Refining Regulatory Networks through Phylogenetic Transfer of Information

Xiuwei Zhang and Bernard M.E. Moret

Abstract—The experimental determination of transcriptional regulatory networks in the laboratory remains difficult and time-consuming, while computational methods to infer these networks provide only modest accuracy. The latter can be attributed partly to the limitations of a single-organism approach. Computational biology has long used comparative and evolutionary approaches to extend the reach and accuracy of its analyses. In this paper, we describe ProPhyC, a probabilistic phylogenetic model and associated inference algorithms, designed to improve the inference of regulatory networks for a family of organisms by using known evolutionary relationships among these organisms. ProPhyC can be used with various network evolutionary models and any existing inference method. Extensive experimental results on both biological and synthetic data confirm that our model (through its associated refinement algorithms) yields substantial improvement in the quality of inferred networks over all current methods. We also compare ProPhyC with a transfer learning approach we design. This approach also uses phylogenetic relationships while inferring regulatory networks for a family of organisms. Using similar input information but designed in a very different framework, this transfer learning approach does not perform better than ProPhyC, which indicates that ProPhyC makes good use of the evolutionary information.

Index Terms—Regulatory networks, network inference, evolution, phylogenetic relationships, ancestral network, refinement, gene duplication, evolutionary model, evolutionary history, reconciliation, maximum likelihood, transfer learning.

1 INTRODUCTION

TRANSCRIPTIO NAL regulatory networks are models of the cellular regulatory system that governs transcription. Because establishing the topology of the network from bench experiments is very difficult and time-consuming, regulatory networks are commonly inferred from gene-expression data. Various computational models, such as Boolean networks [1], Bayesian networks [2], dynamic Bayesian networks (DBNs) [3], and differential equations [4], have been proposed for this purpose. Results, however, have proved mixed: the high-noise level in the data, the paucity of well-studied networks, and the many simplifications in the models all combine to make inference difficult, in terms of both accuracy and computation.

Bioinformatics has long used evolutionary approaches to improve the accuracy of computational analyses. Recent work on the evolution of regulatory networks has demonstrated the applicability of such approaches to regulatory networks. Although regulatory networks produced from bench experiments are available for only a few model organisms, other types of data have been used to assist in the comparative study of regulatory mechanisms across organisms. For example, gene-expression data [5], sequence data such as transcription factor binding site (TFBS) [6], [7], and cis-regulatory elements [5] have all been used in this context. Moreover, a broad range of model organisms have been studied, including bacteria [8], yeast [5], [6], and fly [7]. These studies have identified a number of evolutionary events, such as adding or removing network edges, and the duplication and loss of genes [8], [9], [10]. Results have also appeared on the evolution of metabolic networks and protein interaction networks [11], [12].

Phylogenetic relationships are well established for many groups of organisms; as the regulatory networks evolved along the same lineages, the phylogenetic relationships informed this evolution and so can be used to improve the inference of regulatory networks. Indeed, Bourque and Sankoff [13] developed an integrated algorithm to infer regulatory networks across a group of species whose phylogenetic relationships are known, under a simple parsimony criterion. In previous work [14], [15], we presented refinement algorithms, based on phylogenetic information and using a likelihood framework, that boost the performance of any chosen network inference method, hereafter called a base method. These refinement algorithms, RefineFast and RefineML, are two-step iterative algorithms. The networks to be refined are placed at the corresponding leaves of the known phylogeny. In the first step, ancestral networks for the phylogeny (strings labeling internal nodes) are inferred; in the second step, these ancestral networks are used to refine the leaf networks. These two steps are then repeated as needed. On both simulated and biological data, the receiver-operator characteristic (ROC) curves for our algorithms consistently dominated those of the base methods used alone. Under comparable conditions, RefineFast and RefineML also outperform the Bourque and Sankoff algorithm.

We present ProPhyC, a probabilistic phylogenetic model and associated algorithms, designed to refine regulatory networks for a family of organisms. The input for ProPhyC is the noisy regulatory networks to be refined for the family of organisms, and the output is the refined version of these networks. ProPhyC can accommodate a large variety of
evolutionary models of regulatory networks with only slight modifications, as we demonstrate in the results section. Given that the evolution of regulatory networks is not yet well understood and given the several different models for regulatory network evolution [6], [10], [13], such flexibility is highly desirable. We present algorithms and experimental results in this refinement model for two network evolutionary models: a basic model that includes only gains and losses of regulatory interactions, and an extended model that also accounts for duplications and losses of genes. We also show how to take advantage of position-specific confidence values, if any, assigned to the input networks by the base inference method. Extensive experiments show that ProPhyC model not only brings significant improvement to base network inference algorithms, but also dominates the performance of existing refinement algorithms.

To further analyze and evaluate the performance of ProPhyC, we compare it with an entirely different approach which also incorporates phylogenetic information for network inference, but works under a different scenario of input. It is a new method we designed called tree transfer learning (TTL). Whereas ProPhyC is a framework for refinement that takes the networks to be refined as input, TTL is a direct inference algorithm that uses both gene-expression data and phylogenetic relationships. It combines the concept of transfer learning [16], [17] with a phylogenetic tree, using the basic network evolutionary model. Throughout our experiments, ProPhyC dominates TTL, although the two often return comparable results. That different approaches reach similar accuracy under many settings suggests that ProPhyC (which, unlike TTL, does not have access to the gene-expression data) uses much of the phylogenetic information. However, TTL benefits from its direct access to the gene-expression data and has the advantage in the case of insufficient data.

2 Background

Our approach posits that the evolution of regulatory networks correlates strongly with the evolution of the respective organisms, so that independent network inference errors can be corrected by using the phylogenetic relationships between the networks. To test this approach we use a base network inference method to infer networks from gene-expression data, and use the inferred networks as input to our refinement algorithms. The refinement algorithms involve the inference of ancestral networks. When the extended network evolutionary model is in use, the gene contents of ancestral networks are unknown due to the gene duplication and loss events during evolution. So we need to determine the gene contents of ancestral networks before inferring their network interactions. One way to solve this is to use the reconciliation of gene trees and species tree. In this section we briefly introduce the base inference method we use and the reconciliation of gene trees and species tree.

2.1 Base Network Inference Methods

We chose dynamic Bayesian inference (DBI), the method devised for DBNs, as the base inference method in our experiments. When DBNs are used to model regulatory networks, an associated structure-learning algorithm is used to infer the networks from gene-expression data [3], [18]; so as to avoid overly complex networks, a penalty on graph structure complexity is usually added to the ML score, thereby reducing the number of false positive edges. In [14] we used a coefficient $k_p$ to adjust the weight of this penalty and studied different tradeoffs between sensitivity and specificity, yielding the optimization criterion $\log \Pr(D(G, \hat{\Theta}_G)) - k_p \#G \log N$, where $D$ denotes the data set used in learning, $G$ is the (structure of the) network, $\hat{\Theta}_G$ is the ML estimate of parameters for $G$, $\#G$ is the number of free parameters of $G$, and $N$ is the number of samples in $D$. We adapt Murphy’s Bayesian Network Toolbox [19] as the implementation for DBI.

2.2 Reconciliation of Species Tree and Gene Trees

To recover the gene contents of ancestral networks under the extended model, we need a full history of gene duplications and losses. We reconstruct this history by reconciling the gene trees and the species tree, that is, by using the differences between these trees to infer past duplication and loss events. While reconciliation is a hard computational problem, algorithms have been devised for it in a Bayesian framework [20] or using a simple parsimony criterion, as in the software Notung [21].

3 Models and Methods

In this section, we first present two network evolutionary models, then describe the ProPhyC refinement framework, and give associated refinement algorithms, one for each network evolutionary model. Finally we present the TTL algorithm, which we design to compare with ProPhyC, and to further test ProPhyC.

3.1 Network Evolutionary Models

We present a basic model and an extended model. In both models, the networks are represented by binary adjacency matrices. For the basic model, the evolutionary operations are: edge gain, in which an edge between two genes is generated with probability $p_{10}$, and edge loss, an existing edge is deleted with probability $p_{10}$. The model parameters are thus 1) the base frequencies of 0 and 1 entries in the given networks $\Pi = (\pi_0, \pi_1)$, and 2) the substitution matrix of 0s and 1s, $P = (p_{ij})$.

The extended model has two additional evolutionary operations, gene duplication and gene loss, with corresponding additional model parameters $p_d$ and $p_l$. In gene duplication, a gene is duplicated with probability $p_d$ after duplication, edges for the newly generated copy are assigned according to 1) neutral initialization, where the new copy gets connected to other genes randomly according to the proportion $\pi_1$ of edges in the background network; or 2) inheritance initialization, where the new copy inherits the connections of the original, then loses or gains connections at some fixed rate, following reports of strong correlations between the connections of the new copy and those of the original copy [8], [9], [10]. In gene loss, a gene is deleted along with all its connections with probability $p_l$.

3.2 The ProPhyC Framework

ProPhyC is a probabilistic phylogenetic model designed to refine the inferred (and error-prone) regulatory networks
for a family of organisms by making use of known phylogenetic information for the family. ProPhyC is also a graphical model: the phylogeny of this family is the main information to determine its structure as illustrated in Fig. 1. The shaded nodes labeled in upper case represent the input noisy networks, while the nodes labeled in lower case represent the correct networks for these organisms that we want to infer. In turn, the correct networks are the leaves of the rooted phylogenetic tree of these organisms, while internal nodes correspond to ancestral regulatory networks. The edges in this graph fall into two categories: 1) edges in the phylogenetic tree, representing the evolution from a parent network to a child network, and 2) edges from correct leaf networks to noisy ones, representing the error-prone process of inferring networks from latent correct networks. The parameters for this model are thus the transition matrices \( P \) and \( Q \), where \( P \) represents the transition parameters from an ancestral network to its child network—subject to the network evolutionary model—and \( Q \) represents the difference between the “true” networks and the inferred (observed, from the point of view of the ProPhyC model) noisy networks—associated with one’s confidence in the base network inference method.

The input information is thus the phylogenetic tree and its evolutionary model, and the noisy leaf networks and their noise model. With a dynamic programming algorithm to maximize the likelihood of the whole graph, we can infer the ancestral networks and the “true” leaf networks. These “true” leaf networks inferred are the refined networks for these organisms and the output of the refinement algorithm. The ProPhyC framework can easily be generalized to fit different network evolutionary models.

Some base inference methods can predict regulatory networks with different confidence on different edges or nonedges of the networks, so in this case \( Q \) can vary for different entries of different leaf networks. Our model can incorporate these position-specific confidence values to get better refinements. We developed an extension to ProPhyC to use these position-specific confidence values in the likelihood maximization process; we call this extension ProPhyCC with an additional “C” standing for confidence.

Our previous two-step iterative refinement algorithm also came in two versions, RefineFast and RefineML, with the latter usable only when confidence values are available. In this two-step setup, the likelihood maximization process occurs in the first step, and only involves the noisy networks and the evolutionary model, the consequence of which is that the ancestral networks are inferred from the noisy networks. To get the refined networks as output, we need the second step. In contrast, ProPhyC is designed within a probabilistic graphical model. Within this framework, the likelihood maximization involves both the evolutionary model and a noise model, where the ancestral networks and the refined networks are connected by the evolutionary model, and the noisy networks are connected to the refined networks by the noise model. This enables ProPhyC to appropriately integrate all the input information and the output together, and perform probabilistic inference once to get the output. During the inference, the information of all networks is transferred through its graphical structure, thus ProPhyC can make a good use of the phylogenetic information. In our experiments, ProPhyC always dominates RefineFast and ProPhyCC always dominates RefineML, while in most cases ProPhyC outperforms RefineML, despite the extra confidence value information used by RefineML.

### 3.3 ProPhyC under the Basic Model

Under the basic model, all networks have the same size and gene contents. Each network is represented by its binary adjacency matrix, so the character set is \( S = \{0, 1\} \). The parameters to calculate the likelihood are those from the evolutionary model, \( P \) and \( Q \), and the error parameter for the base inference method, \( Q = (q_{ij}) \). We assume independence between the network entries, so that we can process separately each entry in the adjacency matrices. Let \( i, j, k \) denote nodes in the tree and \( a, b, c \in S \) denote possible values of a character. For each character \( a \) at each node \( i \), we maintain two variables:

- \( L_i(a) \): the likelihood of the best reconstruction of the subtree with root \( i \), given that the parent of \( i \) is assigned character \( a \).
- \( C_i(a) \): the optimal character for \( i \), given that its parent is assigned character \( a \).

When the phylogenetic tree is binary, our inference algorithm works as follows:

1. For each leaf node \( i \), if its corresponding noisy network has character \( b \), then for each \( a \in S \), set \( L_i(a) = \max_{c \in S} L_j(c) \cdot q_{ab} \) and \( C_i(a) = \arg \max_{c \in S} L_j(c) \cdot q_{ab} \).
2. If \( i \) is an internal node and not the root, its children are \( j \) and \( k \), and it has not yet been processed, then for each \( a \in S \), set \( L_i(a) = \max_{c \in S} L_j(c) \cdot L_k(c) \) and \( C_i(a) = \arg \max_{c \in S} L_j(c) \cdot L_k(c) \).
3. If there remain unvisited nonroot nodes, return to Step 2.
4. If \( i \) is the root node, with children \( j \) and \( k \), assign it the value \( a \in S \) that maximizes \( \pi_a \cdot L_j(a) \cdot L_k(a) \).
5. Traverse the tree from the root, assigning to each node its character by \( C_i(a) \).

### 3.4 ProPhyC under the Extended Model

The extended model includes gene duplications and losses, so that the gene content may vary across networks. While the gene content of the leaf networks is known, we need to reconstruct the gene content for ancestral networks, that is,
to reconstruct the history of gene duplications and losses. This part can be solved by using an algorithm to reconcile the gene trees and species tree [20], [21], [22] or by the algorithms that we presented in earlier work under the duplication-only or loss-only model [23].

Under the basic model, we assumed independence among the entries of the adjacency matrices and so greatly simplified the computation. To enable us to do the same under the extended model, we embed each network into a larger one that includes every gene that appears in any network. We then represent a network with a ternary adjacency matrix, where the rows and columns of the network. We then represent a network with a ternary larger one that includes every gene that appears in any network. Under the extended model, we embed each network into a evolutionary step, we have gene duplication and one gene loss can happen at each introducing new parameters. Assuming that at most one evolutionarily step, we have

\[ P' = \begin{pmatrix}
  p_{00}' & p_{01}' & p_{02}' \\
  p_{10}' & p_{11}' & p_{12}' \\
  p_{20}' & p_{21}' & p_{22}'
\end{pmatrix}
\]

\[ = \begin{pmatrix}
  (1 - p_l) \cdot p_{00} & (1 - p_l) \cdot p_{01} & p_l \\
  (1 - p_l) \cdot p_{10} & (1 - p_l) \cdot p_{11} & p_l \\
  p_d \cdot p_{00} & p_d \cdot p_{01} & 1 - p_d
\end{pmatrix}.
\]

We also extend the parameter \( Q \) to be \( Q' \) to fit the new character set \( S' \)

\[ Q' = \begin{pmatrix}
  q_{00}' & q_{01}' & q_{02}' \\
  q_{10}' & q_{11}' & q_{12}' \\
  q_{20}' & q_{21}' & q_{22}'
\end{pmatrix} = \begin{pmatrix}
  q_{00} & q_{01} & 0 \\
  q_{10} & q_{11} & 0 \\
  0 & 0 & 1
\end{pmatrix}.
\]

The transition probabilities in \( Q' \) remain the same as in \( Q \), since the gene contents of the “true” and corresponding noisy network are the same. For each character \( a \) at each tree node \( i \), we calculate \( L_i(a) \) and \( C_i(a) \) for each site with the following procedure:

1. For each leaf node \( i \), if its corresponding noisy network has character \( b \), then for each \( a \in S' \), set
   \[ L_i(a) = \max_{c \in S} p_{ac} \cdot q_{cb}. \]
   and \( C_i(a) = \max_{c \in S} p_{ac} \cdot q_{cb}. \)

2. If \( i \) is an internal node and not the root, its children are \( j \) and \( k \), and it has not yet been processed, then
   a. if \( i \) has character \( x \), for each \( a \in S' \), set \( L_i(a) = p_{ax} \cdot L_j(x) \cdot L_k(x) \) and \( C_i(a) = x; \)
   b. otherwise, for each \( a \in S' \), set \( L_i(a) = \max_{c \in S} p_{ac} \cdot L_j(c) \cdot L_k(c) \) and \( C_i(a) = \arg \max_{c \in S} p_{ac} \cdot L_j(c) \cdot L_k(c) \).

3. If there remain unvisited nonroot nodes, return to Step 2.

4. If \( i \) is the root node, with children \( j \) and \( k \), assign it the value \( a \in S \) that maximizes \( \pi_a \cdot L_j(a) \cdot L_k(a) \), if the character of \( i \) is not already identified as \( x \).

5. Traverse the tree from the root, assigning to each node its character by \( C_i(a) \).

### 3.5 Refinement Algorithm ProPhyCC Using Confidence Values

Parameter \( Q \) (or \( Q' \)) models the errors introduced in the base inference process; its values are obtained from one’s confidence in that method and in the source data. The ProPhyCC algorithm uses the same matrix for all entries in all leaf networks. When sufficient information is available to produce different confidence values for different entries in different networks, we can take advantage of the extra information through the ProPhyCC algorithm.

If the noisy networks are predicted from gene-expression data by DBN models, to obtain the confidence values, we first estimate the conditional probability tables (CPTs) of the DBN inferred networks from the gene-expression data on the inferred structure [24], and then calculate the confidence values from the CPTs. Following [18], we use binary gene-expression levels in our experiments, where 1 and 0 indicate the gene is, respectively, on and off. For each gene \( y_i \), if \( m_i \) nodes have arcs directed to \( y_i \) in the network, let the expression levels of these nodes be denoted by the vector \( y = y_1 y_2 \cdots y_m \), and the confidence values of their arcs by the vector \( c = c_1 c_2 \cdots c_m \). We use signed weights to represent the strength of these arcs, denoted by \( w = w_1 w_2 \cdots w_m \). Considering that if an arc is predicted with high weight, then this arc is very likely to be true, we assign high-confidence values to the arcs predicted with high-absolute weight values. Let \( k \) be a coefficient value to normalize probabilities, we have \( k \cdot w \cdot y = Pr(y_i \text{ is on}|y) \). Since there are \( 2^m \) configurations of \( y \), there are \( 2^m \) such equations. The value of \( Pr(y_i \text{ is on}|y) \) can be directly taken from the CPTs. So \( w \) can be obtained by solving these equations, and \( c \) derived directly from \( w \).

### 3.6 The Tree Transfer Learning Algorithm

TTL is designed to infer the regulatory networks for a family of organisms directly from gene-expression data and all in one step. The TTL approach adapts the concept of transfer learning, which is to learn multiple related tasks simultaneously while applying the relationships among the tasks. In the case of TTL, the “multiple tasks” are the network inference for multiple organisms, and the “relationships among the tasks” are the phylogenetic relationships among these organisms.

The TTL approach is illustrated in Fig. 2. Define a configuration \( G = \{ G_1, G_2, \ldots, G_n \} \) as a set of networks for the leaf organisms; the goal of TTL is to find an optimal configuration \( G^* \). We define an optimization score called TTL score, \( S_{ttl} \), and an optimal configuration \( G^* \) is one that maximizes \( S_{ttl} \).

For each configuration \( G \), the TTL score \( S_{ttl} \) consists of two parts, the fitness of a configuration to the gene-expression data \( S_{data} \) and the score measuring how well the networks are related through the phylogenetic tree \( S_{tree} \). Denote the number of leaves in the phylogenetic tree as \( n_t \), the gene-expression data and the network structure for the \( i \)th leaf as \( D_i \) and \( G_i \), respectively, \( S_{data} \) is the sum of the Bayesian information criterion (BIC) score over all leaves.
where $\hat{G}_r$ is the ML estimate of parameters for $G_r$, $\hat{G}_s$ is the number of free parameters of $G_r$, $N_k$ is the number of samples in $D_k$, and $k_p$ is the penalty coefficient for network structure complexity.

Denote the adjacency matrices of the nodes in the tree as $M_1, M_2, \ldots, M_n$, the number of genes in a network as $n$, and the edges of the tree as $e_1, e_2, \ldots, e_n$, where $n_e$ is the number of edges in the tree; then $S_{tree}$ is calculated as follows:

$$S_{tree} = \sum_{i=1}^{n} \sum_{j=1}^{n} (\log \Pr(M_{root}(i,j) | \Pi) + \sum_{k=1}^{n_e} \log \Pr(M_{parent}(i,j), M_{child}(i,j) | P, e_k)),$$

where $M_{parent}$ and $M_{child}$ are, respectively, the adjacency matrices for the parent and the child networks at the current edge $e_k$. The adjacency matrices for all tree nodes can be obtained while generating the configuration from the root network. Having $S_{data}$ and $S_{tree}$, we can get $S_{ttl}$ by

$$S_{ttl} = S_{data} + k_s \cdot S_{tree},$$

where $k_s$ is the coefficient to adjust the weights for $S_{data}$ and $S_{tree}$.

Since searching in the space of all configurations to find $G^*$ is computationally too expensive, we use the phylogenetic relationships between the leaf networks to reduce the searching space. The strategy is: instead of searching in the space of possible structures of the root network. For each root structure, we generate $n_e$ configurations according to the network evolutionary model, and we choose as $G^*$ the configuration which gives the best TTL score among those generated by all root structures.

We assume that the regulator set for each gene is independent of those of other genes, so in practice we can determine the incoming edges for one gene at a time, and assemble the incoming edges for all genes to get the final networks. That is, for each gene $g$, we find the best configuration of the incoming edges to $g$ over all leaf networks, which we denote as $G^*_g$. The corresponding TTL score for a configuration $G_g$ is denoted as $S_{ttl}^g$. Therefore, with the above definition of the TTL score, the TTL algorithm is shown in Algorithm 1.

**Algorithm 1.** The TTL algorithm

**for** each gene $g$ in the network **do**

$S_{max} \leftarrow -\infty$

**for** each set of incoming edges for $g$ in the root network $G^*_g$ **do**

Generate $n_e$ configurations of incoming edges for $g$ in the leaf networks according to the network evolutionary model;

**for** each configuration $G_g = \{G^*_1, G^*_2, \ldots, G^*_n\}$ **do**

Calculate the current score $S_{ttl}^g$;

if $S_{ttl}^g > S_{max}$ **then**

$S_{max} \leftarrow S_{ttl}^g$;

$G^*_g \leftarrow G^*_g$;

**end if**

**end for**

**end for**

Assemble the $G^*_g$ for all $g$ to get $G^*$.

### 4 Experimental Design

We design a comprehensive collection of experiments to assess our ProPhyC model and its associated algorithms. The accuracy of the output is calculated by comparing the output with the “true” networks for the chosen family of organisms, where the “true” networks are either obtained through simulation or collected from biological data sets.

As a first indicator of the performance of ProPhyC, we design a preliminary comparison between ProPhyC and a previous refinement algorithm RefineFast with simulated networks. On given phylogenetic trees, we evolve networks from a root network along the edges of the phylogenetic tree according to the basic network evolutionary model to obtain networks for modern organisms, which we take as the “true” regulatory networks for these organisms. To get the noisy networks used as input to our refinement methods, we randomly pick entries in the adjacency matrices of the true networks and reverse the values to get erroneous networks. We then apply ProPhyC and RefineFast on these noisy networks and compare the networks refined by these two methods.

Since regulatory networks are usually reconstructed from gene-expression data, we follow the same path in most of our experiments. We compare the accuracies of the networks produced by the base method DBI and of the networks after refinement with ProPhyC and ProPhyCC, to get absolute assessments. We also use our previous refinement algorithms RefineFast and RefineML [14], [15], [23] to refine the same networks and compare the outcome with that of our new refinement model, to get relative assessments. Finally we compare ProPhyC and TTL, using the same gene-expression data sets for TTL and DBI which infers networks for ProPhyC.

Furthermore, we perform experiments with different combinations of network evolutionary models and types of gene-expression data sets. Under each setting, we show both absolute and relative assessments.
4.1 Biological Data Collection

Transcription factor binding site data are used to study regulatory networks, assuming that the regulatory interactions determined by transcription factor binding share many properties with the real interactions [6], [7], [25]. Given this close relationship between regulatory networks and TFBSs and given the large amount of available data on TFBSs, we chose to use TFBS data to derive regulatory networks for the organisms as their “true” networks. The TFBS data are drawn from the work of Kim et al. [26], where the TFBSs are annotated for the *Drosophila* family (whose phylogeny is well studied) with 12 species. They reported TFBS annotations for seven transcription factors on 51 cis-regulatory modules (CRMs) for all 12 species. Since each CRM corresponds to a target gene, we get a regulatory network with 58 nodes for each organism as the “true” network for this organism. We add noise into these “true” networks to obtain noisy networks as input to our refinement algorithm.

4.2 Data Simulation

In simulation experiments, we generate gene-expression data from simulated leaf networks. This step helps in decoupling the generation and the reconstruction phases. The data simulation procedure consists of two main steps: 1) generate the “true” leaf networks according to the evolutionary model and 2) generate the gene-expression data. The whole process starts from three pieces of information: the phylogenetic tree, the network at its root, and the evolutionary model. Since we need quantitative relationships in the networks in order to generate gene-expression data from each network, in the network generation process, we use adjacency matrices with signed weights.

We take specific precautions against systematic bias during data simulation and result analysis. We use a wide variety of phylogenetic trees from the literature (of modest sizes: between 20 and 60 taxa) and several choices of root networks; the latter variations on part of the yeast network from the KEGG database [27]. The root network has between 14 and 17 genes, a relatively easy case for inference from the KEGG database [27], [28]. The root networks are drawn from the work of Kim et al. [26], where the TFBSs are annotated for the *Drosophila* family (whose phylogeny is well studied) with 12 species. They reported TFBS annotations for seven transcription factors on 51 cis-regulatory modules (CRMs) for all 12 species. Since each CRM corresponds to a target gene, we get a regulatory network with 58 nodes for each organism as the “true” network for this organism. We add noise into these “true” networks to obtain noisy networks as input to our refinement algorithm.

4.2.1 Simulating Networks

Denote the weighted adjacency matrix of the root network as $M_P$. Under the basic model, we obtain the adjacency matrix for its child $M_c$ by mutating $M_P$, according to the substitution matrix. By repeating this process as we traverse down the tree we obtain weighted adjacency matrices at the leaves. In other words, we evolve the weighted networks down the tree according to the model parameters, following standard practice in the study of phylogenetic reconstruction [28], [29]. Under the extended model, to get the adjacency matrix for the child network of $M_P$, we follow two steps: evolve the gene contents and evolve the regulatory connections. First, genes are duplicated or lost by $p_d$ and $p_l$. If a duplication happens, a row and column for this new copy will be added to $M_P$, the values initialized either according to the neutral initialization model or the inheritance initialization model. Call this intermediate adjacency matrix $M'$. Now edges in $M'$ are mutated according to $p_{01}$ and $p_{10}$ to get $M_c$. Again we repeat this process as we traverse down the tree to obtain weighted adjacency matrices at the leaves.

4.2.2 Generating Gene-Expression Data

From the “true” networks, we use DBNSim [14], based on the DBN model, to generate time-series gene-expression data. Note that, while DBNSim and DBI are both based on Bayesian networks, which might artificially improve the performance of DBI, this bias can only make it more difficult for ProPhyC to achieve significant improvements.

For all experiments on simulated gene-expression data, we run the generation process 10 times for each choice of tree structure and parameters to compute a mean and a standard deviation. Under the basic model, for each leaf network, we generate 200 time points for its gene-expression matrix. Under the extended model, we generate $13 \times n$ time points for a leaf network with $n$ genes, since larger networks generally need more samples to gain inference accuracy comparable to smaller ones.

4.3 Tests with Biased Leaves

In biology, it is usually the case that in one family, with available data and knowledge, we can get relatively high-quality networks for only a few organisms, while a majority of organisms have poor quality networks due to lack of data and study. This forms a special case for our ProPhyCC algorithm: some input leaf networks have significantly higher confidence values than others. Here we test how ProPhyCC performs when there are only a small number of “good” networks in the input.

We simulate the noisy leaf networks as input to the ProPhyCC algorithm, where a proportion of them have higher noise rate than others. Starting from a root network and a phylogenetic tree, we simulate the evolution according to the basic model, and get the “true” leaf networks. With a fixed number of “good” leaves, we randomly choose the set of “good” leaves. Then we add noise to the “true” leaf networks according to their error rates to get biased noisy leaves. ProPhyCC is then applied to refine these leaf networks, with the confidence values derived from the error rates. In particular, we investigate the case where the specificity is worse than sensitivity in the networks with high noise, since in reality there are usually a large number of false positives in the noisy networks.

We test the performance of ProPhyCC with different numbers of “good” leaves. With each number, we choose different sets from all the leaves and get the average performance. With each chosen set, we also run the steps of adding noise and refinement multiple times to get average performance. Finally, each time we apply ProPhyCC we test the effect of using different parameters for ProPhyCC.

4.4 Comparing ProPhyC, ProPhyCC, and TTL

We compare the performance of ProPhyC, ProPhyCC, and TTL starting with simulated gene-expression data as input. The basic network evolutionary model is applied to all three algorithms. Experiments are conducted with a wide range of parameters for each algorithm to test their overall performance and robustness to parameter settings.

We also test whether combining ProPhyC and TTL will give better results than either ProPhyC or TTL. That is, in these experiments, we take the output networks of TTL and
use ProPhyC to refine these networks. We again apply various parameter settings on both TTL and ProPhyC: first TTL outputs multiple sets of leaf networks corresponding to multiple parameter settings, then each set is refined by ProPhyC using various parameters. The final performance is obtained by averaging over all the output sets from ProPhyC.

4.5 Measurements
We want to examine the predicted networks at different levels of sensitivity and specificity. With DBI, we use a penalty coefficient on structure complexity so as to obtain different tradeoffs between sensitivity and specificity. On each data set, we apply different penalty coefficients to predict regulatory networks, from 0 to 0.5, with an interval of 0.05, which results in 11 discrete coefficients. For each penalty coefficient, we apply our approach (and any method chosen for comparison) on the predicted networks, measure specificity and sensitivity, and plot the values into ROC curves. (In these ROC plots, the closer the curves are to the top left corner of the coordinate space, the better the results.)

5 RESULTS AND ANALYSIS
5.1 Preliminary Comparison with Simulated Networks
In these experiments, for both ProPhyC and RefineFast, we test a wide range of parameters (the substitution probabilities), and plot a point of \((1 - specificity)\) versus sensitivity for each parameter setting. Fig. 3 shows the results on a phylogenetic tree of 37 nodes on six levels. The cloud generated by ProPhyC consistently dominates that generated by RefineFast under various parameters. Within the ProPhyC framework, all ancestral networks, networks of modern organisms, and observed noisy networks are well integrated within the graphical model, and this allows us to take better advantage of the phylogenetic information than in our previous two-step approach.

5.2 Performance under the Basic Model on Simulated Data
5.2.1 Absolute Results
We show experimental results on two representative trees: one has 37 nodes on 7 levels and the other has 41 nodes on 6 levels. We only plot part of the curves within the 11 penalty coefficients to give a more detailed view of the comparison. Fig. 4 shows the results of ProPhyC and ProPhyCC on the networks predicted by DBI. We can see that ProPhyC and ProPhyCC significantly improve both sensitivity and specificity over the base inference algorithm DBI. The improvement remains similar on different tree structures. ProPhyCC further improves ProPhyC, which shows the advantage of using position-specific confidence values. For example, the dots in Fig. 4a marked by triangles correspond to the same penalty coefficient on the three curves, showing that, in going from DBI to ProPhyCC, the sensitivity increases from 77 to 86 percent while the specificity increases from 86 to 96 percent. Similar improvements can be observed with other trees, other evolutionary rates, and other base methods.

5.2.2 Relative Results
Fig. 5 shows the same experiments as in Fig. 4, but adds curves for RefineFast and RefineML to provide a comparison between different refinement approaches. Among the four refinement algorithms, ProPhyCC and RefineML take advantage of the position-specific confidence values, which gives them better performance than ProPhyC and RefineFast. ProPhyCC is obviously the best among all refinement algorithms, while ProPhyC outperforms RefineFast. From

![Fig. 3. Preliminary comparison of ProPhyC and RefineFast.](image)

(a) On the tree with 37 nodes and 7 levels  
(b) On the tree with 41 nodes and 6 levels

Fig. 4. Comparison of ProPhyC and ProPhyCC with base inference algorithm DBI under the basic model. In part (a), the dotted lines join data points for the same model penalty coefficient.
Figs. 4 and 5, we conclude that refinement algorithms under our new model outperform not only base inference algorithms, but also our previous refinement algorithms on simulated data.

5.3 Performance under the Basic Model on Biological Data

Here we show our results on the data sets for 12 species of Drosophila, whose phylogenetic tree is illustrated in Fig. 6. We use different noise rates to get noisy networks with different false positives and false negatives. Then for each set of noisy networks we use ProPhyC to obtain refined networks with different parameter settings. Fig. 7 shows the accuracies of these networks plotted as points. The cloud of points for ProPhyC clearly dominates that of the noisy networks, and the two clouds are well separated; the average improvement brought by ProPhyC is roughly 7 percent in each of sensitivity and specificity. In Fig. 8 we show three versions of the Drosophila melanogaster network in one of the runs: the “true” network, the noisy network with random noise, and the network refined by ProPhyC based on the noisy network.

By adjusting the penalty parameter, we can choose whether to emphasize sensitivity over specificity or the reverse, i.e., we can choose in which part of the ROC curve to operate; Table 1 gives some examples.

5.4 Results with Biased Leaves

We show the results of ProPhyCC refining the leaf networks with different noise rates. The tree we use here has 19 leaves and 7 levels. We test the number of “good” leaves from 1 to 19. With each number of “good” leaves, we randomly choose 100 sets of “good” leaves to get the average results. In the input networks, the “good” leaves have around 80 percent sensitivity and 80 percent specificity, while the “bad” leaves have 40 percent specificity and 60 percent sensitivity.

In Fig. 9, we show the results of ProPhyCC with two different parameter settings. We plot the specificity and sensitivity values of the “good” leaves and “bad” leaves separately, along with the increase of the number of “good” leaves. In Fig. 9a the parameter setting aims to improve both sensitivity and specificity. We can see that both the specificity and sensitivity for the high-noise leaves get improved even when there is only one good leaf, though for the good leaves their accuracy values become lower when there are very few of them. The accuracies represented by all the four solid lines increase along with the increase of the number of good leaves. With 6 good leaves out of 19 the specificity of good leaves improves. With 8 good leaves their sensitivity also improves. The specificity of high-noise leaves, which is the

<table>
<thead>
<tr>
<th>Species</th>
<th>Sensitivity</th>
<th>Specificity</th>
</tr>
</thead>
<tbody>
<tr>
<td>D. simulans</td>
<td>0.80</td>
<td>0.90</td>
</tr>
<tr>
<td>D. sechellia</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>D. melanogaster</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td>D. yakuba</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td>D. erecta</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>D. ananassae</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td>D. pseudoobscura</td>
<td>0.65</td>
<td>0.65</td>
</tr>
<tr>
<td>D. persimilis</td>
<td>0.60</td>
<td>0.60</td>
</tr>
<tr>
<td>D. willistoni</td>
<td>0.55</td>
<td>0.55</td>
</tr>
<tr>
<td>D. mojavensis</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>D. virilis</td>
<td>0.45</td>
<td>0.45</td>
</tr>
<tr>
<td>D. grimshawi</td>
<td>0.40</td>
<td>0.40</td>
</tr>
</tbody>
</table>

Fig. 9. The phylogeny connecting the 12 Drosophila species [30].
lowest measurement in the input networks, has the most significant improvement. These results show that only a very small number of good leaves can lead to significant improvement for the high-noise leaves.

Fig. 9b is obtained with a different parameter setting which favors sensitivity. Therefore, the sensitivity of both low-noise and high-noise leaves are much improved when there is only one good leaf, with loss of specificity of the low-noise leaves. With the increase of the number of good leaves, the two sensitivity values keep improving, while the specificity for the low-noise leaves soon approaches its original value, and that for the high-noise leaves grows even faster and still has the most improvement. All in all, these experimental results show the effectiveness of
ProPhyCC when the input networks are biased, especially its ability of improving the high-noise leaves with a small number of good leaves, which is the most likely scenario with biological data.

5.5 Performance under the Extended Model on Simulated Data

In evaluating performance under the extended model, we must first consider the effect of the first phase, in which the history of gene duplications and losses is reconstructed. In [23] we analyzed various duplication-loss history models and their effect on the performance of RefineFast and RefineML. Our experiments showed that accurate history information with reliable orthology assignments helps the refinement algorithms to get good performance. Here we test ProPhyC and ProPhyCC with two representative histories. One is the “true” history which is available in the framework of simulation experiments; with this history we can exclude the error introduced by the history inference step, and test purely the performance of the refinement algorithms. The other is the history inferred by gene tree and species tree reconciliation algorithms without any prior information. As the rates of gene duplication and loss during evolution can also affect the performance of refinement algorithms, we conduct simulation experiments with different rates of duplication and loss.

We run refinement algorithms with the two gene duplication and loss histories: the true history and the history reconstructed by Notung [21]. In these experiments, with reliable gene tree input, Notung correctly predicts duplication events (modulo changes in the networks), but usually misses recent loss events (it shows those events as happening earlier on the lineages). Furthermore, Notung not only infers the gene contents for ancestral networks, but also alters the gene contents of the leaves, which causes some difficulty for the refinement procedure.

We run refinement algorithms with the two gene duplication and loss histories: the true history and the history reconstructed by Notung [21]. In the following, we show results on one representative phylogenetic tree with 35 nodes and a root network of 15 genes. Since the results of using the neutral initialization model or the inheritance initialization model in data generation are very similar, we only show results with the former. For each experiment we show two plots: the left plot has relatively low rates of gene loss (resulting in 19 duplications and 15 losses along the tree on average), while the right one has significantly higher rates(with 20 duplications and 23 losses).

5.5.1 Absolute Results, with True History

We first compare the performance of ProPhyC and ProPhyCC with that of the base inference algorithm DBI, using the true history of duplications and losses. The results are shown in Fig. 10. Given the size of the tree and the root network, the rates of gene duplication and loss are quite high, yet the improvement gained by our refinement algorithms remains significant in both plots—almost as much as the improvement gained under the basic model shown in Fig. 4. ProPhyCC further dominates ProPhyC in both sensitivity and specificity, thanks to the appropriate use of the position-specific confidence values. Once again, we obtain similar improvements with other trees, other evolutionary rates, and other base methods.

5.5.2 Relative Results, with True History

Fig. 10 also shows the performance of RefineFast and RefineML on the same data sets. Although RefineFast and RefineML still clearly improve on DBI, the improvement is less pronounced than that with the basic model (Fig. 5). Gene duplications and losses give rise to a large overall gene population, yet many of them exist only in a few leaf networks; for these underrepresented genes, phylogenetic information is much reduced and so the refinement is less successful. RefineFast and RefineML are affected by this shortage, however, ProPhyC and ProPhyCC are more robust and easily outperform RefineFast and RefineML.

5.5.3 Absolute Results, with Inferred History

Here we use Notung to reconstruct the history of duplications and losses without any orthology input. In these experiments, with reliable gene tree input, Notung correctly predicts duplication events (modulo changes in the networks), but usually misses recent loss events (it shows those events as happening earlier on the lineages). Furthermore, Notung not only infers the gene contents for ancestral networks, but also alters the gene contents of the leaves, which causes some difficulty for the refinement procedure.

Fig. 11 shows the results of ProPhyC and ProPhyCC with Notung-reconstructed gene contents for the ancestral networks. We see that in Fig. 11a, the two ends of the ProPhyC curve have lost a little specificity while gaining sensitivity or vice versa, a tradeoff rather than an outright gain. However, ProPhyC dominates DBI through the useful range.
of specificity and sensitivity. In Fig. 11b, ProPhyC barely improves DBI, because the high rate of gene loss reduces the performance of refinement algorithms in two ways: first a high rate affects the performance of Notung (which does a poor job at inferring losses); second it increases the total population of genes and decreases the frequency of occurrence of an ortholog in the leaf networks, thus limiting the phylogenetic information. However, ProPhyCC still improves DBI significantly in both plots. Our probabilistic framework can incorporate the prior information in an appropriate way, so as to gain good performance even when the phylogenetic information, including the history of gene duplication and loss, is noisy and incomplete.

5.6 Comparison of ProPhyC, ProPhyCC, and TTL

Here we show the comparison of ProPhyC, ProPhyCC, and TTL. In these experiments, we use a phylogenetic tree of 37 nodes on 6 levels, and a small network size of 7 genes.

The two plots in Fig. 12 show the ROC curves of all three algorithms averaged over multiple runs and again, respectively, averaged over all parameter settings, with different sizes of gene-expression data. The left plot shows the results where five time points of gene-expression data are generated for each organism, while the right plot corresponds to 20 time points. Note that unlike the previous plots, in Fig. 12 the curves are plotted with full scale from 0 to 1 at both axes.

In Fig. 12a, comparing ProPhyC and TTL, we can see that the curves for ProPhyC and TTL are almost coincident, while in Fig. 12b the curve for ProPhyC slightly dominates that of TTL. The two plots together show that the transfer learning approach does not outperform ProPhyC. The observation that TTL performs better in Fig. 12a than in Fig. 12b relative to ProPhyC shows its advantage on small gene-expression data sets. This is because, with smaller data sets, the base inference algorithm (which infers a single network from the corresponding gene-expression data set) outputs networks of low quality; since ProPhyC takes these networks as input, its performance is affected by this limited input information. TTL, on the other hand, uses the gene-expression data sets for all leaf organisms when inferring their networks simultaneously, and the phylogenetic information is also applied at the same time to help obtain better prediction, which brings its overall performance to the level of ProPhyC on small data sets.
the performance of than its corresponding point on the latter curve. However, the former curve has both better sensitivity and specificity according to the mechanism of ProPhyC improve the performance of ProPhyC for the curve below. For example, in Fig. 12a, although the curve the points corresponding to the same penalty coefficients on the points marked on the dominating curve are better than from these figures is that, in some plots, it is not obvious that 13 are shown in full scale to show the AUC. One observation the accuracy of network inference. The plots in Figs. 12 and 13 are shown in full scale to show the AUC. One observation is that, in some plots, it is not obvious that there always exist some points on the TTL curve to its upper left. Similar patterns can be observed occasionally in some of the previous plots from Figs. 4 to 11, with the curves for DBI and ProPhyC in Fig. 11a as an example.

6 CONCLUSIONS

We described ProPhyC, a probabilistic phylogenetic model designed to improve the inference of regulatory networks for a family of organisms by using the phylogenetic relationships among these organisms. This model and its associated refinement algorithms can easily be adapted to work with different network evolutionary models. We conducted experiments on both simulated and biological data to test the performance of the refinement algorithms. With both the basic and extended network evolutionary models, the corresponding versions of ProPhyC and ProPhyCC outperformed those of our previous algorithms RefineFast and RefineML, and all four refinement algorithms outperformed the base inference algorithm. The improvement of ProPhyC and ProPhyCC over RefineFast and RefineML was more significant under the extended model, where the performance of RefineFast and RefineML was affected by the decrease of the phylogenetic information for each ortholog, while ProPhyC and ProPhyCC were hardly influenced. Our probabilistic phylogenetic model is thus quite robust against changes in these network evolutionary models.

We also designed Tree Transfer Learning (TTL), an approach based on inductive transfer learning, which applies the phylogenetic information as it infers the leaf networks. Designed in a very different framework, TTL was compared with ProPhyC and ProPhyCC over a range of parameters. Under various conditions, TTL approached the performance of ProPhyC, but did not outperform it, which again verified the strength of ProPhyC in integrating the phylogenetic information in its probabilistic graphical model. ProPhyCC performed better than the other two, which showed that ProPhyCC not only exploits the phylogenetic information, but also takes advantage of prior information, so as to get the best networks with the information available.
More generally, both ProPhyC and TTL can be viewed as methods for knowledge transfer, using a phylogenetic approach, among multiple organisms. Such a transfer of knowledge using phylogeny goes beyond the widespread comparative approach which involves only pairwise comparisons.

In both the basic and extended network evolutionary models, every interaction in a network has the same probability of being gained or lost. This is a simplification because in reality different regulatory interactions can have different evolutionary rates. However, this simplification does not affect the effectiveness of the ProPhyC refinement model, because: 1) the model can be easily generalized to use different rates for different interactions, or even learn the rates during refinement; 2) the refinement model is robust to deviation of evolutionary rates, that is, it does not require the “true” rates to gain improvement.

In future work, the evolutionary models can be generalized to take into account the dependency between the evolution of different regulatory interactions. For example, the regulatory interactions associated with the same transcription factor (TF) or the TF. Furthermore, our model can be extended to incorporate the evolution of both regulatory networks and binding sites.

ACKNOWLEDGMENTS

Xiwei Zhang would like to thank Dr. Jaebum Kim for providing much of the data we used for the Drosophila networks. A preliminary version of this paper appeared in the Proceedings of the Seventh International Symposium on Bioinformatics Research and Applications (ISBRA’11) [31].

REFERENCES

Xiuwei Zhang received the MS and BS degrees in computer science, respectively, from Tsinghua University and Dalian Jiaotong University in China and the PhD degree in computer science from EPFL. She is a postdoctoral researcher in Laboratory for Computational Biology and Bioinformatics (LCBB) at Swiss Federal Institute of Technology in Lausanne (EPFL), Switzerland. Her research interests include modeling and algorithm design in computational biology.

Bernard M.E. Moret received the PhD degree in 1980 from the University of Tennessee and was on the faculty of the Department of Computer Science at the University of New Mexico until 2006, serving as the chairman from 1991 until 1993. He is a professor of computer science, holding the Chair of Bioinformatics at the EPFL, the Swiss Federal Institute of Technology in Lausanne, Switzerland. His research interests include the area of algorithms and applications, particularly in computational molecular biology. He founded the ACM Journal of Experimental Algorithmics in 1995, serving as its editor-in-chief for seven years. Since 2000, he has focused on the development of models and algorithms for evolutionary genomics, publishing around 80 peer-reviewed articles in the area and founding, in 2001, the Annual Workshop on Algorithms in Bioinformatics (WABI).

> For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/publications/dlib.