Transcriptional Regulatory Networks across Species: Evolution, Inference, and Refinement

THÈSE N^{O} 5145 (2011)

PRÉSENTÉE LE 2 SEPTEMBRE 2011

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

LABORATOIRE DE BIOLOGIE COMPUTATIONNELLE ET BIOINFORMATIQUE

PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Xiuwei ZHANG

acceptée sur proposition du jury:

Prof. E. Telatar, président du jury
Prof. B. Moret, directeur de thèse
Prof. M. Kellis, rapporteur
Prof. M. Seeger, rapporteur
Dr S. Teichmann, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2011

Abstract

The determination of transcriptional regulatory networks is key to the understanding of biological systems. However, the experimental determination of transcriptional regulatory networks in the laboratory remains difficult and time-consuming, while current computational methods to infer these networks (typically from gene-expression data) achieve only modest accuracy.

The latter can be attributed in part to the limitations of a single-organism approach. Computational biology has long used comparative and, more generally, evolutionary approaches to extend the reach and accuracy of its analyses. We therefore use an evolutionary approach to the inference of regulatory networks, which enables us to study evolutionary models for these networks as well as to improve the accuracy of inferred networks. Since the regulatory networks evolve along with the genomes, we consider that the regulatory networks for a family of organisms are related to each other through the same phylogenetic tree. These relationships contain information that can be used to improve the accuracy of inferred networks. Advances in the study of evolution of regulatory networks provide evidence to establish evolutionary models for regulatory networks, which is an important component of our evolutionary approach. We use two network evolutionary models, a *basic* model that considers only the gains and losses of regulatory connections during evolution, and an *extended* model that also takes into account the duplications and losses of genes.

With the network evolutionary models, we design refinement algorithms to make use of the phylogenetic relationships to refine noisy regulatory networks for a family of organisms. These refinement algorithms include: *RefineFast* and *RefineML*, which are two-step iterative algorithms, and *ProPhyC* and *ProPhyCC*, which are based on a probabilistic phylogenetic model. For each algorithm we first design it with the basic network evolutionary model and then generalize it to the extended evolutionary model. All these algorithms are computationally efficient and are supported by extensive experimental results showing that they yield substantial improvement in the quality of the input noisy networks. In particular, *ProPhyC* and *ProPhyCC* further improve the performance of *RefineFast* and *RefineML*.

Besides the four refinement algorithms mentioned above, we also design an algorithm based on transfer learning theory called tree transfer learning (*TTL*). *TTL* differs from the previous four refinement algorithms in the sense that it takes the gene-expression data for the family of organisms as input, instead of their inferred noisy networks. *TTL* then learns the network structures for all the organisms at once, meanwhile taking advantage of the phylogenetic relationships. Although this approach outperforms an inference algorithm used alone, it does not perform better than *ProPhyC*, which indicates that the *ProPhyC* framework makes good use of the phylogenetic information.

keywords: regulatory networks, network inference, evolution, phylogenetic relationships, ancestral network, refinement, gene duplication, evolutionary model, evolutionary history, reconciliation, orthology, maximum likelihood, transfer learning

Résumé

La détermination des réseaux de contrôle transcriptionnel est essentielle à la compréhension des systèmes biologiques. Cependant, la détermination expérimentale des réseaux de contrôle transcriptionnel dans le laboratoire reste difficile et laborieuse, tandis que les méthodes computationnelles pour ces réseaux (généralement à partir des données d' expression génique) n'atteignent qu'une précision modeste. Cette dernière peut être attribuée en partie à des limitations d'une approche restreinte à un seul organisme.

La biologie computationnelle a longtemps utilisé des approches comparatives et, plus généralement, évolutives pour étendre la portée et la précision de ses analyses. Nous utilisons donc une approche évolutive pour l'inférence des réseaux de contrôle, ce qui nous permet d'étudier les modèles d'évolution de ces réseaux ainsi que d'améliorer la précision de l'infére rence. Comme les réseaux de contrôle évoluent avec les génomes, nous considérons que les réseaux de contrôle pour une famille d'organismes sont liés les uns aux autres au travers de l'arbre phylogénétique lui-mêmes. Ces relations contiennent des informations qui peuvent être utilisées pour améliorer la précision des réseaux reconstruits. Les progrès dans l'étude de l'évolution des réseaux de contrôle fournissent des preuves pour établir des modèles évolutifs pour les réseaux de contrôle—ceci est un élément important de notre approche évolutive. Nous utilisons deux modèles pour l'évolution des réseaux, un modèle *de base* qui ne considère que les gains et pertes de connexions au cours de l'évolution, et un modèle *complexe* qui prend également en compte les duplications et les pertes de gènes.

Avec les modèles d'évolution des réseaux, nous élaborons des algorithmes qui utilisent des relations phylogénétiques pour affiner des réseaux perturbés pour une famille d' organismes. Ces algorithmes de raffinement comprennent: *RefineFast* et *RefineML* qui sont algorithmes itératifs en deux étapes, et *ProPhyC* et *ProPhyCC* qui sont basés sur un modèle probabiliste phylogénétique. Pour chaque algorithme nous avons d'abord le concevoir avec le modèle évolutif de base et ensuite le généraliser au modèle complexe. Tous ces algorithmes sont informatiquement efficaces. Quantité de résultats expérimentaux démontrent qu'ils donnent une amélioration notable de la qualité des réseaux d'entrée. En particulier, *ProPhyC* et *ProPhyCC* améliorent encore les performances de *RefineFast* et *RefineML*.

En sus des quatre algorithmes mentionnés ci-dessus, nous avons également conçu un algorithme basé sur la théorie du transfert d'apprentissage, un algorithme de transfert d'appretissage sur arbre (*TTL*). *TTL* diffère des quatre algorithmes précédents dans le sens où il prend les données d'expression génique pour la famille d'organismes comme entrée, au lieu de leurs réseaux reconstruits. *TTL* apprend alors les structures des réseaux pour tous les organismes à la fois, en prenant parti des relations phylogénétiques. Bien que cette approche surpasse un algorithme d' inférence utilisé sur chaque réseau séparément, il ne donne pas de meilleurs résultats que *ProPhyC*, ce qui indique que *ProPhyC* fait bon usage de l'information phylogénétique.

mots clés: regulatory networks, network inference, evolution, phylogenetic relationships, ancestral network, refinement, gene duplication, evolutionary model, evolutionary history, reconciliation, orthology, maximum likelihood, transfer learning

Acknowledgements

I am very fortunate to have Prof. Bernard Moret as my supervisor. I can not find the right words to express what I really want to say now – so I am borrowing a sentence from the dissertation of our former PhD student Krister Swenson: “Bernard Moret is the best advisor a research student could hope for.” With this, I can not agree more. Not a single piece of this work could have been completed without his guidance, his active participation, his constant encouragement, and a lot more.

I am even more fortunate to have great labmates around – Krister, Yu, Vaibhav, Yann, Alexis, Wei, Cristina, Nishanth, Alisa, Kremena, Olga, Oana, Avinash, thank you all for making my PhD life so enjoyable and unforgettable.

I would like to thank Prof. Emre Telatar, Prof. Sarah Teichmann, Prof. Manolis Kellis and Prof. Matthias Seeger for having accepted to be on my jury, and for their interest and time.

Last but not the least, I am very grateful to my family for their unconditional support – thank you very much.

Contents

Chapter 1

Introduction

Transcriptional regulatory networks are models of the cellular regulatory system that governs transcription. They show how genes are up- and down- regulated by their associated transcription factors in response to signals. Transcriptional regulatory networks are often modelled as directed graphs [1], with nodes representing the genes and arcs (directed edges) representing the regulatory relationships between these genes. The arcs can have different properties like sign (denoting activation or inhibition), weight, etc., that give more details about the interactions.

Due to their importance in biological processes, transcriptional regulatory networks are studied from various aspects. First of all, the discovery of transcriptional regulatory networks, that is, the determination of regulatory interactions between transcription factors and target genes, is of great interest in biology and medicine. Wet-lab techniques such as chromatin immunoprecipitation (ChIP) can be used to determine the DNA binding sites for transcription factors and thus find their target genes. Since regulatory networks are determined only for a few organisms and this data produced by biological experiments is growing very slowly, computational methods are developed to infer regulatory networks. Then, with the networks known, the network topology properties are studied. For example, large regulatory networks are regarded as scale-free networks, whose degree distribution follows a power law distribution [2,3]; the network structures are hierarchical and have high modularity [4–6], and there are highly repetitive subgraph patterns called network motifs [7,8]. Furthermore, dynamic analysis of regulatory networks is also performed, to study how regulatory interactions are activated or deactivated under different conditions in one network [9], how networks grow along with the gene duplication events [4, 10], and finally how networks evolve from one organism to another [10, 11]. Knowledge of the dynamics and evolution can explain how the networks have formed into what they are. Furthermore, with sufficient data and knowledge we can predict future networks.

During my PhD I mainly worked on two of the topics above: the computational inference of regulatory networks, and the evolution of regulatory networks. In particular, I use the evolution of regulatory networks to improve their inference.

The inference of regulatory networks is important because the determination of regulatory networks is basic to all other studies, and because of the large number of genes of interest, and the limit of wet-lab techniques, it is still difficult and time-consuming to establish regulatory connections from bench experiments. Given high-throughput genome sequence data and microarray gene-expression data, computational methods are used to predict transcription factor binding sites (TFBS) and infer regulatory networks [12]. In particular, microarray gene-expression data, as phenotypical level data in contrast to genome sequence data, is used to infer regulatory networks. Methods using Boolean networks [13], Bayesian networks [14], dynamic Bayesian networks (DBNs) [15], and differential equations [16, 17], and so on, have been proposed for this purpose. The networks predicted by these algorithms, however, suffer from a high error rate. The high noise level in the data, the paucity of well studied networks, the many simplifications made in the models, combined with other factors (such as

the typically large number of genes tested vs. the small number of test samples—the so-called "tall dataset" problem), make inference difficult, in terms of both accuracy and computation.

Much effort is being made to improve the regulatory network inference in the community, most of which focuses on improving the standalone inference model or integrating additional data to infer the network for a single organism [18–20]. For example, methods were developed to use time-series expression data [15,21–23], and biological techniques have been improved to obtain richer data [24]. On the other hand, computational biology has long used comparative and, more generally, evolutionary approaches to extend the reach and accuracy of its analyses. As species evolve, their regulatory networks also evolve along the same lineages, so that the regulatory networks for a family of organisms are related through the organismal phylogeny. These relationships can be used as additional information to correct errors in currently available networks and obtain higher-quality networks for a family of organisms. Therefore, instead of focusing on a single-organism inference approach, we consider the regulatory networks for a family of species, and use an evolutionary approach to improve the inference of regulatory networks, which enables us to study evolutionary models for these networks as well as to obtain improved networks.

To design such an evolutionary approach, we have to consider two problems. First, although phylogenetic relationships are well established for many groups of organisms, we do not know how their regulatory networks evolve along the phylogeny, that is, we need a model for the evolution of regulatory networks to apply the evolutionary relationships among networks. Second, we need methods and algorithms which output the desired regulatory networks.

We turn to recent work on the evolution of biological networks to find a solution for the first problem. The evolution of biological networks—regulatory networks, metabolic networks and protein interaction networks—has drawn great interest among researchers [25–27]. Among these three types of networks, the evolution of regulatory networks is more difficult to study mainly due to the lack of benchmark data, since transcriptional regulatory networks produced from bench experiments are available only for a few model organisms. However, other types of data have been used to assist in the comparative study of regulatory mechanisms across organisms. For example, gene-expression data [11], sequence data such as transcription factor binding sites (TFBS) [28,29], and *cis*-regulatory elements [11] have all been used in this context. Moreover, a broad range of model organisms have been studied, including bacteria [30], yeast [11, 28], and fruit fly [29]. Although these studies have not to date sufficed to establish a clear model for regulatory network evolution, they have identified a number of evolutionary events, such as adding or removing network edges, and the duplication and loss of genes [4, 30–32]. In particular, Babu and his colleagues pioneered an evolutionary approach to the study of regulatory networks in *E. coli* and in *S. cerevisiae* [4, 30, 32, 33], where they posit a simple evolutionary model for regulatory networks, which amounts to adding edges to, or removing edges from the network, and proceed to investigate how well such a model accounts for the dynamic evolution of two of the best studied networks.

These studies have provided the basis for introducing evolutionary models for regulatory networks. We summarize two evolutionary models for regulatory networks. One is called a *basic* model, where we consider only gain and loss of regulatory connections while the gene contents stay the same. The other is an *extended* model, and in this model we also take into account duplications and losses of genes. These two models are formalized in Chapter 3.

Then, with a network evolutionary model, we design a computational framework that uses phylogenetic information to yield better networks than those derived with current inference algorithms. There are two scenarios for this problem with respect to the input information:

1. The input can be the regulatory networks for a family of species inferred independently (with any inference method), hereafter called a *base* method. In this case we design refinement algorithms which take these noisy inferred networks as input, and output the refined version of these networks. The refinement algorithms we have designed for this scenario are: *RefineFast*,

RefineML, *ProPhyC*, *ProPhyCC*.

2. The input can also be the gene-expression data of these species. In this case, we devise a network inference algorithm that infers the networks for all the species at the same time, while taking the phylogenetic relationships as part of the input and constraints. For this scenario, we have designed the *Tree Transfer Learning (TTL)* algorithm to infer the optimal configuration of the networks for all organisms at a time.

We have worked on both scenarios while focusing on the first one, since it is more general and not limited to certain source of regulatory networks in terms of both data and inference method.

When our input is the networks for the family of organisms to be refined, we first place these networks at the corresponding leaves of the phylogeny of this family, and consider using networks of their ancestors as a media to store and propagate the phylogenetic information. In this case, the ancestral networks will be inferred from these leaf networks by an ancestral reconstruction algorithm. However, since these leaf networks are error-prone, the reconstructed ancestral networks can help us get refined networks only if they have lower error rate than the leaf networks. In fact, an appropriate algorithm should be able to exclude errors and use the correct information in the leaves during ancestor reconstruction, given that the errors are independent across the leaves. We adapt FastML [34] to reconstruct ancestral networks with a maximum likelihood criterion. We perform experiments to test the accuracy of the ancestors reconstructed by FastML. Our results, which are shown in Chapter 3, show that the ancestral networks have less error than the leaf networks. With this guarantee we proceed to design refinement algorithms *RefineFast* and *RefineML*, which directly use ancestral information to obtain refined networks, and later on two improved refinement algorithms *ProPhyC* and *ProPhyCC*, which are based on a probabilistic graphical model and elaborately integrate the input noisy networks, ancestral networks and the refined networks all together.

We also design a tree transfer learning (*TTL*) algorithm which works with the second scenario mainly for the purpose of comparison and analysis. Prior to our work, Bourque and Sankoff [35] also developed an algorithm to infer regulatory networks across a group of species whose phylogenetic relationships are known; they used the phylogeny to reconstruct networks from the gene-expression data of these species, under a simple parsimony criterion.

For each refinement algorithm we present, we first show how it works with the basic evolutionary model, and then show how we extend it to work with the extended model. In our experiments, the noisy networks as input to our refinement algorithms are generated by different means: by using various basic inference methods to infer networks from gene-expression data, or by adding artificial noise from various distributions to the "true" regulatory networks. We use "true" regulatory networks both generated from data simulation and from biological data collection. We perform extensive experiments to test our algorithms from multiple aspects. We compare the accuracy of networks inferred from the base inference algorithm, output from Bourque and Sankoff's algorithm, and refined by each of our refinement algorithms respectively. We show that, under all comparable settings, *RefineFast* and *RefineML* outperform the base inference algorithm and Bourque and Sankoff's algorithm, and *ProPhyC* and *ProPhyCC* further improve *RefineFast* and *RefineML*.

In Chapter 2, we give the computational and biological backgrounds of our work, as well as a brief introduction of the algorithm from Bourque and Sankoff. In Chapter 3, before getting to our core algorithms, we give the details of two preliminaries, which are the two formalized networks evolutionary models we use, and the accuracy tests of the ancestral networks reconstructed by FastML. In Chapter 4 we describe our two-step iterative algorithms *RefineFast* and *RefineML*, and show how to extend them from the setting of basic evolutionary model to that of the extended model. In Chapter 5 we describe the *ProPhyC* and *ProPhyCC* algorithms which further improve *RefineFast* and *RefineML* with both the basic and extended network evolutionary models. In Chapter 6 we present the *TTL* algorithm and compare it with *ProPhyC* and *ProPhyCC*. Finally in Chapter 7 we give conclusions.

All the work presented here is the author's. All of it was carried out in close collaboration with Prof. Bernard Moret. Two MS students also participated in the research: Ms. Maryam Zaheri, who collaborated on the design of the *RefineFast* algorithm for the basic network evolutionary model, and Ms. Kremena Diatchka who collaborated on extending the *RefineFast* and *RefineML* to fit the extended evolutionary model.

Chapter 2

Background

Our main refinement algorithms (*RefineFast*, *RefineML*, *ProPhyC* and *ProPhyCC*) take the the noisy regulatory networks for a family of organisms as input, and output refined version of this set of networks. In our experiments, we obtain the noisy networks as input to the refinement by using an existing network inference algorithm (which we call a *base* inference method) to infer networks from gene-expression data. To test the generality of our refinement algorithms, we use two different base inference methods to infer regulatory networks from gene-expression data, one based on dynamic Bayesian networks (DBN) and the other based on differential equations, which are two widely used models for network inference. For the former, we use the implementation in Murphy's Bayesian Network Toolbox [36]; for the latter, we use TRNinfer [17].

RefineFast, *RefineML*, *ProPhyC*, and *ProPhyCC* all use networks of ancestral species in various ways. *RefineFast* and *RefineML* are two-step iterative algorithms. The noisy input networks are placed at the corresponding leaves of the (known) phylogeny. In the first step, from the input networks at the leaves, we infer ancestral networks; in the second step, these ancestral networks are used to refine the leaf networks. These two steps are then repeated as needed. To infer ancestral networks, we use our adaptation of FastML [34], which was initially designed to reconstruct ancestral protein sequences with a given phylogeny.

When the basic network evolutionary model is used, all networks have equal gene contents, so to resolve the ancestral networks we only need to infer the connections. However, with the extended evolutionary model, since it includes gene duplications and losses, 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. A standard approach to address this problem is to reconcile the gene trees and species tree [37–39].

In this chapter we briefly introduce the relevant topics we need for later chapters.

2.1 DBNs for Network Inference

When DBNs are used to model regulatory networks, an associated structure learning algorithm is used to infer the networks from gene-expression data [15, 40–42]. The implementation of this algorithm in the Bayesian Network Toolbox provides two optimization functions: a maximum likelihood (ML) score and a Bayesian information criterion (BIC) score.

Let D denote the dataset used in learning and G the (structure of the) network; the algorithm using ML scoring aims to return the structure $G^* = \arg\max_G \log Pr(D|G)$. However, transcriptional regulatory networks are typically sparse graphs, so ML inferences often produce many false positive edges. The BIC score introduces a penalty on the complexity of G to get a tradeoff between fit and

complexity, which is defined as

$$\log Pr(D|G,\hat{\Theta}_G) - 0.5\#G \log N \tag{2.1}$$

where $\hat{\Theta}_G$ is the ML estimate of network parameters for structure G, N is the number of samples in dataset D, and $\#G$ is the *dimension* of structure G, defined as the number of free parameters of G. The penalty for model complexity makes networks inferred under this criterion more conservative, reducing the number of false positives in the networks, and thus gaining specificity at the expense of sensitivity.

In practice, heuristic search methods are used, as well as mild restrictions on the structure of the model, the latter aimed at reducing the huge number of possible network structures—such as a bound on the maximum indegree of the nodes, a restriction that appears well supported by the data [13, 42] and that we use in our simulations.

2.2 Differential Equations for Network Inference

Differential equations can describe causal relationships among components in a quantitative manner and are thus well suited to model transcriptional regulatory networks [16, 17]. A regulatory system is represented by the equation $d\boldsymbol{x}/dt = f(\boldsymbol{x}(t)) - K\boldsymbol{x}(t)$, where $\boldsymbol{x}(t) = (x_1(t), \cdots, x_n(t))$ denotes the expression levels of the n genes at time t and K (a matrix) denotes the degradation rates of the genes. The regulatory relationships among genes are then characterized by $f(\cdot)$. Wang *et al* [17] produced a tool, TRNinfer, that solves the differential equations by formulating them into linear programming problems.

2.3 ML-based Reconstruction of Ancestral Nodes

Reconstructing ancestral information in phylogenetic work is typically in the nature of an anchoring step in the computation, particularly in parsimony-based approaches. When we have high confidence in the tree and the edge lengths are modest, however, an ML approach to ancestral inference can yield accurate results; FastML [34], using a user-specified character substitution matrix, infers labels for the internal nodes (on a site-by-site basis) that maximize the overall likelihood of the tree. The algorithm was initially designed for protein sequences, but can be used for any type of sequence with a suitable substitution matrix.

This algorithm assumes that each site in the protein sequences evolve independently, so it can infer the ancestral characters for one site at a time. Fix a site, i.e., a character position in the sequence. Let i denote a node in the tree, l_i the length of the edge between node i and its parent, and a the value of a character at a node in the tree, chosen from a given set S of possible character values. For each node i and each character a, 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 assigned to i given that its parent is assigned as a.

Finally, let π_a denote the initial distribution of character a and $p_{ab}(l)$ the probability of substitution of a with b along an edge of length l. For simplicity, assume that the given tree is binary; then our adaptation of the FastML algorithm carries out these steps (see Fig. 2.1):

1. If leaf i has character b, then, for each $a \in S$, set $C_i(a) = b$ and $L_i(a) = p_{ab}(l_i)$.
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

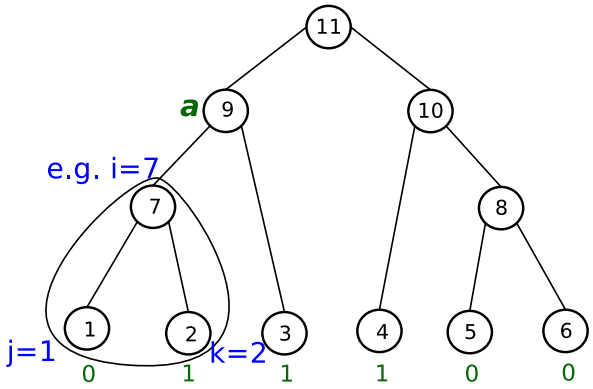


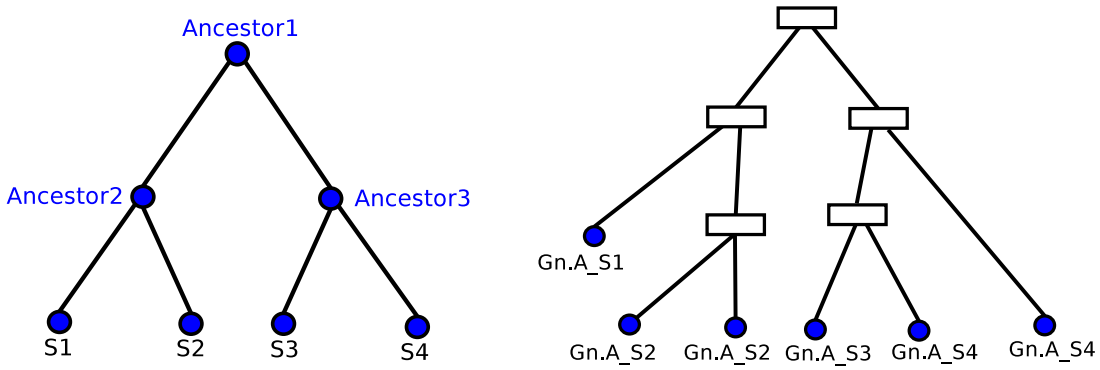
Figure 2.1: Illustration of the FastML algorithm: calculating $L_i(a)$ and $C_i(a)$ for a node i with two children j and k.

- $L_i(a) = \max_{c \in S} p_{ac}(l_i) \cdot L_j(c) \cdot L_k(c)$
- $C_i(a) = \arg\max_{c \in S} p_{ac}(l_i) \cdot L_j(c) \cdot L_k(c)$

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

2.4 Reconciliation of Species Tree and Gene Trees

To infer ancestral networks with the extended network evolution model, we need a full history of gene duplications and losses. We reconstruct this history by reconciling the gene trees and the species tree. The species tree is the phylogenetic tree whose leaves correspond to the modern organisms; gene duplications and losses occur along the branches of this tree. A gene tree is a phylogenetic tree whose leaves correspond to genes in orthologous gene families across the organisms of interest; in such a tree, gene *duplication* and *speciation* events are associated with internal nodes. Fig. 2.2 shows an example.



(a) *The species tree of 4 species*

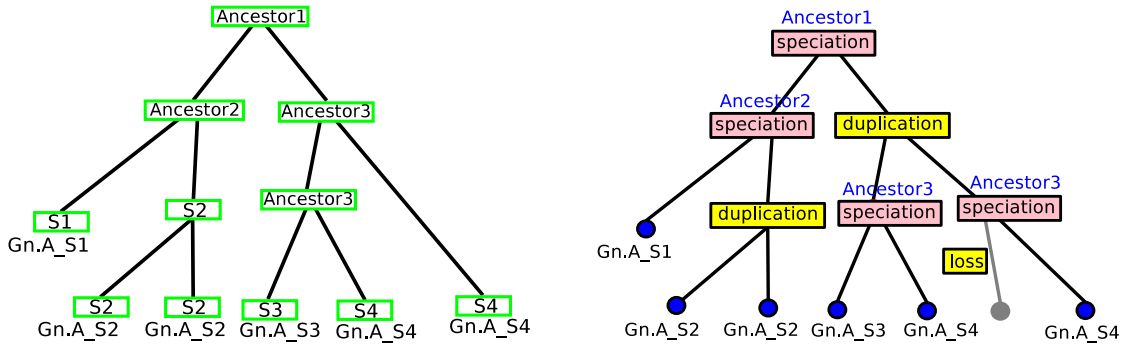
(b) *The gene tree of the gene family "A"*

Figure 2.2: The left plot shows the species tree for 4 species: S1, S2, S3 and S4. The right plot shows the gene tree of the gene family "A" across the 4 species. The events at its internal nodes can be determined by reconciling the two trees.

When gene duplications and losses occur, the species trees and the gene trees may legitimately differ in topology. Reconciling these superficially conflicting topologies—that is, explaining the differences through a history of gene duplications and losses—is known as *lineage sorting* or *reconciliation*. Given a gene tree for a gene family and the corresponding species tree (as shown in Fig. 2.2), the reconciliation process can label the internal nodes of the gene tree with speciation and duplication events [38, 43]. This is usually done by creating a mapping between the gene tree T_G and the species tree T_S. The mapping M maps every node v in T_G to a target node, $M(v)$, in T_S. Do a post-order traversal on the gene tree, then for each v, $M(v)$ is assigned as follows:

- If v is a leaf node in T_G, $M(v)$ is the species from which the gene at v is obtained.
- If v is an internal node in T_G, v is mapped to the least common ancestor, *lca*, of the target nodes of its children, that is, $M(v) = lca(M(leftchild(v)), M(rightchild(v)))$.

Fig. 2.3(a) shows the same gene tree in Fig 2.2(b) with all the nodes labelled with their target nodes obtained by the mapping M. Under the mapping M, a node in T_G is determined as a duplication node if its target node is the same as at least one of its children's target nodes, otherwise it is a speciation node. Thus we can determine all the speciation and duplication events in the gene tree. Then necessary gene loss events can be inferred to consist with the speciation and duplication events. Fig. 2.3(b) shows the same gene tree with all gene duplication, gene loss, and speciation events labelled.



(a) *The gene tree showing target nodes from the mapping M*

(b) *The gene tree showing a solved history*

Figure 2.3: The gene tree for gene family "A" across 4 species. Left: the text at each node shows the target node $M(v)$ of each node v in the gene tree. Right: the gene tree showing a complete gene duplication and loss history.

In practice, when a gene tree is not pre-determined, the reconstruction of the gene tree uses two criteria: 1. the gene tree should fit the sequence data of the genes; 2. the gene tree should yield an optimal gene duplication and loss history by the reconciliation process introduced above. While reconstructing the optimal gene tree and history is a hard computational problem, algorithms have been devised for it in a Bayesian framework [37] or using a simple parsimony criterion [38].

2.5 Evolution and Dynamics of Regulatory Networks

The evolution of regulatory networks is mainly attributed to the evolution of genomes [10, 44]. Mutations in the genome of an organism can change the regulatory interactions in various ways. First, mutations in the regions of *cis*-regulatory elements can cause the loss of transcription factors which bind to these elements. For example, in [11] the authors suggest that *cis*-regulatory elements can be gained or lost during the evolution of a family of 17 fungi genomes. Second, genome-level mutations

for the transcription factors (TFs) can change their function such that they may not regulate the same target genes. In particular, studies have shown that transcription factors evolve faster than their target genes in prokaryote organisms, and orthologous transcription factors can regulate different genes in different organisms [30, 45]. These studies show that even with a network which contains only orthologous TFs and target genes, the regulatory interactions can be lost or gained during evolution.

Another important evolutionary event during genome evolution is gene duplication. As a main source of new gene functions, gene duplication also plays an important role in the evolution of regulatory networks [31, 32, 44]. Although the studies of how gene duplication affects network evolution are mainly performed for a single organism, where the time scale of dynamics is smaller than that in species evolution, the results and observations can easily be extended to cross-species studies. The duplicated copies of a gene family tend to inherit the regulatory interactions from the original copy, but since the duplicated copies also diverge quickly, loss and gain of interactions can happen after the duplication. Other studies suggest a *preferential attachment* model for the interactions of duplicated gene copies, that is, the new copies tend to connect to genes with high degree [3, 46]. Therefore, gene duplications result in much difference between an ancestral network and its child network. In single organism studies, these mechanisms of gene duplication contribute to the growth of regulatory networks, and can also explain to certain extent the structure and connectivity attributes of large regulatory networks.

Both the gain and loss of regulatory interactions and the duplication and loss of genes describe the evolutionary changes on the level of a single gene or interaction. On a higher structure level in regulatory networks, there are network motifs which appear frequently throughout the networks. Network motifs are small subnetworks with certain patterns, like single input, multiple input and feed-forward loop motifs [4, 7]. These motifs, however, are not conserved during evolution, though the evolved network still has similar abundance of these motifs [4, 10].

Besides the changes in regulatory networks on large evolutionary time scale, the dynamics of these networks on small time scale like a certain life period within an individual have also been studied [9, 47]. Although a static network structure has been used to represent the regulatory network for a certain organism, the network in an individual is not static with respect to time and change of conditions. In [9], based on a static network of *S. cerevisiae*, the authors derived the active subsets of interactions under different conditions from the corresponding gene-expression data, and found significant differences between the subsets. The studies of short-time dynamics of regulatory networks can provide insights for network evolution across species, but not much has been done so far to incorporate the dynamics into cross-species studies, while most comparative analysis of regulatory networks across species still use static network structures.

Despite the possible evolutionary changes of regulatory networks we describe above, researchers also found that a large portion of orthologous TFs and target genes tend to share the same regulatory interactions across species, and this conservation is related to the phylogenetic distance between the organisms [4, 30, 48]. This provides further support for our evolutionary approach.

2.6 Transfer Learning

The design of our *TTL* algorithm is inspired by the inductive transfer learning theories in machine learning. Transfer learning, also called multitask learning, is an approach to learn *related* tasks simultaneously, such that what is learned for each task can interactively help other tasks to be learned better [49–51]. It is especially useful when the data for some of the tasks is not sufficient – they can be learned better by transferring knowledge from other well learned tasks. Transfer learning is widely used in various problems such as classification, regression, clustering, and so on. In particular, in [50], the authors applied the transfer learning idea to learn the structure of a set of Bayesian networks. The relationships between different Bayesian networks are modeled as a prior probability,

where one should specify parameters to quantify difference between any two networks. This prior then links the different tasks (the different Bayesian networks) in this way. The prior contributes to the posterior probability, and the best set of network structures (which is a *configuration*) is the one which yields optimal posterior probability.

In our case, if we model the regulatory networks by Bayesian networks, then our goal is to find the best configuration of networks for all organisms in the family, from the gene-expression data of these organisms. The relationships between the networks are well defined and represented by the phylogenetic tree. Our *TTL* algorithm uses the tree to do the knowledge transfer while learning the network structures.

2.7 The Algorithm of Bourque and Sankoff

In [35], Bourque and Sankoff presented a method to generalize a single-organism network inference algorithm to infer the network for a family of organisms simultaneously, with a parsimony criterion. They first described the algorithm they used to infer a single network. This algorithm takes time-series gene-expression data as input, and employs a system of differential equations to model the regulatory network. For a gene x, denote its gene-expression level at time t as $x_i(t)$. Let $a_{i,j}$ be the coefficient corresponding to the regulatory impact of gene j on gene i. Then

$$\frac{dx_i(t)}{dt} = \sum_{j=0,\ldots,n} a_{i,j}x_j(t)$$

Also let $y_i(t)$ denote $dx_i(t)/dt$, that is, $y_i(t) = \sum_{j=0,\ldots,n} a_{i,j}x_j(t)$. From the gene-expression data, we have the values of $x_j(t)$ for all genes and all time points, and the problem is to solve the values of $a_{i,j}$ for all i and all j.

Then if the set of regulators for gene x is R_i, we have

$$y_i(t,R_i) = \sum_{j\in R_i} a_{i,j}x_j(t) \tag{2.2}$$

The task of inferring a networks is to solve for coefficients $a_{i,j}$. In their case they find estimates of $a_{i,j}$, $\hat{a}_{i,j}$, by minimizing the square error

$$SSE(R_i) = \sum_t (y_i(t) - \hat{y}_i(t,R_i))^2 \tag{2.3}$$

where $\hat{y}_i(t,R_i) = \sum_{j\in R_i} \hat{a}_{i,j}x_j(t)$. The size of R_i is controlled to limit the number of non-zero coefficients, and thus reduce the complexity of the network.

To extend this method to consider the networks of a family of organisms at once, they modified the optimization function so that it contains two parts: the total square error and the complexity of the network of all the modern organisms, and the total *evolutionary cost* over the edges of the phylogeny. This algorithm assumes the same gene content throughout the networks of all species, and only considers the insertion and deletion of regulatory connections for each gene, which is equivalent to our basic network evolutionary model. Denote the phylogenetic tree as a graph $G = (V,E)$, assume for each gene i and for each vertex in G, v, the set of regulators is R_i^v. Then the evolutionary cost is:

$$COST(R_i^1, R_i^2, \ldots, R_i^{|V|}) = \sum_{(u,v)\in E} |R_i^u \ominus R_i^v| \tag{2.4}$$

where $\ominus$ is the symmetric difference between two sets. This *COST* term is added to the *SSE* score (defined in Eq. 2.3) of all modern species with a weight coefficient. This combined score is used as a criterion to find the best sets of regulators for all the genes in all the networks in G.

This algorithm, hereafter called the B&S algorithm, has provided a framework for using phylogenetic information to infer regulatory networks under the differential equation model. However, since the algorithm requires time-series gene-expression for all the organisms as input, its application may be limited by the input information. Furthermore, the optimization problem has high computational complexity. So far it has only used the structure of the phylogeny when calculating *COST*, however, taking into account more information provided by the phylogeny such as edge lengths may help improve the scoring.

Chapter 3

Preliminaries

In this chapter we formalize the network evolutionary models which we use for our refinement algorithms, and report the tests of FastML in the accuracy of reconstructing ancestral networks from noisy leaf networks.

3.1 Regulatory Network Evolutionary Models

We summarize two network evolutionary models, a *basic* model and an *extended* model. In both models, the networks are represented by binary adjacency matrices, with a 1 in the (i, j) entry denoting an edge from node i to node j. We use binary matrices for simplicity's sake: generalization to weighted matrices is immediate and, indeed, the additional information present in a weighted matrix should further improve the results.

For the basic model, the evolutionary operations are:

- *Edge gain*: an edge between two genes is generated with probability p_{01}.
- *Edge loss*: an existing edge is deleted with probability p_{10}.

We also assume that all the edges in the networks have the same probability to be lost, and all the non-existing edges have the same probability to be gained at any evolutionary step. The model parameters are thus:

- the base frequencies of 0 and 1 entries in the given networks $\Pi = \begin{pmatrix} \pi_0 & \pi_1 \end{pmatrix}$;
- the substitution matrix of 0s and 1s, $P = \begin{pmatrix} p_{00} & p_{01} \\ p_{10} & p_{11} \end{pmatrix}$.

The extended model has two additional evolutionary operations, gene duplication and gene loss, with corresponding additional model parameters p_d and p_l. We assume that all the genes have the same duplication and loss rates. So the extended model not only has the the gene gain and gene loss operations, but also has the following two operations:

- *Gene duplication*: a gene is duplicated with probability p_d. After a duplication, edges for the newly generated copy can be assigned as follows:

  Neutral initialization: Create connections between the new copy and other genes randomly according to the proportion π_1 of edges in the background network independently of the original copy. The directions of connections are also random.

Inheritance initialization: Connections of the duplicated copy are reported to correlate with those of the original copy [30–32]. This observation suggests letting the new copy inherit the connections of the original while keeping the directions of connections, then lose some of them or gain new ones at some fixed rate [46].

Preferential attachment: The new copy gets connected to genes with high connectivity [3,46].

- *Gene loss*: a gene is deleted along with all its connections with probability p_l.

The parameters of the extended model are thus: Π, P, p_l and p_d.

3.2 Verifying the Ancestral Reconstruction Procedure

We examine the accuracy of the ancestral networks reconstructed by FastML when the leaf networks are noisy. We conjecture that during ancestor reconstruction FastML is able to eliminate much of the noise in the leaf networks, such that the ancestral networks have lower error rate than the leaf networks. This may not be true for all the ancestral networks, since for the ancestors which are far from the leaves (for example, those close to the root), the distance between these ancestors and the leaves can be big enough for some correct information to be lost on the way. However, we can proceed to use the ancestral information for our design of a refinement algorithm, as long as the ancestors to certain height are more accurate than the leaves.

We perform simulation experiments to test the above conjecture. We use the basic network evolutionary model in these tests. Starting from a tree and a root network, we simulate the "real" evolution along the tree, according to the network evolutionary model, to generate the "true" regulatory networks for all ancestors and all modern organisms. Then with certain error rates, we obtain noisy leaf networks from the true ones, which are then used to reconstruct ancestral networks by FastML.

The data used by FastML are:

- the proportions of 0s and 1s in the networks, $\Pi = \begin{pmatrix}\pi_0 & \pi_1\end{pmatrix}$
- the topology of the phylogenetic tree;
- the *edge length* l_e of each edge e, i.e., the number of changes along this edge;
- for each edge length l_e, its corresponding substitution matrix, $P_s(l_e)$, which represents the mutation probability between 0 and 1

$$P_s(l_e) = \begin{pmatrix} p_{00}(l_e) & p_{01}(l_e) \\ p_{10}(l_e) & p_{11}(l_e) \end{pmatrix}$$

The substitution matrices depend on edge length: the longer the edge, the higher the mutation probabilities. We choose a $P_s(1)$ for edge length 1 and calculate $P_s(l_e)$ for $l_e \geq 2$ using an exponential distribution, $P_s(l_e) = P_s^{l_e}(1)$.

The reconstructed ancestral networks are compared with the "true" ones. For each noisy leaf network and FastML-reconstructed ancestral network, we compare it with the corresponding true network from simulation, and calculate the *sensitivity* and *specificity* values as measurement of its accuracy. If we compare a noisy/reconstructed network G_1 to the true network G_2, then the sensitivity and specificity of G_1 are defined as follows:

$$\text{sensitivity} = \frac{TP}{TP+FN}$$

$$\text{specificity} = \frac{TN}{TN+FP}$$

where TP is the number of connections which are in both G_1 and G_2; FP is the number of connections which are in G_1 but not in G_2; TN is the number of connections which are in neither G_1 nor G_2; FN is the number of connections which are not in G_1 but are present in G_2.

The specificity and sensitivity values calculated for all the networks are then averaged over networks on the same tree level to get the "sensitivity and specificity of a level". Fig. 3.1 shows an illustration of experimental setup.

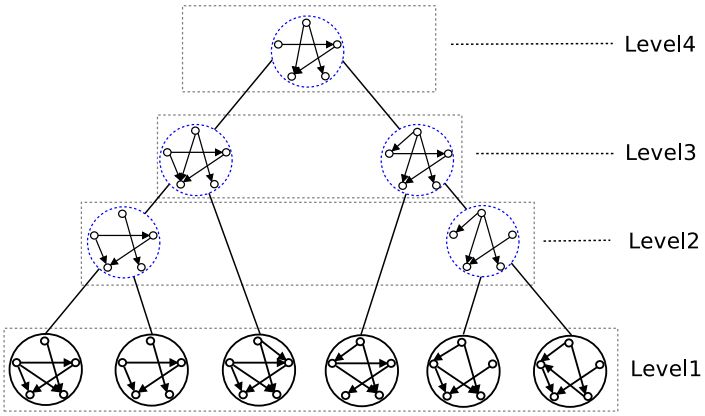
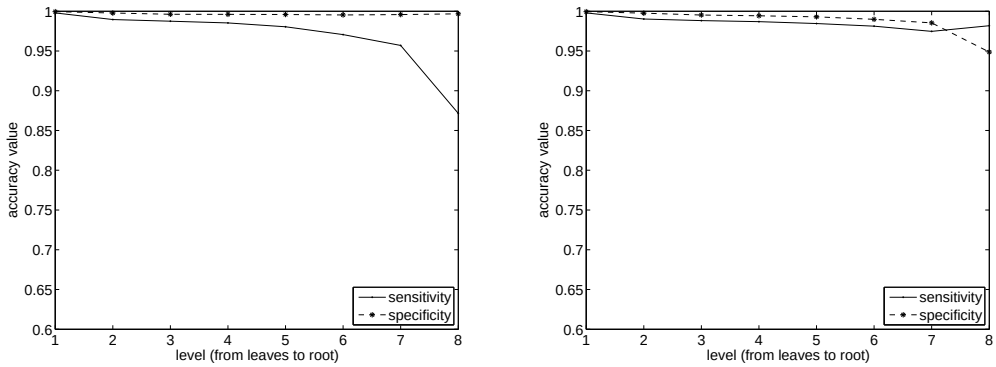


Figure 3.1: An illustration of the experimental setup to evaluate the performance of FastML on each tree level

In these experiments we use trees which have 100 leaves and 8 levels. The regulatory networks each have 16 genes. We let FastML reconstruct ancestors from leaf networks with different error rates. Finally, since there can be different structures of a tree with given numbers of leaves and levels, we generate 100 random trees for each setting of experiments and report the averaged results.

Fig. 3.2 shows the average sensitivity and specificity values of networks on each level, which



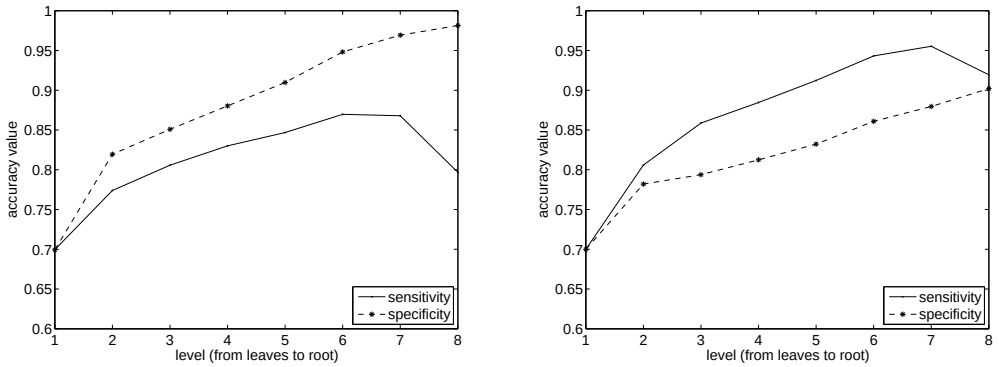
(a) with same evolutionary rates as parameters

(b) with different evolutionary rates as parameters

Figure 3.2: Leaf networks are not noisy

is again averaged on 100 tree structures. In these plots the input networks are correct networks. In Fig. 3.2(a) the substitution rate for FastML is the same as the one we use in simulation, that is, the "true" parameters. We see that with these parameters the specificity is better maintained from the leaves to the root. However we can tune the parameters so that they favor the sensitivity more—an example is shown in Fig. 3.2(b) with tuned parameters.

In Fig. 3.3 the input networks have respectively about 30% error in each of sensitivity and specificity. Fig. 3.3(a) shows the results where we use the same substitution rates as the simulation. We



(a) with same evolutionary rates as parameters

(b) with different evolutionary rates as parameters

Figure 3.3: Leaf networks are noisy

observe that when the leaf networks are noisy, the accuracy of reconstructed ancestral networks does increase, up to some level in the phylogenetic tree. In Fig. 3.3(a) the specificity keeps going up as we go towards the root, while the sensitivity starts decreasing when we go too far. This matches our conjecture and provides confidence to design refinement algorithms using ancestral information. Similarly, if we want more improvement on sensitivity we can change the parameters of FastML so that the sensitivity gets more improvement with the tradeoff of specificity, as shown in Fig. 3.3(b). This confirms the flexibility of having different tradeoffs between sensitivity and specificity of the reconstructed ancestral networks, thus establishing the potential for our refinement algorithms to inherit this flexibility. Experiments also show that the increase of accuracy from leaves to ancestors can be obtained with a large range of parameters, though the increase can be allocated between sensitivity and specificity in different ways. This provides the robustness basis for our refinement algorithms.

Chapter 4

Two-step Refinement Algorithms RefineFast and RefineML

The principle of our refinement approach is as follows: since regulatory networks evolve along with genomes, we posit that the regulatory networks for a family of organisms are related to each other through the same phylogenetic tree. Then if there are errors in the regulatory networks for a family of organisms, the phylogenetic relationships can be used to improve the accuracy of these networks. We consider a scenario where regulatory networks have been (separately) inferred for a number of related organisms whose phylogenetic relationships are known. Our algorithms *RefineFast* and *RefineML* refine these networks by considering all of them at once, within the known phylogeny of the organisms, to produce networks with much higher specificity and sensitivity. To make use of the phylogenetic relationships among this group of species for the refinement purpose, we consider ancestral information, that is, using networks of ancestors as media to store and propagate the phylogenetic information.

RefineFast and *RefineML* [52–55] work iteratively in two phases after an initialization step, which is to obtain the regulatory networks for the family of organisms. Typically, these networks are inferred from gene-expression data for these organisms, using standard inference methods. We place these networks at the corresponding leaves of the phylogeny of the family of organisms and encode them into binary strings by simply concatenating the rows of their adjacency matrix. We then enter the iterative refinement cycle. In the first phase, we infer ancestral networks for the phylogeny (strings labelling internal nodes), using our own adaptation of the FastML [34] algorithm; in the second phase, these ancestral networks are used to refine the leaf networks. These two phases are then repeated as needed. Our refinement algorithms are formulated within a maximum likelihood (ML) framework and focus solely on refinement—they are algorithmic boosters for one's preferred network inference method.

RefineFast and *RefineML* were firstly designed on the basic network evolutionary model, then we generalized them to fit the extended network evolutionary model so that they work in a broader framework. The generalization includes many changes to use the duplication/loss data and handle the more complicated cases caused by the extended model. One of the main problems to solve is to reconstruct the gene duplication and loss history. Besides using the existing reconciliation algorithms for gene trees and species tree [37–39], we also designed and tested other history models like *duplication-only* and *loss-only* models to analyze the effect of different duplication and loss history predictions on the performance of refinement algorithms.

We did experiments to test the performance of *RefineFast* and *RefineML* under various settings, on both simulated data and biological data. With simulated datasets, we tested the algorithms in different aspects by altering the following factors: the size and shape of the phylogenetic tree, the size of the networks (that is, number of genes in the networks), the evolutionary rates for networks

(including both the rates for gain and loss of connections and those for gene duplications and losses). To further test the generalization of our refinement algorithms, we apply different network inference algorithms as base algorithm to predict networks as input for our refinement algorithms, and during the data simulation procedure we use data generation methods to verify that our algorithms work under all circumstances.

We also perform further tests to exclude confounding factors and test different aspects of our algorithms. We investigate the source of these improvements, eliminating various simple possibilities such as noise averaging and thus demonstrate that it is indeed the phylogenetic data that enables our algorithm to improve upon the standard approach. *RefineLocal* and *RefineRandomTree* were designed and used in these tests.

We compare the networks predicted by a base inference used standalone, and those output from our refinement algorithms. We also apply the algorithm of Bourque and Sankoff on the same datasets and compare its output with that of *RefineFast* and *RefineML*. Accuracy of networks is measured by their *sensitivity* and *specificity*. We plot *receiver-operator characteristic (ROC)* curves with different tradeoffs of sensitivity and specificity. The ROC curves for our algorithms consistently dominate those of the standard approaches used alone; under comparable conditions, they also dominate the results from Bourque and Sankoff.

In this chapter we first present the *RefineFast* and *RefineML* algorithms on the basic network evolutionary model, followed by a description of their versions for the extended model. Then we show in detail our experimental design, including the data generation for the simulated datasets, and the experimental results, firstly with the basic network evolutionary model and then with the extended model.

4.1 RefineFast and RefineML under the Basic Model

4.1.1 Overview

To get the orthologous networks to be refined by our algorithms, we use a standard network inference method to infer networks from gene-expression data. So the input of the whole procedure is a set of gene-expression data matrices, collected under similar experimental conditions, for several related organisms, along with a known phylogeny (with edge lengths) for this group of organisms. (Such phylogenies are typically well established though the edge lengths remain to be explored.) Thus there are three dimensions to the data: the number of organisms (the number of matrices), the number of genes (the number of rows in each matrix), and the number of test conditions (the number of columns in each matrix).

The first step is simply to run one's preferred algorithm for regulatory network inference, independently on each of the data matrices; in this study, we use two types of inference algorithms, respectively based on DBN and differential equations. The resulting networks are used to label the corresponding leaves of the phylogeny. We encode a network by the concatenation of the rows of its adjacency matrix—every code thus represents a valid network. Note that the initial networks themselves are the real inputs to our algorithm; we use the gene-expression data stage in our tests solely in order to enhance the verisimilitude of our simulations.

We then use our adaptation of the FastML algorithm to infer ancestral networks, which in turn are used to refine the sequences at the leaves. We present below two algorithms to carry out this refinement, both based on the intuition (verified in simulations) that ancestral sequences are more accurate than those at the leaves, but only up to some height in the tree—as distant ancestral sequences suffer from the inference errors of FastML. The two middle steps can be iterated: starting from the newly refined networks, we can once again infer ancestral networks and use the results to refine the leaves.

We realize that edge lengths obtained from an analysis of the sequences of (typically) a few genes

need not reflect the amount of evolution in the regulatory networks—while both evolved on the same tree, their respective rates of evolution could differ considerably. As we still lack the knowledge required to formulate a more precise model of network evolution, using the same edge lengths is just the neutral choice.

4.1.2 Inferring the initial networks

The inference algorithm we use to initialize the process is the *DBN*, as implemented in the Bayesian Network Toolbox [36]. In our application, however, we want to examine the ROC curves and so need to be able to trade off specificity and sensitivity. To this end, we modify the inference method based on DBN by generalizing Eq. 2.1 with a *penalty coefficient* k_p to adjust the penalty:

$$\log Pr(D|G,\hat{\Theta}_G) - k_p \#G \log N \tag{4.1}$$

where k_p varies from 0 to 0.5. With $k_p = 0$, we have the ML score; this is equivalent to the objective function used in REVEAL [42, 56], which maximizes the mutual information between parents and child. With $k_p = 0.5$, the score of Eq. 4.1 reduces to the original BIC score from Eq. 2.1.

For the TRNinfer algorithm, the parameter that it provides to adjust the sparseness of the networks does not afford sufficient control to generate sparse enough networks. We thus supplement it by applying different thresholds to the output connection matrix to choose final edges. We shall refer to these modified inference methods as *DBI* for that based on the DBN model and as *DEI* for that based on TRNinfer.

4.1.3 Inferring the ancestral networks

In this study our adjacency matrices are binary, with a 1 in the (i, j) entry denoting an edge from node i to node j. We use binary matrices for simplicity's sake: generalization to weighted matrices is immediate and, indeed, the additional information present in a weighted matrix should further improve the results. Similar to the tests in Sec. 3.2, the data used by FastML are thus:

- the proportions of 0s and 1s in the networks, $\Pi = \begin{pmatrix}\pi_0 & \pi_1\end{pmatrix}$;
- the topology of the phylogenetic tree;
- the *edge length* l_e of each edge e, i.e., the number of changes along this edge;
- for each edge length l_e, its corresponding substitution matrix, $P_s(l_e)$, which represents the mutation probability between 0 and 1

$$P_s(l_e) = \begin{pmatrix} p_{00}(l_e) & p_{01}(l_e) \\ p_{10}(l_e) & p_{11}(l_e) \end{pmatrix} \tag{4.2}$$

The substitution matrices depend on edge length: the longer the edge, the higher the mutation probabilities. We choose a $P_s(1)$ for edge length 1 and calculate $P_s(l_e)$ for $l_e \geq 2$ using an exponential distribution, $P_s(l_e) = P_s^{l_e}(1)$.

4.1.4 Refining the leaves

The underlying principle is simple: phylogenetically close organisms are likely to have similar regulatory networks; thus independent network inference errors at the leaves get corrected in the ancestral reconstruction process. Obviously, however, if too much evolution occurred, the ancestral reconstruction process itself generates errors. Thus a crucial aspect of our algorithm is how to use ancestral networks at various heights above the leaves to refine the leaves. We ran large series of experiments

under various conditions (not shown); all showed an expected increase in accuracy when moving to the parents of the leaves, eventually replaced by a decrease when moving too far above the leaves. On the basis of our results, we chose to use only the immediate parents of the leaves for refinement—but note that these parents are themselves the product of a global ML inference and thus reflect the structure of the entire phylogeny.

A fast oblivious refinement algorithm: RefineFast

Our first algorithm, *RefineFast*, is designed to run quickly; it reposes complete trust in the networks associated with the parents of the leaves, using them to replace, rather than refine, the leaf networks.

1. From the current leaves, infer ancestral nodes using FastML.
2. For each leaf, pick its parent and evolve it (according to the length of the edge to the leaf and its substitution matrix) to generate a new child.
3. Use these new children to replace the old leaves.
4. Repeat Steps 1–3 until the total size of the leaf networks stabilizes.

We can use the same substitution matrices $P_s(l_e)$ in both Step 1 and Step 2, but choosing different substitution matrices can accelerate convergence. In practice the algorithm converges very fast, that is, in less than 5 iterations. Denoting the number of genes in each network by n, and the number of leaves in the phylogenetic tree by n_l, the running time of this algorithms is $O(n_l \cdot n^2)$.

The algorithm is deliberately oblivious: it uses the original networks only in the ancestral reconstruction, after which it replaces them with a sample network drawn from the distribution of possible children of the parent. When the original networks are noisy (a common occurrence), this simplistic procedure does quite well.

A nonoblivious refinement algorithm: RefineML

To use the information still present in the original leaf networks in the refinement step, we developed an ML-based refinement algorithm, *RefineML*. To use the existing leaf sequences, we assign each site of each leaf (that is, each entry of the adjacency matrix of each leaf network) a *belief coefficient*, k_b, which varies between 0.5 and 1. This value represents the confidence we have for each entry in the input networks. In the DBN framework, to obtain the confidence coefficient values, we first estimate the conditional probability tables (CPTs) of the *DBI* inferred networks from the gene-expression data on the inferred structure [57], and then calculate the confidence values from the CPTs. We introduce this procedure below.

For each gene g_i, if m_i nodes have arcs directed to g_i in the inferred network, we define the following notations:

- the expression levels of these nodes are denoted by vector $\boldsymbol{y} = y_1 y_2 \cdots y_{m_i}$;
- the confidence values of these arcs are denoted by vector $\boldsymbol{v_b} = v_b^1 v_b^2 \cdots v_b^{m_i}$;
- we use signed weights to represent the strength of these arcs, denoted by vector $\boldsymbol{w} = w_1 w_2 \cdots w_{m_i}$.

We assume that the gene-expression of a gene has two states, *on* and *off*. 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

$$k \cdot \boldsymbol{w} \cdot \boldsymbol{y} = Pr(g_i \text{ is } on | \boldsymbol{y})$$

Since there are 2^{m_i} configurations of $\boldsymbol{y}$, there are 2^{m_i} such equations. The value of $Pr(g_i \text{ is } on|\boldsymbol{y})$ can be directly taken from the CPTs. So $\boldsymbol{w}$ can be obtained by solving these equations, and $\boldsymbol{v_b}$ derived directly from $\boldsymbol{w}$.

Having the belief coefficient values, *RefineML* can proceed to calculate the variables $L_i(a)$ and $C_i(a)$ for each leaf i, as defined in Sec. 2.3, where a is the character value of the parent of leaf i inferred by FastML. For each site in the network adjacency matrices, its belief coefficient value k_b can be obtained from the corresponding vector $\boldsymbol{v_b}$. Then, using b and c to denote a character value where $b, c \in S$, the complete *RefineML* algorithm can be described as follows:

1. Learn the CPT parameters for the leaf networks reconstructed by the base inference algorithm and calculate the belief coefficient k_b for every site.
2. From the current leaves, infer ancestral sequences using FastML.
3. For each leaf i with value b, set
   - $L_i(a) = \max_{c \in S} p_{ac}(l_i) \cdot Q_i(c)$
   - $C_i(a) = \arg\max_{c \in S} p_{ac}(l_i) \cdot Q_i(c)$

   where
   $$Q_i(c) = \begin{cases} k_b, & b = c \\ 1 - k_b, & \text{otherwise.} \end{cases}$$
4. For each leaf i, assign its most likely character from the variable $C_i(a)$.

4.2 RefineFast and RefineML under the Extended Model

Since the basic network evolutionary model considers only edge gains and losses, refinement algorithms with this model require the input networks all have the same number of genes (orthologous across all species). Moreover, the gain or loss of an edge in that model is independent of any other event. However, this process accounts for only a small part of regulatory network evolution; in particular, gene duplication is known to be a crucial source of new genetic function and a mechanism of evolutionary novelty [31, 32].

The extended network evolutionary model not only enables broader application and more flexible parameterization, but also provides a direct evolutionary mechanism for edge gains and losses. For example, in the networks to be refined, the genes can have different numbers of copies for different organisms.

Within this broader framework, the phylogenetic information that we use lies on two levels: the evolution of gene contents of the networks and the regulatory interactions of the networks. The former can be regarded as the basis of the latter, and can be obtained by inferring the history of gene duplications and losses during evolution. We then extend our refinement algorithms [53] to handle this data and use different models of gene duplications and losses to study their effect on the performance of the refinement algorithms.

4.2.1 Models of gene duplications and losses

While networks evolve according to the extended network evolutionary model, a history of gene duplications and losses is created along the evolution. However, during reconstruction, this history may not be exactly reconstructed. Therefore, we propose other models of gene duplications and losses to approximate the true history:

- *The duplication-only model*: We assume that different gene contents are due exclusively to gene duplication events.
- *The loss-only model*: We assume that different gene contents are due exclusively to gene loss events.

We also compare outcomes when the true history is known.

4.2.2 Algorithm overview

We begin by collecting the regulatory networks to be refined. These networks may have already been inferred or they can be inferred from gene-expression data at this point using any of the standard network inference methods. The genes in these networks are not required to be orthologous across all species, as the duplication/loss model allows for gene families of various sizes. Refinement proceeds in the two-phase iterative manner already described, but adding a step for reconstruction of gene duplication and loss history and suitably modified algorithms for ancestral reconstruction and leaf refinement:

1. Reconstruct the history of gene duplications and losses, from which the gene contents for the ancestral regulatory networks (at each internal node of the species tree) can be determined. We present algorithms for history reconstruction with different gene duplication and loss models.
2. Infer the edges in the ancestral networks once we have the genes of these networks. We do this using a revised version of FastML.
3. Refine the leaf networks with new versions of *RefineFast* and *RefineML*.
4. Repeat steps 2 and 3 as needed.

4.2.3 Inferring gene duplication and loss history

With different gene history models and input information we have different ways to infer the gene duplication and loss history. The *duplication-only* and *loss-only* models allow simplifying the inference of the gene duplication and loss history and of the gene contents of the ancestors. For a certain internal node of the phylogenetic tree, with the *duplication-only* assumption, the intersection of the genes of all the leaves in the subtree rooted at this internal node is its set of genes, while with the *loss-only* assumption, the union of genes in all the leaves of the subtree is the set of genes. Gene duplication and loss histories inferred with these methods have a minimum number of gene duplications, respectively losses — they are optimal under the model.

With both the gene duplication and gene loss operations allowed, we use two different ways to infer this history. One is the reconciliation algorithms introduced earlier. This method requires the least amount of input information. It takes all the genes for each gene family, reconstructs the gene tree which is reconciled with the species tree so as to obtain the gene duplication and loss history. In our experiments, we use the parsimony-based reconciliation tool Notung [38] to get such duplication and loss histories.

When we have the orthology assignment for each gene family across species, this information can be leveraged for better inference of the history. FastML [34], which was designed to infer ancestral sequences given the sequences of a family of modern organisms, can be applied in this case after the following preprocessing. Suppose there are N different genes in all the modern organisms, we then represent the gene content of each organism with a binary sequence of length N, where the value at each position is assigned as 1 if the corresponding gene or its ortholog is present, otherwise 0. FastML can be used to obtain an estimate of these sequences for the ancestral organisms, with a character set

$\{0,1\}$ and the substitution matrix:

$$P_h = \begin{pmatrix} 1-p_d & p_d \\ p_l & 1-p_l \end{pmatrix} \quad .$$

Note that this approach assumes 1–1 orthologies, whereas orthology is a many-to-many relationship. In biological practice, however, 1–1 orthologies are by far the most common.

4.2.4 Inferring ancestral networks

We obtain the gene contents for all the networks over the tree from the previous step. In this step, we use the FastML framework to infer the regulatory connections in the ancestral networks.

Recall that FastML assumes independence among the entries of the adjacency matrices and reconstructs ancestral characters one site at a time. When the basic network evolutionary model is used, the gene content is the same in all networks, we can assign corresponding entries across networks, and use FastML to infer the ancestral characters for each entry at a time.

In the extended model, however, the gene content varies across networks, so it is not direct to assign corresponding entries across networks. We solve this problem by embedding all networks into one that includes every gene that appears in any network, taking the union of all gene sets. We then represent a network with a ternary adjacency matrix, where the rows and columns of the missing genes are filled with a special character x. All networks are thus represented with adjacency matrices of the same size. Since the gene contents of ancestral networks are known thanks to reconciliation, the entries with x are already identified in their matrices; other entries are reconstructed by our revised version of FastML, with a new character set $S' = \{0,1,x\}$. The substitution matrix P' for S' can be derived from the model parameters in Chapter 3, without introducing new parameters. Without loss of generality, we assume at each evolutionary step at most one gene duplication event and one gene loss event can happen. This simplifies the calculation of P', which is now calculated as following:

$$P' = \begin{pmatrix} p'_{00} & p'_{01} & p'_{0x} \\ p'_{10} & p'_{11} & p'_{1x} \\ p'_{x0} & p'_{x1} & p'_{xx} \end{pmatrix} = \begin{pmatrix} (1-p_l)\cdot p_{00} & (1-p_l)\cdot p_{01} & p_l \\ (1-p_l)\cdot p_{10} & (1-p_l)\cdot p_{11} & p_l \\ p_d \cdot \pi_0 & p_d \cdot \pi_1 & 1-p_d \end{pmatrix}$$

During inference of ancestral characters for each entry, we take special measures for x during calculation. Given P', let i, j, k denote a tree node, and $a,b,c \in S'$ possible values of a character at some node. 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.

On a binary phylogenetic tree, for each site, the revised FastML then works as follows:

1. If leaf i has character b, then, for each $a \in S'$, set $C_i(a) = b$ and $L_i(a) = p'_{ab}$.
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
   - 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 $C_i(a) = x$;
   - otherwise, for each $a \in S'$, set $L_i(a) = \max_{c\in\{0,1\}} p'_{ac} \cdot L_j(c) \cdot L_k(c)$ and $C_i(a) = \arg\max_{c\in\{0,1\}} p'_{ac} \cdot L_j(c) \cdot L_k(c)$.
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 $a \in \{0,1\}$ 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 $C_i(a)$.

4.2.5 Refining leaf networks: RefineFast

RefineFast uses the parent networks inferred by FastML to evolve new sample leaf networks. Because the strategy is just one of sampling, we do not alter the gene contents of the original leaves—duplication and loss are not taken into account in this refinement step. Let A_l and A_p be the adjacency matrices of a leaf network and its parent network, respectively, and let A'_l stand for the refined network for A_l; then the revised *RefineFast* algorithm carries out the following steps:

1. For each entry (i, j) of each leaf network A_l,
   - if $A_l(i, j) \neq x$ and $A_p(i, j) \neq x$, evolve $A_p(i, j)$ by P to get $A'_l(i, j)$;
   - otherwise, assign $A'_l(i, j) = A_l(i, j)$.
2. Use the $A'_l(i, j)$ to replace $A_l(i, j)$.

In this algorithm, the original leaf networks are used only in the first round of ancestral reconstruction, after which they are replaced with the sample networks drawn from the distribution of possible children of the parents.

4.2.6 Refining leaf networks: RefineML

To make use of the prior information (in the original leaf networks), *RefineML* uses a *belief coefficient* k_b for each entry of the adjacency matrices of these networks, which represents how much we trust the prediction by the base network inference algorithm. With the extended network evolution model, the value of k_b is the combination of two items. One is the weights of the edges given by the inference algorithm, which can be calculated from the CPT parameters of the predicted networks in the DBN framework, as described in Sec. 4.1.4. The other depends on the distribution of the orthologs of corresponding genes over other leaves. Denote the number of leaves by n_l, and the distance between leaf i and leaf j in the phylogenetic tree by d_{ij}, then the second item of k_b of a certain entry for leaf l can be calculated by

$$\frac{\sum_{i=1,\ldots,n_l,\ i\neq l} h_i d_{il}^{-1}}{\sum_{i=1,\ldots,n_l,\ i\neq l} d_{il}^{-1}}$$

where $h_i = 1$ if leaf i has the corresponding genes, $h_i = 0$ otherwise. This provides a weighting system to enable the entries which are shared by more leaves to have higher confidence values, subject to the distance between these leaves.

As in *RefineFast*, the refinement procedure does not alter the gene contents of the leaves. Using the same notations as for FastML and *RefineFast*, *RefineML* aims to find the A'_l which maximizes the likelihood of the subtree between A_p and A'_l. The revised *RefineML* algorithm thus works as follows:

1. Learn the CPT parameters for the leaf networks reconstructed by the base inference algorithm and calculate the *belief coefficient* k_b for every site.
2. For each entry (i, j) of each leaf network A_l, do:
   - If $A_l(i, j) \neq x$ and $A_p(i, j) \neq x$, let $a = A_p(i, j)$, $b = A_l(i, j)$,
     (a) let $Q(c) = k_b$ if $b = c$, $1 - k_b$ otherwise;
     (b) calculate the likelihood $L(a) = \max_{c\in\{0,1\}} p_{ac} \cdot Q(c)$;
     (c) assign $A'_l(i, j) = \arg\max_{c\in\{0,1\}} p_{ac} \cdot Q(c)$.
   - Otherwise, assign $A'_l(i, j) = A_l(i, j)$.
3. Use $A'_l(i, j)$ to replace $A_l(i, j)$.

4.3 Experimental Design under the Basic Model

The purpose of our experiments is to provide evidence for our hypothesis through a detailed examination of the sensitivity and specificity characteristics of our algorithm compared to the base inference algorithm. To test the performance of our approach, we need "noisy" regulatory networks as the input to our refinement algorithms. We obtain these networks by applying a base inference method on the gene-expression datasets of the family of species. We also need the "true" networks for these species to calculate the accuracy of output networks.

In our simulation experiments, we generate both the "true" regulatory networks and the gene-expression datasets. we evolve networks along a given tree from a chosen root network to obtain the "true" leaf networks. Then, in order to reduce the correlation between generation and reconstruction of networks, we use the leaf networks to create simulated expression data and use our preferred network inference method to reconstruct networks from the expression data. These inferred networks are the true starting point of our refinement procedure—we use the simulated gene expression data only to achieve better separation between the generation of networks and their refinement, and also to provide a glimpse of a full analysis pipeline for biological data. We then compare the inferred networks after and before refinement against the "true" networks (generated in the first step).

4.3.1 Simulated data generation

We generate test data from three pieces of information: the phylogenetic tree, the network at the root, and the network evolutionary model (which includes evolutionary operations, and evolutionary rates for each operation). We first generate the leaf networks from the root according to the network evolutionary model, and use these networks as the "true" networks; then generate gene-expression data for these leaf networks.

We need CPT parameters for each network to generate its corresponding gene-expression dataset. These CPT parameters come from quantitative relationships in the networks, so we need a step of calculating CPTs from the weights. Fig. 4.1 illustrates the whole data generation process; in the

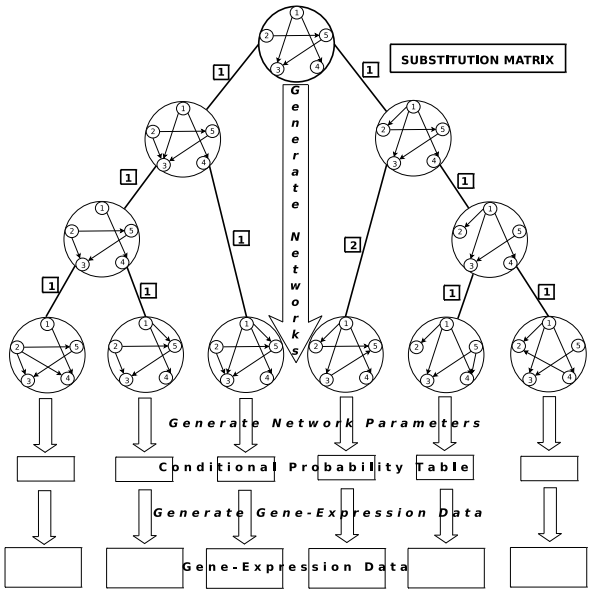


Figure 4.1: The data generation process

figure, the known conditions are shown with bold lines, characters in bold boxes, and the steps are labelled with italic characters.

To get consistent CPTs, we also evolve the quantitative relationships when generating networks along evolution. That is, we use a root network represented by a weighted adjacency matrix with signed weights. Then we do the following:

Denote the weighted adjacency matrix of the root network as A_p. Since in the basic network evolutionary model, there is only edge gain and loss operations, we can obtain the adjacency matrix for its child A_c by mutating A_p according to the substitution matrix. By repeating this process as we traverse down the tree we can obtain weighted adjacency matrices at the leaves. In other words, we evolve the weighted networks down the tree according to the model parameters, which is standard practice in the study of phylogenetic reconstruction [58, 59].

Having the weighted leaf networks we can generate gene-expression data from them. For this task we use both Yu's GeneSim [60] and *DBNSim* [53], our own design based on the DBN model, which are presented in more details below.

Gene-expression data generated by DBNSim

For *DBNSim*, we follow [42], using binary gene-expression levels, where 1 and 0 indicate that the gene is, respectively, *on* and *off*. Denote the expression level of gene g_i by x_i, $x_i \in \{0,1\}$; if m_i nodes have arcs directed to g_i in the network, let the expression levels of these nodes be denoted by the vector $\boldsymbol{y} = y_1 y_2 \cdots y_{m_i}$ and the weights of their arcs by the vector $\boldsymbol{w} = w_1 w_2 \cdots w_{m_i}$. From $\boldsymbol{y}$ and $\boldsymbol{w}$, we can get the conditional probability $Pr(x_i|\boldsymbol{y})$. Once we have the full parameters of the leaf networks, we generate simulated time-series gene-expression data. At the initial time point, the expression level of gene g_i is generated by the initial distribution $Pr(x_i)$; at time t, its expression level is generated based on $\boldsymbol{y}$ at time $t-1$ and the conditional probability $Pr(x_i|\boldsymbol{y})$.

Gene-expression data generated by GeneSim

GeneSim [60] can produce simulated gene-expression values for a given weighted network as well as generate arbitrary network structures. In contrast to our *DBNSim* method, GeneSim gives continuous gene-expression levels. Denoting the gene-expression levels of the genes at time t by the vector $\boldsymbol{x}(t)$, the values at time $t+1$ are calculated according to $\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + (\boldsymbol{x}(t) - \boldsymbol{z})C + \varepsilon$, where C is the weighted adjacency matrix of the network, the vector $\boldsymbol{z}$ represents *constitutive expression values* for each gene, and ε models noise in the data. The values of $\boldsymbol{x}(0)$ and $x_i(t)$ for those genes without parents are chosen uniformly at random from the range $[0, 100]$, while the values of $\boldsymbol{z}$ are all set to 50. The term $(\boldsymbol{x}(t) - \boldsymbol{z})C$ represents the effect of the regulators on the genes; this term needs to be amplified for the use of *DBI*, because of the required discretization. We use a factor k_e with the regulation term (set to 7 in our experiments), yielding the new equation $\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + k_e(\boldsymbol{x}(t) - \boldsymbol{z})C + \varepsilon$.

4.3.2 Tests

Simulated data allows us to control the parameters and, more importantly, to get an absolute assessment of accuracy. Other than possible issues about the biological verisimilitude of the simulated data, such simulations create the risk of introducing a systematic bias in the results. We take specific precautions against such bias, both in the design of the simulations and in the 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 [61], as also used by Kim *et al.* [15]; we also explore a wide range of evolutionary rates. Our networks are of modest size, with 16 genes each—this selection makes the gene-expression tables less "tall" and thus, at least in principle, less prone to generate errors in reconstruction, thus presenting a more challenging case for a boosting algorithm.

With two data generation methods, *DBNSim* and GeneSim, and two network inference algorithms as our base algorithms, *DBI* and *DEI*, we conduct experiments with different combinations of data generation methods and inference algorithms to verify that our boosting algorithms work under all circumstances. First, we use different data generation methods with the same inference algorithm. Since the binary gene-expression data generated by *DBNSim* does not fit *DEI*, we use *DBNSim* and GeneSim to generate data for *DBI*. We generate 200 time points for each gene-expression matrix, running the generation process 10 times to obtain the mean and standard deviation. Second, we apply *DBI* and *DEI* to datasets generated by GeneSim to infer the networks. Since *DEI* does not accept large datasets (with many time points), here we used smaller datasets than the previous group of experiments with 75 time points, yielding expression level matrix of size 16×75. Since the generation process is random according to the substitution probabilities and CPTs, we run the generation process 20 times for each choice of tree structure and parameters and calculate the mean and standard deviation. Finally, we conduct experiments with various evolutionary rates.

Comparing with the Bourque and Sankoff approach

Bourque and Sankoff's algorithm [35], thereafter the B&S algorithm, also uses phylogenetic information to improve the inference of gene networks. We therefore conduct experiments, using continuous data, to compare our approach to theirs.

Where is the Important Information?

Although we use only the direct parents to refine the leaves at each iteration, the leaves receive information from the whole tree, since the FastML algorithm assigns states to every internal node based on global information. We claim that the use of this global information is necessary. To verify this claim, we build a variation of our algorithms, that we call *RefineLocal*, where the ancestral reconstruction stops once the parents of leaves are reached. The resulting ancestral reconstruction, in other words, is now limited to exactly the parts of the tree used in the leaf refinement. *RefineLocal* works with both *RefineFast* and with *RefineML*, since it does not alter the refinement phase of the algorithm.

Part of the improvement is due to noise averaging, taking advantage of the independence in errors among the leaf networks. We claim that noise averaging not based on the correct phylogeny cannot produce the type of improvement we see. To verify this claim, we build a procedure that we call *RefineRandomTree*, which runs our full refinement procedures (either one), but does it on a tree where the initial inferred networks were randomly assigned to leaves. Since the tree topology is unchanged, the averaging effect over the data remains globally similar, but the phylogenetic relationships are destroyed. We run 100 such randomized tests and report the mean behavior.

4.3.3 Measurements

We want to examine the predicted networks at different levels of sensitivity and specificity. For *DBI*, on each dataset, 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 penalty coefficients. For each penalty coefficient, we apply *RefineFast*, *RefineML*, *RefineLocal*, and *RefineRandomTree* on the predicted networks. For *DEI*, we also choose 11 thresholds for each predicted weighted connection matrix to get networks on various sparseness levels. For each threshold, we apply *RefineFast*, *RefineLocal*, and *RefineRandomTree* on the predicted networks. We measure specificity and sensitivity to evaluate the performance of the algorithms and plot the values, as measured on the results for various penalty coefficients (for *DBI*) and thresholds (for *DEI*) to yield ROC curves. Recall that in such plots, the larger the area under the curve, the better the results.

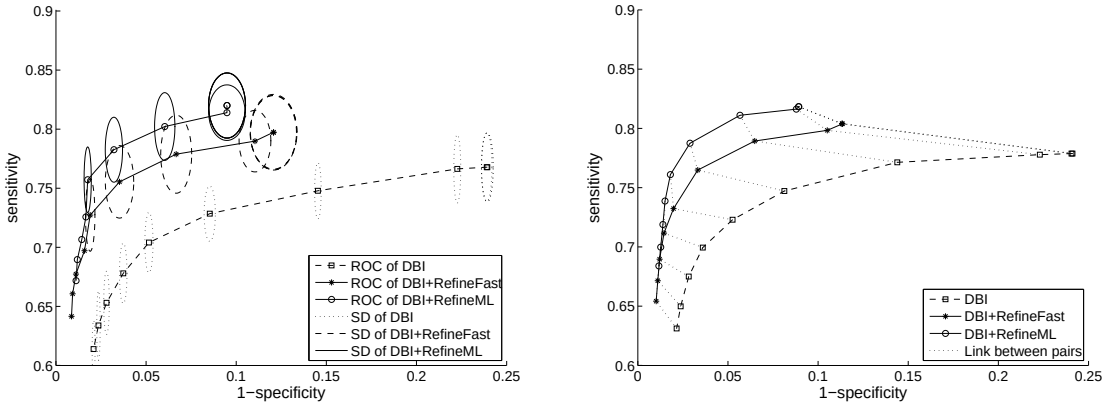
4.4 Results and Discussion under the Basic Model

We show results on two representative trees: tree *T1* has 41 nodes on 6 levels and is better balanced than tree *T2*, which has 37 nodes on 7 levels. Both trees were generated with an expected evolutionary rate of 2.2 events (gain or loss of a regulatory arc in the network) per edge and resulting leaf networks have from 23 to 38 edges.

4.4.1 On boosting under different experimental settings

Different gene-expression data generation methods, same inference algorithm

Fig. 4.2 shows the average performance of *RefineFast*, *RefineML*, and *DBI* on 10 noiseless datasets



(a) *Results on* T1*, with standard deviations of sensitivity and specificity*

(b) *Results on* T2*, with dotted lines indicating matched penalty coefficients*

Figure 4.2: ROC curves for DBI and boosting algorithms on the datasets generated by DBNSim

generated by *DBNSim* on trees *T1* (left) and *T2* (right). Throughout the range of parameters, our two algorithms clearly dominate *DBI*, with *RefineML* also dominating the simpler *RefineFast*: for every penalty coefficient, both sensitivity and specificity are improved from *DBI* to *RefineFast* and further improved from *RefineFast* to *RefineML*, as easily seen on the right. Sample standard deviations of sensitivity and specificity for these three methods on the noiseless datasets on *T1* are shown as ellipses, the loci of one standard deviation around each point. The separation between the curves is almost always larger than the standard deviations, so that our assertions of dominance of one method over another hold, not only on average, but also in the vast majority of cases. Also, as this figure demonstrates, the boosting effect remains similar on different phylogenies—and so we present results only on *T1* hereafter.

All three algorithms behave on the noisy datasets much as on the noiseless ones. Our refinement algorithms yield more improvement on the noisy datasets, which are closer to the real data and thus cause more difficulties for *DBI* methods, yielding a larger margin for improvement. We thus show results for noiseless datasets only, as the level of improvement caused by our algorithms can only increase as the noise level in the data increases. Fig. 4.3 shows the results of the three algorithms on the noiseless datasets generated by GeneSim on *T1*. The boosting effects are much the same as seen in Fig. 4.2, but it is clear that the *DBI* base algorithm does worse on the datasets generated by GeneSim than on those generated by *DBNSim*, as might be expected.

Different inference algorithms, same gene-expression data generation method

The datasets used in this experiment are generated by GeneSim. Fig. 4.4 shows the ROC curves of *DBI* and *DEI*, along with *RefineFast* boosting, on the same datasets; the refinement algorithm clearly dominates the base algorithms.

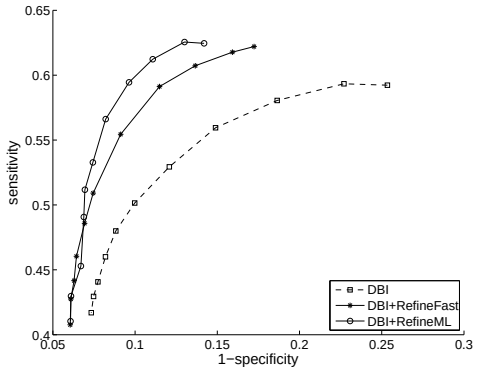


Figure 4.3: ROC curves for DBI and boosting algorithms on the datasets generated by GeneSim

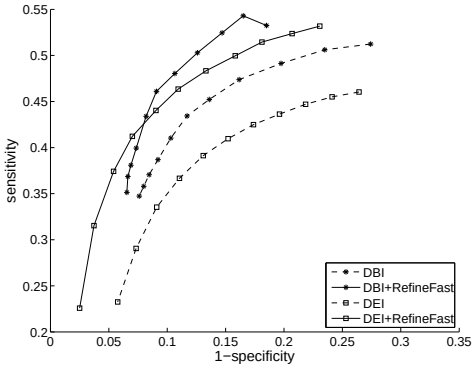


Figure 4.4: Results for DBI and DEI with RefineFast boosting on GeneSim generated datasets

Different evolutionary rates

The expected evolutionary rate (average edge length) was fixed in all experiments presented above. High rates of evolution cause various difficulties in phylogenetic reconstruction; In particular, they cause *saturation*, where the apparent number of evolutionary changes needed to explain the observed differences underestimates the actual number of changes that occurred over time. This problem is aggravated when each character has only two states, as two changes to the same character cancel each other. We thus expect our method to become less effective as evolutionary rates increase. To study this problem, we conducted experiments on tree *T1* with a root network of 16 nodes and 24 edges, using different evolutionary rates to generate the leaf networks. Fig. 4.5 shows ROC curves for *RefineML* and *DBI* with evolutionary rates of 2.32, 4.76 and 6.67 on noiseless datasets. The loss in performance as the rate of evolution increases is clear for both methods; since *DBI* itself suffers (perhaps because some networks produced in the simulation violate implicit assumptions), the loss in performance of *RefineML* is a combination of worsened leaf networks returned by *DBI* and worsened ancestral reconstruction by FastML. Yet boosting is evident in all cases and performance remains excellent at the very high evolutionary rate of 4.76: most paths from the root to a leaf in the tree have 5 edges and so, at that rate, have an expected length of 23.5, so that the expected number of changes from the root network almost equals the number of edges of that network—a very challenging problem and one that is remarkably well solved here.

4.4.2 On performance with respect to the B&S algorithm

Since B&S requires continuous time-series gene-expression data, we use the same datasets, generated by GeneSim, as in Fig. 4.4. Fig. 4.6 presents the performance of B&S and *RefineFast* based on both *DBI* and *DEI*. The results of B&S are shown as a cloud of points, obtained under different parameter settings. B&S does better than plain *DEI*, but is clearly dominated by our *RefineFast* based on *DEI*, meaning that our refinement algorithm gains more improvement than B&S does.

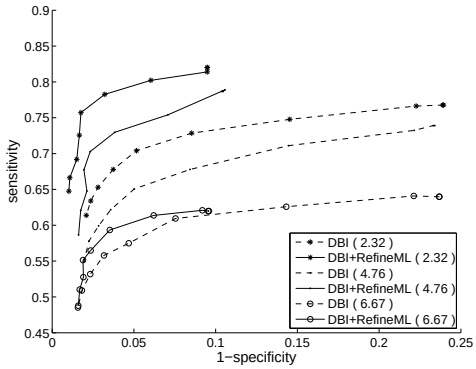


Figure 4.5: ROC curves for DBI and RefineML under various evolutionary rates

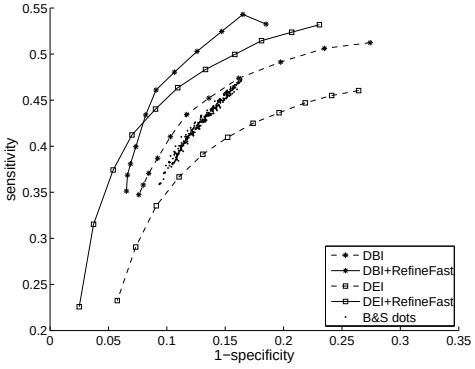
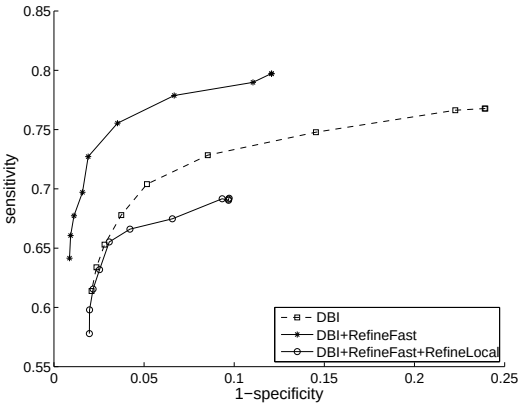


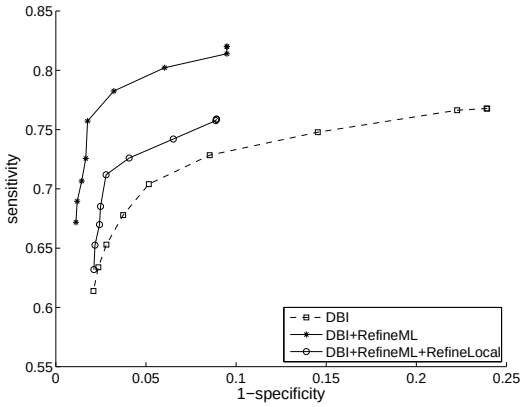
Figure 4.6: Performance for B&S and RefineFast based on DBI and DEI

4.4.3 On applying ML globally

We described earlier *RefineLocal*, a variant of our algorithms that infers ancestral networks only for the part of the tree that is used in the refinement phase. We use this algorithm to show that the improvement wrought in the leaves by our algorithms uses the phylogenetic information of the whole tree, not just the information present in the subforest induced by direct parents of leaves. Fig. 4.7(a) compares the performance of *RefineFast* with that of its localized version on noiseless datasets generated by *DBNSim* (the same datasets as in Sec. 4.4.1), while Fig. 4.7(b) does the same for *RefineML* on the same datasets. The plots are very similar: *RefineLocal* is clearly worse than the original algorithms, especially in terms of sensitivity. In fact, *RefineLocal* based on *RefineFast* does worse than *DBI*—due to the fact that the ancestral inference procedure introduces significant additional errors when limited to small subtrees. On the other hand, *RefineLocal* based on *RefineML* outperforms *DBI*—indicating that there is significant information present in the leaves, independent of the ancestral reconstruction.



(a) *ROC curves with RefineFast*

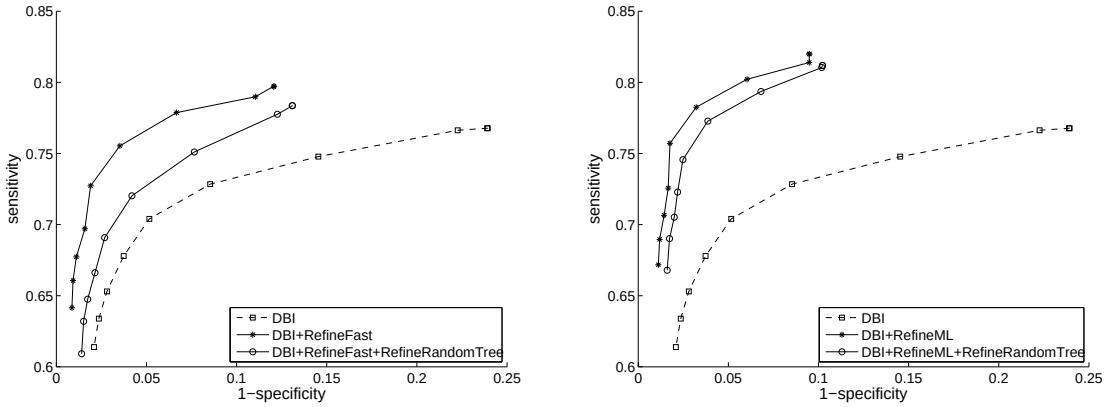


(b) *ROC curves with RefineML*

Figure 4.7: ROC curves for DBI and RefineLocal, showing RefineFast (left) and RefineML (right)

4.4.4 On phylogenetic information

In Sec. 4.3.2 we introduced *RefineRandomTree*, which carries out our full algorithms, but on a tree where the leaves have been reshuffled randomly. Its purpose is to demonstrate that the improvements we observe are not due entirely to noise averaging among the leaf networks. Fig. 4.8 compares the



(a) *ROC curves with RefineFast* (b) *ROC curves with RefineML*

Figure 4.8: ROC curves for DBI and RefineRandomTree, with RefineFast (left) and RefineML (right)

performance of *RefineFast* (left) and *RefineML* (right) run on the correct phylogenetic tree with the average performance (over 100 runs) of the same algorithm run after randomly reshuffling leaf labels. Both *RefineFast* and *RefineML* show clearly worse performance on the reshuffled trees than on the correct one. The results on the shuffled trees are still better than the base algorithm *DBI*, which shows the error averaging effect of the trees. However, this improvement depends on the performance of the base algorithm: in other experiments (not shown) with larger gene-expression datasets, where *DBI* does better, *RefineFast* on the shuffled trees does not outperform *DBI*, while *RefineML* with shuffled trees does. Overall, the results demonstrate the value of correct phylogenetic data, the value of the information present in the original leaf networks, and the averaging effect of the trees.

4.5 Experimental Design under the Extended Model

4.5.1 Data simulation

Similar to Sec. 4.3.1, in these experiments, the "true" networks for the organisms and their gene-expression data are both generated, starting from three pieces of input information: the phylogenetic tree, the network at the root, and the evolutionary model. To reduce the systematic bias during data simulation and result analysis, we use various phylogenetic trees from the literature and several choices of root networks. We also explore a wide range of evolutionary rates, especially different rates of gene duplication and loss. The root network is of modest size, between 14 and 17 genes, a relatively easy case for inference algorithms and thus also a more challenging case for a boosting algorithm.

We first generate the leaf networks that are used as the "true" regulatory networks for the chosen organisms. Since we need quantitative relationships in the networks in order to generate gene-expression data from each network, in the data generation process, we use adjacency matrices with signed weights. Weight values are assigned to the root network, yielding a weighted adjacency matrix A_p. To get the adjacency matrix for its child A_c, according to the extended network evolution model, 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 A_p, the values initialized either according to the *neutral initialization* model or the *inheritance initialization* model. (We conducted experiments under both models.) We denote the current adjacency matrix as A'_c. Secondly, edges in A'_c are mutated according to p_{01} and p_{10} to get A_c. We repeat this process as we traverse down the tree to obtain weighted adjacency matrices at the leaves, which is standard practice in the study of phylogenetic reconstruction [58, 59].

To test our refinement algorithms on different kinds of data, we use both *DBNSim* and Yu's GeneSim [60] (which are introduced in Sec. 4.3.1) to generate gene-expression data from the weighted leaf networks.

4.5.2 Groups of experiments

With two data generation methods, *DBNSim* and GeneSim, and two base inference algorithms, *DBI* and *DEI*, we conduct experiments with different combinations of data generation methods and inference algorithms to verify that our boosting algorithms work under all circumstances. First, we use *DBNSim* to generate data for *DBI*. We generate $13n$ time points for a network with n genes, since larger networks generally need more samples to gain inference accuracy comparable to smaller ones. Second, we apply *DEI* to datasets generated by GeneSim to infer the networks. Since the *DEI* tool TRNinfer does not accept large datasets (with many time points), here we use smaller datasets than the previous group of experiments with at most 75 time points. For each setup, experiments with different rates of gene duplication and loss are conducted.

For each combination of rates of gene duplication and loss, data generation methods, and base network inference methods, we get the networks inferred by *DBI* or *DEI* for the family of organisms. We then run refinement algorithms on each set of networks with different gene duplication and loss histories: the *duplication-only* history, the *loss-only* history, the history reconstructed by FastML given the true orthology assignment, and that reconstructed by Notung [38] without orthology information as input. Besides, since simulation experiments allow us to record the true gene duplication and loss history during data generation, we can also test the accuracy of the refinement algorithms with the true history, without mixing their performance with that of gene tree reconstruction or reconciliation. Each experiment is run 10 times to obtain average performance.

We again show the performance of the algorithms in ROC curves based on different settings of sensitivity and specificity. With *DBI*, to get inferred networks with different tradeoffs of sensitivity and specificity, 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 penalty coefficients. With *DEI*, we choose 11 thresholds for each predicted weighted connection matrix to get networks on various sparseness levels.

4.6 Results and Discussion under the Extended Model

We used different evolutionary rates to generate the networks for the simulation experiments. In [53] we tested mainly edge gain or loss rates; here we focus on testing different gene duplication and loss rates. We also conducted experiments on various combinations of gene-expression data generation methods and network inference methods. The inferred networks were then refined by refinement algorithms with different models of gene duplications and losses.

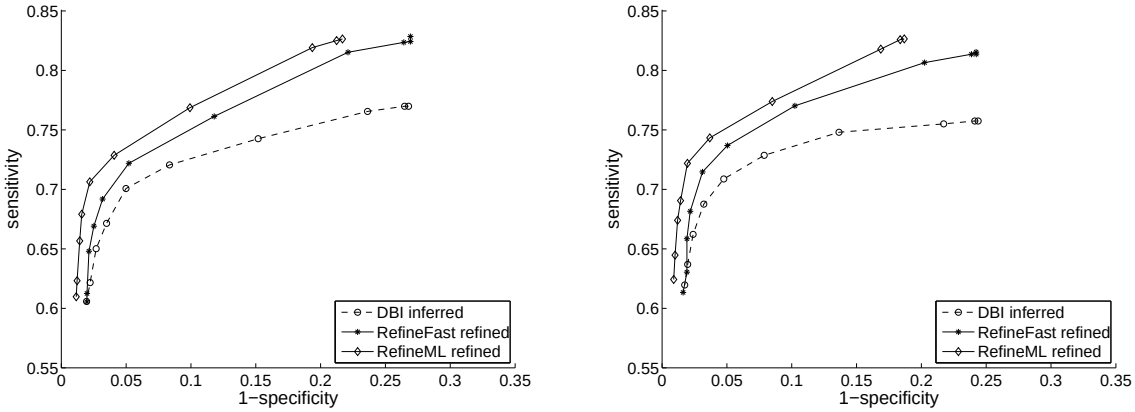
We do not directly compare the extended model with the basic, as the two do not lend themselves to a fair comparison—for instance, the basic model requires equal gene contents across all leaves, something that can only be achieved by restricting the data to a common intersection, thereby catastrophically reducing sensitivity.

Since the results of using *neutral initialization* and *inheritance initialization* in data generation are very similar, we only show results with the *neutral initialization* model. We first refine networks with the true gene duplication and loss history to test the pure performance of the refinement algorithms, then we present and discuss the results of refinement algorithms with several other gene evolution histories, which are more suitable for the application on real biological data. All results we show below are averages over 10 runs.

4.6.1 Refine with true history of gene duplications and losses

In Fig. 4.9, we show the results of the experiments with *DBNSim* used to generate gene-expression data, and *DBI* as base inference algorithm. All results with *DBI* inference that we show are on one representative phylogenetic tree with 35 nodes on 7 levels, and the root network has 15 genes. The left plot has a relatively high rate of gene duplication and loss (resulting in 20 duplications and 23 losses along the tree), while the right one has a slightly lower rate (with 19 duplications and 15 losses), again averaged over 10 runs.

Given the size of the tree and the root network, these are high rates of gene duplication and loss, yet, as we can see from Fig. 4.9, the improvement gained by our refinement algorithms remains clear in both plots, while *RefineML* further dominates *RefineFast* in both sensitivity and specificity, thanks to the appropriate reuse of the inferred leaf networks.



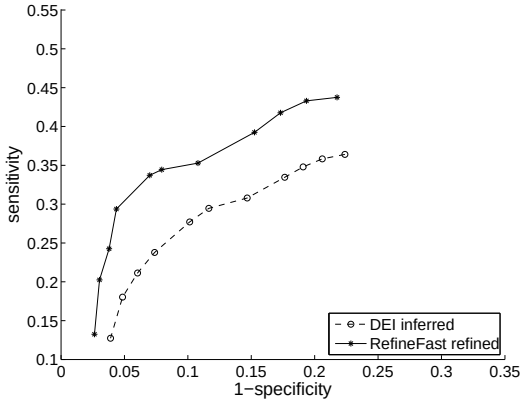
(a) *Results with higher gene duplication and loss rates* (b) *Results with lower gene duplication and loss rates*

Figure 4.9: Performance with extended evolution model and *DBI* inference method, and true history of gene duplications and losses.

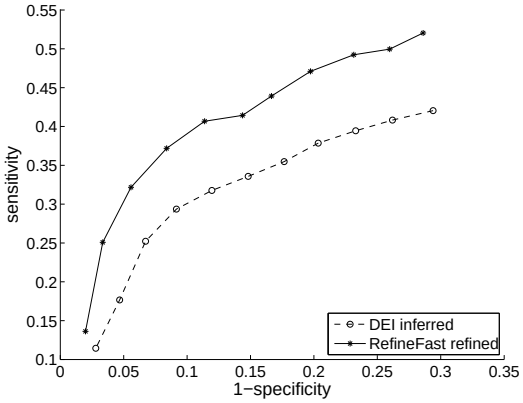
In the experiments with *DEI* network inference, GeneSim is used to generate continuous gene-expression data. In these experiments, the root network has 14 genes, and the phylogenetic tree has 37 nodes on 7 levels. The average performance of *DEI* and *RefineFast* over 10 runs is shown in Fig. 4.10. We also show results for two different evolutionary rates: Fig. 4.10(a) has higher gene duplication and loss rates, resulting in 15 duplications and 7 losses, while datasets in Fig. 4.10(b) have an average of 8 duplications and 3 losses. The *DEI* tool aims to infer networks with small gene-expression datasets. *RefineFast* significantly improves the performance of the base algorithm, especially the sensitivity. (Sensitivity for *DEI* is poor in these experiments, because of the inherent lower sensitivity of TRNinfer, as seen in [53] and also because of the reduced size of the gene-expression datasets.) Since the difference between the gene duplication and loss rates in Fig. 4.10(a) and Fig. 4.10(b) is large, we can observe more improvement in Fig. 4.10(b), which has lower rates. This is because high duplication and loss rates give rise to a large overall gene population, yet many of them exist only in a few leaves, so that there is not much phylogenetic information to be used to correct the prediction of the connections for these genes.

4.6.2 Refine with *duplication-only* and *loss-only* histories

We have seen from Figs. 4.9 and 4.10 that our two refinement algorithms improve the networks inferred by both *DBI* and *DEI*. Since the accuracy of *DBI* is much better than that of *DEI*, which causes more difficulty for refinement algorithms, and since *RefineML* does clearly better than *RefineFast*, hereafter we only show results with *DBI* inference and *RefineFast* refinement, which are on the



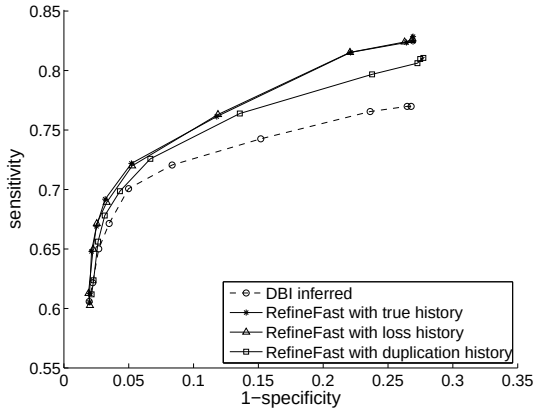
(a) *Results with higher gene duplication and loss rates*



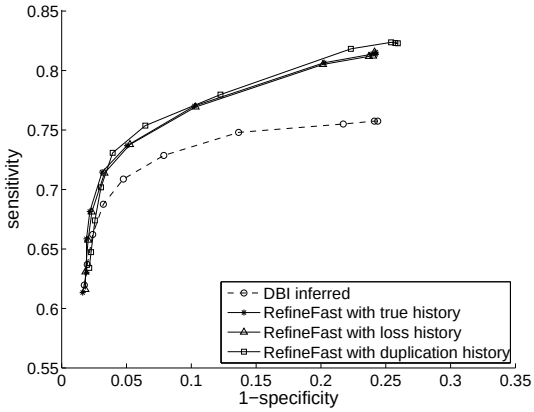
(b) *Results with lower gene duplication and loss rates*

Figure 4.10: Performance with extended evolution model and *DEI* inference method, and true history of gene duplications and losses.

same datasets as used in Fig. 4.9.



(a) *Results with higher gene duplication and loss rates*



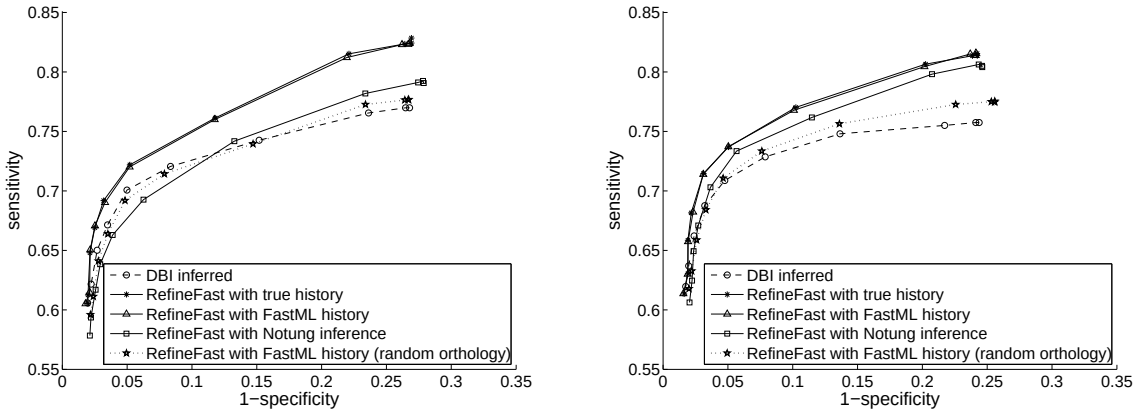
(b) *Results with lower gene duplication and loss rates*

Figure 4.11: Performance with extended evolution model and *DBI* inference method, with *duplication-only* and *loss-only* histories.

Fig. 4.11 shows the comparison of the performance of *DBI* and *RefineFast* with respectively the true gene duplication and loss history, the *duplication-only* history and the *loss-only* history assuming correct orthology assignment. The *duplication-only* and *loss-only* assumptions are at the opposite (and equally unrealistic) extremes of possible models of gene family evolution — their only positive attribute is that they facilitate the reconstruction of that evolution. Yet we see that *RefineFast* still improves the base network inference algorithm with both models. The performance of the *duplication-only* history differs between Fig. 4.11(a) and Fig. 4.11(b): in Fig. 4.11(a), it does worse than the true history and the *loss-only* history, while in Fig. 4.11(b), its performance is comparable with the other two. This is because there are more gene losses than gene duplications in the left plot, but more gene duplications than gene losses in the right plot, which the *duplication-only* history matches better. The performance of the *loss-only* history appears to be steady and not much affected by different evolutionary rates.

4.6.3 Refine with inferred histories of gene duplications and losses

In Fig. 4.12, we show the performance of *RefineFast* with various inferred gene duplication and loss histories, compared to that with the true history. FastML is applied to infer history with correct orthology information as described earlier. To test the value of having good orthology information, we also assign orthologies at random and then use FastML to infer ancestral gene contents. In each run, the refinement procedure with this history is repeated 20 times to get average results over 20 random orthology assignments. Finally, we use Notung to reconstruct a gene duplication and loss history without orthology input; Notung not only infers the gene contents for ancestral networks, but also alters the gene contents of the leaves.



(a) *Results with higher gene duplication and loss rates* (b) *Results with lower gene duplication and loss rates*

Figure 4.12: Performance with extended evolution model and *DBI* inference method, with inferred mixture histories.

In both Fig. 4.12(a) and Fig. 4.12(b) the FastML reconstructed history with correct orthology does as well as the true history. In fact, the history is very accurately reconstructed, which explains why the two curves agree so much. However, with the history reconstructed by FastML under random orthology assignments, the refinement algorithm only improves slightly over the base algorithm. With Notung inference *RefineFast* still dominates *DBI* in Fig. 4.12(b), but not in Fig. 4.12(a) which has higher evolutionary rates.

4.6.4 On using histories of gene duplications and losses, and orthology assignments

Our experiments with various evolutionary histories lead to several conclusions:

1. Good orthology assignments are important.

2. When we have good orthology assignments, the refinement algorithms need not rely on the true history of gene duplications and losses. We can use the *loss-only* history or the history reconstructed by FastML, both of which are easy to build and lead to performance similar to that of the true history.

4.7 Discussion and Conclusions

We present algorithms, models and experimental support for our claim that phylogenetic information can be used to improve the inference of regulatory networks for a family of related organisms. Our approach is best viewed as a booster for existing inference algorithms and can, in principle, be used with any favored network inference tool and any favored phylogenetic reconstruction algorithm.

Specifically, we present versions of our evolutionary approach for two network evolutionary models, the basic model and the extended model. As the extended model takes into account gene duplication and loss events during evolution, which are thought to play a crucial role in evolving new functions and interactions [31, 32], the algorithms with this extension have a broader range of applicability.

Furthermore, to give a comprehensive analysis of the factors which affect the performance of the refinement algorithms under the extended evolutionary model, we conducted experiments with different histories of gene duplications and losses, and different orthology assignments. Results of experiments under various settings show the effectiveness of our refinement algorithms with the new model throughout a broad range of gene duplications and losses.

Chapter 5

Probabilistic Phylogenetic Refinement Models ProPhyC and ProPhyCC

In Chapter 4, we presented *refinement* algorithms *RefineFast* and *RefineML*, based on phylogenetic information and using a likelihood framework, that boost the performance of any chosen base network inference method. They 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 labelling 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.

Although *RefineFast* and *RefineML* allow us to exploit from the phylogenetic information, we wonder whether there are ways to make further use, and, ideally, make full use of this information. For example, when using ancestral networks to refine the leaf networks in *RefineFast* and *RefineML*, we only used the direct parents of the leaves to correct the leaves. We choose to use only the direct parents as a tradeoff between the accuracy of ancestral network reconstruction and the distance to the leaves, but a way to reasonably use all ancestors should make better use of the ancestral information.

Therefore, we design a probabilistic phylogenetic model and associated algorithms, that we call *ProPhyC*, to refine regulatory networks for a family of organisms [62]. As with *RefineFast* and *RefineML*, *ProPhyC* takes as input a phylogenetic tree and inferred networks for a family of organisms, and uses the phylogenetic relationships to produce refined networks. Compared to the previous two-step algorithms, *ProPhyC* is an integrated model which has input noisy networks, output refined networks, and ancestral networks all in one graphical model. This framework can accommodate a large variety of evolutionary models of regulatory networks with only slight modifications, as we demonstrate in the methods section. Given that the evolution of regulatory networks is not yet well understood and given the several different models for regulatory network evolution [28, 32, 35], a comprehensive refinement model like this is highly desirable. We present algorithms and experimental results in this refinement model for both the basic and the extended network evolutionary models. We also show how to calculate and incorporate position-specific confidence values from input networks predicted by base inference methods.

We begin by describing *ProPhyC*, our probabilistic phylogenetic model to refine regulatory networks and the associated refinement algorithms under the two network evolutionary models. We then present an analysis of a comprehensive collection of experiments designed to assess our 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 datasets. We compare the accuracies of the networks produced by the base methods (especially dynamic Bayesian inference, *DBI*, the method devised for

DBNs) and of the networks after refinement, to get absolute assessments. In order to get relative assessments, we also use *RefineFast* and *RefineML* to refine the same networks and compare the outcome with that of *ProPhyC*. 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, including our own *RefineFast* and *RefineML*.

5.1 Models and Methods

5.1.1 The *ProPhyC* model: probabilistic phylogenetic refinement

ProPhyC is a probabilistic phylogenetic model designed to refine the inferred 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. 5.1. The shaded nodes labeled in upper case represent the input noisy networks, while the nodes labeled in lower case represent the correct networks that we want to infer. In turn, the correct networks are the leaves of the rooted phylogenetic tree of these organisms; internal nodes in this tree correspond to ancestral regulatory networks. The edges in this graph can thus be classified into two categories: (i) edges in the phylogenetic tree and (ii) edges from correct leaf networks to noisy ones. The first category of edges represents the evolution from a parent network to a child network, while the second category represents the error-prone process of inferring networks from latent correct networks. The parameters for this model are the substitution matrices P and Q. P represents the transition parameters from an ancestral network to its child network, subject to the network evolutionary model. Q represents the difference from the "true" networks to the inferred (observed, from the point of view of the *ProPhyC* model) noisy networks, which is associated with one's confidence in the base network inference method.

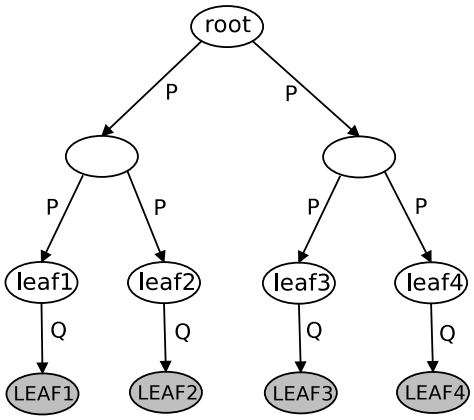


Figure 5.1: The *ProPhyC* model

The input information for this model is thus the phylogenetic tree, the noisy leaf networks, and the network evolutionary model. With a dynamic programming algorithm to maximize the likelihood of the whole graph, we can infer all of the ancestral networks and the "true" leaf networks. These "true" leaf networks inferred are the refined versions of the noisy input networks for these organisms. This framework can be generalized to fit different network evolutionary models. We name the basic refinement algorithm after the model and call it the *ProPhyC* algorithm.

Some base inference methods can predict regulatory networks with different confidence on different edges or non-edges, 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 name this version of the refinement algorithm *ProPhyCC*.

In our phylogenetic probabilistic model as illustrated in Fig. 5.1, we know the networks only for the shaded nodes, and all other networks are to be inferred. We use a dynamic programming algorithm to find the configuration of these networks which maximizes the likelihood of the entire model. We number the unknown nodes in Fig. 5.1 from 1 to n_t, where n_t denotes the number of nodes in the phylogenetic tree.

5.1.2 *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, Π and P, 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 $L_i(a) = \max_{c \in S} p_{ac} \cdot q_{cb}$ and $C_i(a) = \arg\max_{c \in S} p_{ac} \cdot q_{cb}$.
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 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 $\pi_a \cdot L_j(a) \cdot L_k(a)$.
5. Traverse the tree from the root, assigning to each node its character by $C_i(a)$.

The running time of this algorithm is $O(n_l \cdot n^2)$, where n is the number of genes in each network, and n_l is the number of leaves in the phylogenetic tree.

5.1.3 *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 [37–39] or by the algorithms that we presented in earlier work under the *duplication-only* or *loss-only* model [55].

Under the basic model, we assume independence among the entries of the adjacency matrices and so greatly simplify 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 missing genes are filled with a special character x. All networks are thus represented with adjacency matrices of the same size. Since the gene contents of ancestral networks are known thanks to reconciliation,

the entries with x are already identified in their matrices; the other entries are reconstructed by the refinement algorithm using the new character set $S' = \{0,1,x\}$. The substitution matrix P' for S' can be derived from the model parameters, without introducing new parameters. Assuming that at most one gene duplication and one gene loss can happen at each evolutionary step, we have:

$$P' = \begin{pmatrix} p'_{00} & p'_{01} & p'_{0x} \\ p'_{10} & p'_{11} & p'_{1x} \\ p'_{x0} & p'_{x1} & p'_{xx} \end{pmatrix} = \begin{pmatrix} (1-p_l)\cdot p_{00} & (1-p_l)\cdot p_{01} & p_l \\ (1-p_l)\cdot p_{10} & (1-p_l)\cdot p_{11} & p_l \\ p_d \cdot \pi_0 & p_d \cdot \pi_1 & 1-p_d \end{pmatrix} \quad .$$

We also extend the parameter Q to be Q' to fit the new character set S':

$$Q' = \begin{pmatrix} q'_{00} & q'_{01} & q'_{0x} \\ q'_{10} & q'_{11} & q'_{1x} \\ q'_{x0} & q'_{x1} & q'_{xx} \end{pmatrix} = \begin{pmatrix} q_{00} & q_{01} & 0 \\ q_{10} & q_{11} & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad .$$

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 $C_i(a) = \arg\max_{c \in S'} p'_{ac} \cdot q'_{cb}$.
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
   - 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 $C_i(a) = x$;
   - 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 $C_i(a)$.

5.1.4 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 *ProPhyC* 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. That is, *ProPhyCC* is an extended version of *ProPhyC* which takes advantage of position-specific confidence values for different entries in different networks. These values are embedded in Q'.

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 *DBI* inferred networks from the gene-expression data on the inferred structure [57], and then calculate the confidence values from the CPTs, as described in Sec. 4.1.4. Under the extended network evolutionary model, the confidence values also take into account the distribution of the orthologs of a certain gene family over all leaf networks, as described in Sec. 4.2.6.

5.2 Experimental Design under the Basic Model

As a first indicator of the performance of *ProPhyC*, we design a preliminary comparison between *ProPhyC* and *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 the following experiments. We use standard network inference algorithms to infer regulatory networks for the family of organisms from their gene-expression data, and then use our approach to refine the inferred networks. In the results presented here, the base algorithm is dynamic Bayesian inference (*DBI*). To obtain a detailed assessment of the performance of the *ProPhyC* model, we conduct simulation experiments for both network evolutionary models. With the basic network evolutionary model, we also apply our refinement algorithms to biological data that we assembled for 12 *Drosophila* species.

In experiments with both the basic and the extended network evolutionary model, 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 [61], as also used by Kim *et al.* [15]. The root network is of modest size, between 14 and 17 genes, a relatively easy case for inference algorithms and thus also a more challenging case for a boosting algorithm. We explore a wide range of evolutionary rates, including rates of gene duplication and loss, and of edge gain and loss, to verify that our approach works under all circumstances.

5.2.1 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: (i) generate the "true" leaf networks and (ii) generate the gene-expression data, the whole process starting from three pieces of input information: the phylogenetic tree, the network at its root, and the evolutionary model. *DBNSim*, based on the DBN model [53], is used to generate gene-expression data from the "true" networks. The details of the generation of simulated data are described in Sec. 4.3.1.

For all experiments on simulated gene-expression data, since the data generation process is sampling from a distribution, for each choice of tree structure and parameters, we run the generation process 10 times to obtain mean and standard deviation. When the networks are evolved under the basic network evolutionary model, for each leaf network, we generate 200 time points for its gene-expression matrix with *DBNSim*.

5.2.2 Biological data collection

Despite the advantages of simulation experiments (which allow an exact assessment of the performance of the inference and refinement algorithms), results on biological data are highly desirable, as such data may prove quite different from what was generated in our simulations.

To test the refinement algorithms on biological data, we need the "true" networks for the chosen organisms as benchmark to calculate the accuracies of the predicted and refined networks. *Transcription factor binding site* (TFBS) data is used to study regulatory networks, assuming that the

regulatory interactions determined by transcription factor binding share many properties with the real interactions [28, 29, 63]. Given this close relationship between regulatory networks and TFBSs and given the large amount of available data on TFBSs, we choose to use TFBS data to derive regulatory networks for the organisms as their "true" networks. We add noise into these "true" networks to obtain noisy networks as input of our refinement algorithm.

We use transcription factor binding site (TFBS) data for the *Drosophila* family (whose phylogeny is well studied) with 12 organisms: *D. simulans*, *D. sechellia*, *D. melanogaster*, *D. yakuba*, *D. erecta*, *D. ananassae*, *D. pseudoobscura*, *D. persimilis*, *D. willistoni*, *D. mojavensis*, *D. virilis*, and *D. grimshawi*. The TFBS data is drawn from the work of Kim *et al.* [64], where the TFBSs are annotated for all 12 organisms on 51 *cis*-regulatory modules (CRMs). 7 transcription factors were studied in their work, which are *Dstat*, *Bicoid*, *Caudal*, *Hunchback*, *Knirps*, *Kruppel*, and *Tailless*. Since each CRM corresponds to a target gene, we get a regulatory network with 58 nodes for each organism. These networks are used as the "true" regulatory networks for these 12 organisms.

5.2.3 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*.

5.2.4 Measurements

We want to examine the predicted networks at different levels of sensitivity and specificity. With *DBI*, we can use a penalty coefficient to modulate the weight of the penalty on structure complexity when inferring the regulatory networks in a DBN framework, so as to obtain different tradeoffs between sensitivity and specificity. On each dataset, 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.)

5.3 Experimental Results under the Basic Model

We begin with a preliminary comparison between *ProPhyC* and *RefineFast* on simulated noisy networks, to demonstrate the large improvement over the best prior results. We then proceed to more detailed results. With networks inferred from gene-expression data as input for *ProPhyC*, *ProPhyCC*, *RefineFast* and *RefineML*, we conducted experiments with different combinations of networks evolutionary models and types of datasets. Under each setting, we show both the absolute and relative assessments. Part of the data we use comes from the Drosophila family—we briefly discuss our results for this family.

5.3.1 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 $(1 - \textit{specificity})$ vs. *sensitivity* for each parameter setting. Fig. 5.2 shows the results on a phylogenetic tree of 37 nodes on 6 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.

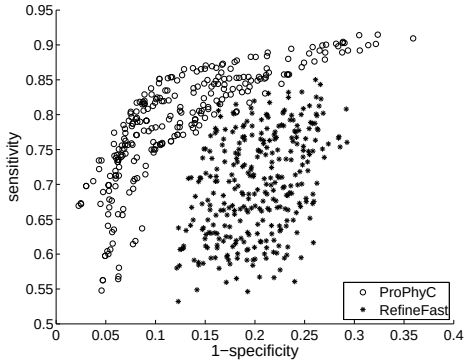
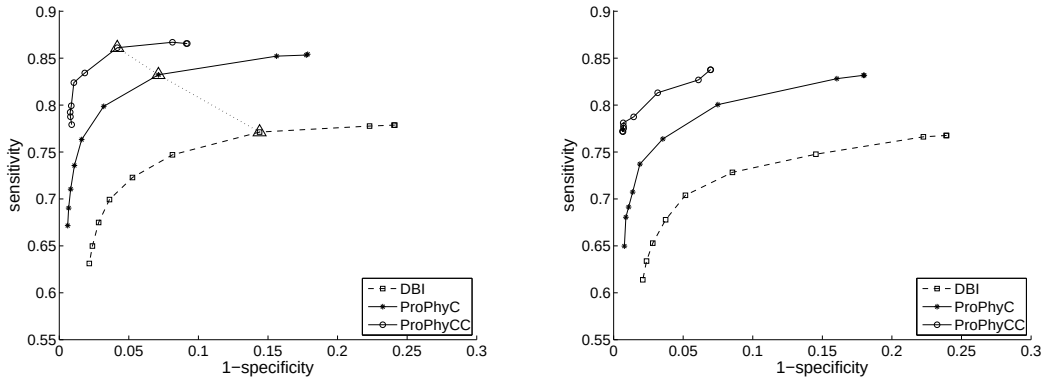


Figure 5.2: Preliminary comparison of *ProPhyC* and *RefineFast*

5.3.2 Performance on simulated data

Absolute results: comparison with the base inference algorithm DBI

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. 5.3 shows the results of *ProPhyC* and *ProPhyCC* on the networks predicted by *DBI*. We can see that *ProPhyC* and *ProPhyCC* improve both sensitivity and specificity significantly 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. 5.3(a) marked by triangles correspond to the same penalty coefficient on the three curves. We can see that in going from *DBI* to *ProPhyCC*, the sensitivity increases from 77% to 86%, while the specificity increases from 86% to 96%. Similar improvements can be observed with (i) other trees; (ii) other evolutionary rates; (iii) other base methods.

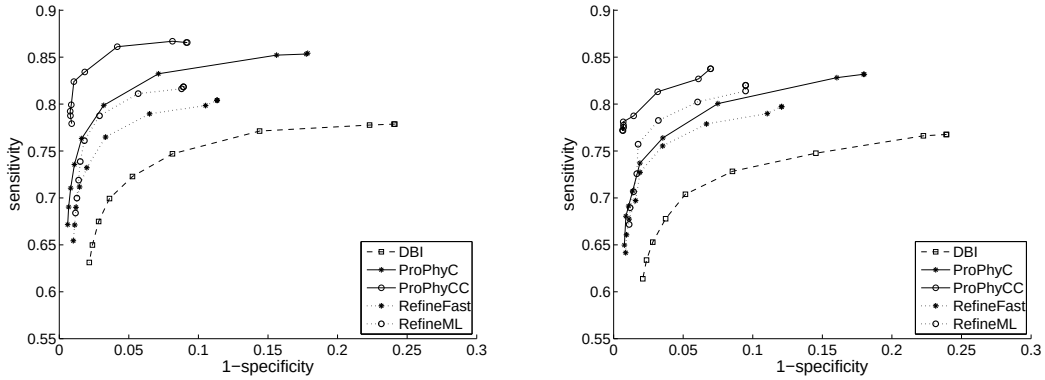


(a) *On the tree with 37 nodes and 7 levels* (b) *On the tree with 41 nodes and 6 levels*

Figure 5.3: Results of refinement algorithms with basic network evolutionary model, comparison of *ProPhyC* and *ProPhyCC* with base inference algorithm *DBI*. In part (a), the dotted lines join data points for the same model penalty coefficient

Relative results: comparison with the previous best

Fig. 5.4 shows the same experiments as in Fig. 5.3, but adds curves for *RefineFast* and *RefineML* to provide a comparison between different refinement approaches. Among the four refinement al-



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

Figure 5.4: Results of refinement algorithms with the basic network evolutionary model, comparison of *ProPhyC* and *ProPhyCC* with RefineFast and RefineML

gorithms, *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. 5.3 and Fig. 5.4, we conclude that refinement algorithms under our new model outperform not only the base inference algorithm, but also previous refinement algorithms on simulated data.

5.3.3 Performance on biological data

In these experiments we use datasets collected for 12 species of *Drosophila*, whose phylogenetic tree is illustrated in Fig. 5.5. The nodes of the regulatory networks consist of 7 transcription factors and 51 CRMs, such that an interaction between a transcription factor and a CRM implies an interaction between this transcription factor and the target gene of this CRM. The transcription factors and CRMs we choose are involved in the control of anterior-posterior segmentation in the blastoderm

stage embryo.

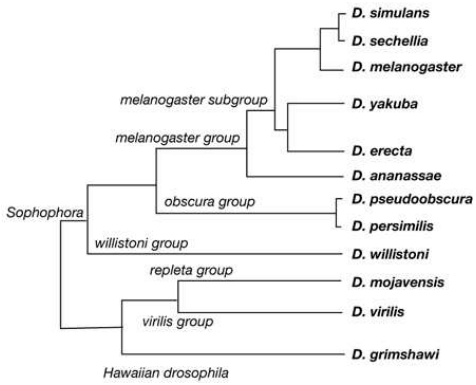


Figure 5.5: The phylogeny connecting the 12 Drosophila species [65].

Two parameters are used to add noise into the "true" networks to obtain noisy networks: one is the rate to introduce false positive, the other to introduce false negative. We use different noisy 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.5.6 shows the accuracies of these networks plotted as points. The cloud of points for *ProPhyC* clearly

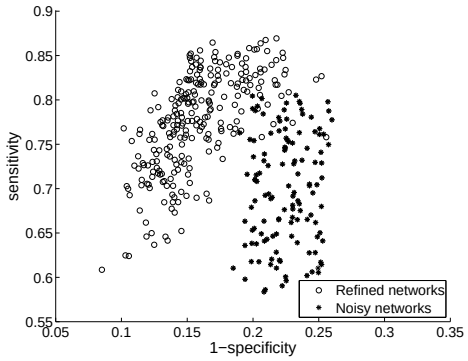


Figure 5.6: Results of *ProPhyC* with basic network evolutionary model on biological datasets

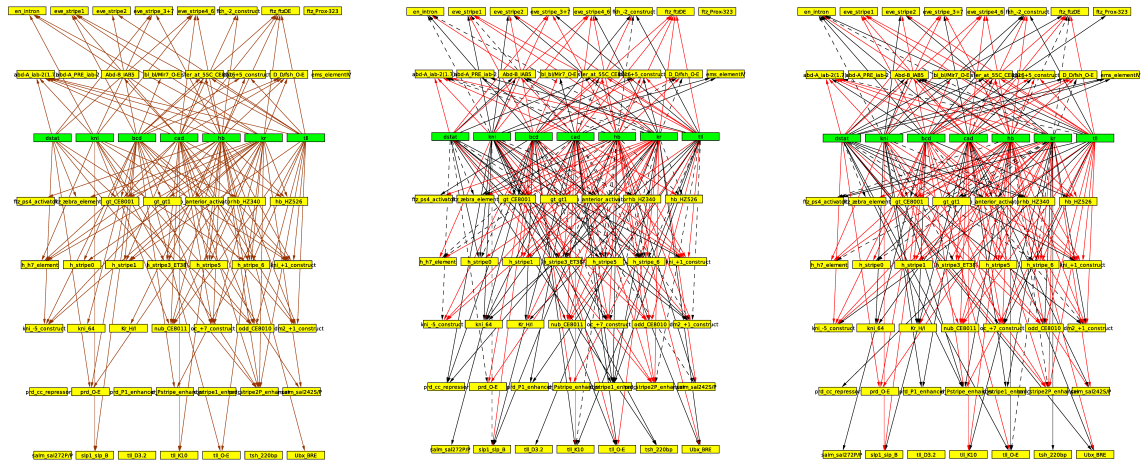
dominates that of the noisy networks, and the two clouds are well separated; the average improvement brought by *ProPhyC* is roughly 7% in each of sensitivity and specificity.

ProPhyC allows tradeoffs between sensitivity and specificity by using different parameters. Table 5.1 shows three examples.

Table 5.1: Examples of performance of ProPhyC with difference emphasis of improvement

	sensitivity (noisy $\rightarrow$ refined)	specificity (noisy $\rightarrow$ refined)
improve both	59.9% $\rightarrow$ 66.3%	80.0% $\rightarrow$ 86.5%
focus on sensitivity	59.5% $\rightarrow$ 69.2%	69.3% $\rightarrow$ 72.7%
focus on specificity	57.7% $\rightarrow$ 58.5%	70.1% $\rightarrow$ 80.0%

In Fig. 5.7 we show 3 versions of the Drosophila melanogaster network: the "true" network, the noisy network with random noisy, and the network refined by *ProPhyC* based on the noisy network.



(a) *The "true" network for D. melanogaster*

(b) *The noisy network for D. melanogaster. (Sensitivity 73.19%, specificity 70.42%)*

(c) *The refined network for D. melanogaster. (Sensitivity 84.20%, specificity 73.02%)*

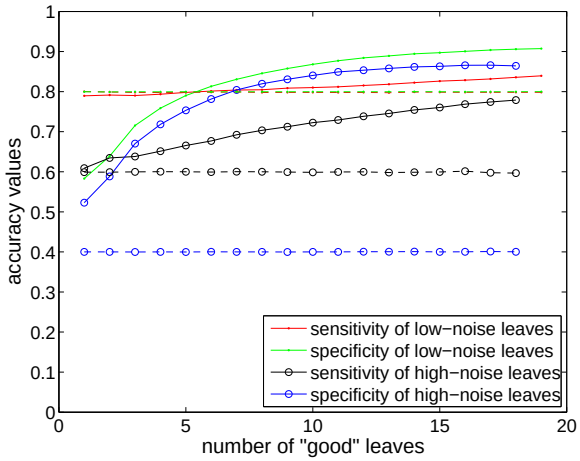
Figure 5.7: Comparison of the "true" network, the noisy network, and the refined network for D. melanogaster in one run of our test. Nodes in green are transcription factors. In parts (b) and (c), edges in red are true edges (present in the true network) and those in solid black are false positive edges (not present in the true network), while those in dashed black are false negatives (present in the true network, but not in the network under study).

5.3.4 Results with biased leaves

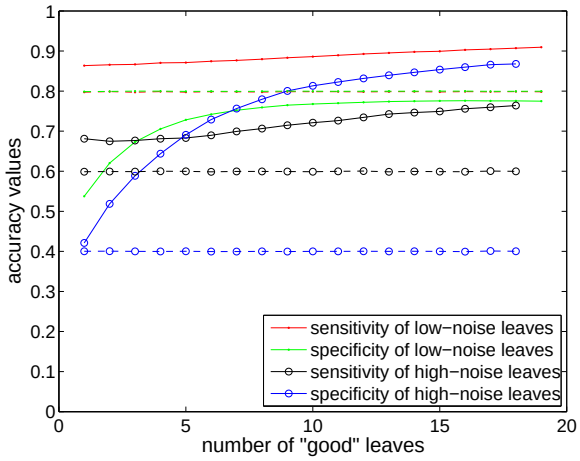
We show the results of *ProPhyCC* refining the leaf networks with different noise rates. The tree we use here has 19 leaves and 7 levels. We test the number of "good" leaves from 1 to 19. With each number of "good" leaves, we randomly choose 100 sets of "good" leaves to get the average results. In the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

In Fig. 5.8 we show the results of *ProPhyCC* with 2 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 5.8(a) 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 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. 5.8(b) 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.



(a) With parameter setting 1 for *ProPhyCC*



(b) With parameter setting 2 for *ProPhyCC*

Figure 5.8: Results of *ProPhyC* with biased leaves. Dashed lines show the accuracy values before refinement, while solid lines after refinement by *ProPhyCC*.

5.4 Experimental Design under the Extended Model

With the extended evolutionary model, conducting experiments with real data involves several extra steps besides the refinement step, each of which is a potential source of errors. For example, assuming we have identified gene families of interest, we need to build gene trees or assign orthologies for these genes to be able to reconstruct a history of duplications and losses. Any error in gene tree reconstruction or orthology determination leads to magnified errors in the history of duplications and losses. Assessing the results under such circumstances (no knowledge of the true networks and many complex sources of error) is not possible, so we turned to simulation for this part of the testing. This decision does not prejudice our ability to apply our approach to real data and to infer high-quality networks: it only reflects our inability to compute precise accuracy scores on biological data.

5.4.1 Data

In these experiments we use simulated networks and gene-expression data for the modern organisms. The “true” networks are simulated with the same method as the one used for testing the *RefineFast* and *RefineML* algorithms described in Sec. 4.5.1, and the gene-expression data is generated with

the same *DBNSim* procedure. We generate $13n$ time points of gene-expression data for a leaf network with n genes, since larger networks generally need more samples to gain inference accuracy comparable to smaller ones.

5.4.2 Tests

When the leaf networks are evolved under an evolutionary model that includes gene loss and duplication (the extended model), the networks can have different gene contents across organisms, that is, the genes can have different numbers of copies in different organisms. In this case, we know the gene contents only for the input leaf networks, not for the ancestral networks. Therefore, before running the refinement algorithms *ProPhyC* or *ProPhyCC*, we add a preprocessing step to obtain the gene contents of the ancestral networks, by inferring the gene duplication and loss history during evolution.

In Chapter 4 we analyzed various duplication-loss history models and their effect on the performance of *RefineFast* and *RefineML*. The simulation experiments showed that accurate history information with reliable orthology assignments help 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, the only option when dealing with biological data. We use Notung [38] as the reconciliation tool.

The rates of gene duplication and loss during evolution is another factor that can affect the performance of refinement algorithms. To get a comprehensive assessment of *ProPhyC* and *ProPhyCC* under different conditions, we conduct simulation experiments with different gene duplication and loss rates.

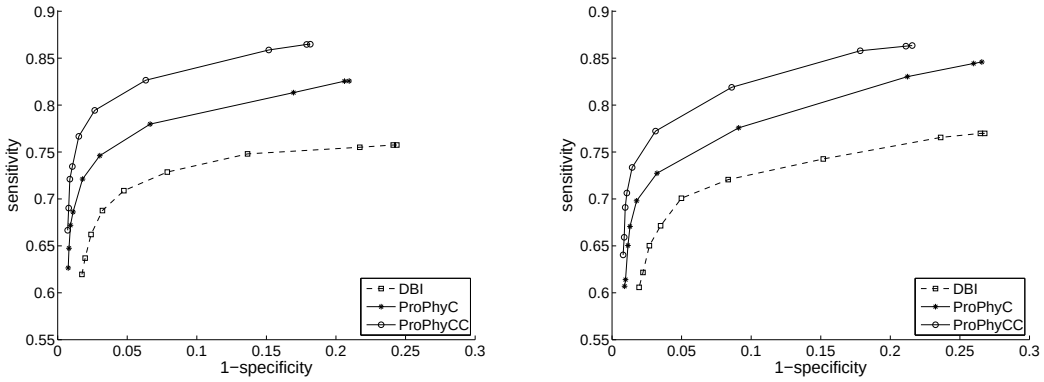
We start our inference and refinement procedures with gene-expression data. We first use *DBI* to infer networks for the leaf organisms, then run refinement algorithms on each set of networks with the two gene duplication and loss histories: the true history and the history reconstructed by Notung [38]. In the following we show results on one representative phylogenetic tree with 35 nodes on 7 levels, 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 neutral initialization model. 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 high rates of gene loss (with 20 duplications and 23 losses).

The results are also shown in ROC curves, where different sensitivity and specificity settings are obtained in the same fashion as described in Sec. 5.2.4.

5.5 Experimental Results under the Extended Model

5.5.1 Absolute comparison, with true history

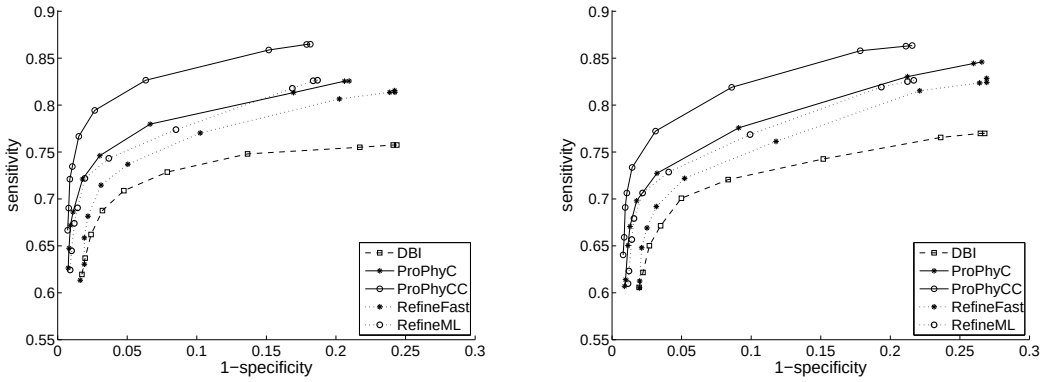
Figs. 5.9 shows the comparison of *ProPhyC*, *ProPhyCC* and the base inference algorithm *DBI*, with the true gene duplication and loss history. Given the size of the tree and the root network, the rates of gene duplication and loss are quite high, yet, as we can see from Fig. 5.9, the improvement gained by our refinement algorithms remains significant in both plots – almost as much as the improvement gained with the basic network evolutionary model shown in Fig. 5.3. *ProPhyCC* further dominates *ProPhyC* in both sensitivity and specificity, thanks to the appropriate use of the position-specific confidence values. We obtain similar improvements with (i) other trees; (ii) other evolutionary rates; and (iii) other base methods.



(a) *Results with lower gene duplication and loss rates* (b) *Results with higher gene duplication and loss rates*

Figure 5.9: Results of refinement algorithms with extended network evolutionary model, comparison of *ProPhyC* and *ProPhyCC* with base inference algorithm *DBI*, with true gene duplication and loss history

5.5.2 Relative comparison, with true history



(a) *Results with lower gene duplication and loss rates* (b) *Results with higher gene duplication and loss rates*

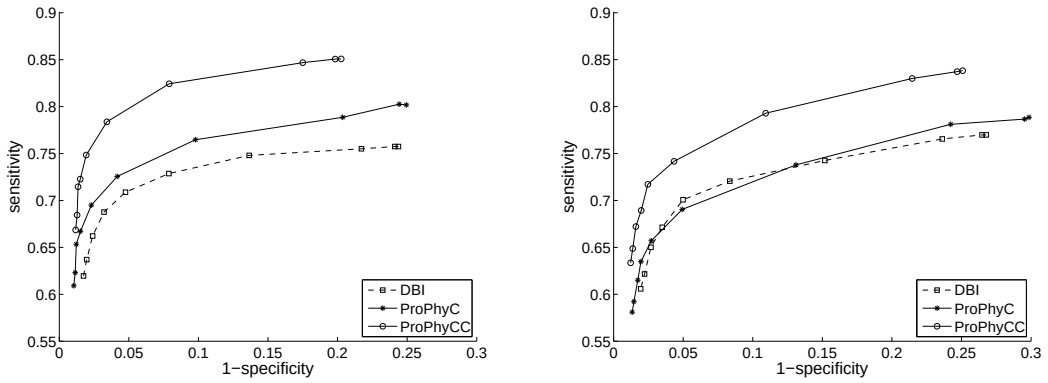
Figure 5.10: Results of refinement algorithms with extended network evolutionary model, comparison of *ProPhyC* and *ProPhyCC* with RefineFast and RefineML, with true gene duplication and loss history

Fig. 5.10 shows us results of the same experiments as in Fig. 5.9, but with the performance of *RefineFast* and *RefineML*. We see that although *RefineFast* and *RefineML* still clearly improve *DBI*, the improvement is not as big as that in Fig. 5.4 with the basic evolutionary model. This is because the gene duplication and loss events during evolution give rise to a large overall gene population, yet many of them exist only in a few leaf networks, so that there is not much phylogenetic information to be used to correct the prediction of the connections for these genes. *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 comparison, with inferred history

Here we use Notung to reconstruct the gene duplication and loss history without any orthology input. In these experiments, with reliable gene tree input, Notung correctly predicts gene duplication events (modulo changes in the networks), but usually misses the gene loss events when they happen to leaf

species (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.



(a) *Results with lower gene duplication and loss rates* (b) *Results with higher gene duplication and loss rates*

Figure 5.11: Results of refinement algorithms with extended network evolutionary model, comparison of *ProPhyC* and *ProPhyCC* with *DBI*, with inferred gene duplication and loss history by Notung

Fig. 5.11 shows the results of *ProPhyC* and *ProPhyCC* with Notung-reconstructed gene contents for the ancestral networks. We see that in Fig. 5.11(a), 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. 5.11(b), *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); secondly 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 Discussion and Conclusion

In this chapter, we propose a probabilistic phylogenetic model designed to improve the regulatory network inference for a family of organisms by using the phylogenetic relationships among these organisms. This model and its refinement algorithms *ProPhyC* and *ProPhyCC* can easily be adapted to work with different network evolutionary models.

We conduct experiments on both simulated and biological data to test the performance of the refinement algorithms, and compare them with our previous refinement algorithms *RefineFast* and *RefineML*. With both the basic and extended network evolutionary models, the corresponding versions of *ProPhyC* and *ProPhyCC* outperform those of *RefineFast* and *RefineML*, and all four refinement algorithms improve the base inference algorithm *DBI*. The improvement of *ProPhyC* and *ProPhyCC* over *RefineFast* and *RefineML* is more significant with the extended network evolutionary model, where the performance of *RefineFast* and *RefineML* is affected by the decrease of the phylogenetic information for each ortholog, yet *ProPhyC* and *ProPhyCC* are hardly influenced. Our probabilistic phylogenetic model is thus quite robust against changes in these network evolutionary models.

These refinement algorithms not only output the refined networks, but also the ancestral networks which can help in analyzing the evolution of regulatory networks.

Our probabilistic phylogenetic model can easily be extended into a probabilistic graphical model to incorporate the evolution of both the regulatory networks and the binding sites.

Chapter 6

Tree Transfer Learning Algorithm

In Chapter 4 and 5 we presented our refinement algorithms *RefineFast*, *RefineML*, *ProPhyC*, and *ProPhyCC*, all of which attempt to refine the regulatory networks for a family or organisms using the phylogenetic relationships; the latter two further improve the performance of the former two. All four algorithms work under the scenario where the input is the noisy regulatory networks of the family of organisms, and the output is the refined version of these networks.

The positive results from extensive tests on these models and algorithms confirm the usefulness of phylogenetic information in obtaining better inference of regulatory networks. Clearly, however, there is a limit to the improvement brought by the phylogenetic information. Does *ProPhyC* come close to this limit? With the same input and output setting, we can not find or design an algorithm which outperform *ProPhyC*, so we try another scenario, where the input data is the gene-expression data for the family of organisms instead of the noisy networks. Under this scenario, we devise an entirely different approach to the incorporation of phylogenetic information, *Tree Transfer Learning (TTL)*. *TTL* combines the concept of transfer learning [49, 66] with a phylogenetic tree, using the basic network evolutionary model. 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. Throughout our experiments, *ProPhyC* dominates *TTL*, although the two often return comparable results. That such 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, perhaps most, of the phylogenetic information.

6.1 The Tree Transfer Learning (*TTL*) Algorithm

Our *TTL* approach is illustrated in Fig. 6.1. It infers the regulatory networks for a family of organisms directly from gene-expression data and all in one step. This algorithm is inspired by the transfer learning algorithms in machine learning. Transfer learning is to learn multiple (related) tasks simultaneously while applying the relationships among the tasks. In our case, the multiple tasks are the inference of regulatory networks for the organisms in the family, and the relationships among the tasks are the phylogenetic relationships among the organisms.

Define a configuration $G = \{G_1, G_2, ..., G_{n_l}\}$ 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_l, the gene-expression data and the network structure for the ith leaf as D_i and G_i respectively, S_{data} is the sum

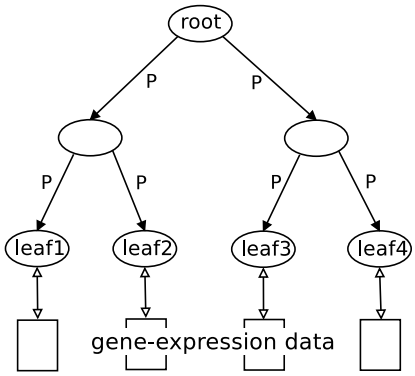


Figure 6.1: Illustration of the *TTL* approach

of the Bayesian information criterion (BIC) score over all leaves:

$$S_{data} = \sum_{i=1}^{n_l} \log Pr(D_i|G_i, \hat{\Theta}_{G_i}) - k_p \#G_i \log N_i$$

where $\hat{\Theta}_{G_i}$ is the ML estimate of parameters for G_i, $\#G_i$ is the number of free parameters of G_i, N_i is the number of samples in D_i, and k_p is the penalty coefficient for network structure complexity.

Denote the adjacency matrices of the nodes in the tree as $A_1, A_2, \ldots, A_{n_t}$, the number of genes in a network as n, and the edges of the tree as $e_1, e_2, \ldots, e_{n_e}$ where n_e is the number of edges in the tree; then S_{tree} is calculated as follows:

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

where A_p and A_c 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 configurations, we search in the space of possible structures of the root network. For each root structure, we generate n_c 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.

6.2 Comparison of *ProPhyC*, *ProPhyCC*, and *TTL*

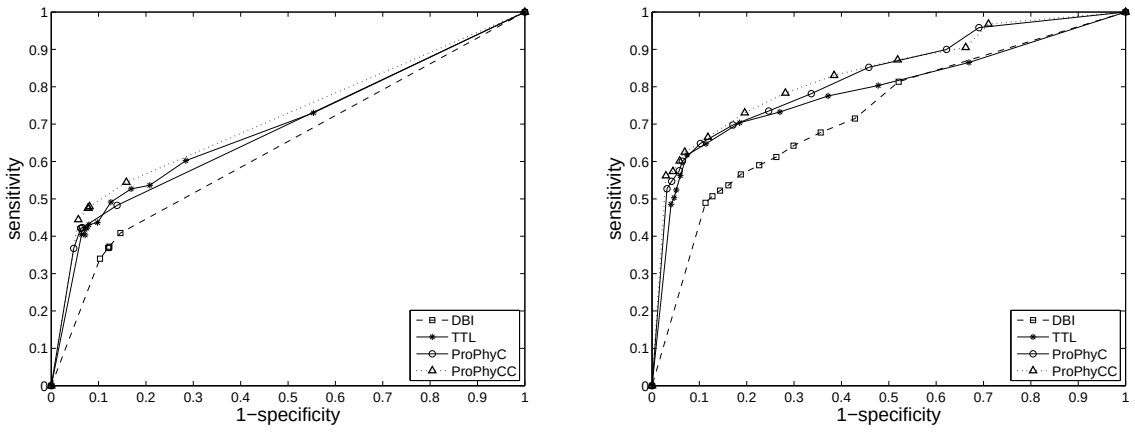
Here we show the comparison of *ProPhyC*, *ProPhyCC* and *TTL* based on the basic network evolutionary model. In these experiments, we use a phylogenetic tree of 37 nodes on 6 levels, and

Algorithm 1 the *TTL* algorithm

for each gene g in the network **do**
  $S^g_{max} \Leftarrow -\infty$;
  for each set of incoming edges for g in the root network G^g_{root} **do**
    Generate n_c configurations of incoming edges for g in the leaf networks according to the network evolutionary model;
    for each configuration $G_g = \{G^g_1, G^g_2, ..., G^g_{n_l}\}$ **do**
      Calculate the current score S^g_{ttl};
      if $S^g_{ttl} > S^g_{max}$ **then**
        $S^g_{max} \Leftarrow S^g_{ttl}$;
        $G^*_g \Leftarrow G_g$;
      end if
    end for
  end for
end for
Assemble the G^*_g for all g to get G^*.

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, and we use a small network size of 7 genes. Experiments are conducted with a wide range of parameters for each algorithm to test their overall performance and robustness to parameter settings.

The two plots in Fig. 6.2 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 5 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. 6.2 the curves are plotted with full scale from 0 to 1 at both axes.



(a) *On gene-expression datasets with 5 time points* (b) *On gene-expression datasets with 20 time points*

Figure 6.2: Comparison of *ProPhyC*, *ProPhyCC* and *TTL*

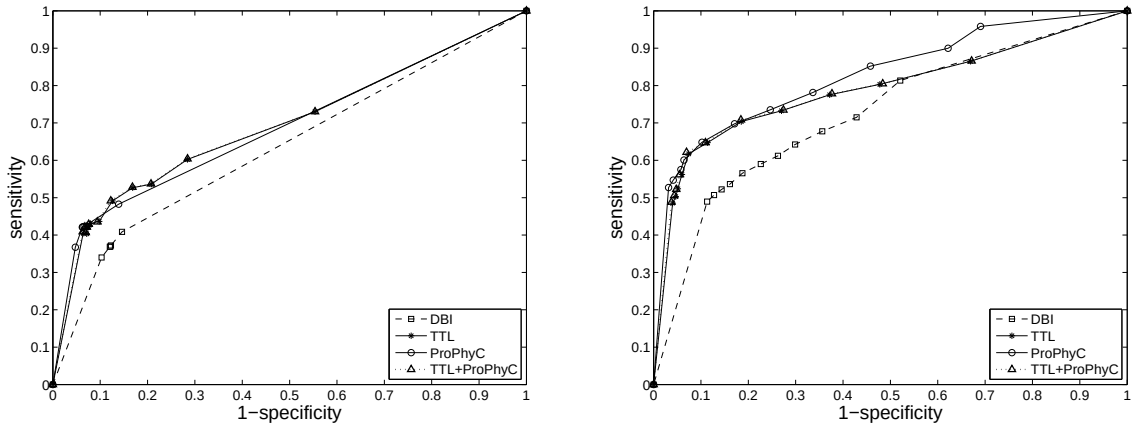
In Fig. 6.2(a), comparing *ProPhyC* and *TTL*, we can see that the curves for *ProPhyC* and *TTL* are almost coincident, while in Fig. 6.2(b) the curve for *ProPhyC* slightly dominates that of *TTL*. The two plots together show that the transfer learning approach does not outperform *ProPhyC*. The observation that *TTL* performs better in Fig. 6.2(a) than in Fig. 6.2(b) relative to *ProPhyC* shows its advantage on small gene-expression datasets. This is because, with smaller datasets, the base inference algorithm (which infers a single network from the corresponding gene-expression dataset)

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 datasets 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 datasets. The running time of *TTL* is n_c times of that of *ProPhyC* including the time to run *DBI*, where n_c is the number of configurations generated for each root structure. This value increases with the scale of the tree or networks, so *TTL* is much slower than *ProPhyC*.

Although *ProPhyC* is affected by the poor performance of *DBI* on small datasets, *ProPhyCC* benefits from the confidence values of the prediction of the base inference algorithm, which gives us a distribution of the leaf networks instead of a single configuration, and leads to better performance even with small datasets (see Fig. 6.2(a)).

We also test whether combining *ProPhyC* and *TTL* will allow *ProPhyC* to benefit from the advantage of *TTL* with small datasets, so that this combined method 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*: firstly *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*.

Fig. 6.3 shows the results of this combined algorithm on the same datasets as in Fig. 6.2. The ROC curves are averaged over different parameter settings of *ProPhyC* applied onto different outcomes of *TTL*. In both plots of Fig. 6.3, the curves of the combined algorithm are almost identical to but very slightly above those of *TTL*, so they do not improve over the curves of *ProPhyC*. Thus, combining the two algorithms does not help improve the performance of *ProPhyC*. Since the output networks already fit the phylogenetic relationships well according to the mechanism of *TTL*, *ProPhyC* does not alter the networks much in such a case. Therefore, we claim that, when the input information is of low-quality or limited, there is not much space to improve over *ProPhyC*, since it has already made good use of the available information.



(a) *On gene-expression datasets with 5 time points* (b) *On gene-expression datasets with 20 time points*

Figure 6.3: Comparison of *ProPhyC* and *TTL*, and the combination of *ProPhyC* and *TTL*

We have presented most of our results with ROC curves, and the area under curves (AUC) is a standard measure for the accuracy of network inference. The plots in Figs. 6.2 and 6.3 are shown in full scale to show the AUC. One observation from these figures is that, in some plots, it is not obvious that the points marked on the dominating curve are better than the points corresponding to the same penalty coefficients on the curve below. For example, in Fig. 6.2(a), although the curve for

TTL has larger AUC than that of *DBI*, not every point on the former curve has both better sensitivity and specificity than its corresponding point on the latter curve. However, the performance of *TTL* is still better than *DBI* according to the AUC measure, since for any point on the *DBI* curve, there always exist some points on the *TTL* curve to its upper left. Similar patterns can be observed in some of the previous plots from Fig. 5.3 to Fig. 5.11, with the curves for *DBI* and *ProPhyC* in Fig. 5.11(a) as an example.

6.3 Discussion and Conclusion

Tree Transfer Learning (*TTL*) is an approach based on inductive transfer learning, which applies the phylogenetic information as it infers the leaf networks. Devised in a very different framework, *TTL* is compared with *ProPhyC* and *ProPhyCC* over a range of parameters. Under various conditions, *TTL* approaches the performance of *ProPhyC* but does not outperform it, which again verifies the strength of *ProPhyC* in integrating the phylogenetic information in its probabilistic graphical model. *ProPhyCC* performs better than the other two, which shows 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.

Chapter 7

Conclusion and Discussion

In this dissertation, I presented the main algorithms we developed for refining regulatory networks for a family of organisms by the phylogenetic relationships among these organisms. These algorithms require an evolutionary model for regulatory networks, and I proposed two such models.

The two topics, the computational inference and the evolutionary analysis of regulatory networks are related to each other, in the sense that advances in one can assist research in the other. On the one hand, improvement in the inference can provide more reliable data for the study of evolution; on the other hand, progress of studies on network evolution will allow us to better model the phylogenetic relationships between regulatory networks of multiple organisms. Through a refinement procedure (like *RefineFast*, *RefineML*, *ProPhyC* and *ProPhyCC*) or a transfer learning method (like *TTL*) we can then get better inference of the networks. In this manner these two lines of research assist each other especially when there is a third step in-between, which is to obtain more benchmark data via wet-lab experiments. These experiments can again be guided by the output from the refinement algorithms: during the generation of new data in biology, benchmark experiments are often guided by results from computational predictions, especially from comparative studies, so as to save time and cost.

In previous chapters we have described our refinement algorithms *RefineFast*, *RefineML*, *ProPhyC* and *ProPhyCC*. These algorithms aim to use phylogenetic information to refine the (noisy) networks of a family of organisms, and their effectiveness has been confirmed by a large collection of experiments. We also designed a tree transfer learning (*TTL*) algorithm which takes the gene-expression data of the organisms as input, and infers their regulatory networks all at once while taking into account their phylogenetic relationships. *ProPhyC* and *ProPhyCC*, which use a probabilistic phylogenetic model, are shown to have the best performance among all.

In all the algorithms mentioned above, we use simple network evolutionary models which are the basic model and the extended model introduced in Chapter 3. Simple models often turn out to be safe and robust in computation, and when we want to use a more complex model to include more factors, we often need to seek a tradeoff between model complexity and exactness, therefore it is prudent to start with simple models. On the other hand, we hope that as knowledge of network evolution advances, we will be able to formulate more realistic models which are also widely accepted. We expect that with these improved models our refinement framework will work better. For example, it would be interesting to take into account the effect of external environmental factors on the evolution of regulatory networks.

In our network evolutionary models we represent the regulatory networks by their binary adjacency matrices. We know that in reality the regulatory connections are not binary – they exist in various strengths. In fact we have worked out versions of our refinement algorithms where the regulatory connections are represented by continuous values, but we could not evaluate their performance since there is no standard measurement to assess the quality of quantitative networks, so they

are not presented in this thesis. As more data becomes available regulatory networks will be better quantified.

Furthermore, during the calculation of all the five algorithms, to simplify the computation we assumed that the entries in the adjacency matrices are independent of each other, so that when reconstructing ancestral networks (in *RefineFast* and *RefineML*) or inferring the unknown "true" networks (in *ProPhyC* and *ProPhyCC*), or calculating the S_{tree} score (in *TTL*), we could deal with only one entry in all the networks at one time, instead of using the whole network for each organism. A similar independency assumption is widely used for genome and protein sequences in various contexts, such as the ancestral reconstruction for protein sequences [34], or phylogenetic tree reconstruction [67]. Both assumptions are false in biology, that is, the interactions in regulatory networks or nucleotides clearly do not evolve independently. In the case of regulatory networks, it can be useful to consider the dependency between some interactions, such as the interactions of genes from the same gene family. To solve the increased complexity caused to our refinement algorithms, one may consider using the *variational inference* technique from machine learning, which provides an efficient approximation when calculating the global likelihood for a set of variables with complex dependencies [68, 69].

Bibliography

[1] de Jong H: **Modeling and simulation of genetic regulatory systems: a literature review**. *J. Comput. Bio.* 2002, **9**:67–103.

[2] Albert R: **Scale-free networks in cell biology**. *Journal of Cell Science* 2005, **118**(21):4947–4957.

[3] Barabási AL, Oltvai ZN: **Network biology: understanding the cell's functional organization**. *Nature Reviews Genetics* 2004, **5**:101 – 113.

[4] Babu MM, Luscombe NM, Aravind L, Gerstein M, Teichmann SA: **Structure and evolution of transcriptional regulatory networks**. *Curr. Opinion in Struct. Bio.* 2004, **14**(3):283–291.

[5] Bhardwaj N, Kim PM, Gerstein MB: **Rewiring of Transcriptional Regulatory Networks: Hierarchy, Rather Than Connectivity, Better Reflects the Importance of Regulators**. *Science Signaling* 2010, **3**(146):ra79.

[6] Wagner GP, Pavlicev M, Cheverud JM: **The road to modularity**. *Nature Reviews Genetics* 2007, **302B**(8):921–931.

[7] Alon U: **Network motifs: theory and experimental approaches**. *Nature Reviews Genetics* 2007, **8**:450–461.

[8] Lee TI, Rinaldi NJ, Robert F, *et al*: **Transcriptional regulatory networks in *Saccharomyces cerevisiae***. *Science* 2002, **298**(5594):799 – 804.

[9] Luscombe NM, Babu MM, Yu H, Snyder M, Teichmann SA, Gerstein M: **Genomic analysis of regulatory network dynamics reveals large topological changes**. *Nature* 2004, **431**:308 – 312.

[10] Babu MM: **Structure, evolution and dynamics of transcriptional regulatory networks**. *Biochem Soc Trans.* 2010, **38**(5):1155–78.

[11] Tanay A, Regev A, Shamir R: **Conservation and evolvability in regulatory networks: The evolution of ribosomal regulation in yeast**. *Proc. Nat'l Acad. Sci., USA* 2005, **102**(20):7203–7208.

[12] Schlitt T, Brazma A: **Current approaches to gene regulatory network modelling**. *BMC Bioinformatics* 2007, **8**(Suppl 6):S9.

[13] Akutsu T, Miyano S, Kuhara S: **Identification of genetic networks from a small number of gene expression patterns under the Boolean network model**. In *Proc. 4th Pacific Symp. on Biocomputing (PSB'99), Volume 4*, World Scientific 1999:17–28.

[14] Friedman N, Linial M, Nachman I, Pe'er D: **Using Bayesian Networks to Analyze Expression Data**. *J. Comput. Bio.* 2000, **7**(3-4):601–620.

[15] Kim SY, Imoto S, Miyano S: **Inferring gene networks from time series microarray data using dynamic Bayesian networks**. *Briefings in Bioinformatics* 2003, **4**(3):228–235.

[16] Chen T, He HL, Church GM: **Modeling gene expression with differential equations**. In *Proc. 4th Pacific Symp. on Biocomputing (PSB'99)*, World Scientific 1999:29–40.

[17] Wang R, Wang Y, Zhang X, Chen L: **Inferring Transcriptional Regulatory Networks from High-throughput Data**. *Bioinformatics* 2007, **23**(22):3056–3064.

[18] Cooke EJ, Savage RS, Wild DL: **Computational approaches to the integration of gene expression, ChIP-chip and sequence data in the inference of gene regulatory networks**. *Seminars in Cell & Developmental Biology* 2009, **20**(7):863–868.

[19] Prill RJ, Marbach D, Saez-Rodriguez J, Sorger PK, Alexopoulos LG, Xue X, Clarke ND, Altan-Bonnet G, Stolovitzky G: **Towards a Rigorous Assessment of Systems Biology Models: The DREAM3 Challenges**. *PLoS ONE* 2010, **5**:e9202.

[20] Marbach D, Prill RJ, Schaffter T, Mattiussi C, Floreano D, Stolovitzky G: **Revealing strengths and weaknesses of methods for gene network inference**. *Proc. Nat'l Acad. Sci., USA* 2010, **107**(14):6286–6291.

[21] Bar-Joseph Z: **Analyzing time series gene expression data**. *Bioinformatics* 2004, **20**(16):2493–2503.

[22] Xu R, Hu X, Wunsch DC: **Inference of genetic regulatory networks from time series gene expression data**. In *Proc. IEEE Int'l Joint Conf. on Neural Networks, Volume 2*, IEEE Press, Piscataway, NJ 2004:1215–1220.

[23] Zhao W, Serpedin E, Dougherty ER: **Inferring gene regulatory networks from time series data using the minimum length description principle**. *Bioinformatics* 2006, **22**(17):2129–2135.

[24] Kim HD, Shay T, OShea EK, Regev A: **Transcriptional Regulatory Circuits: Predicting Numbers from Alphabets**. *Science* 2009, **325**(5939):429–432.

[25] Berg J, Lassig M, Wagner A: **Structure and evolution of protein interaction networks: a statistical model for link dynamics and gene duplications**. *BMC Evolutionary Biology* 2004, **4**:51.

[26] Mithani A, Preston GM, Hein J: **A Bayesian Approach to the Evolution of Metabolic Networks on a Phylogeny**. *PLoS Comput Biol* 2010, **6**(8):e1000868.

[27] Yamada T, Bork P: **Evolution of biomolecular networks lessons from metabolic and protein interactions**. *Nature Reviews Molecular Cell Biology* 2009, **10**:791–803.

[28] Crombach A, Hogeweg P: **Evolution of Evolvability in Gene Regulatory Networks**. *PLoS Comput Biol* 2008, **4**(7):e1000112.

[29] Stark A, Kheradpour P, Roy S, Kellis M: **Reliable prediction of regulator targets using 12 *Drosophila* genomes.** *Genome Research* 2007, **17**:1919–1931.

[30] Babu MM, Teichmann SA, Aravind L: **Evolutionary Dynamics of Prokaryotic Transcriptional Regulatory Networks**. *J. Mol. Bio.* 2006, **358**(2):614–633.

[31] Roth C, Rastogi S, Arvestad L, Dittmar K, Light S, Ekman D, Liberles DA: **Evolution after gene duplication: models, mechanisms, sequences, systems, and organisms**. *Journal of Experimental Zoology Part B: Molecular and Developmental Evolution* 2007, **308B**:58 – 73.

[32] Teichmann SA, Babu MM: **Gene regulatory network growth by duplication**. *Nature Genetics* 2004, **36**(5):492–496.

[33] Babu MM, Teichmann SA: **Evolution of transcription factors and the gene regulatory network in Escherichia coli**. *Nucleic Acids Research* 2003, **31**(4):1234–1244.

[34] Pupko T, Pe'er I, Shamir R, Graur D: **A Fast Algorithm for Joint Reconstruction of Ancestral Amino Acid Sequences**. *Mol. Bio. Evol.* 2000, **17**(6):890–896.

[35] Bourque G, Sankoff D: **Improving gene network inference by comparing expression time-series across species, developmental stages or tissues**. *J. Bioinform. Comput. Biol* 2004, **2**(4):765–783.

[36] Murphy KP: **The Bayes Net Toolbox for MATLAB**. *Computing Sci. and Statistics* 2001, **33**:331–351.

[37] Arvestad L, Berglund AC, Lagergren J, Sennblad B: **Gene tree reconstruction and orthology analysis based on an integrated model for duplications and sequence evolution**. In *Proc. 8th Ann. Int'l Conf. Comput. Mol. Bio. (RECOMB'04)*, New York, NY, USA: ACM 2004:326 – 335.

[38] Durand D, Halldórsson BV, Vernot B: **A Hybrid Micro–Macroevolutionary Approach to Gene Tree Reconstruction**. *J. Comput. Bio.* 2006, **13**(2):320–335.

[39] Page RDM, Charleston MA: **From Gene to Organismal Phylogeny: Reconciled Trees and the Gene Tree/Species Tree Problem**. *Molecular Phylogenetics and Evolution* 1997, **7**(2):231–240.

[40] Conant RC: **Extended dependency analysis of large systems**. *Int'l J. General Systems* 1988, **14**(2):97–141.

[41] Friedman N, Murphy KP, Russell S: **Learning the structure of dynamic probabilistic networks**. In *Proc. 14th Conf. on Uncertainty in Art. Intell. UAI'98* 1998:139–147.

[42] Liang S, Fuhrman S, Somogyi R: **REVEAL, a general reverse engineering algorithm for inference of genetic network architectures**. In *Proc. 3rd Pacific Symp. on Biocomputing (PSB'98), Volume 3*, World Scientific 1998:18–29.

[43] Zmasek CM, Eddy SR: **A simple algorithm to infer gene duplication and speciation events on a gene tree**. *Bioinformatics* 2001, **17**(9):821–828.

[44] Evangelisti AM, Wagner A: **Molecular evolution in the yeast transcriptional regulation network**. *Journal of Experimental Zoology Part B: Molecular and Developmental Evolution* 2004, **302B**(4):392–411.

[45] Price MN, Dehal PS, Arkin AP: **Orthologous Transcription Factors in Bacteria Have Different Functions and Regulate Different Genes**. *PLoS Comput Biol* 2007, **3**(9):e175.

[46] Bhan A, Galas DJ, Dewey TG: **A duplication growth model of gene expression networks**. *Bioinformatics* 2002, **18**(11):1486 – 1493.

[47] Przytycka TM, Singh M, Slonim DK: **Toward the dynamic interactome: it's about time**. *Briefings in Bioinformatics* 2010, **11**:15–29.

[48] Yu H, Luscombe NM, Lu HX, Zhu X, Xia Y, Han JDJ, Bertin N, Chung S, Vidal M, Gerstein M: **Annotation Transfer Between Genomes: Protein–Protein Interologs and Protein–DNA Regulogs**. *Genome Research* 2004, **14**:1107–1118.

[49] Caruana R: **Multitask Learning**. *Machine Learning* 1997, **28**:41 – 75.

[50] Niculescu-mizil A, Caruana R: **Inductive transfer for Bayesian network structure learning**. In *Proceedings of the 11th international conference on AI and statistics (AISTATS 07)* 2007.

[51] Pan SJ, Yang Q: **A Survey on Transfer Learning**. *IEEE Transactions on Knowledge and Data Engineering* 2010, **22**:1345–1359.

[52] Zhang X, Zaheri M, Moret BME: **Using Phylogenetic Relationships to Improve the Inference of Transcriptional Regulatory Networks**. In *Proc. 1st Int'l Conf. on BioMedical Engineering and Informatics. BMEI'08* 2008:186–193.

[53] Zhang X, Moret BME: **Boosting the Performance of Inference Algorithms for Transcriptional Regulatory Networks Using a Phylogenetic Approach**. In *Proc. 8th Int'l Workshop Algs. in Bioinformatics (WABI'08), Volume 5251 of* Lecture Notes in Computer Science, Springer 2008:245 – 258.

[54] Zhang X, Moret BME: **Improving Inference of Transcriptional Regulatory Networks Based on Network Evolutionary Models**. In *Proc. 9th Int'l Workshop Algs. in Bioinformatics (WABI'09), Volume 5724 of* Lecture Notes in Computer Science, Springer 2009:412–425.

[55] Zhang X, Moret BME: **Refining transcriptional regulatory networks using network evolutionary models and gene histories**. *Algorithms for Molecular Biology* 2010, **5**:1.

[56] Murphy KP, Mian S: **Modelling gene expression data using dynamic Bayesian networks**. Tech. rep., University of California, Berkeley 1999. [`www.csail.mit.edu/~murphyk/Papers/ismb99.ps.gz`].

[57] Heckerman D: *Learning in graphical models*, Cambridge, MA, USA: MIT Press 1999 chap. A tutorial on learning with Bayesian networks, :301–354.

[58] Hillis DM: **Approaches for assessing phylogenetic accuracy**. *Syst. Bio.* 1995, **44**:3–16.

[59] Moret BME, Warnow T: **Reconstructing optimal phylogenetic trees: A challenge in experimental algorithmics**. In *Experimental Algorithmics, Volume 2547 of* Lecture Notes in Computer Science. Edited by Fleischer R, Moret B, Schmidt E, Springer Verlag 2002:163–180.

[60] Yu J, Smith VA, Wang PP, Hartemink AJ, Jarvis ED: **Advances to Bayesian network inference for generating causal networks from observational biological data**. *Bioinformatics* 2004, **20**(18):3594–3603.

[61] Kanehisa M, Goto S, Hattori M, Aoki-Kinoshita KF, Itoh M, Kawashima S, Katayama T, Araki M, Hirakawa M: **From genomics to chemical genomics: new developments in KEGG**. *Nucleic Acids Research* 2006, **34**:D354–D357.

[62] Zhang X, Moret BME: **ProPhyC: A Probabilistic Phylogenetic Model for Refining Regulatory Networks**. In *Bioinformatics Research and Applications, Volume 6674 of* Lecture Notes in Computer Science. Edited by Chen J, Wang J, Zelikovsky A, Springer Berlin / Heidelberg 2011:344–357.

[63] Harbison CT, Gordon DB, Lee TI, Rinaldi NJ, Macisaac KD, Danford TW, Hannett NM, Tagne JB, Reynolds DB, Yoo J, Jennings EG, Zeitlinger J, Pokholok DK, Kellis M: **Transcriptional regulatory code of a eukaryotic genome**. *Nature* 2004, **431**:99–104.

[64] Kim J, He X, Sinha S: **Evolution of Regulatory Sequences in 12 *Drosophila* Species**. *PLoS Genet* 2009, **5**:e1000330.

[65] Tweedie S, Ashburner M, Falls K, Leyland P, McQuilton P, Marygold S, Millburn G, Osumi-Sutherland D, Schroeder A, Seal R, Zhang H, The FlyBase Consortium: **FlyBase: enhancing Drosophila Gene Ontology annotations**. *Nucleic Acids Research* 2009, **37**:D555–D559.

[66] Baxter J: **A Bayesian/Information Theoretic Model of Learning to Learn via Multiple Task Sampling**. *Machine Learning* 1997, **28**:7 – 39.

[67] Felsenstein J: **Evolutionary trees from DNA sequences: A maximum likelihood approach**. *Journal of Molecular Evolution* 1981, **17**:368–376.

[68] Jordan MI, Ghahramani Z, Jaakkola TS, Saul LK: **An Introduction to Variational Methods for Graphical Models**. *Machine Learning* 1999, **37**:183–233.

[69] Wainwright MJ, Jordan MI: **Graphical Models, Exponential Families, and Variational Inference**. *Found. Trends Mach. Learn.* 2008, **1**:1–305.

CURRICULUM VITAE

Xiuwei Zhang

Personal Information

Gender: female

Birth Date: Oct. 05, 1981

Contact Information

EPFL IC IIF LCBB
INJ211 Station 14
CH-1015, Lausanne

Voice: (+41)21 69 38183
E-mail: xiuwei.zhang@epfl.ch

Education

Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland

Ph.D. Candidate, Computer science, started Aug. 2006

- Advisor: Prof. Bernard M.E. Moret

Tsinghua University, Beijing, China

M.S., Computer science, Sep. 2003 – Jul. 2006

- Advisor: Prof. Zhidong Deng

Dalian Jiaotong University, Dalian, China

B.S., Computer science, Sep. 1999 – Jul. 2003

Research Interests

Algorithms and models in computational biology. In particular, the inference of transcriptional regulatory networks, and models and analysis of their evolution and dynamics.

Selected Publications

Xiuwei Zhang and Bernard M.E. Moret. ProPhyC: a probabilistic phylogenetic model for refining regulatory networks. In *Proc. 7th Int'l Symp. Bioinformatics Research & Appls (ISBRA'11)*, volume of *Lecture Notes in Computer Science*, 6674: 344-357, Springer, 2011.

Xiuwei Zhang, Martin Kupiec, Uri Gophna and Tamir Tuller. Analysis of Co-evolving Gene Families Using Mutually Exclusive Orthologous Modules. *Genome Biology and Evolution*, 2011.

Xiuwei Zhang and Bernard M.E. Moret. Refining transcriptional regulatory networks using network evolutionary models and gene histories. *BMC Algorithms for Molecular Biology*, 5(1):1, 2010.

Xiuwei Zhang and Bernard M.E. Moret. Improving inference of transcriptional regulatory networks based on network evolutionary models. In *Proc. 9th Workshop on Algs. in Bioinformatics (WABI'09)*, volume of *Lecture Notes in Computer Science*, 5724: 415–428, Springer, 2009.

Xiuwei Zhang and Bernard M.E. Moret. Boosting the performance of inference algorithms for transcriptional regulatory networks using a phylogenetic approach. In *Proc. 8th Workshop on Algs. in Bioinformatics (WABI'08)*, *Lecture Notes in Computer Science*, 5251: 245–258. Springer, 2008.

Xiuwei Zhang, Maryam Zaheri, Bernard M.E. Moret. Using phylogenetic relationships to improve the inference of transcriptional regulatory networks. In *Proc. 1st*

International Conference on BioMedical Engineering and Informatics (BMEI'08), 1: 186–193, 2008.

Xiuwei Zhang, Zhidong Deng and Dandan Song. A new neural network approach for RNA secondary structure prediction, *Journal of Tsinghua University (Science and Technology)*, 46(10): 1793-1796, 2006.

Academic Experience

Ecole Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland,

Teaching Assistant Sep. 2006 – 2010

Have worked as teaching assistant for the following courses:

- Advanced Algorithms, given by Prof. Bernard M.E. Moret.
- Computational molecular biology, given by Prof. Bernard M.E. Moret.
- Topics in Bioinformatics I, given by Prof. Bernard M.E. Moret, Prof. Philipp Bucher, and Prof. Felix Naef.

Journal Referee PLoS ONE
BMC Proceedings

Conference Referee RECOMB, ISMB, etc.

Conference presentations

May 2011 Oral presentation at the 7th Int'l Symp. Bioinformatics Research & Appls (ISBRA'11), Changsha (China), for the paper "ProPhyC: a Probabilistic Phylogenetic Model for Refining Regulatory Networks".

Feb. 2011 Oral presentation at the SIB (Swiss Institute of Bioinformatics) Days, Biel (Switzerland).

Nov. 2010 Oral presentation at the 7th Annual RECOMB Satellite on Regulatory Genomics and the 6th Annual RECOMB Satellite on Systems Biology, New York. Title: "Phylogenetic modeling uncovers finer structure of regulatory networks".

Feb. 2010 Poster presentation at the SIB (Swiss Institute of Bioinformatics) Days, Montreux (Switzerland).

Dec. 2009 Poster presentation at the 6th Annual RECOMB Satellite on Regulatory Genomics and the 5th Annual RECOMB Satellite on Systems Biology, Boston.

Sep. 2009 Oral presentation at the 9th Workshop on Algorithms in Bioinformatics WABI'09, Philadelphia, for the paper "Improving inference of transcriptional regulatory networks based on network evolutionary models".

Apr. 2009 Oral presentation at the 8th International Conf. on Information Processing in Cells and Tissues IPCAT'09, Ascona (Switzerland).

Jan. 2009 Poster presentation at the SIB (Swiss Institute of Bioinformatics) Days, Fribourg (Switzerland).

Nov. 2008 Poster presentation at the 5th Annual RECOMB Satellite on Regulatory Genomics, Boston.

Sep. 2008 Oral presentation at the 8th Workshop on Algorithms in Bioinformatics WABI'08, Karlsruhe (Germany), for the paper "Boosting the performance of inference

algorithms for transcriptional regulatory networks using a phylogenetic approach".

May 2008 Oral presentation at the 1st IEEE Conference on Biomedical Engineering and Informatics BMEI'08, Sanya (China), for the paper "Using phylogenetic relationships to improve the inference of transcriptional regulatory networks".