query, Græmlin first finds for each node in the d-cluster all homologs in the other network. It considers as potential d-cluster matches only d-clusters containing those homologs, which, since most nodes have relatively few homologs, dramatically reduces the number of potential d-cluster matches. While the effectiveness of this algorithm depends on the requirement that nodes can only be aligned to homologs, it is easy to extend it to allow for alignments of arbitrary nodes. In a more general setting, rather than finding all nodes that are homologous to those in the d-cluster, Græmlin considers all nodes scoring above some threshold T_1 with respect to a node in the d-cluster. The value of T_1 depends on the specific values of d and T; intuitively, it should be on the order of $\frac{T}{d}$.

Theory While the specific values of d and T are somewhat intuitive, by making some simplifying assumptions we can statistically quantify their effects on the speed-sensitivity trade-off presented by Græmlin. Our goal is to assess for various values of d and T the probability that no d-cluster pair scoring higher than T will be found within an alignment with total score greater than some threshold S, where S is the raw score above which alignments are high-scoring.

To begin with, we introduce a model of a high-scoring local alignment, in which we assume that each equivalence class is the same size. Rather than a set of equivalence classes, we can view the alignment as a set of node random variables, each of which has value equal to the node score of the corresponding equivalence class. Similarly, we can view each set of edges between two equivalence classes as an edge random variable. This model is analogous to the concept of an ungapped homology used in studies of spaced seeds (Ma et al., 2002). Conditioned on the event that the alignment scores greater than S, we assume that the node scores and edge scores are separate i.i.d. random variables; that is, they are conditionally independent identically distributed random variables. Obviously, these i.i.d distributions depend on the size of the equivalence classes.

Each node random variable n_i has score $S(n_i)$, and we denote 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, we presume that $S_E + S_N > S$. The probability that a d-cluster match does not occur within an alignment is then

$$p_{miss} = \Pr(\neg\exists c \in D \text{ s.t. } \sum_{n_i \in c} S(n_i) > T | S_N + S_E > S),$$

which we refer to as the miss probability.

To simplify matters, we condition only on the node score S_N by using a value for it such that a very high percentage of alignments with total score greater than S have node scores greater than S_N. In addition, we assume that a module of size N has $\frac{N}{d}$ disjoint d-cluster pairs within it. This enables us to obtain an approximate miss probability in a straightforward manner and effectively amounts to the assumption that d-clusters are disjoint and completely cover each module.

With the above assumptions, we must calculate the intersection of $\frac{N}{d}$ identical events, each with

probability

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

If we denote Y_i as a random variable that corresponds to the sum of i variables drawn from the node distribution, we can write each event as

$$\begin{aligned}
&\Pr(\sum_{i=1}^{d} S(n_i) < T | \sum_{n_i \in M} S(n_i) > S_N) \\
&\qquad = \Pr(Y_d < T | Y_N > S_N) \\
&\qquad = \frac{\Pr(Y_d < T \wedge Y_N > S_N)}{\Pr(Y_N > S_N)} \\
&\qquad = \frac{\int_{-\infty}^{T} \Pr(Y_d = x \wedge Y_{N-d} > S_N - x) dx}{\Pr(Y_N > S_N)} \\
&\qquad = \frac{\int_{-\infty}^{T} \Pr(Y_d = x) \Pr(Y_{N-d} > S_N - x) dx}{\Pr(Y_N > S_N)}. \qquad (1)
\end{aligned}$$

Using normal distributions for the Y_i is an adequate approximation for our present purposes. Because non-orthologs are rarely aligned as part of a high-scoring alignment, we can estimate μ and σ by sampling many known orthologous pairs of proteins; when nodes consist of equivalence classes, we must modify this technique slightly to take into account the linear scaling of node scores.

Equation (1) represents a constraint on the values of d and T; it allows us to evaluate whether a given pair of values has a miss probability no greater than a user-specified p_{miss} parameter. We can also quantify the effect that a pair of values for d and T has on the computational cost of Græmlin. Each d-cluster pair that it compares takes $O(d^2)$ time, and each high-scoring d-cluster pair initiates a greedy extension. Using λ as a parameter specifying the relative costs of these two operations, the expected computational cost of a given choice for d and T is proportional to

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

where N_1 and N_2 are the number of d-clusters in each network. Z_d is a distribution similar to Y_d except that the node distributions are taken over random pairs of proteins, since here we are assessing arbitrary pairs of d-clusters and not only d-clusters that are a part of a high-scoring alignment.

With this in place, we can optimize equation (2) subject to the constraint involving equation (1). This will give us values for d and T that present approximately the best speed/sensitivity trade-off subject to a constraint on sensitivity loss due to seeding. Using a constraint of $p_{miss} = .05$, we obtained values of $d = 4$, $T = 7$, which we used for all of our tests.

A.3.2 Seed extension

Græmlin uses a greedy algorithm to try to transform each seed between two networks into a high-scoring alignment. The algorithm proceeds in phases; at each step Græmlin aligns a pair of nodes that maximally increases the score of the growing alignment.

In the first step in the extension phase, Græmlin aligns two nodes, one from each d-cluster in the seed. It aligns the pair of nodes resulting in the maximal score, and then places all nodes that neighbor either of these two nodes, as well as the nodes themselves, into a frontier.

Græmlin then examines all pairs of nodes on the frontier, computing for each the change in score that would result from adding the aligned pair to the alignment. If one or both nodes are present in the alignment, this entails augmenting one equivalence class or merging two equivalence classes; if neither is present Græmlin must create a new equivalence class. Græmlin also considers adding each node by itself to the alignment in case it has no ortholog with which to align. In both cases, any time an equivalence class is created or modified Græmlin must assign it the label resulting in the highest score according to the ESM.

If after examining the frontier any scores are positive, Græmlin updates the alignment and recomputes the frontier by adding to it any nodes that neighbor the newly aligned nodes. It then iterates the process until no changes can yield an increase in the score of the alignment. As a final post-processing step, Græmlin removes any equivalence classes from the alignment if it can further increase the score; this serves to partially rectify any mistakes made early in the greedy process. Figure 3 shows pseudocode for the basic extension algorithm.

One problem with the greedy extension algorithm is that the size of the frontier grows exponentially as nodes are added to the alignment. Græmlin uses several heuristics to handle this. First, when considering pairs to add to the alignment, it only considers pairs that have a positive node score; this dramatically reduces the search space size but prevents Græmlin from aligning functional orthologs with no sequence homology. Græmlin also does not recompute the score of all pairings on the frontier at each iteration but rather updates the scores only to reflect the most recent change in the alignment. Furthermore, it does not update all pairs on the frontier; rather, it stores pairs 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 that change the alignment score by a value greater than some threshold; if at any point the change in score resulting from aligning a pair of nodes falls below that threshold, Græmlin moves that pair into the lower echelon. It only updates nodes in the lower echelon if no pairs in the upper echelon can increase the alignment score, at which point it then continues the extension if possible.

One final issue with the extension phase is that it typically results in many overlapping alignments, which is typically not desirable. To handle this, Græmlin accepts a parameter governing the maximum allowable pairwise overlap of alignments and makes alignments disjoint after all seeds have been extended. It does this by examining all pairs of alignments overlapping by more than the accepted threshold and successively removing equivalence classes from one or the other, at each step choosing the equivalence class resulting in the minimum decrease in score of either alignment. Græmlin also controls for alignment overlap during the extension phase by checking if the growing alignment overlaps any already extended alignments by more than the allowable threshold. If it does, it removes all nodes in the overlapped alignment from the frontier. For example, suppose high scoring alignment A contains 10 nodes and the overlap threshold is 25%. When Græmlin is extending alignment B, as soon as it includes in B 3 nodes that are present in A, Græmlin will remove all nodes from the frontier that are present in alignment A.

As a final note, Græmlin supports simultaneous search with multiple ESMs. It can do this by running each ESM independently, but it can also prioritize ESMs and ignore parts of the search space already containing high-scoring alignments found using higher priority ESMs. For example, Græmlin can first search with a Complex ESM and search with a Pathway ESM only parts of the networks that do not contain high-scoring protein complexes; this can decrease search times, particularly if there is a large complex that contains many high-scoring pathways.

A.3.3 Progressive multiple alignment

Given a set of interaction networks and a phylogenetic tree relating them, Græmlin builds alignments progressively using an extension of the pairwise alignment algorithm. It begins by performing a pairwise alignment of the two closest species in the tree, A and B. At the parent node, it constructs the following networks:

1. Network AB contains all high-scoring alignments of the child networks; in it, there is one node for each equivalence class in these alignments. In addition, it contains every unaligned node in the child networks that is connected to at least f nodes of a high-scoring alignment; f is a parameter that defaults to 3. This heuristic is meant to allow Græmlin to extend the pairwise alignments as it adds more species to the multiple alignment. Within network AB, we consider species A and B to be *present*.

2. Networks A' and B' contain all unaligned nodes within networks A and B, including those nodes that Græmlin appended to AB. Therefore, some nodes appear in AB as well as A' or B', but in practice only a small number of nodes are duplicated. Within network A', we consider only species A to be present, and within network B', only species B is present.

Græmlin then progressively works its way up the tree, at each node aligning 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 proceeds by performing 9 pairwise alignments. While the number of networks at an internal node increases exponentially in principle, 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, only the species that are considered present in the networks participate during scoring. For example, when aligning networks A' and CD, an equivalence class does not incur a protein deletion penalty for missing proteins from species B. In this way, 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 species that are not phylogenetically near one another.

A.4 Performance evaluation

A.4.1 Circularity of network construction and alignment

A.4.2 Complete results

Tables 1 through 4 show more detailed results of our performance tests.

A.4.3 Further biological results

Module analysis In addition to their identification, Græmlin has the potential to aid in detailed studies of conserved modules. Figure 4, which shows several alignments of the proteins *rfbA*, *rfbB*, *rfbC*, and *rfbD*, provides an example of this. These four proteins help catalyze the production of dTDP-L-rhamnose from D-glucose-1-phosphate and are involved in several metabolic pathways (Stevenson et al., 1994); Figure 4A shows the steps of this process. The alignment in Figure 4B results from performing a pairwise alignment of the proteins in *E. coli* and *S. typhimurium* using a Complex ESM. While it correctly aligns proteins to their orthologs, this alignment does not yield any information about the details of the pathway. On the other hand, the alignment in

Figure 4C, resulting from the use of a Pathway ESM, provides a rough ordering of the proteins. From the alignment, we can infer that proteins with labels that are adjacent in the ESM directly interact; while the resulting order is not entirely correct, the alignment predicts two out of the three interactions correctly. The alignment in Figure 4D, which results from a multiple alignment including *C. crescentus* and *Synechocystis*, recapitulates the order of the pathway exactly; in this case, the presence of more than two species provides additional information about edge conservation that reorders the pathway. While this is a simple example, it does indicate that Græmlin has use in detailed analyses of functional modules, both by its ability to search for pathways and to perform multiple alignments.

References

Breiman, L., 1996. Bagging predictors. *Machine Learning*, **24**(2):123–140.

Kanehisa, M. and Goto, S., 2000. KEGG: kyoto encyclopedia of genes and genomes. *Nucleic Acids Res*, **28**(1):27–30.

Kelley, B. P., Sharan, R., Karp, R. M., Sittler, T., Root, D. E., Stockwell, B. R., and Ideker, T., 2003. Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proc Natl Acad Sci U S A*, **100**(20):11394–11399.

Koyuturk, M., Grama, A., and Szpankowski, W., 2005. Pairwise local alignment of protein interaction networks guided by models of evolution. *Lecture Notes in Bioinformatics (RECOMB)*, **3500**(9):48–65.

Li, M., Ma, B., Kisman, D., and Tromp, J., 2003. PatternHunter II: highly sensitive and fast homology search. *Genome Inform Ser Workshop Genome Inform*, **14**:164–175.

Ma, B., Tromp, J., and Li, M., 2002. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, **18**(3):440–445.

Pellegrini, M., Marcotte, E. M., Thompson, M. J., Eisenberg, D., and Yeates, T. O., 1999. Assigning protein functions by comparative genome analysis: protein phylogenetic profiles. *Proc Natl Acad Sci U S A*, **96**(8):4285–4288.

Sharan, R., Ideker, T., Kelley, B., Shamir, R., and Karp, R. M., 2005. Identification of protein complexes by comparative analysis of yeast and bacterial protein interaction data. *J Comput Biol*, **12**(6):835–846.

Srinivasan, B. S., Caberoy, N. B., Suen, G., Taylor, R. G., Shah, R., Tengra, F., Goldman, B. S., Garza, A. G., and Welch, R. D., 2005. Functional genome annotation through phylogenomic mapping. *Nat Biotechnol*, **23**(6):691–698.

Srinivasan, B. S., Novak, A., Flannick, J., Batzoglou, S., and McAdams, H. H., 2006a. Integrated protein interaction networks for 11 microbes. *Proceedings of the 10th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2006)*, **In Press**.

Srinivasan, B. S., Novak, A. F., Flannick, J. A., Batzoglou, S., and McAdams, H. H., 2006b. Network Integration Reveals Hidden Biology in 312 Microbes. *(Submitted)*, .

Stevenson, G., Neal, B., Liu, D., Hobbs, M., Packer, N. H., Batley, M., Redmond, J. W., Lindquist, L., and Reeves, P., 1994. Structure of the O antigen of *Escherichia coli* K-12 and the sequence of its rfb gene cluster. *J Bacteriol*, **176**(13):4144–4156.

Sun, Y. and Buhler, J., 2005. Designing multiple simultaneous seeds for DNA similarity search. *J Comput Biol*, **12**(6):847–861.

X_i	Annotation
CC1	0025,0030
CC2	0025,0040
CC3	0050
CC4	-

→

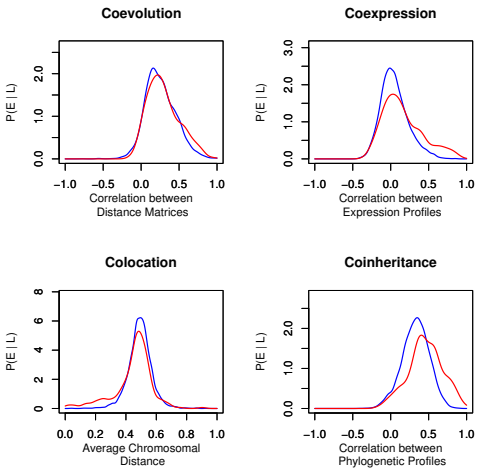
X_i	X_j	$L(X_i, X_j)$
CC1	CC2	1
CC1	CC3	0
CC1	CC4	?
CC2	CC3	0
CC2	CC4	?
CC3	CC4	?

L=1 (shared functional category)

L=0 (different functional category)

L=? (unknown linkage)

(a) Training Set Generation



(b) Evidence vs. Training Set

Figure 1: Training Sets and Evidence. (a) Genome-scale systematic annotations such as COG, GO or KEGG give functions for proteins X_i. As described in the text and shown on example data, we use this annotation to build an initial classification of protein pairs (X_i, X_j) with three categories: a relatively small set of likely linked (red) pairs and unlinked (blue) pairs, and a much larger set of uncertain (gray) pairs. (b) We observe that proteins which share an annotation category generally have more significant levels of evidence, as seen in the shifted distribution of linked (red) vs. unlinked (blue) pairs. Even subtle distributional differences contribute statistical resolution to our algorithm.

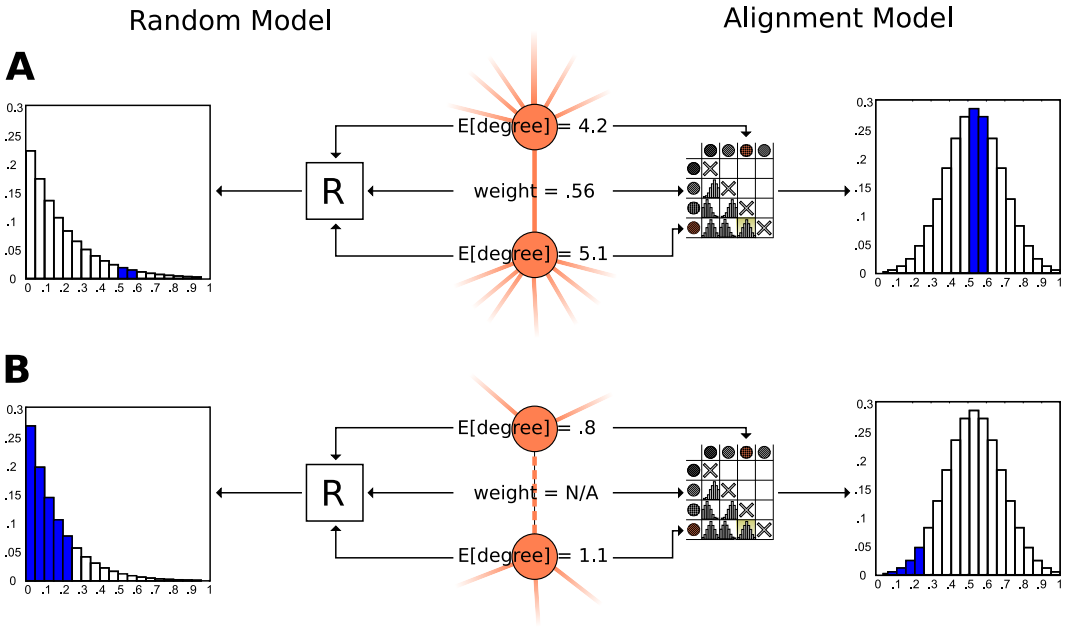


Figure 2: **The general methodology for scoring a single edge in an alignment of thresholded networks. A.** Græmlin scores an edge of moderate weight by integrating both the random model distribution function and the alignment model distribution function over a small area centered at the edge weight. 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.** The procedure for scoring an edge with weight below the minimum edge weight threshold is similar, but Græmlin now integrates both distributions from 0 to the threshold below which weights are discarded.

```
ExtendSeed(d_1, d_2 : pair of nodes )
 1  var A ← ∅ //the alignment
 2  var F ← ∅ //the frontier
 3   Create an equivalence class containing d_1 and d_2, add to A
 4  while true
 5  do for  nodes n in A
 6       do  Ensure n and all neighbors of n are in F
 7       (Δ_p, A'_p) ← max_{n_1,n_2∈F}  DeltaPair (n_1, n_2, A)
 8       (Δ_s, A'_s) ← max_{n∈F}  DeltaSingleton (n, A)
 9       if Δ_p > 0 or Δ_s > 0
10          then if Δ_p > Δ_s
11                  then A ← A'_p
12                  else  A ← A'_s
13          else  break
14  return A

DeltaPair (n_1, n_2 : pair of proteins , A : alignment )
 1  var Δ //the change in score of aligning n_1 and n_2 in A
 2  var A' //the alignment resulting from aligning n_1 and n_2
 3  switch
 4     case n_1 ∈ A and n_2 ∈ A :
 5           [n'] := [n_1] ∪ [n_2]
 6           A' := A − [n_1] − [n_2] + [n']
 7     case n_1 ∈ A and n_2 ∉ A :
 8           [n'] := [n_1] ∪ n_2
 9           A' := A − [n_1] + [n']
10     case n_1 ∉ A and n_2 ∈ A :
11           [n'] := [n_2] ∪ n_1
12           A' := A − [n_2] + [n']
13     case n_1 ∉ A and n_2 ∉ A :
14           [n'] := n_2 ∪ n_1
15           A' := A + [n']
16  Δ = Score(A') − Score(A)
17  return (Δ, A')

DeltaSingleton (n : protein , A : alignment )
1  var Δ //the change in score of adding n to A
2  var A' //the alignment resulting from adding n to A
3  A' := A + [n]
4  Δ = Score(A') − Score(A)
5  return (Δ, A')
```

Figure 3: **Pseudocode for extending a seed.** The input to the procedure is an aligned pair of nodes, which Græmlin obtains by aligning the two highest scoring nodes from a pair of d-clusters. The procedure makes use of two utility functions, which compute the score of adding a pair of proteins, or a single protein, to the growing alignment.

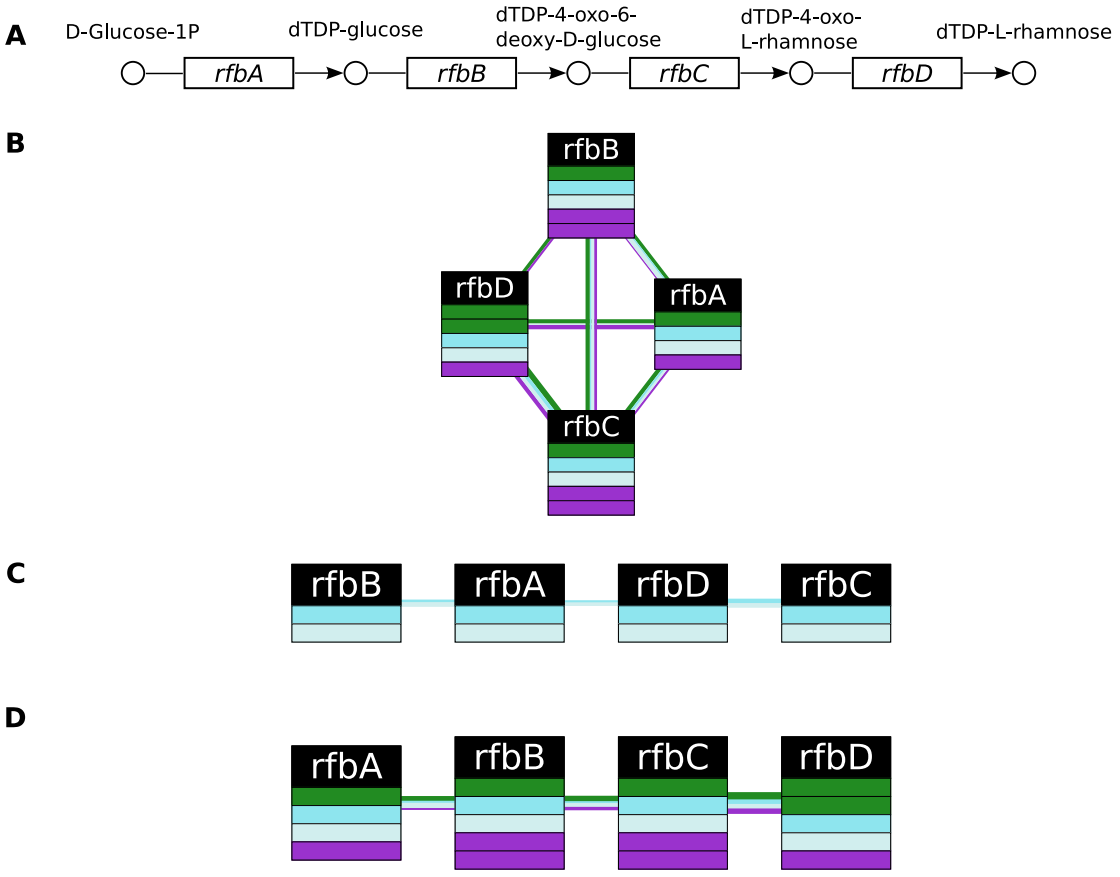


Figure 4: **An alignment of the *rfb* gene cluster.** **A.** The reference KEGG pathway (Kanehisa and Goto, 2000). **B.** Using a Complex ESM, Græmlin covers the entire pathway but provides no information about the ordering of the proteins in the pathway. **C.** Although not completely correct, the use of a Pathway ESM gives a rough ordering of the nodes, grouping *rfbA*and *rfbB* as neighbors as well as *rfbC* and *rfbD*. **D.** When Græmlin extends the alignment to four species, the ordering of the proteins is completely correct. For clarity, C and D show only the edges between adjacent proteins.

		KEGGs Hit	KEGG Coverage	Alignments Enriched	Nodes Aligned	Align-ments	Nodes Misaligned	Running Time
E. coli vs. *C. crescentus*								
MaWISh		11 (20%)	37 %	86 %	378	24	43 %	12 s
Græmlin	Pathway	26 (47%)	61 %	78 %	839	59	27 %	116 s
	Complex	31 (56%)	60 %	81 %	1191	82	37 %	66 s
E. coli vs. *M. tuberculosis*								
MaWISh		16 (27%)	33 %	82 %	397	56	41 %	14 s
Græmlin	Pathway	24 (40%)	45 %	85 %	793	79	30 %	61 s
	Complex	29 (48%)	50 %	79 %	888	71	33 %	47 s
E. coli vs. *V. cholerae*								
MaWISh		32 (59%)	49 %	81 %	1556	57	13 %	13 s
Græmlin	Pathway	32 (59%)	54 %	71 %	1684	101	12 %	159 s
	Complex	38 (70%)	64 %	82 %	2058	113	14 %	164 s
E. coli vs. *S. coelicolor*								
MaWISh		N/A	N/A	N/A	N/A	N/A	N/A	N/A
Græmlin	Pathway	17 (30%)	62 %	82 %	1105	68	49 %	2,976 s
	Complex	25 (44%)	63 %	82 %	1729	96	55 %	1,224 s

Table 1: **Results on pairwise alignment of complete networks thresholded at** 0.25**.** For each pair of species, we performed complete network-to-network alignment using MaWISh and Græmlin. For each tested method, shown, from left, is the total number of KEGG pathways hit by an alignment, the fraction of KEGG pathways hit by an alignment, the average coverage of a KEGG pathway, the percentage of enriched alignments, the total number of aligned proteins, the total number of significant alignments returned, the fraction of aligned proteins that were 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.

		KEGGs Hit	KEGG Coverage	Alignments Enriched	Nodes Aligned	Align-ments	Nodes Misaligned	Running Time
E. coli vs. *C. crescentus*								
MaWISh		9 (20%)	32 %	72 %	346	67	35 %	3 s
NetworkBLAST	Pathway	6 (14%)	28 %	61 %	233	33	32 %	9,624 s
	Complex	12 (27%)	49 %	72 %	466	23	52 %	
Græmlin	Pathway	15 (34%)	47 %	68 %	594	55	26 %	21 s
	Complex	17 (39%)	45 %	67 %	677	58	27 %	11 s
E. coli vs. *M. tuberculosis*								
MaWISh		7 (13%)	20 %	85 %	202	33	28 %	3 s
NetworkBLAST	Pathway	7 (13%)	24 %	88 %	220	33	30 %	301 s
	Complex	7 (13%)	32 %	88 %	262	15	40 %	
Græmlin	Pathway	8 (15%)	36 %	89 %	454	58	27 %	11 s
	Complex	8 (15%)	39 %	89 %	518	65	23 %	8 s
E. coli vs. *V. cholerae*								
MaWISh		12 (31%)	35 %	64 %	819	131	13 %	3 s
NetworkBLAST	Pathway	10 (26%)	35 %	58 %	774	123	16 %	8,797 s
	Complex	11 (28%)	41 %	64 %	1044	48	16 %	
Græmlin	Pathway	19 (49%)	48 %	75 %	1342	115	12 %	13 s
	Complex	15 (38%)	55 %	74 %	1414	100	11 %	12 s
E. coli vs. *S. coelicolor*								
MaWISh		N/A	N/A	N/A	N/A	N/A	N/A	N/A
NetworkBLAST	Pathway	6 (14%)	23 %	46 %	381	2041	60 %	122,168 s
	Complex	10 (23%)	67 %	95 %	1637	26272	82 %	
Græmlin	Pathway	8 (19%)	58 %	88 %	947	85	47 %	734 s
	Complex	9 (21%)	59 %	85 %	1006	78	45 %	829 s

Table 2: **Results on pairwise alignment of complete networks thresholded at** 0.5**.** For each pair of species, we performed complete network-to-network alignment using MaWISh, NetworkBLAST, and Græmlin; the columns in this table are analogous to those in Table 1. As NetworkBLAST does not have an option to search separately for pathways and complexes, the table lists the combined running time of both searches.

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

Table 3: **Results on alignment of a query network to a database thresholded at** 0.25**.** For each pair of species, using MaWISh and Græmlin, we successively aligned each KEGG pathway in the query species to the complete network of the database species. For each tested method, 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 per query of the fraction of aligned proteins that were misaligned, and the total running time.

		KEGGs Hit	KEGG Coverage	Nodes Misaligned	Running Time
E. coli vs. *C. crescentus*					
MaWISh		15 (34%)	31 %	29 %	37 s
NetworkBLAST	Pathway	8 (18%)	32 %	23 %	3,453 s
	Complex	10 (23%)	49 %	36 %	
Græmlin	Pathway	20 (45%)	45 %	16 %	17 s
	Complex	20 (45%)	47 %	25 %	3 s
	Module	20 (45%)	48 %	23 %	23 s
C. crescentus vs. *E. coli*					
MaWISh		9 (20%)	32 %	15 %	130 s
NetworkBLAST	Pathway	10 (23%)	37 %	27 %	4,788 s
	Complex	10 (23%)	41 %	26 %	
Græmlin	Pathway	15 (34%)	39 %	23 %	6 s
	Complex	15 (34%)	42 %	19 %	5 s
	Module	15 (34%)	42 %	21 %	33 s
E. coli vs. *M. tuberculosis*					
MaWISh		10 (19%)	19 %	26 %	93 s
NetworkBLAST	Pathway	12 (22%)	23 %	15 %	3,947 s
	Complex	12 (22%)	29 %	18 %	
Græmlin	Pathway	17 (31%)	31 %	27 %	3 s
	Complex	17 (31%)	35 %	25 %	3 s
	Module	17 (31%)	35 %	24 %	22 s
M. tuberculosis vs. *E. coli*					
MaWISh		6 (11%)	12 %	11 %	138 s
NetworkBLAST	Pathway	10 (19%)	19 %	29 %	5,047 s
	Complex	7 (13%)	22 %	29 %	
Græmlin	Pathway	13 (24%)	25 %	13 %	5 s
	Complex	14 (26%)	26 %	13 %	5 s
	Module	14 (26%)	27 %	15 %	28 s

Table 4: **Results on alignment of a query network to a database thresholded at** 0.5**.** For each pair of species, using MaWISh, NetworkBLAST, and Græmlin, we successively aligned each KEGG pathway in the query species to the complete network of the database species; the columns in this table are analogous to those in Table 3. As NetworkBLAST does not have an option to search separately for pathways and complexes, the table lists the combined running time of both searches.