Computational Evolutionary Genomics: 
Phylogenomic Models 
Spanning Domains, Genes, Individuals, and Species

by

Yi-Chieh Wu

B.S.E.E., Rice University (2007) 
S.M., Massachusetts Institute of Technology (2009)

Submitted to the Department of Electrical Engineering and Computer Science 
in partial fulfillment of the requirements for the degree of 
Doctor of Philosophy in Electrical Engineering and Computer Science 
at the 
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2014

© Massachusetts Institute of Technology 2014. All rights reserved.
Computational Evolutionary Genomics:
Phylogenomic Models
Spanning Domains, Genes, Individuals, and Species
by
Yi-Chieh Wu

Submitted to the Department of Electrical Engineering and Computer Science
on January 31, 2014, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Computational techniques have long been applied to biological data to address a wide range of evolutionary questions. In phylogenetics, methods for reconstructing gene histories from sequence data have enabled researchers to better understand how evolution shapes gene content; for example, the identification of evolutionarily-related genes has allowed for the mapping of functions across species and the discovery of novel functions. Such predictions have become increasingly important over the last ten to fifteen years, as technology has reduced the cost of sequencing and increased processing power, leading to vast genomic datasets with little, if any, functional information. In turn, the growing availability of genomes has led to increased power for biological signal discovery and revealed insight into the core evolutionary forces that govern our existence.

However, to realize the full potential in genomic and evolutionary studies, we require accurate, efficient, and scalable methods that are widely applicable. In this thesis, I address this need by developing novel computational approaches for reconstructing gene evolutionary histories. In particular, I consider models for gene family evolution that take into account (1) nucleotide or amino acid substitution, (2) genetic drift (leading to deep coalescence), (3) gene duplication and loss, (4) horizontal gene transfer, and (5) domain rearrangement, and I present new phylogenetic algorithms for (1) eukaryotic gene tree reconstruction, (2) prokaryotic gene tree reconstruction, (3) gene tree-species tree reconciliation, and (4) sub-gene-level reconstruction. Through extensive benchmarking, I show that these methods dramatically improve reconstructions compared to state-of-the-art programs; in addition, they are efficient and require few modeling assumptions or parameters, making them applicable to a broad range of species and large datasets. As evidence, I apply these methods to clades of 12 Drosophila, 16 fungi, 15 primates, and 11 cyanobacteria, as well as to simulated phylogenies with up to 200 taxa, and demonstrate the large impact of accurate phylogenetic inference on downstream evolutionary analyses.

These results demonstrate the power of computational phylogenetics, and I believe that with the continued development and adoption of such methods, we can address fundamental biological questions with many important implications for future investigations of gene and genome evolution.

Thesis Supervisor: Manolis Kellis
Title: Professor of Computer Science
Acknowledgments

I express my sincerest thanks to the following, without whom this thesis would not have been possible:

To my advisor, Manolis Kellis, for his advice, support, and mentorship throughout my graduate career. I am so grateful and privileged to have met an advisor willing to take the risk of letting a graduate student with little biological background enter the computational biology field. His continuous encouragement and guidance have been invaluable. I also appreciate the enormous freedom granted in choosing projects and the opportunities to collaborate with various consortia.

To members of the Computational Biology Group, for the many discussions and insights, and for their friendship. I would particularly like to thank Matt Rasmussen and Mukul Bansal, who taught me much about evolution and phylogenetics and who contributed greatly to the work presented in this thesis. Many thanks also to the rest of the evolution office: Ran Libeskind-Hadas, Mike Lin, Irwin Jungreis, Rob Waterhouse, Olivier Jaillon, Max Wolf, and Eva Novoa.

To my collaborators, in particular, Javier Herrero, for discussions on phylogenetic pipelines. Thanks also to various members of the ENCODE, modENCODE, and mouseENCODE consortia and the Boston Evolutionary Genomics Supergroup.

To my thesis committee, Eric Alm, Daniel Hartl, and Constantinos Daskalakis, for discussions at various stages of this work.

To various MIT EECS professors and staff, for their advice, support, and guidance. In particular, I would like to acknowledge my academic advisor, Asuman Ozdaglar; my Masters advisor, John Wyatt; EECS Graduate Officers, Leslie Kolodziejski and Terry Orlando; and EECS Graduate Office staff, Janet Fischer and Alicia Duarte.

To Rice University professors Lydia Kavraki, Richard Baraniuk, Michael Orchard, Stephen Wong, and Steven Cox, for their advice, support, and mentorship, and for encouraging me to pursue research and graduate school.

To my friends, for the many conversations, diversions, and support.

Finally, to my parents, Rongsong and Manhung, and my sisters, Emmy and Diana, for their constant love, support, and encouragement.

This work was made possible due to financial support from the National Science Foundation, the Kambourides Fellowship, and the MIT/Whitehead/Broad Computational Genetics Training Program through the National Institutes of Health.
# Contents

1 Introduction  
  1.1 Thesis contributions .................................................. 11  
  1.2 Gene and genome evolution ......................................... 12  
    1.2.1 Mutation .......................................................... 13  
    1.2.2 Genetic drift ..................................................... 13  
    1.2.3 Gene duplication (and loss) ..................................... 13  
    1.2.4 Horizontal gene transfer ........................................ 14  
    1.2.5 Gene fusion and fission ......................................... 14  
    1.2.6 Other evolutionary events ....................................... 15  
  1.3 Phylogenetics and phylogenomics .................................. 15  
    1.3.1 Gene trees and species trees .................................. 16  
    1.3.2 Reconciliation .................................................... 17  
    1.3.3 Orthologs and paralogs ........................................... 17  
    1.3.4 Phylogenetic methods ............................................. 18  
    1.3.5 Phylogenomic methods ............................................ 19  

2 Statistically informed gene tree reconstruction with gene duplication and loss: TreeFix  
  2.1 Eukaryotic gene tree reconstruction ................................ 21  
    2.1.1 Existing work .................................................... 21  
    2.1.2 Contributions .................................................... 22  
  2.2 Gene tree landscape .................................................. 22  
    2.2.1 Statistical comparison of sequence support for multiple topologies .......... 23  
    2.2.2 Reconciliation cost as a heuristic for topological accuracy ............... 24  
    2.2.3 Tree search ...................................................... 25  
  2.3 The TreeFix algorithm ................................................ 25  
  2.4 Benchmarks ............................................................. 26  
    2.4.1 Simulated datasets ............................................... 26  
    2.4.2 Biological dataset of 16 fungal genomes .......................... 27  
    2.4.3 Alternative species tree topologies ................................ 28  
  2.5 Discussion ............................................................. 29  

3 Species-aware gene tree reconstruction with gene duplication, transfer, and loss: TreeFix-DTL  
  3.1 Prokaryotic gene tree reconstruction ............................... 30  
    3.1.1 Existing work .................................................... 30  
    3.1.2 Contributions .................................................... 31  
  3.2 The TreeFix-DTL algorithm .......................................... 31  
  3.3 Benchmarks ............................................................. 32  
    3.3.1 Simulated datasets ............................................... 32  
  3.4 Resolving gene tree-species tree conflict in 11 cyanobacteria genomes .......... 33

7
Chapter 1

Introduction

Evolution of genes and genomes is responsible for the immense biological diversity of our planet, ranging from the multitude of life forms, to the assortment of protein structural domains and gene functions, to the variation of traits between individuals within and across populations of species. However, despite its central role as the most fundamental property of life, the process of evolution remains poorly understood, and current models have typically been unable to span the diversity of scales at which evolution can act.

The field of phylogenetics provides rigorous tools for the study of evolution. At its most basic level, the goal of phylogenetics is to reconstruct the evolutionary history, or phylogeny, of a group of related taxonomic units, for example, of genes or species. Whereas historically, phylogenies relied on observable, typically morphological, traits and were computed by hand, with new technological advances, modern phylogenies use molecular data and increasingly sophisticated computational tools. With the explosion of molecular biology techniques, it is now commonplace to infer phylogenies of hundreds and sometimes thousands of sequences, thus requiring new computational models for their systematic reconstruction.

The field of phylogenomics deals specifically with the study of phylogenetics at the genome scale [58, 59] and is uniquely enabled by the ability to compare multiple complete genomes. Working at the genome scale enables us to study parallels in the effects and dynamics of evolutionary processes across genes in the same species, and across species for the same gene. This has led to a better understanding of how evolution shapes the genomes of many different species, including humans [73, 92, 95, 144, 166, 202, 213, 214, 280], flies [231], fungi [26], plants [122], yeast [130], and cyanobacteria [287]. Even more ambitious goals are on the horizon, as in the next few years, thanks to next-generation sequencing, we expect thousands of human genomes [245], tens of thousands of vertebrate genomes [77], and millions of bacterial genomes.

Despite rapid technological advances, however, we have yet to realize the full potential in
genomic and evolutionary studies. In particular, phylogenomic methods still suffer from a number of limitations, mostly stemming from statistical challenges of distinguishing biological signal from noise, algorithmic challenges of scaling to large datasets, and modeling challenges of generalizing across genomes and genomic features at multiple scales. This leads to inaccurate analyses and can disastrously affect evolutionary-based inferences.

1.1 Thesis contributions

The goal of this thesis is to develop accurate, efficient, scalable, and general computational methods for understanding gene and genome evolution. The thesis is organized as follows:

- I begin with a review of gene and genome evolution, introduce basic phylogenetic concepts, and discuss existing approaches to the problem of reconstructing and analyzing evolutionary histories (Chapter 1).

- I discuss gene tree reconstruction in the presence of duplications and losses, focusing on the problem of gene tree topological errors caused by lack of phylogenetic signal. I then present a statistical framework for gene tree error correction in eukaryotic species, implement this framework in the program TreeFix, and demonstrate the accuracy of this method through extensive sets of benchmarks from both real and simulated data (Chapter 2).

- I discuss how gene tree reconstruction is complicated by the addition of horizontal gene transfer, which plays an important role in prokaryotic gene family evolution. I then present a method, TreeFix-DTL, for gene tree error correction in prokaryotic species. In addition, I apply this method to a data set of eleven cyanobacteria genomes and show that the extent of horizontal gene transfer within this clade has likely been substantially overestimated (Chapter 3).

- I discuss gene family evolution within eukaryotic species separated by small evolutionary distances, in which it becomes important to consider not only evolutionary events such as duplication and loss, but also population-related phenomenon such as deep coalescence, which can result in incomplete lineage sorting (ILS). To simultaneously model the duplication-loss and coalescent history of a gene family, I introduce a new reconciliation structure, the labeled coalescent tree (LCT), then present a dynamic programming algorithm, DLCpar, for inferring a most parsimonious LCT. I show that the LCT representation enables, for the first time, an exhaustive and efficient search over the space of ILS-aware reconciliations and demonstrate how these properties allow DLCpar to achieve increased accuracy at reduced runtime compared
to sophisticated probabilistic approaches, thereby enabling inference of complex evolution of gene families across a broad range of species and large datasets (Chapter 4).

- I discuss the need for phylogenetic methods for reconstructing and systematically analyzing gene family evolution at the sub-gene level, in particular taking into account domain rearrangements. I present a phylogenomic pipeline for identifying homologous domains and develop an evolutionary model and reconstruction algorithm, STAR-MP (Species Tree informed Architecture Reconstruction through Maximum Parsimony), that incorporates domain-level events such as generation, duplication, loss, fusion, and fission. I then apply this pipeline to a clade of nine \textit{Drosophila} species and report numerous striking examples of architecture evolution that cannot be captured by traditional gene-level methods (Chapter 5).

- I discuss the implications of this work to the field, including the central role of computational techniques in modern biological analysis, and I consider possibilities for future directions (Chapter 6).

1.2 Gene and genome evolution

In this section, we review several major processes responsible for gene and genome evolution.

1.2.1 Mutation

At the most basic level, the genome can change due to a point mutation, that is, via the replacement (substitution) of a single nucleotide with another. Within a gene, mutations can be functionally categorized into three types: silent, in which the mutation results in no change to the amino acid sequence (due to degeneracy of the genetic code); nonsense, in which the mutation results in a premature stop codon; or missense, in which the mutation results in a codon for a different amino acid. A point mutation can also occur via the insertion or deletion of a single base pair, resulting in a frameshift (change in reading frame) and typically substantially altering the resulting protein as, in general, such a mutation causes subsequent nucleotides to code for different amino acids and alters the first stop codon encountered in the sequence.

1.2.2 Genetic drift

We might then be interested in how the frequency of a genetic variant, or allele, in a population changes over time as a result of random sampling. That is, alleles in the offspring are a sample of those in the parents, and which individuals survive and reproduce are determined partly by chance.
Therefore, randomness alone can change allele frequencies from generation to generation. This process of genetic drift will ultimately result in an allele becoming fixed (that is, reach a frequency of 1) or lost (that is, reach a frequency of 0). Thus, because random sampling cannot create new alleles but can cause existing alleles to disappear, genetic drift acts to reduce genetic variation over time (for more on this process, see [105, Ch. 3]).

Note that genetic drift is a neutral process, distinct from the directed process of change that is natural selection. For this reason, it is sometimes referred to as “random genetic drift”. The idea that molecular evolution is mainly caused by genetic drift acting on neutral mutations is termed the “neutral theory of molecular evolution” [131, 133, 134] and was later expanded into the “nearly-neutral theory of molecular evolution” [177, 178]. There remains some debate over the relative importance of natural selection versus genetic drift [90, 136] though it is generally agreed that in natural populations, both forces play a role.

1.2.3 Gene duplication (and loss)

At a larger scale, evolutionary events can occur at the gene-level. In particular, gene duplication has long been identified as a major mechanism for generating new genes and functions [154, 158, 176, 191] whereas gene loss plays a similarly important role in shaping genomic content [98, 173]. In the classic model of duplication [133, 176], the ancestral copy maintains its original functions and the duplicate gene, free from selective pressure, can accumulate mutations and evolve new functions; this is known as neofunctionalization. Other evolutionary fates are also possible: in subfunctionalization, the functionality of the original gene is distributed among the two copies [71, 118], and in non-functionalization, the duplicate gene degenerates into a pseudogene or is lost altogether [67, 100].

In addition to small-scale duplications of individual genes, large-scale duplications are also possible at the segmental chromosomal or genome levels; for example, whole-genome duplication is believed to have occurred in yeast [130] and is responsible for polyploidy in plants [39, 176].

1.2.4 Horizontal gene transfer

In contrast to the vertical transmission of genetic material from a parental generation to its offspring, in horizontal, or lateral, gene transfer (HGT or LGT), a gene is transmitted between organisms [175]. HGT has played a major role in prokaryotic evolution [21, 47, 74]; for example, it is the primary reason for bacterial antibiotic resistance [137] and can convert benign bacteria into pathogens [175]. There is also growing evidence that HGT is common among unicellular eukaryotes [15], and HGT has been observed in the evolution of plant mitochondrial genes [18] and from intracellular bacteria symbionts to their multicellular eukaryotic hosts [115].
1.2.5 Gene fusion and fission

The last process that we shall consider is gene fusion and fission. That is, new genes can arise when multiple genes fuse into a single gene, for example, when two adjacent genes fuse by readthrough transcription, or when a single gene splits into multiple genes, though the underlying mechanism for fission remains unclear [154]. Gene fusions and fissions have been involved in the formation of \(\sim 0.5\%\) of prokaryotic genes [227], and gene fusions have also been reported in eukaryotes [252]. More generally, fusions and fissions can occur in the context of domain rearrangements [20].

1.2.6 Other evolutionary events

We have highlighted only a few evolutionary processes involved in the origin of new genes and functions; others include gene conversion, in which one version of a gene replaces a different version [113, 174, 223] and de novo origination, in which a coding region originates from previously non-coding genomic region [154]. In addition, new genes can be created by a combination of mechanisms; for example, the *Drosophila* gene *jingwei* arose through exon shuffling, retroposition and gene duplication [155].

1.3 Phylogenetics and phylogenomics

*Phylogenetics* provides an important foundation for studying evolution. In this field, the goal is to understand the evolutionary history of a set of taxa, which can be species or genes, or, as is more often the case in population genetics, populations or individuals. The most commonly used representation for depicting an evolutionary history, or *phylogeny*, is the *phylogenetic tree*. Nodes within the tree represent taxa, with terminal nodes, or leaves, representing extant taxa and internal, or ancestral, nodes representing ancestral taxa. Taxa are joined together through connecting edges or branches (with no cycles allowed), and together, these nodes and branches form a tree topology that describes how taxa are related. To measure the degree to which taxa are related, branches of the tree may additionally have lengths often represented in units of time (e.g. millions of years) or sequence divergence (e.g. number of substitutions per site).

If the most recent common ancestor (MRCA, also known as the least common ancestor, or LCA) of all taxa under consideration is known, it is called the root and the tree is said to be rooted. In this case, the tree is directed with each node corresponding to the MRCA of all descendants; otherwise, the tree is unrooted and simply illustrates the relatedness of the leaves without any assumptions of ancestry. Several useful relationships exist between nodes in a rooted tree; in particular, each non-root node has a parent and each internal node a set of children. If all internal nodes have at
most two children, the tree is said to be binary; furthermore, if all internal nodes have exactly two children, the tree is said to be a full binary tree. Unless otherwise stated, all trees in this thesis are full, rooted, and binary.

1.3.1 Gene trees and species trees

For a cluster of genes with detectable share ancestry, or a gene family, we wish to infer the evolutionary events likely responsible for its creation. To describe this evolutionary history, we will be primarily interested in two types of trees: the gene tree, which describes the evolutionary relationship of a set of genes, and the species tree, which describes the evolutionary relationship of a set of species. In a species tree, the bifurcations represent speciation events, or points in time at which an ancestral population divided into non-interbreeding populations that each began to evolve as distinct species. Similarly, bifurcations in a gene tree represent points at which a gene replicated into distinct sequences that each began to evolve independently. To understand the events responsible for this replication, we can think of the gene tree as evolving “inside” of the species tree [82, 159, 181] (Figure 1.1).

In the simplest scenario, the gene tree and species tree are congruent, that is, they share the same topology, indicating that a single gene was present in the common ancestral species (represented by the species tree root), and every time an ancestral population speciated, each descendant species inherited a copy. This has resulted in a single copy of the gene in all modern species, and within the phylogeny, every bifurcation in the gene tree is a speciation event imposed by the species tree (Figure 1.1A).
In more complex scenarios in which the two tree topologies differ, evolutionary events other than speciations must be postulated to explain the observed incongruence. For example, in eukaryotic species separated by sufficiently large distances, discordance between the gene tree and species tree typically arises due to gene duplications and gene losses (Figure 1.1B). These can be viewed as complementary events: a gene duplication creates an additional copy of the gene in the genome and is represented as a bifurcation in the gene tree that occurs in the middle of a species branch, and a gene loss deletes a gene from the genome and is represented as “missing” branches in the gene tree.

1.3.2 Reconciliation

The nesting of the gene tree within the species tree is represented by a reconciliation. While several definitions of reconciliation exist [8, 50, 84, 86], the simplest is a mapping from each node in the gene tree to a node in the species tree that specifies the species to which the gene belongs [82, 181] (Figure 1.1C). For a given gene tree and species tree, it is this reconciliation that indicates the particular number and order of evolutionary events that have given rise to the gene tree. Furthermore, there exist multiple ways to reconcile a gene tree to a species tree, with each reconciliation implying a different evolutionary history.

The earliest reconciliation method, Most Parsimonious Reconciliation (MPR), defines an “optimal” reconciliation as one that minimizes the number of implied duplications [181]. Given a mapping that, for each extant gene, defines the extant species to which it belongs, MPR works recursively up the gene tree and determines, for each ancestral node in the gene tree, the species to which its children belong, then maps the ancestral node to the LCA of these species [290]. In addition to minimizing the number of duplications, the LCA mapping also minimizes the number of duplications and losses (mutation cost) [86].

While traditionally, most reconciliation algorithms have used a parsimony framework to minimize the number of implied events for a binary gene tree and species tree, more recently, non-binary [27, 259] and probabilistic [8, 51] formulations have also been developed, as have models incorporating horizontal gene transfer [33, 52, 84, 102, 137] or incomplete lineage sorting [201].

1.3.3 Orthologs and paralogs

Gene histories are fundamental for inferring relationships between extant genes. In particular, for two genes with shared ancestry, or homologs, it is often useful to distinguish between shared ancestry due to speciation, in which case the genes are orthologs, and shared ancestry due to duplication, in which case the genes are paralogs [68]. Whereas orthologs often (but not always) have the same function, duplication allows for paralogs to evolve new functions through neofunctionalization.
and subfunctionalization [64]; identifying orthologous and paralogous relationships therefore plays a major role in functional annotation [58]. Furthermore, tracing the history of one-to-one orthologous families can aid researchers in inferring population sizes and structures [44] or in the reconstruction of species histories [32].

1.3.4 Phylogenetic methods

Due to their many powerful applications, accurate gene tree reconstruction, species tree reconstruction, and gene tree-species tree reconciliation are fundamental problems in phylogenetics. For the problem of species tree reconstruction, traditional phylogenetics relies on morphological traits, which suffers from a number of disadvantages: for example, the traits need to be measured and quantified, the selection of traits necessarily reflects a hypothesis about their evolutionary relevance, the distribution in values for continuously varying traits can overlap between species, different traits may yield different phylogenies, and similar traits may have evolved independently in different lineages due to convergent evolution. While it does not address all of these problems, modern phylogenetics relying on molecular data offers the benefit in that the characters in biological sequence data, that is, nucleotides or amino acids, are unambiguous and discretely defined.

Furthermore, unlike morphological phylogenetic analysis, molecular phylogenetic techniques can be applied to the problem of gene tree reconstruction, thereby aiding in our understanding of gene family evolution. In general, these methods take as input a multiple sequence alignment and output a gene tree, and they can be broadly classified into two categories.

In distance-based approaches, the alignment is first converted into a pairwise distance matrix, and this matrix is then used to build a tree such that the discrepancy between the distances observed and those implied by the tree is minimized. Examples of distance-based methods include Unweighted Pair Group Method with Arithmetic Mean (UPGMA) [228], neighbor-joining (NJ) [215], and least-square error (LSE) [69]. Many theoretical studies have focused on understanding the consistency of these approaches (that is, whether they correctly reconstruct a tree when the distances fit perfectly on that tree) or on the biological relevance of the optimization criterion [10, 23, 38, 76, 167]. Furthermore, distance-based methods remain popular as the trees can be reconstructed in polynomial time [24, 169, 234].

Nevertheless, character-based approaches are now the method of choice for phylogenetic reconstruction. These can be further subdivided into three categories: maximum parsimony (MP) [68], maximum likelihood (ML) [65], and Bayesian inference [195]. All build a tree directly from the alignment, are inherently statistical and based on an evolutionary model, and, by using the character data directly, are able to incorporate character state distributions and substitution
rate variations, thus yielding more accurate reconstructions. However, these advantages come at the cost of increased model complexity, which is problematic as violations of the assumed model can lead to incorrect trees. Furthermore, the space of possible tree topologies grows exponentially with the number of leaves; thus, computing a score for all topologies is intractable, and current approaches use heuristic methods that search only a fraction of the tree space, either through local rearrangements [66] or, for Bayesian methods, through Markov Chain Monte Carlo sampling [91, 108, 165]. Finally, existing algorithms are also computationally intensive, making them impractical for analyzing large datasets.

A key component of molecular phylogenetic techniques is the substitution model under which sequences evolve. The most common models are neutral, independent, finite-sites, stationary, time-reversible models, and for nucleotide evolution, range from the simplest, Jukes-Cantor [125], through the more complex, including Kimura-2-parameter [132] and Hasegawa, Kishino, and Yano [107], to the most general, generalized time-reversible model [143, 242]. When applied to phylogenetic trees, ancestral sequences states are unknown and are thus marginalized out, sequences along different branches are assumed to evolve independently and often according to a molecular clock, and the prior distribution for nucleotides at the root of the tree is assumed to be equal to the base frequencies in the substitution model [65].

1.3.5 Phylogenomic methods

With the growing availability of genome sequences, phylogenetic analysis can be carried across both sizable clades and whole genomes, in a research field called phylogenomics [58, 59]. This has led to a better understanding of evolutionary conservation and divergence within species such as yeasts [26, 267], flies [99], vertebrates [260], and plants [122]. At the same time, phylogenomic approaches also benefit from the ability to exploit genome-wide information, for example, by incorporating species tree branch lengths, duplication-loss rates, or synteny [8, 199, 200, 260].

These studies often utilize one or more of the several computational methods that have been developed for detecting gene families and inferring their histories. The phylogenomic pipeline typically proceeds as follows: (1) We are given as input multiple fully sequenced genomes along with their gene models. (2) We determine the sequence similarity among the genes and cluster the genes into families. (3) We reconstruct gene trees for each family. (4) We reconcile the gene trees to a common species tree, allowing us to infer homology relationships and all evolutionary events. Note that this pipeline is only one way in which to conduct a phylogenomic study. For example, studies of gene family expansion and contraction need not reconstruct the underlying gene trees [97].

In this thesis, we assume that the species tree is known and that gene annotations and families...
exist in literature, in online databases, or can be computed to a high degree of accuracy using existing tools. Therefore, we will focus mainly on methods for improving the last two stages of this pipeline, that of gene tree reconstruction and gene tree-species tree reconciliation.
Chapter 2

Statistically informed gene tree reconstruction with gene duplication and loss: TreeFix

Accurate gene tree reconstruction is a fundamental problem in phylogenetics, with many important applications. However, sequence data alone often lack enough information to confidently support one gene tree topology over many competing alternatives. Here, we present a novel framework for combining sequence data and species tree information, and we describe an implementation of this framework in TreeFix, a new phylogenetic program for improving gene tree reconstructions. Given a gene tree (preferably computed using a maximum likelihood phylogenetic program), TreeFix finds a “statistically equivalent” gene tree that minimizes a species tree based cost function. We have applied TreeFix to two clades of 12 *Drosophila* and 16 fungal genomes, as well as to simulated phylogenies, and show that it dramatically improves reconstructions compared to current state-of-the-art programs. Given its accuracy, speed, and simplicity, TreeFix should be applicable to a wide range of analyses and have many important implications for future investigations of gene evolution.

2.1 Eukaryotic gene tree reconstruction

Although gene trees have many powerful applications, tree-based analyses depend strongly on the accuracy of the reconstruction [96, 199]. However, unlike species tree reconstruction, which can

benefit from the use of well-behaved gene families as well as multi-gene phylogeny construction methods [25, 46], gene tree reconstruction is complicated by the fact that many genes lack enough information to confidently support a single gene tree topology. Thus, "sequence-only" algorithms that reconstruct gene trees using only the sequence data (e.g. PAUP* [238], BioNJ [75], PhyML [94], RAxML [230], MrBayes [208]) often produce incorrect and poorly supported gene trees.

2.1.1 Existing work

However, within eukaryotic genomes, recent studies have found that incorporating species tree information can drastically improve gene tree accuracy [200, 260]. This has led to the formulation of "species tree aware" methods, which often combine sequence likelihood with a topology prior based on a known species tree, with the most principled methods adopting a Bayesian approach (e.g. PrIME-GSR [8], SPIMAP [200]), though simpler models (e.g. TreeBest [260], SPIDIR [199]) also exist. However, these models often require additional parameters, such as estimates of divergence times and duplication-loss rates, and they tend to be very computationally intensive.

In parallel, several "hybrid" methods have been developed for resolving gene tree and species tree incongruence to produce "error-corrected" gene trees. These are often based on a reconciliation framework and attempt to minimize a species tree aware cost function based on the inferred evolutionary events. For example, both NOTUNG [30, 55] and tt [85] consider local rearrangements around an initial gene tree to find an error-corrected gene tree that has minimum duplication-loss cost after reconciliation. While these algorithms only require a known species tree topology and are therefore much simpler than model-based species tree aware approaches, they suffer from two important drawbacks: (1) they limit their search space and can therefore miss the correct tree topology if it is distant from the initial tree, and (2) because they ignore whether the corrected gene tree is supported by the sequence data, they cannot guarantee that the corrected gene tree does not overfit to the species tree.

2.1.2 Contributions

To address these shortcomings, we present a novel hybrid method TreeFix. Like other hybrid methods, TreeFix rearranges an input gene tree to minimize the number of inferred duplications and losses. However, TreeFix is novel in that it also uses the sequence data (i.e. nucleotide or peptide alignment) to guarantee that the final corrected gene tree is "statistically equivalent" in likelihood to the initial input tree (i.e. the difference in likelihood between the two trees is not significant). In essence, TreeFix recognizes that although phylogenetic programs often return a single optimal gene tree (or possibly a consensus tree across multiple bootstraps), multiple gene trees are
often statistically equivalent, as measured by likelihood ratio tests, such as the Kishino-Hasegawa (KH) test [135], the Shimodaira-Hasegawa (SH) test [220], or many others [221]. Furthermore, one of these statistically equivalent gene trees will often more accurately reflect the true gene tree topology. By incorporating this statistical test with a reconciliation cost, such as the duplication-loss cost, we can therefore improve phylogenetic accuracy. In addition, because of this guarantee, TreeFix is free to employ an expanded search algorithm that can explore more distant parts of the tree space. We find that together, these techniques lead to a simple yet powerful method that requires few modeling assumptions or parameters and produces highly accurate gene trees ideal for inferring the evolutionary history of gene families. The TreeFix program is available at http://compbio.mit.edu/treefix.

We have applied TreeFix to both real and simulated datasets and compared its performance to that of several other gene tree reconstruction methods. We find that TreeFix shows drastic improvement over existing sequence-only and hybrid approaches, with performance comparable to the most sophisticated species tree aware Bayesian approaches.

2.2 Gene tree landscape

To understand the basic idea behind TreeFix, consider the likelihood landscape of the gene tree space (Figure 2.1A). Ideally, TreeFix is given as input the ML tree (models with non-unique ML trees should not be used). This tree corresponds to the highest peak in the landscape, but often, this peak is located in a plateau of high likelihood topologies. Methods such as NOTUNG and tt make local rearrangements to explore this surrounding plateau for the topology that minimizes some user-defined cost function (e.g. the number of inferred duplications and losses), where this cost function is used as a heuristic for improving gene tree accuracy. However, these local moves may result in a topology outside the plateau that has significantly worse likelihood than the ML topology. Furthermore, the likelihood landscape may also contain multiple peaks and valleys, necessitating a larger search to explore distant plateaus. TreeFix essentially searches among topologies within the landscape that lie above a certain threshold, using reconciliation cost as a heuristic to determine an optimal tree among these topologies. In this way, TreeFix is able to move beyond local rearrangements to find a minimum cost gene tree without overfitting to the species tree.

Note that TreeFix inherently assumes that regions of high sequence likelihood and low reconciliation cost overlap, an assumption held up in practice (Figure 2.1B). When this is not the case, TreeFix errs on the side of sequence support (rather than species tree support) and returns a gene tree with high sequence likelihood and high reconciliation cost.

Our goal is to find, among all gene tree topologies that are statistically equivalent to the maximum
reconciliation cost (e.g. duplication-loss cost)
Given a significance level \( \alpha \) that represents the probability of false rejection (i.e. we believe the trees are not equally supported when they actually are), we then reject the null hypothesis and say the trees are not equally supported if \( p < \alpha \), or we fail to reject the null hypothesis and say that the trees are equally supported if \( p \geq \alpha \). Note that if \( \alpha = 0 \), all trees are equally supported, effectively removing the statistical test and causing TreeFix to return the minimum cost gene tree regardless of sequence support, whereas if \( \alpha = 1 \), no trees are equally supported, effectively causing TreeFix to return only the tree with highest sequence support, e.g. the ML tree.

While users may implement their own statistical module, by default, TreeFix uses the SH test provided by the RAxML package. For further information on likelihood tests, including a discussion of statistical power, how to correct for multiple comparisons, and an error rate analysis, see Sections A.1 and A.2.

2.2.2 Reconciliation cost as a heuristic for topological accuracy

To calculate a species tree aware cost, we make use of the reconciliation framework, in which any incongruence between the gene tree and species tree topologies is explained by postulating evolutionary events, such as gene duplication, gene loss, horizontal gene transfer, or incomplete lineage sorting [159]. While users may implement their own cost module, by default, TreeFix uses maximum parsimony reconciliation (MPR) [181, 290] with the duplication-loss cost function [82], which seeks the reconciliation with the fewest total number of inferred duplications and losses. In addition to being the standard model used in many species tree aware and hybrid approaches, e.g. in SPIMAP, TreeBest, NOTUNG, and tt, we found the reconciliation cost to be highly correlated with gene tree topological accuracy (Section A.3, Table A.3), lending support to our approach of using this metric to incorporate information from the species tree into the gene tree reconstruction.

2.2.3 Tree search

Since it is impractical to search through the space of all possible gene tree topologies, we use a heuristic hill climbing search strategy. The idea is to start with the given input gene tree and find a better tree in its neighborhood (defined using some tree edit operation). This constitutes one local search step. This better tree then becomes the starting point for the next local search step, and so on, for either a predefined number of local search steps or until a local optima is reached. Local search forms the basis of almost all known parsimony and likelihood based phylogeny construction programs, e.g., in PAUP*, RAxML, PhyML, and others, and has been employed for gene tree error correction as well.

In our implementation, we use nearest neighbor interchange (NNI) and subtree prune and regraft
(SPR) [66, Ch. 4] to define the neighborhood in each local search step. In addition, we use the reconciliation cost to prescreen topology proposals. In particular, on each iteration, we perform a random NNI or SPR operation on the current optimal gene tree and compute its cost. This proposal is always accepted if it has a lower cost and is accepted with some predefined probability if it has a higher cost, and we repeat this local search until we have $n_q$ proposals, after which only those proposals with a cost lower than the optimal are retained. We then sort the proposals by their costs, set the first proposal with statistically equivalent likelihood as the new optimal, and repeat this entire process for $n_i$ iterations. Notably, this search strategy allows us to “jump” over valleys of low tree cost or low likelihood and explore distant parts of the gene tree landscape.

### 2.3 The TreeFix algorithm

TreeFix takes as input a gene tree $T_i$, a multiple alignment $A$, a species tree $T$, a test statistic $stat$ and significance level $\alpha \in [0,1]$ for likelihood equivalence. Additionally, it takes three search parameters: $n_i, n_q \geq 1$ that control the number of tree proposals, and $f \in [0,1]$ that specifies the fraction of proposals to reroot.

For an arbitrary gene tree $T_x$, TreeFix evaluates two functions in order to determine how the tree fits within the likelihood landscape: (1) $c_x = cost(T_x; T_s) \geq 0$, i.e. the cost of the gene tree based on the species tree, and (2) $p_x, \delta_x = LH_{stat}(T_x; A, T_i)$, i.e. the statistical significance and change in log-likelihood $ln L_i - ln L_*$ of the gene tree (against the input gene tree) based on the test statistic, multiple alignment, and input tree.

TreeFix outputs the optimal rooted gene tree $T^*$, i.e. the gene tree with minimum cost and statistically equivalent likelihood, if such a tree is found. If multiple trees have minimum cost and are statistically equivalent, the one with minimum change in likelihood is returned. The main algorithm is as follows (Figure 2.2):

1. Initialize $T^* = T_i$, $c^* = c_i$, and $\delta^* = ln L_i - ln L^* = 0$.

2. (Tree search) Make $n_q$ proposals $\{T_x\}$, compute $c_x$ for all $T_x$ in $\{T_x\}$, and reroot $f$ of them to have minimum cost.

3. (Reconciliation cost) Sort $\{T_x\}$ in order of increasing cost, and retain only those for which $c_x \leq c^*$. Call this $[T_x]$.

4. (Likelihood test) Consider proposal $T_x$ in $[T_x]$, and compute $p_x$ and $\delta_x$. If $p_x \geq \alpha$ and either (a) $c_x < c^*$ or (b) $c_x = c^*$ and $\delta_x < \delta^*$, set $T^* = T_x$, $c^* = c_x$, and $\delta^* = \delta_x$, and go to step 5. Else consider the next proposal.
Figure 2.2: TreeFix algorithm. After initialization (step 1), TreeFix proposes \( n_q \) trees, with \( f \) of them rerooted to have minimum cost (step 2, details of the search strategy omitted for clarity). It then accepts or rejects trees based on their costs (step 3) and likelihood scores (step 4), returning a new optimal tree \( T^* \) that is statistically equivalent to the input tree \( T_{in} \) but has minimum cost. This is repeated for \( n_i \) iterations or until the optimal cost is 0 (step 5).

5. Repeat steps 2–4 for \( n_i \) iterations or until \( c^* = 0 \).

Though users can input any tree, in practice, we recommend that users input a ML tree, e.g. from RAxML or PhyML, and use the same likelihood model as the input tree when computing the test statistic.

Note that TreeFix does nothing if the input gene tree contains \( \leq 2 \) genes, is unrooted and contains \( \leq 3 \) genes, or has a reconciliation cost of 0. Furthermore, if the gene family contains one gene per

2.3 The TreeFix algorithm
species, TreeFix first checks the likelihood and reconciliation cost of the gene tree topology that is congruent to the species tree topology.

To measure the uncertainty of different topologies and events, we also implemented a bootstrapping procedure. To bootstrap the entire pipeline would require resampling the alignments, reconstructing the ML trees using these alignments, then passing both the resampled alignments and associated ML trees to TreeFix for error correction. However, such a procedure would be computationally expensive and infeasible for large datasets. Furthermore, we envision TreeFix as a tool to be used in conjunction with existing ML programs, most of which store only the bootstrap trees and not the bootstrap alignments. Therefore, we have implemented an approximation in which we bootstrap only the TreeFix stage of the pipeline. If bootstrapping is enabled, then TreeFix resamples the alignment and error corrects the input gene tree for each resampled alignment. Note that we reuse the original ML tree topology (reconstructed from the full data) across these bootstraps; that is, we do not explore the uncertainty in the topology of $T_{in}$. However, the likelihood test does optimize the branch lengths and recalculate the likelihood of $T_{in}$ on each resampled alignment. Therefore, as long as the ML tree topology reconstructed from the full alignment can be considered as a good proxy for the ML tree topology that would have been reconstructed from the bootstrapped alignments, this approximation should have little effect on the statistical significance of alternative topologies and the resulting TreeFix corrected gene trees.

2.4 Benchmarks

We evaluated TreeFix using two clades of 12 *Drosophila* and 16 fungi (Figure 2.3) and using the same datasets used to evaluate SPIMAP [200]. This included 1000 simulated gene families (generated under the SPIMAP model) across each clade, as well as 5351 real gene families across the 16 fungal genomes.

For comparison, we also evaluated several phylogenetic programs, including the “sequence-only” probabilistic program RAxML, the “species tree aware” programs SPIMAP and TreeBest, and the “hybrid” programs NOTUNG and tt (pipeline and algorithm parameters provided in Section A.4; results using sequence-only methods BioNJ, PHYML, and MrBayes, and species tree aware method PriME-GSR can be found in Rasmussen and Kellis [200]).

2.4.1 Simulated datasets

In the simulated dataset, the correct phylogeny is unambiguously known, allowing us to analyze several different aspects of the phylogenetic programs.
To measure accuracy, we evaluated several different metrics including topological accuracy, branch accuracy, and sensitivity and precision of duplication, loss, and ortholog inference (Figure 2.4). While we ran TreeFix on both the fly and fungi clades, we focus our discussion here on the results of the larger fungi clade, as the phylogenetic programs performed similarly across many of the metrics using the smaller fly clade.

We found that TreeFix significantly improves on the input RAxML trees, improving topological accuracy by 82.8-84.8%, branch accuracy by 22.7-23.2%, and duplication and loss precision by 64.6-69.6% and 82.9-89.6%. (The other metrics are less sensitive to gene tree errors, showing between 0.3% reduction and 13.6% improvement.) Additionally, TreeFix performance is comparable to that of the most sophisticated reconstruction method analyzed (SPIMAP), and both of these dramatically outperform all other methods.

The low performance of RAxML is, of course, expected as it utilizes only sequence data. Among the species tree aware and hybrid methods, TreeBest performs by far the worst, which we believe can be attributed to its relatively simple penalized likelihood approach. (Results presented here used default parameters for TreeBest. Analysis using a variety of parameter settings for TreeBest did not show an appreciable change in accuracy.)

Using quick search parameters, TreeFix performs slightly better than NOTUNG (3.9% improvement in topological accuracy, 2.3% and 7.2% in duplication/loss precision) and significantly better than tt (20.4%, 25.1%, 53.5% improvement). Furthermore, we found that 1.8% of the NOTUNG trees and 1.1% of tt trees fail the SH test (compared to the input RAxML trees at α = 0.05), suggesting that the decreased performance of these hybrid methods is at least partially a result of overfitting to the species tree.

In light of this, we analyzed TreeFix to determine the effect of overfitting on gene tree error.

### 2.4 Benchmarks
Figure 2.4: Phylogenetic accuracy and runtime using several phylogenetic methods on simulated fly and fungal datasets. (A) Both hybrid and species tree aware methods have high reconstruction accuracy for correctly inferring the full gene tree topology for the fly dataset. TreeFix has the highest reconstruction accuracy for the larger fungal dataset. (B) The percent of accurately reconstructed branches is similar across all methods for the fly dataset, but the hybrid methods and SPIMAP show significant improvement over TreeBest and RAxML for the fungal dataset. (C) Despite topological and branch inaccuracies, pairwise ortholog detection is robust across all methods in both precision and sensitivity. (D,E) TreeFix (long) and SPIMAP infer duplications and losses with a high degree of sensitivity and precision, with both these methods offering a slight improvement over TreeFix and NOTUNG (100). Again, the hybrid methods and SPIMAP greatly outperform TreeBest and RAxML, particularly in terms of precision for the fungal dataset. (F) TreeFix achieves performance comparable to SPIMAP at a fraction of the runtime (average runtimes provided for the fungal dataset). Note that TreeBest and RAxML were run with 100 bootstraps, whereas all other methods were run without bootstrapping. (G) TreeFix runtime can likely be improved if the program were ported to a more efficient programming language. For more metrics, see Table A.1.
Figure 2.5: Phylogenetic accuracy of TreeFix for varying α on simulated fungal datasets. TreeFix was run for a range of α between 0 and 1 (α = 0, 10^{-10}, 10^{-5}, 10^{-3}, 0.01, 0.05, 0.1, 0.2, 0.25, 0.5, 1). Note that α = 0 returns a gene tree with minimum (duplication-loss) reconciliation cost, and α = 1 returns the maximum likelihood gene tree. Ideally, running TreeFix with α = 1 should return the input RAxML gene tree, but likelihood estimation may cause TreeFix to return a different tree. That is, though ε differences in likelihood are indistinguishable to RAxML, TreeFix still returns the gene tree with higher likelihood.

We ran TreeFix with α = 0, effectively removing the likelihood test and finding the minimum duplication-loss tree. We found that 7.4% of the resulting trees fail the SH test (at α = 0.05), and that topological accuracy decreases by 5.2% to 88.8% and duplication precision decreases by 9.0% to 79.8% (loss precision decreases by < 1.4%). This suggests that, as expected, ignoring sequence information is detrimental to gene tree accuracy and further supports our strategy of balancing a tree search strategy with a likelihood test. Further analysis showed that TreeFix performance remains relatively stable for 10^{-3} ≤ α ≤ 0.2 (Figure 2.5). Within this range, the Robinson-Foulds (RF) distances [204] against the TreeFix trees at α = 0.05 are also highly similar (average RF < 0.009), suggesting there exists a good compromise between sequence and species tree information.

Impressively, TreeFix performance is comparable to, and, using expanded search parameters, even exceeds, that of SPIMAP (4.8% and 6.6% improvement in duplication/loss precision, < 3.3% difference in all other metrics). This is despite the SPIMAP model being used to generate the simulated trees, and SPIMAP incorporating a number of additional parameters, including species information.

2.4 Benchmarks
divergence times, duplication and loss rates, and lineage-specific gene rates, in its model, whereas TreeFix uses only the species tree topology. It is possible that given a longer search time, SPIMAP would perform the best, though it is also worthwhile to note that SPIMAP already has an average runtime 2.3× that of RAxML+TreeFix (long) and 16.4× that of RAxML+TreeFix (with default search parameters).

Our goal with TreeFix was to develop a method that is feasible enough to include in a phylogenomic pipeline containing thousands of trees and a variety of family sizes; thus, we also evaluated its reconstruction speed and scalability. We found that TreeFix reconstructs gene trees with accuracy comparable to species-tree-aware Bayesian methods at a fraction of the runtime (Figure 2.4G, Section A.5, Table A.2). Furthermore, TreeFix consistently outperforms other methods across a range of gene family sizes (Section A.6, Figure A.3) and simulated species trees (Section A.7, Figure A.4), and its runtime increases at the same rate as that of RAxML. Thus, TreeFix can easily be inserted into existing phylogenetic pipelines without a noticeable increase in runtime complexity.

2.4.2 Biological dataset of 16 fungal genomes

We have also assessed the performance of TreeFix on a real dataset. Most TreeFix trees were well-supported: the minimum and mean bootstraps averaged over all trees were 51.8% and 85.8%, respectively. (This is similar to RAxML support at 54.0% and 86.8%.) As the ground truth is not known for real data, we used several informative metrics to assess the quality of reconstructed gene trees (Table 2.1).

For the first metric, we assessed each program’s ability to infer syntenic orthologs, defined as pairwise orthologs that are highly likely to be orthologous given their surrounding conserved gene order. We found that TreeFix recovers syntenic orthologs with 95.2–97.6% sensitivity, comparable to SPIMAP at 96.5% and NOTUNG at 96.1% and a significant improvement over RAxML, which performed the worst, at 63.8%. However, note that these high sensitivities are also accompanied by predictions of as many as 24.4% more orthologs compared to other methods, suggesting that the improvement in sensitivity could be tied to a loss in specificity.

The second metric evaluates the total number of inferred duplications and losses across the clade. We, of course, expect the hybrid methods to infer much fewer duplications and losses compared to RAxML, as their objective is to minimize the duplication-loss cost by rearranging the input RAxML gene tree. Here, SPIMAP infers the fewest number of events, though again, TreeFix, SPIMAP, and NOTUNG found far fewer events than the other methods, inferring at least 33.5% fewer duplications and 47.6% fewer losses.

For the third metric, we used the duplication consistency score [260] to evaluate the plausibility
Table 2.1: Evaluation of several phylogenetic programs on real fungal dataset

<table>
<thead>
<tr>
<th>program</th>
<th>% orths</th>
<th># orths</th>
<th># dups</th>
<th># losses</th>
<th>DCSd</th>
<th>% GCc</th>
<th>% fail</th>
<th>SHf</th>
<th>RFg</th>
<th>runtimeh</th>
</tr>
</thead>
<tbody>
<tr>
<td>TreeFix (long)</td>
<td>96.4</td>
<td>574,946</td>
<td>6,062</td>
<td>10,981</td>
<td>0.649</td>
<td>97.3</td>
<td>*</td>
<td>0.306</td>
<td>21.35 min</td>
<td></td>
</tr>
<tr>
<td>TreeFix</td>
<td>95.2</td>
<td>569,664</td>
<td>6,505</td>
<td>12,532</td>
<td>0.609</td>
<td>94.6</td>
<td>*</td>
<td>0.302</td>
<td>45.68 sec</td>
<td></td>
</tr>
<tr>
<td>NOTUNG (100)</td>
<td>96.1</td>
<td>582,581</td>
<td>6,161</td>
<td>10,835</td>
<td>0.659</td>
<td>89.2</td>
<td>19.2</td>
<td>0.285</td>
<td>0.41 sec</td>
<td></td>
</tr>
<tr>
<td>NOTUNG (90)</td>
<td>89.1</td>
<td>556,685</td>
<td>9,906</td>
<td>23,917</td>
<td>0.395</td>
<td>94.6</td>
<td>7.5</td>
<td>0.211</td>
<td>0.38 sec</td>
<td></td>
</tr>
<tr>
<td>NOTUNG (50)</td>
<td>70.1</td>
<td>487,875</td>
<td>18,322</td>
<td>54,101</td>
<td>0.191</td>
<td>89.2</td>
<td>0.8</td>
<td>0.051</td>
<td>0.40 sec</td>
<td></td>
</tr>
<tr>
<td>tt (3)</td>
<td>82.8</td>
<td>522,834</td>
<td>9,780</td>
<td>26,621</td>
<td>0.354</td>
<td>89.2</td>
<td>16.3</td>
<td>0.224</td>
<td>3.29 sec</td>
<td></td>
</tr>
<tr>
<td>tt (2)</td>
<td>76.5</td>
<td>503,323</td>
<td>12,552</td>
<td>35,898</td>
<td>0.272</td>
<td>89.2</td>
<td>10.7</td>
<td>0.171</td>
<td>0.18 sec</td>
<td></td>
</tr>
<tr>
<td>tt (1)</td>
<td>70.0</td>
<td>482,439</td>
<td>16,310</td>
<td>48,036</td>
<td>0.206</td>
<td>89.2</td>
<td>5.1</td>
<td>0.100</td>
<td>0.05 sec</td>
<td></td>
</tr>
<tr>
<td>SPIMAP</td>
<td>96.5</td>
<td>557,981</td>
<td>5,407</td>
<td>10,384</td>
<td>0.650</td>
<td>83.8</td>
<td>–</td>
<td>–</td>
<td>21.89 min</td>
<td></td>
</tr>
<tr>
<td>TreeBest</td>
<td>79.5</td>
<td>480,680</td>
<td>11,240</td>
<td>34,287</td>
<td>0.266</td>
<td>81.1</td>
<td>–</td>
<td>–</td>
<td>25.72 sec</td>
<td></td>
</tr>
<tr>
<td>RAxML</td>
<td>63.8</td>
<td>462,039</td>
<td>21,083</td>
<td>64,037</td>
<td>0.159</td>
<td>89.2</td>
<td>–</td>
<td>–</td>
<td>4.32 min</td>
<td></td>
</tr>
</tbody>
</table>

a See Section A.4 for program details.
b Percentage of 183,374 syntenic orthologs recovered.
c Number of pairwise orthologs, duplications, and losses inferred across all gene trees.
d Average duplication consistency score.
e Percentage of 37 recent gene converted paralogs recovered.
f For the hybrid methods, percentage of trees that fail the SH test compared to the input RAxML trees at \( \alpha = 0.05 \). By design, TreeFix always returns a statistically equivalent tree.
g For the hybrid methods, average Robinson-Foulds distance compared to the input RAxML trees.
h Average runtime for reconstructing each gene tree. (TreeBest and RAxML were run with 100 bootstraps.)

of the inferred duplications by each method. For each duplication node, this score computes the percentage of species overlap in the two child subtrees, under the assumption that a low species overlap is indicative of a false duplication (followed by many compensating losses). We found that TreeFix, SPIMAP, and NOTUNG show similar duplication consistency distributions (Figure A.1) and again outperform the other methods, with SPIMAP detecting the fewest fully inconsistent duplications (score = 0) and NOTUNG detecting the most fully consistent duplications (score = 1).

The fourth metric assesses each program's ability to recover more recent duplications due to gene conversion events, which effectively tests the ability of species tree aware and hybrid methods to properly weigh conflicting species information and sequence information. Here, we found that TreeFix performs the best, recovering 94.6% of recent gene converted paralogs compared to the next best methods at 89.2%. This suggests that our approach of balancing species and sequence information using a likelihood ratio test may be superior to Bayesian (SPIMAP) or penalized likelihood (TreeBest) approaches, as well as approaches that ignore sequence information (NOTUNG, tt) or species information (RAxML) in the final tree. (One parameterization of NOTUNG was also able to recover 94.6% of these paralogs but only at the expense of far lower scores across the other metrics.) Interestingly, while neither the species tree aware nor the hybrid methods model gene conversion, the hybrid methods (> 89.2%) outperform the species tree aware methods (81.1 83.8%) and are able to attain performance at least as well as the sequence-only method (89.2%), suggesting

2.4 Benchmarks
that the underlying assumptions of these hybrid methods do not tend to cause species information to override sequence information at the expense of recovering gene conversion events.

Note that, according to our metrics, TreeFix shows performance comparable to that of SPIMAP and NOTUNG. The high performance of SPIMAP is unsurprising since it employs a sophisticated Bayesian approach. In addition, SPIMAP requires extensive preparation and training, and due to excessive run times, cannot easily be applied to larger datasets. In fact, we believe this demonstrates the advantages of TreeFix, as it is able to achieve these high levels of accuracy while requiring far fewer modeling assumptions and no additional training or parameter estimation.

We also believe that TreeFix has advantages over NOTUNG in several ways. For example, as we previously showed (Figure 2.4), our first metric of ortholog sensitivity may not reflect topological inaccuracies, and NOTUNG was only able to achieve these high levels of performance by considering all branches with bootstrap support < 100% of the maximum bootstrap as weak. Furthermore, both NOTUNG and tt operate under the implicit assumption that by only using minor rearrangements, the final output tree is still supported by the sequence data. This assumption may not hold, however: if we compare output trees against input trees, we found that RF distance and p-value have little correlation (Figure A.2). While NOTUNG can attain high performance using our metrics, almost 19.2% of the resulting trees are no longer supported by the sequence data (e.g. they fail the SH test at \( \alpha = 0.05 \)). In comparison, TreeFix can attain higher RF scores, reaching parts of the tree space unreachable by the other hybrid methods, while also returning a tree supported by the sequence data.

### 2.4.3 Alternative species tree topologies

In practice, the true species tree is not known with certainty. Therefore, we also evaluated how robust TreeFix is to incorrect species tree topologies. For the 16 fungi dataset, we assumed the correct species tree topology matched that of Butler et al. [26], which used additional evidence from genomic rearrangements, synteny, and nucleotide, purine + pyrimidine, and peptide alignments. Here, we re-evaluated the simulated fungi datasets using four alternative species tree topologies (Figure 2.6B). These topologies were chosen by evaluating a real dataset of 739 one-to-one syntenic orthologs. A concatenated nucleotide alignment of these 739 families supported topology \( T_4 \), which differs from our assumed species tree topology \( T \) in three branches (the single branch differences make up topologies \( T_1 - T_3 \)). However, the differences in sequence support between the alternative topologies and the assumed true topology were also found to be negligible (SH-test, \( p \approx 1 \) in all cases).

For the simulated dataset, topological accuracy decreases dramatically (by 62.5–92.3%) when
an incorrect species tree is used (Figure 2.6C-G), as expected. Surprisingly, however, using this metric, TreeFix using a species tree topology with a single branch error outperformed RAxML using the correct topology (recall that while RAxML is a sequence-only method, the resulting tree is still reconciled against the correct species tree). This suggests that incorrect species tree topologies still provide a lot of correct information, and furthermore, this additional information is sufficient to overcome gene tree errors caused by uncertainty in the sequence data. Using more robust metrics, we found that branch accuracy and duplication precision are reduced, though to a lesser effect (by 10.0 - 22.6% and 5.0 - 24.6%), and that ortholog inference is robust to species tree topology errors (> 98.6% accuracy), as is duplication sensitivity (> 92.2% accuracy, < 0.7% difference). Again, using branch accuracy as our metric, TreeFix using species tree topologies with a single error outperformed TreeBest and RAxML using the correct topology. Even using a species tree topology with three branch errors, TreeFix performed as well as RAxML using the correct topology. This improvement is even more pronounced for duplication accuracy: TreeFix using an incorrect topology is able to outperform all other evaluated phylogenetic methods except SPIMAP using the correct topology. However, species tree topology errors do seem to cause errors in loss inference (17.1 - 60.7% decrease in sensitivity, 32.3 – 82.3% decrease in precision), an unsurprising result, as many of the inferred losses have likely migrated to other parts of the gene tree.

For the real dataset, we found that using alternative species tree topologies, TreeFix shows similar improvement over RAxML as with the assumed correct species tree topology (e.g. higher recovery of syntenic orthologs, lower number of inferred duplications and losses, higher duplication consistency, higher recovery of gene conversion events; Tables 2.2 and 2.3). Furthermore, we found high agreement between ortholog calls (> 97.3%), with less agreement in duplication and loss inference (55.6 – 69.2%, 29.7 – 71.1%).

Using the real dataset, we also analyzed support for our alternative species tree topologies using RAxML and TreeFix gene trees. Indeed, one of the main applications of gene tree error correction is to determine the species tree topology most supported by the corrected gene trees [85], that is, the species tree topology with the fewest event counts. One advantage of this approach is that it allows us to make use of all gene families rather than only one-to-one gene families, as is the case when testing sequence support using concatenated alignments. Rather than looking only at event counts, however, we tested whether the differences in event counts between multiple species tree topologies are significant. That is, we found event count distributions using bootstrapped gene trees with the assumed correct species tree topology, then determined the p-value of the event counts with the alternative species tree topologies. If p < α, the gene trees support one species tree topology over the other, whereas if p ≥ α, there is not enough evidence to support a single species tree.

2.4 Benchmarks
Figure 2.6: Phylogenetic accuracy of TreeFix on simulated fungal datasets using incorrect species tree topologies. (A,B) The true species tree topology (T, used in generation) and alternative topologies (T1 - T4, used in reconstruction) in our analysis. For TreeFix, dark colors correspond to long search parameters, and light colors to default search parameters. (C) Using incorrect species tree topologies dramatically reduces the reconstruction accuracy of TreeFix for correctly inferring the full gene tree topology. No effect is seen for RAxML as the gene tree remains unchanged. (D) For TreeFix, the percent of accurately reconstructed branches is similarly reduced, but to a lesser effect. (E) Pairwise ortholog detection is robust to errors in the species tree topology. (F) For TreeFix, sensitivity of inferred duplications remains high while precision decreases slightly. For RAxML, inferred duplications remain largely unchanged. (G) In contrast, both sensitivity and precision of inferred losses decrease significantly.
Table 2.2: Evaluation of TreeFix on the real fungal dataset using several alternative species tree topologies

<table>
<thead>
<tr>
<th>topology</th>
<th>% orths</th>
<th># orths</th>
<th># dups</th>
<th># losses</th>
<th>DCS</th>
<th>% GC</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>T (long)</td>
<td>96.4</td>
<td>574,946</td>
<td>6,062</td>
<td>10,981</td>
<td>0.351</td>
<td>97.3</td>
<td>0.306</td>
</tr>
<tr>
<td>T</td>
<td>95.2</td>
<td>569,664</td>
<td>6,505</td>
<td>12,532</td>
<td>0.391</td>
<td>94.6</td>
<td>0.302</td>
</tr>
<tr>
<td>T1</td>
<td>96.3</td>
<td>572,407</td>
<td>6,075</td>
<td>11,016</td>
<td>0.339</td>
<td>97.3</td>
<td>0.290</td>
</tr>
<tr>
<td>T2</td>
<td>95.4</td>
<td>569,862</td>
<td>6,281</td>
<td>11,500</td>
<td>0.366</td>
<td>97.3</td>
<td>0.290</td>
</tr>
<tr>
<td>T3</td>
<td>95.7</td>
<td>570,400</td>
<td>6,221</td>
<td>11,560</td>
<td>0.363</td>
<td>94.6</td>
<td>0.285</td>
</tr>
<tr>
<td>T4</td>
<td>96.7</td>
<td>573,353</td>
<td>5,615</td>
<td>9,346</td>
<td>0.286</td>
<td>94.6</td>
<td>0.260</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>topology</th>
<th>RFₜ</th>
<th>% top</th>
<th>% branch</th>
<th>orth</th>
<th>sn</th>
<th>orth</th>
<th>ppv</th>
<th>dup</th>
<th>sn</th>
<th>dup</th>
<th>ppv</th>
<th>loss</th>
<th>sn</th>
<th>loss</th>
<th>ppv</th>
</tr>
</thead>
<tbody>
<tr>
<td>T (long)</td>
<td>0.028</td>
<td>94.9</td>
<td>94.8</td>
<td>98.6</td>
<td>97.7</td>
<td>72.5</td>
<td>77.8</td>
<td>76.0</td>
<td>86.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>0.091</td>
<td>27.2</td>
<td>84.7</td>
<td>98.2</td>
<td>97.7</td>
<td>64.6</td>
<td>69.2</td>
<td>62.5</td>
<td>71.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td>0.107</td>
<td>25.0</td>
<td>84.1</td>
<td>97.9</td>
<td>97.9</td>
<td>64.9</td>
<td>67.2</td>
<td>59.2</td>
<td>63.4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td>0.086</td>
<td>32.3</td>
<td>85.6</td>
<td>98.0</td>
<td>97.9</td>
<td>64.8</td>
<td>67.8</td>
<td>57.7</td>
<td>62.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T4</td>
<td>0.209</td>
<td>8.9</td>
<td>73.6</td>
<td>97.9</td>
<td>97.3</td>
<td>55.6</td>
<td>64.4</td>
<td>29.7</td>
<td>39.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Tables 2.2 and 2.3 provide a detailed comparison of TreeFix and RAxML on the real fungal dataset. The tables show the performance of both methods under various species tree topologies, with metrics such as the percentage of orthologs, number of orthologs, number of duplicates, number of losses, DC score, and RF score. The topologies are labeled T, T1, T2, T3, and T4, each representing a different species tree. For RAxML, the tables also include the percentage of true branches and their sensitivities, positive predictive values, and losses. The RFₜ metric measures the average RF score compared to TreeFix gene trees reconstructed with the correct species tree topology, providing insight into the agreement between different sets of gene trees. Some columns are omitted since the species tree topology only affects the reconciliation of RAxML gene trees, i.e., the gene trees are identical but have different inferred orthologs and events.

2.4 Benchmarks
topology. As with the concatenated alignment, we found that all alternative topologies are equally supported (RAxML - T1: $p = 0.27$, T2: $p = 0.28$, T3: $p = 0.26$, T4: $p = 0.14$; TreeFix - T1: $p = 0.61$, T2: $p = 0.64$, T3: $p = 0.64$, T4: $p = 0.51$; one-sided test). Interestingly, TreeFix returns higher $p$-values than RAxML. Thus, if we used a different significance level, e.g. an $\alpha$ for which $p_{\text{RAxML}} < \alpha < p_{\text{TreeFix}}$, we might conclude that an alternative species tree topology is not supported by the RAxML gene trees but is supported by the TreeFix trees. Since TreeFix returns gene trees that are statistically equivalent to the RAxML trees, this suggests that when using sequence-only methods, the evidence supporting one species tree topology over another can be partly attributed to insufficient sequence information. Therefore, care should be taken when using gene tree parsimony methods that reconstruct species trees based on sequence-only gene trees.

In summary, we found that TreeFix is more accurate than RAxML even when the species tree topology contains possible errors. In practice, if the species tree topology is uncertain, one conservative approach is to run TreeFix with multiple topologies and take the intersection of their inferred orthologs and events. Alternatively, if the multiple topologies are believed to be equally likely, then one could combine gene tree bootstraps into a single pool and from that pool, perform tree consensus to get branch bootstrap values.

2.5 Discussion

In this chapter, we have presented a new framework for combining sequence and species tree information in a principled manner. In addition, we have described a novel phylogenetic algorithm TreeFix that uses this framework to dramatically improve gene tree reconstructions, with performance comparable to the most sophisticated gene tree reconstruction algorithms achievable at a fraction of the runtime.

A major feature of TreeFix is its simplicity. There are few assumptions or parameters, and the algorithm’s behavior is easy to understand and control. The output gene tree from TreeFix is guaranteed to be statistically equivalent to the input tree, which is a condition that is directly interpretable and can be easily controlled through the test statistic and significance level. In contrast, other hybrid methods are not as careful in balancing the trade-off between species tree congruence and sequence data likelihood, likely resulting in their lower performance in our evaluations. Methods that do balance these two types of information in a more principled way (e.g. SPIMAP, PrIME-GSR) require more modeling assumptions and parameters, which may be difficult to make or obtain in many situations. Thus, we feel that TreeFix offers a useful and powerful approach that will likely be applicable to many phylogenetic datasets.

Additionally, TreeFix’s modeling assumptions are fully contained within the sequence likelihood
and reconciliation model. We have used some of the most basic phylogenetic assumptions in our analysis (GTR-Γ model of evolution and a duplication-loss reconciliation cost). Therefore, unlike species tree aware methods that tend to make many assumptions on gene tree evolution, TreeFix is applicable to a variety of datasets and compatible with a wide range of downstream algorithms, including those that account for more complicated evolutionary events. Used in conjunction, TreeFix would correct for gene tree errors due to uncertainty, and other algorithms would correct for errors due to events such as horizontal gene transfer and deep coalescence.

TreeFix is also highly modular. It has no dependency on the method used to compute the input gene tree, and it is very straightforward to substitute other likelihood tests and cost functions. For example, while our current implementation uses RAxML-computed likelihoods and the SH test, users who prefer other likelihood models or test statistics only need to implement a small Python plugin module. We have also implemented a similar modular approach to the cost heuristic, thereby allowing users to incorporate other reconciliation models, such as those that account for horizontal gene transfer [33, 37, 52], deep coalescence [44, 159], or non-binary reconciliations [55, 259], or other sources of information, such as synteny or local region properties. This leaves the complexity of the likelihood and reconciliation models to the user, and allows TreeFix to serve as a useful framework for measuring the effects of different statistical tests or cost heuristics on gene tree accuracy.

Along these lines, note that while many statistical tests compare the likelihoods of multiple trees, and we have formulated the gene tree landscape using likelihood values, the framework discussed here could easily be extended to other probabilistic measures, in particular to penalized likelihoods, as in TreeBest, or to a posteriori probabilities, as in Bayesian reconstruction methods. Of course, some of these probabilities rely on topology priors and therefore, our statistical comparisons would no longer be based solely on sequence support. In our approach, we have chosen to differentiate sequence support and species tree support, using likelihood-based statistical tests for the former and a reconciliation cost for the latter.

As with any phylogenetic method, users must decide whether the models for sequence evolution and gene tree-species tree reconciliation used in TreeFix are appropriate for a given dataset. The duplication-loss reconciliation cost was chosen as the default in TreeFix as it is broadly applicable, especially in eukaryotes [25, 82, 171, 181, 183, 260, 267], and is the model used by many species tree aware and hybrid methods [8, 55, 85, 199, 200, 260]. Still, if the species under analysis are closely related, it may be appropriate to use more advanced models that combine the duplication-loss and deep coalescence processes [201]. Similarly, when working with prokaryotic species, models that incorporate horizontal gene transfer may yield better performance [37, 52, 254] (see also Chapter 3). As we have mentioned, users have multiple options for correcting possible model

2.5 Discussion
mismatches. Because a TreeFix tree is statistically equivalent to the ML tree and equally supported by the sequence data, users can simply run TreeFix as an intermediate step between initial ML gene tree reconstruction and other methods that account for these evolutionary events. Alternatively, users may incorporate the evolutionary model directly into TreeFix by implementing their own reconciliation cost function.

Aside from improved gene tree reconstruction, because TreeFix guarantees that the final error-corrected gene tree is equivalent to the input tree in terms of sequence likelihood, it can also be used to measure the robustness of existing trees in a manner similar to bootstrapping. Furthermore, TreeFix can be used to validate biological conclusions based on phylogenetic analysis. For example, studies that compare duplication and loss counts or posit genome-wide events based on event distributions across the species trees can run TreeFix on their existing gene trees to determine whether their conclusions hold for the TreeFix-corrected gene trees. Those conclusions that are no longer supported should then be further analyzed using other methods. Similarly, TreeFix can be used to test multiple species tree topologies. If multiple topologies are equally supported, then gene trees may be insufficient to resolve the underlying species phylogeny, and other metrics, such as synteny, should be used.

While we have shown that TreeFix likely would not dramatically increase runtime complexity if applied to RAxML trees, it does currently perform significantly slower than other hybrid methods. This is most likely because we have implemented TreeFix using Python compared to the efficient programming languages (NOTUNG: Java, tt: C++) used by the other programs. Furthermore, while we have not implemented many optimizations thus far, many of the techniques used by NOTUNG and tt to reuse computation between tree proposals could also be applied to TreeFix, or the species tree topology or likelihood landscape could be used to more efficiently guide the tree search. Recent advances using GPU computation [235] may also be leveraged. Thus, future speed improvements are likely possible.

In conclusion, we believe that the concepts presented here offer a simple yet powerful alternative to existing hybrid and Bayesian models of gene tree reconstruction, and we feel that TreeFix will be a valuable addition to the phylogenetists' toolkit, as it can be easily integrated into existing phylogenomic pipelines to significantly improve gene tree reconstructions.
Chapter 3

Species-aware gene tree reconstruction with gene duplication, transfer, and loss: TreeFix-DTL

While the problem of accurate gene tree reconstruction has received considerable attention in eukaryotes, little attention has been paid to prokaryotes. The main reason for this disparity is that gene tree reconstruction in prokaryotes is complicated by the extensive presence of horizontal gene transfer. Most existing methods for gene tree reconstruction account only for gene duplication and loss, and are therefore only applicable to (multi-cellular) eukaryotes. As a result, the accurate reconstruction of prokaryotic gene trees remains a largely unaddressed problem, limiting the use of gene trees in prokaryotic evolutionary analyses.

In this chapter, we address this important problem by introducing a new and highly effective method for prokaryotic gene tree reconstruction. Our method efficiently models horizontal gene transfers, gene duplications, and losses, and uses a statistical hypothesis testing framework to balance sequence likelihood with topological information from a known species tree. Using a thorough simulation study, we show that existing phylogenetic methods yield inaccurate gene trees when applied to prokaryotes and that our method dramatically improves reconstruction accuracy. We

The work presented in this chapter has led to the following publication: Bansal MS\textsuperscript{1}, Wu YC\textsuperscript{1}, Alm EJ, Kellis M. Improved gene tree reconstruction for deciphering microbial evolution. Under review. (\textsuperscript{1} Equal contribution) Parts of this chapter were written by Mukul Bansal for the publication and have been reproduced here with his permission.
Figure 3.1: **Reconciliation with duplication, transfer, and loss.** A gene tree (black) evolves "inside" a species tree (blue), with incongruence explained by postulating gene duplications (yellow star), horizontal gene transfers (green triangle, with transfer edges as dashed lines), and gene losses (red 'x'). For a given gene tree and species tree, multiple scenarios are often possible; for example, the scenarios shown postulate (A) two transfers and one loss, (B) one duplication and two losses, and (C) two transfers.

apply our method to a dataset of 11 cyanobacterial species and demonstrate the large impact of gene tree accuracy on downstream evolutionary analyses.

### 3.1 Prokaryotic gene tree reconstruction

In eukaryotes (especially multi-cellular eukaryotes), duplication and loss play the dominant role in gene family evolution. Therefore, eukaryotic gene trees are typically studied under the duplication-loss reconciliation model [82, 181], and almost all existing species tree aware gene tree reconstruction methods are based on the duplication-loss reconciliation model [1, 55, 85, 200, 260, 276]. In prokaryotes, however, gene family evolution is more complex and is driven not just by duplication and loss but also by horizontal gene transfer, or simply transfer for short (Figure 3.1). Indeed, it has become increasingly apparent that transfer has played a major role in prokaryotic genome evolution [31, 40, 121, 137, 175, 225, 226, 286] and is perhaps the dominant form of genetic transfer among single-celled organisms [162, 255], and as a result, methods based on the duplication-loss model perform poorly and cannot be used for gene tree reconstruction in prokaryotes. Moreover, due to the complexity of the reconciliation model based on duplication, transfer, and loss (DTL-reconciliation model), there has been little systematic development of formal species tree aware methods for prokaryotic gene tree reconstruction. Consequently, prokaryotic gene trees continue to be notoriously difficult to reconstruct, leading to gross inaccuracies in prokaryotic evolutionary histories, and biologists are often unable to trust gene trees for understanding prokaryotic evolution.
3.1.1 Existing work

Two methods, AnGST [37] and MowgliNNI [172], have recently been proposed for dealing with the uncertainty of prokaryotic gene tree topologies in the context of reconciliation. Both work by considering alternate gene tree topologies, reconciling them to the species tree, and choosing the gene tree that yields the best (most parsimonious) reconciliation. Thus, by incorporating information from the species tree, both approaches produce potentially more accurate gene trees. For instance, AnGST accounts for uncertainty in gene tree topologies by taking as input, not a single gene tree, but a set of gene trees (e.g., bootstrap replicates) and creating a new gene tree that minimizes the reconciliation cost by selecting and piecing together clades from the set of input gene trees. Similarly, MowgliNNI deals with topological uncertainty by considering alternate gene tree topologies obtained by performing NNI operations (see, e.g., [66]) on the weakly-supported branches of the original input gene tree, then reconciling the alternative gene trees to the species tree, and selecting a gene tree topology with minimum reconciliation cost as the reconstructed gene tree. While AnGST and MowgliNNI are excellent first attempts at dealing with the problem of gene tree accuracy in prokaryotes, they have several limitations. First, they are both based on a simple parsimony criterion that ignores sequence support beyond the input gene tree(s), making them prone to over-fitting the gene tree to the species tree topology. Second, both consider only a limited search space, affecting their ability to recover the true gene tree topology. Third, MowgliNNI requires the input species trees be fully dated (i.e., all internal nodes of the species tree must be relatively ordered by time), which is notoriously difficult [212], making MowgliNNI less broadly applicable in practice. Fourth, since the development of these methods was driven by the need to perform more accurate reconciliation, they have not been rigorously tested from the point of view of gene tree reconstruction accuracy under a variety of conditions.

3.1.2 Contributions

In this chapter, we introduce a principled species tree aware approach for gene tree reconstruction in prokaryotes. Our method TreeFix-DTL attempts to find a tree that has the lowest reconciliation cost, under the DTL-reconciliation model, among all gene trees that have a likelihood that is "statistically equivalent" to that of the maximum likelihood tree. By balancing sequence likelihood with species divergence information from the species tree topology, TreeFix-DTL avoids the pitfalls of the existing approaches and finds more accurate gene trees. Similar ideas have been successfully used before in the context of gene tree reconstruction for eukaryotes (Chapter 2). The idea of statistical equivalence of gene tree likelihoods has also been used earlier to study the congruence between species tree and gene tree topologies in bacteria [146].
To evaluate the performance of TreeFix-DTL, we develop a thorough simulation framework that accounts for variations in species tree sizes, rates of gene evolution, alignment lengths, rates of duplication, transfer and loss, and species tree errors. We use this framework to compare TreeFix-DTL to AnGST, MowgliNNI, and some of the most popular phylogenetic reconstruction programs including RAxML, one of the most accurate and widely used sequence-only phylogeny reconstruction methods, NOTUNG, a well known species tree aware gene tree reconstruction method for eukaryotes, and TreeFix, a state-of-the-art gene tree reconstruction program for eukaryotes that we previously developed. We observe that TreeFix-DTL outperforms all other methods and improves the reconstruction accuracy of gene trees across the entire range of simulation parameters. We also analyze both simulated and biological datasets to show how the improved topological accuracy translates into significantly more accurate inference of duplication, loss, and transfer events. Specifically, our contributions are as follows:

- **We demonstrate that even the most sophisticated methods for accurate gene tree reconstruction in eukaryotes do not work well for prokaryotes.** We test the performance of NOTUNG and TreeFix on simulated prokaryotic gene families and observe that they show poor performance even when the datasets had low rates of transfer. On datasets with high rates of duplication, transfer and loss, these methods actually perform worse, on average, than the baseline (sequence-only) method RAxML.

- **We introduce TreeFix-DTL, a statistically founded approach for species tree aware gene tree reconstruction in prokaryotes.** As we demonstrate in our simulation study, TreeFix-DTL outperforms all other methods (including AnGST and MowgliNNI) at accurately reconstructing prokaryotic gene trees and achieves a great improvement in accuracy compared to RAxML. We also show that TreeFix-DTL can be scaled to gene trees with hundreds of leaves and that its performance is robust to the choice of parameters used.

- **We show that more accurate gene trees dramatically improve the inference of evolutionary events.** We reconcile the gene trees reconstructed using RAxML and those reconstructed using TreeFix-DTL with their corresponding species trees and compare the resulting reconciliations with the true history of implanted duplications, transfers, and losses in the simulated datasets. The results show that the great improvement in topological accuracy compared to RAxML translates into a dramatic increase in the ability to infer evolutionary events correctly.

- **We show that TreeFix-DTL leads to an almost 10-fold reduction in the number of inferred transfer events on a dataset of 11 cyanobacterial species.** We apply...
TreeFix-DTL to a published dataset of 11 cyanobacterial species that has been previously analyzed for horizontal gene transfer and incomplete lineage sorting, and demonstrate that, strikingly, most of the evolutionary inferences drawn using the original dataset change when more accurate gene trees are used.

Thus, we provide the first thorough evaluation of gene tree reconstruction accuracy in prokaryotes, demonstrate the limitation of existing methods, and introduce a new approach with dramatically higher accuracy. In using our approach to find that most previously inferred transfer events cannot be supported by the sequence evidence, we also demonstrate the dramatic biological impact of using the more accurate gene trees. The TreeFix-DTL program is available at http://compbio.mit.edu/treefix-dtl/.

Throughout the rest of this chapter, all trees are assumed to be binary and, unless otherwise stated, gene trees are assumed to be unrooted and species trees rooted.

3.2 The TreeFix-DTL algorithm

TreeFix-DTL is our new method for reconstructing prokaryotic gene trees and is based on the same principle as TreeFix (Section 2.3). That is, TreeFix-DTL takes as input a single maximum likelihood (ML) gene tree (obtained from programs such as RAxML [230] or PhyML [93]), the gene sequence alignment on which the gene tree was built, and a (trusted) rooted species tree. TreeFix-DTL then seeks a gene tree that has minimum reconciliation cost (under the DTL-reconciliation model) among all gene trees that have likelihood statistically-equivalent to that of the ML gene tree. To implement this strategy, TreeFix-DTL performs a local search procedure starting with the given ML gene tree and successively finds statistically equivalent gene trees that have a lower reconciliation cost.

TreeFix-DTL, like TreeFix, consists of three basic components (Section 2.2): (i) a test of statistical equivalence to filter out gene tree topologies that are suboptimal, (ii) a gene tree and species tree reconciliation method to compute the reconciliation cost, and (iii) a tree search to explore the space of alternative gene tree topologies. By default, TreeFix-DTL uses the same test for statistical equivalence as TreeFix (Section 2.2.1); that is, sequence support for an alternative tree topology is determined by computing the Shimodaira-Hasegawa (SH) test statistic [220] and comparing the p-value to a significance level α. Similarly, TreeFix-DTL employs the same heuristic hill-climbing strategy as TreeFix (Section 2.2.3), in which trees are proposed by searching the local neighborhood [66, Ch. 4], and, on each search iteration, a proposal is always accepted if it is statistically equivalent (to the input gene tree) and has lower reconciliation cost (than the current optimal gene tree) and accepted with some predefined probability otherwise. In case the search finds
multiple gene trees with statistically equivalent likelihood and the same minimum reconciliation cost, it outputs the one that has the highest likelihood.

The major difference between TreeFix-DTL and TreeFix lies in their approaches for measuring incongruence between the gene tree and species tree: TreeFix, meant for use with eukaryotic gene trees, incorporates species tree information by postulating duplication and loss events (Section 2.2.2); however, TreeFix-DTL, meant for use with prokaryotic gene trees, incorporates species tree information by postulating duplications, losses, and transfers (for more detail on the DTL-reconciliation model, see Section B.1). In particular, TreeFix-DTL uses a parsimony-based DTL-reconciliation framework in which a lower total reconciliation cost implies a better fit with the species tree topology and consequently, a more optimal gene tree. In its current implementation, TreeFix-DTL computes the DTL-reconciliation cost using RANGER-DTL [12], which assumes an undated species tree and infers a most parsimonious DTL-reconciliation. For efficiency, the inferred DTL-reconciliation is allowed to violate temporal constraints; that is, inferred transfers may induce contradictory constraints on the dates for the internal nodes of the species tree. However, RANGER-DTL is able to work with unrooted gene trees by considering all possible rootings of the gene tree and picking one that minimizes the reconciliation cost; thus, for each tree proposal, TreeFix-DTL evaluates the reconciliation cost for all possible rootings, and at the end, outputs the best unrooted gene tree found during its search.

While the correct reconciliation may not always be the most parsimonious one, our analyses suggest that parsimonious DTL-reconciliation is highly accurate, even for high rates of duplication, transfer, and loss (Figure 3.3). We also point out that TreeFix-DTL only considers the cost of the reconciliation, not the actual reconciliation itself. In our simulation study (described later), we observed that the true gene trees indeed tended to have the lowest reconciliation cost among all the gene trees considered during the heuristic search, suggesting that minimizing the reconciliation cost is an appropriate search strategy for finding more accurate gene trees. Only in 21.5% of the cases did TreeFix-DTL output a gene tree that had lower reconciliation cost than the true gene tree, and in such cases, the reconciliation cost tended to be only slightly lower.

Users may choose to customize TreeFix-DTL by (i) specifying the significance level $\alpha \in [0, 1]$ (default 0.05) for likelihood equivalence, (ii) specifying alternative costs for duplication, transfer, and loss events to be used by DTL reconciliation (defaults 2, 3, and 1, respectively), and (iii) changing the thoroughness of the local search heuristic by specifying the number of search iterations (default 1000). The default values of these parameters have been set based on extensive experimental analyses. We found performance to be robust to the choice of $\alpha$ (we tested values 0.01, 0.05, and 0.1) and to the exact event costs used; furthermore, default $\alpha$ and event costs worked well for a wide
range of datasets. Finally, we chose a relatively high default of 1000 search iterations because this yielded good performance (in terms of gene tree accuracy) even for datasets with up to 200 taxa.

3.3 Benchmarks

3.3.1 Simulated datasets

Basic experimental setup

To study the performance of the different methods at accurately reconstructing prokaryotic gene trees, we created multiple simulated datasets, accounting for the key parameters likely to have an impact on reconstruction accuracy. This includes rates of duplication, transfer, and loss; rates of sequence evolution; alignment length; and species tree size, where our choice of parameters are meant to simulate a wide variety of real gene families and are based on an analysis of 4736 gene tree, 100 species dataset [37], which consists of predominantly prokaryotic species sampled broadly from across the tree of life (Section B.3).

Our basic simulation setup is as follows: we simulated under a Yule process [104, 284] 100 random species trees with 50 taxa. For each species trees, we then simulated gene trees under a probabilistic model of gene duplication, transfer, and loss [253, 254] with three different settings (low, medium, and high) of event rates. Next, for each of the three sets of 100 gene-tree/species-tree pairs, we simulated four different mutation rates by scaling the branch lengths of the gene trees so as to correspond to 1, 3, 5, and 10 mutations per site. Finally, we simulated amino acid sequences of length 173 and 333 down the gene trees under the JTT model with gamma rate heterogeneity shape parameter 1 and 4 rate categories using the program Seq-Gen [194]. Thus, we created a total of 24 datasets, each with 100 gene-tree/species-tree pairs, and corresponding to a fixed rate of duplication, transfer, and loss (low-DTL, medium-DTL, and high-DTL), a fixed mutation rate (rate-1, rate-3, rate-5, and rate-10), and a fixed alignment length (length-173, length-333).

As part of our study, we applied several gene tree reconstruction methods, including RAxML [230], NOTUNG [55], TreeFix [276], MowgliNNI [172], AnGST [37], and our new method TreeFix-DTL to a variety of simulated datasets and evaluated the accuracy of the reconstructed gene trees. RAxML is a well-known method used for reconstructing a maximum likelihood tree from sequence data and represents the baseline method for gene tree reconstruction. NOTUNG and TreeFix are two of the best performing methods for gene tree reconstruction in eukaryotes (both are species tree aware), and MowgliNNI, AnGST, and TreeFix-DTL are the three species tree aware methods for prokaryotic gene tree reconstruction considered in this study. Details on the exact settings used for these phylogenetic programs are provided in Section B.4.
Table 3.1: Accuracy of the different programs at reconstructing gene trees

<table>
<thead>
<tr>
<th>program</th>
<th>RF distance</th>
<th>topological accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAxML</td>
<td>0.097</td>
<td>3.04</td>
</tr>
<tr>
<td>NOTUNG</td>
<td>0.088</td>
<td>13.08</td>
</tr>
<tr>
<td>TreeFix</td>
<td>0.079</td>
<td>10.29</td>
</tr>
<tr>
<td>MowgliNNI</td>
<td>0.039</td>
<td>22.17</td>
</tr>
<tr>
<td>AnGST</td>
<td>0.032</td>
<td>29.08</td>
</tr>
<tr>
<td>TreeFix-DTL</td>
<td>0.028</td>
<td>38.21</td>
</tr>
</tbody>
</table>

- For each program, results are averaged over all 24 simulated datasets of 50 taxa.
- The normalized RF distance between the reconstructed gene trees and the true gene trees.
- The percentage of reconstructed gene trees that have (unrooted) topology identical to the true gene trees.

Reconstruction accuracy

We evaluated the different methods on our simulated datasets (24 datasets, each with 100 gene-tree/species-tree pairs) and measured the accuracy of the reconstructed gene trees against the true gene tree topologies. We quantified topological error in terms of the widely used normalized Robinson-Foulds (RF) distance, which captures the fraction of splits in the two trees that are present in only one of the two trees [204]. For example, a normalized RF distance of 0.05 between two trees implies that 5% of the splits in the reconstructed tree are not present in the true tree and, consequently, that 5% of the splits from the true tree do not appear in the reconstructed tree. Thus, the value of the normalized RF distance always lies between 0 and 1, and the closer the value is to 0, the more accurate the reconstruction. We found that TreeFix-DTL produced highly accurate gene trees over a range of evolutionary scenarios (Figures 3.2 and B.1, Table 3.1). Specifically, we observed the following:

- RAxML trees yield high error rates: The maximum likelihood trees produced using sequence-only methods tend to have high error rates, with the normalized RF distance varying from a minimum of 0.06 (i.e., an error rate of 6%) on the datasets with sequence length 333 and mutation rates 1 and 3 (substitutions per site), to a maximum of 0.17 on the datasets with sequence length 173 and a mutation rate of 10. As expected, error rates are consistently higher for datasets with high rates of sequence evolution (mutation rates 5 and 10) as well as for those with shorter sequence length. Biologists often collapse poorly supported branches in the inferred maximum likelihood trees; however, we found that if we collapse the RAxML tree branches with low bootstrap support before computing the normalized RF distances, the error rate actually tends to increase. For example, at the commonly used threshold of 80%,
Figure 3.2: **Reconstruction error of the different methods on simulated datasets of 50 taxa.** Error rates in terms of the normalized Robinson-Foulds (RF) distance are shown for gene trees reconstructed using RAxML, NOTUNG, TreeFix, MowgliNNI, AnGST, and TreeFix-DTL on the simulated datasets of 50 taxa. TreeFix-DTL produces the most accurate gene trees among all tested programs. (A) Results for the datasets with a low rate of duplication, transfer, and loss, a sequence length of 333 amino acids, and with varying mutation rates. (B) Results for datasets with a mutation rate of 1, a sequence length of 333 amino acids, and with varying rates of duplication, transfer, and loss. Detailed results for all 24 simulated datasets are available in Figure B.1.

we observed that the collapsed RAxML trees had an average normalized RF distance of 0.14, compared to 0.097 for the original (uncollapsed) RAxML trees.

- **NOTUNG and TreeFix are ineffective:** Eukaryotic gene tree reconstruction methods are ineffective at reconstructing prokaryotic gene trees, with average normalized RF distance over all datasets of 0.088 (NOTUNG) and 0.079 (TreeFix). At low rates of duplication, transfer, and loss, NOTUNG and TreeFix do improve on the error rate of the RAxML trees by approximately 30–40%, but for higher rates, they often performed worse than RAxML. Thus, species tree aware methods developed for eukaryotes simply do not work well for prokaryotes.

- **AnGST outperforms MowgliNNI:** Both AnGST and MowgliNNI improve significantly on the accuracy of the RAxML trees, with AnGST outperforming MowgliNNI on 23 of the 24 datasets. This is in spite of the fact that MowgliNNI was run on simulated species trees with perfect

3.3 Benchmarks
dating information while AnGST ignored all dating information. The average normalized RF distance over all datasets are 0.032 (AnGST) and 0.039 (MowgliNNI), suggesting that, in general, AnGST is a more effective method than MowgliNNI.

- **TreeFix-DTL reconstructs the most accurate gene trees**: TreeFix-DTL outperforms all other methods at accurately reconstructing prokaryotic gene trees and decreases the error rate of RAxML trees by an average of 71.1%. The normalized RF distance of the TreeFix-DTL trees varies from 0.006 (on the datasets with sequence length 333 and mutation rates 1 and 3) to 0.074 (on the datasets with sequence length 173 and a mutation rate of 10), and the average normalized RF distance over all datasets is 0.028. Furthermore, TreeFix-DTL produced gene tree topologies completely identical to the true gene trees (i.e., with a normalized RF distance of zero) 38.2% of the time (averaged over all datasets), compared with 3.0% for RAxML and 29.1% for AnGST.

In summary, our analysis demonstrates the ineffectiveness of RAxML, and of methods developed for eukaryotic gene tree reconstruction, at accurately inferring prokaryotic gene trees. We found great improvement in accuracy by using gene tree reconstruction methods designed for prokaryotes, and TreeFix-DTL is the most accurate method overall, producing gene trees that have, on average, 71.1% fewer topological errors than RAxML and over 10-fold higher rates of reconstructing the gene tree topology perfectly. TreeFix-DTL also has 12.5% fewer topological errors and 31.8% higher rates of reconstructing perfect gene tree topologies compared to AnGST. Importantly, neither TreeFix-DTL nor AnGST require species trees to be dated, while MowgliNNI does; yet both TreeFix-DTL and AnGST significantly outperform MowgliNNI even when MowgliNNI is provided with perfect species tree dating.

**Scalability and speed**

We also evaluated the performance of TreeFix-DTL on larger 100- and 200-taxon datasets and observed that the error rates of the TreeFix-DTL trees was generally similar to those observed on the corresponding 50-taxon datasets (Section B.5, Figure B.3), suggesting that the performance of the method does not deteriorate as the number of taxa in the input trees increases. Furthermore, irrespective of the dataset size, TreeFix-DTL requires only about three times the time required to build the RAxML trees themselves (Table B.1).

**Robustness to event costs**

To use the DTL-reconciliation model, one must assign costs for the different evolutionary events. Based on existing literature [37], we chose costs of 2, 3, and 1 for duplication, transfer, and loss.
respectively. In general, however, it is hard to know which cost assignment will yield the most accurate reconciliation on the particular dataset being analyzed. To assess the impact of using different costs on the performance of prokaryotic gene tree reconstruction, we ran TreeFix-DTL on the 50-taxon datasets from the basic simulation setup with all three event costs set to 1 (the simplest and most unbiased setting of event costs possible). We observed that the performance of TreeFix-DTL was only slightly affected by using these alternative event costs (Figure B.2), thus demonstrating the robustness of the gene tree reconstruction method to the actual event costs used for DTL-reconciliation. Therefore, even with its default cost assignment, TreeFix-DTL can be expected to be accurate at reconstructing prokaryotic gene trees across a wide range of datasets.

Robustness to inaccurate species tree topologies

TreeFix-DTL uses a known species tree topology to guide the reconstruction of the gene tree; however, in some cases, the species tree topology may not be known with certainty and the given species tree may contain topological errors. To assess the impact of species tree errors on the reconstruction accuracy of TreeFix-DTL, we ran TreeFix-DTL on the 50-taxon gene trees from the basic experimental setup but reconciled to species trees that had been corrupted with one and three random NNI operations. Species trees with just one NNI represent cases in which the species tree is only slightly incorrect while species trees with 3 NNIs represent cases in which there is a fair amount of topological error in the species tree. We found that TreeFix-DTL is robust to species tree errors and still substantially improves on RAxML gene trees despite species tree topological inaccuracies (Table 3.2). For example, for the datasets in which the species trees had one [three] NNI error(s), the average error rate of the TreeFix-DTL trees was 61.3% [40.0%] smaller than that of the RAxML trees.

Additional simulated datasets

We also evaluated the performance of TreeFix-DTL on additional datasets with different and higher rates of duplication, transfer and loss, shorter alignments, and branch-specific mutation rate variation and found that TreeFix-DTL performed remarkably well in all cases (Section B.6, Figure B.4). Overall, our results demonstrate that TreeFix-DTL should yield substantially more accurate prokaryotic gene trees compared to the best sequence-only methods irrespective of the underlying characteristics of the dataset being analyzed.

3.3 Benchmarks
### Impact of species tree error on gene tree reconstruction

<table>
<thead>
<tr>
<th>Program</th>
<th>Species Tree</th>
<th>RF Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAxML</td>
<td>none (sequence-only)</td>
<td>0.088</td>
</tr>
<tr>
<td>AnGST</td>
<td>true</td>
<td>0.028</td>
</tr>
<tr>
<td>TreeFix-DTL</td>
<td>true</td>
<td>0.027</td>
</tr>
<tr>
<td>AnGST 1 NNI</td>
<td></td>
<td>0.036</td>
</tr>
<tr>
<td>AnGST 3 NNIs</td>
<td></td>
<td>0.054</td>
</tr>
<tr>
<td>TreeFix-DTL 1 NNI</td>
<td></td>
<td>0.034</td>
</tr>
<tr>
<td>TreeFix-DTL 3 NNIs</td>
<td></td>
<td>0.053</td>
</tr>
</tbody>
</table>

*Gene trees were reconstructed using both the true species tree and topologically incorrect species trees (with 1 NNI and 3 NNI operations). These reconstructions were performed for the 50-taxon dataset with medium-DTL, mutation rates 1 and 5, and sequence lengths 173 and 333.*

*Error rates are averaged over the four datasets, each with 100 gene-tree/species-tree pairs.*

### Impact of gene tree accuracy on inferring duplications, transfers, and losses

Here, we demonstrate the impact of using more accurate gene trees on the ability to correctly infer duplication, transfer, and loss events. For the gene trees from the 50-taxon simulated datasets, reconstructed using RAxML, AnGST, and TreeFix-DTL, we applied DTL-reconciliation using RANGER-DTL [12] with default parameters. We then compared the inferred reconciliations to the true evolutionary histories, looking at both inferred event counts (Figure 3.3) and the accuracy of inferred events (Figure 3.3).

First, while applying DTL-reconciliation on the true gene trees does not recover the true evolutionary history with 100% accuracy, DTL-reconciliation is still highly accurate at inferring duplications, transfers, and losses. Specifically, applying DTL-reconciliation on the true gene tree topologies yielded an average sensitivity and precision, respectively, of 97.9% and 93.8% for duplications, 91.6% and 95.6% for transfers, and 91.9% and 84.6% for losses.

For events based on reconstructed gene trees, we found that inferences based on RAxML trees are misleading and those based on TreeFix-DTL trees are much more accurate. In particular, RAxML trees greatly overestimate transfers and losses, inferring roughly twice as many transfers and losses as TreeFix-DTL and the actual implanted counts; this is also reflected in the very low precision of RAxML + RANGER-DTL at recovering transfers and losses. Compared to RAxML trees, DTL-reconciliation on TreeFix-DTL trees yields better sensitivity for inferred duplications (89.9% vs 80.0%), transfers (72.6% vs 68.3%), and losses (86.5% vs 76.3%) and substantially better precision for inferred transfers (76.2% vs 38.4%) and losses (76.3% vs 31.0%) (precision of inferred duplications is comparable at 80.9% vs 80.3%). By comparing the counts inferred on the TreeFix-DTL trees to...
Figure 3.3: **Accuracy of inferred duplication, transfer, and loss events on simulated datasets of 50 taxa.** The precision and sensitivity of various phylogenetic programs at recovering duplication, transfer, and loss events are averaged over all gene tree-species tree pairs from all 24 simulated datasets of 50 taxa. While applying DTL-reconciliation to the true gene trees yields the highest precision and sensitivity, due to multiple optimal solutions, neither metric reaches 100%. Among all reconstructed gene trees, those from TreeFix-DTL yield the most accurate events.

the true counts, we observed that TreeFix-DTL has a tendency to slightly underestimate the number of transfers at the cost of slightly overestimating the number of duplications and losses; however, TreeFix-DTL shows the best overall performance at inferring duplications, transfers, and losses among all gene tree reconstruction programs.

### 3.4 Resolving gene tree-species tree conflict in 11 cyanobacteria genomes

To demonstrate the impact of accurate gene tree reconstruction in practice, we applied TreeFix-DTL to a set of 1128 protein-coding gene families from 11 completely sequenced cyanobacterial genomes [287]. Each of these gene families is single-copy and represents a gene shared by at least 9 of the 11 cyanobacterial genomes. Study of such single-copy, presumably orthologous, genes families plays an important role in determining species phylogenies, but analysis has revealed many competing results. For example, Lerat et al. [146] analyzed core families within 13 γ-Proteobacterial genomes and found that almost all are congruent to the species phylogeny, thereby claiming that these families were resistant to horizontal transfer. However, Bapteste et al. [14] later found that the sequences are compatible with many conflicting topologies and there was insufficient phylogenetic signal to either support or reject the claim from Lerat et al. [146]. In our analysis, we revisit some
previous claims on evolution within cyanobacteria [232, 287], in particular that transfer is frequent and that incomplete lineage sorting (ILS) plays a major role in shaping gene tree topologies. Given that all the gene trees in the dataset are single-copy, we make the standard assumption that any gene tree-species tree incongruence is a result of (additive) horizontal gene transfer (and compensating losses) and that duplications do not occur.

As previous analyses have relied on gene trees reconstructed through neighbor-joining (NJ), we first investigated whether a ML method resulted in substantially different reconstructions. We found that, while gene trees inferred by RAxML (using the PROTGAMMAJTT model) showed slight topological differences, total event counts inferred by RANGER-DTL [12] were reduced by only 4.4% (NJ: 2693 transfers, 1899 losses; RAxML: 2636 transfers, 1756 losses). Interestingly, while many NJ and ML gene trees were incongruent to the species phylogeny (NJ: 16 trees with normalized RF distance of 0, mean distance = 0.430; RAxML: 16 trees with normalized RF distance of 0, mean distance = 0.423) and thus imply many transfer events, TreeFix-DTL removed nearly all gene tree-species tree incongruence (856 trees with normalized RF distance of 0, mean distance = 0.065) and infers far fewer events (301 transfers, 573 losses). Thus, accounting for gene tree error reduced the number of inferred transfers and losses by 88.6% and 67.4%, respectively, compared to the RAxML trees. Our analysis not only shows that the number of transfer events on this dataset have been likely greatly overestimated using the NJ trees, but also identifies approximately 300 transfer events that are well-supported by sequence information.

Recently, Stolzer et al. [232] proposed a reconciliation model for explaining gene tree-species tree incongruence by postulating duplication, transfer, loss, and deep coalescence (also known as ILS), finding that accounting for ILS significantly reduced the number of inferred transfers. In contrast, by repeating their analysis using only Transfer-Loss (TL) and Transfer-Loss-ILS (TLI) models of reconciliation (with default event costs) applied to both NJ and TreeFix-DTL gene trees, we found that support for the presence of ILS in this dataset greatly diminishes (Figure 3.4). In detail, Stolzer et al. [232] considered TL and TLI models, as well as Duplication-Transfer-Loss (DTL) and Duplication-Transfer-Loss-ILS (DTLI) models, and filtered gene families (by removing any families that contained either temporally infeasible or conflicting multiple optimal solutions) to obtain a set of 314 families in which they compared inferred event counts. However, by considering only TL and TLI models and using the same filtering criterion, we obtain a set of 769 gene families, or 2.45× that considered by Stolzer et al. [232]. With this larger set of families, applying the TLI reconciliation model to the original NJ trees decreased the number of inferred transfers and losses by only 4.48% and 4.60%, respectively, over the TL model; this is a far smaller reduction than that reported by Stolzer et al. [232] (15–18% decrease in duplications + transfers, up to 20% decrease in losses).

543 Species-aware gene tree reconstruction with gene duplication, transfer, and loss: TreeFix-DTL
Figure 3.4: Cyanobacterial gene family evolution. Event counts are aggregated over 769 cyanobacterial gene families for which the reconciliation model of Stolzer et al. [232] did not infer temporally infeasible or conflicting degenerate solutions for any combination of gene tree program and reconciliation model. Applying TreeFix-DTL to the gene families substantially reduces the number of inferred events and the incidence of incomplete lineage sorting, as inferred under the TLI reconciliation model [232].

With the more accurate TreeFix-DTL gene trees, the similarity between inferred events using the two models is even more dramatic, with the TLI reconciliation model decreasing the inferred number of transfers and losses by only 1.52% and 0.94%, respectively, over the TL model. Altogether, these results suggest that many ILS events can be accounted for by gene tree error, and that ILS may play a much smaller role in cyanobacterial gene family evolution than previously reported.

Finally, we reanalyzed “highways” of transfer events [13, 232, 287] that have been previously inferred within cyanobacteria, where a highway is defined as an undirected horizontal transfer edge between two species for which an unusually large number of transfers have occurred relative to other horizontal transfer edges. Notably, as we previously mentioned, applying TreeFix-DTL dramatically decreased the number of inferred transfers; thus, even if highways still exist, their “magnitude” is far smaller. As in Stolzer et al. [232], we predict highways based on the assumption that transfer counts between pairs of species are normally distributed, and highways exist when these counts exceed 5 standard deviations (sd) above the mean. We note that Stolzer et al. [232] considered a threshold of 1.5 sd above the mean, but our analysis revealed that our higher threshold more often clearly distinguished edges with abnormally high counts. Using all three reconciliation programs, inferred transfers using NJ and RAxML gene trees corroborated previously found highways between Prochlorococcus and Synechococcus. Our higher threshold for highway detection also resulted in this being the only inferred highway across all programs, thus removing disagreements in inferred highways between the TL and TLI models of reconciliation and again suggesting that ILS may not play a role in bacterial gene tree-species tree incongruence. In the filtered set of families, TreeFix-DTL still nominally supports this highway and further infers a highway between Gloeobacter and...
the ancestor of *Synechococcus*, but the counts are so reduced (21 and 20 events) that the presence of highways is called into question.

In conclusion, we find that accounting for gene tree error has a dramatic impact of the amount of inferred transfer events within cyanobacteria. Furthermore, it reduces the effect of applying a reconciliation model that incorporates ILS and reduces the number of transfers along highways, suggesting that the existence of ILS and highways of transfers within the evolution of these genomes is not well supported by existing data.

3.5 Discussion

In this chapter, we have shed light on the complications of gene tree reconstruction in prokaryotes and proposed effective solutions for the problem. Our extensive experimental evaluation shows that our new method, TreeFix-DTL, outperforms all existing methods at reconstructing prokaryotic gene trees accurately. Our experimental results also strongly suggest that the adoption of TreeFix-DTL would result in substantial improvements in the accuracy of inferring gene trees and in our ability to understand the role of evolutionary events such as duplication, transfer and loss in gene family evolution.

The accuracy and scalability of TreeFix-DTL can be further improved by making the local search step more efficient. More efficient search techniques or faster algorithms for local search would make it possible to efficiently handle even larger input instances and to execute a larger number of search steps to further improve the reconstruction accuracy. Currently, if TreeFix-DTL encounters multiple gene trees with statistically equivalent likelihood and with the same minimum reconciliation cost, it outputs the gene tree with the highest likelihood score. In the future, it might be instructive to search for and report all gene trees with statistically equivalent likelihood scores and minimum reconciliation costs, and to study the similarities and differences in the alternative gene tree topologies.

TreeFix-DTL assumes that among all statistically equivalent candidates for the gene tree (in terms of likelihood score), the one with minimum reconciliation cost is most likely to be the true tree. However, the true evolutionary history of a gene family need not always be the most parsimonious one. It may therefore help, in some cases, to relax this assumption, and it would be interesting to develop such an approach further, possibly by incorporating probabilistic models of DTL-reconciliation \[253\] that would allow for integrating over all possible reconciliations for a given gene tree.

Species tree aware methods for gene tree reconstruction require that the species tree topology be given as input. As we have demonstrated, TreeFix-DTL is fairly robust to inaccuracies in the given species tree topology. However, to explicitly handle those cases in which it is hard to derive the
species tree topology accurately, it would be useful to develop approaches for simultaneous inference of gene and species tree topologies; such methods have been developed, for instance, in the eukaryotic setting for dealing with the effects of duplications and losses Boussau:GR:2013:phyldog or coalescent processes [111].

In our analysis, we observed that AnGST performed quite well on our test datasets. Thus, the simple approach used by AnGST may be worth studying and refining further. One of the difficulties with using AnGST is that users must decide on the number of bootstrap replicates to use as input. This number has a direct impact on the accuracy of the reconstructed gene tree, as using too few bootstraps reduces the search space, but using too many leads to over-fitting to the species tree. We were able to optimize the number of bootstrap replicates used in our simulation study by comparing the reconstructed gene tree against the known gene tree topology, but this is not possible to do with real datasets. It may be possible to develop effective techniques for estimating the ideal number of replicates to be used with any given dataset. Similarly, the current implementation of AnGST uses inefficient algorithms and is therefore not scalable much further beyond 200 taxa. By incorporating recent algorithmic advances [12] into the AnGST framework, it can be made to work efficiently even on trees with thousands of taxa. The AnGST approach could thus be useful for reconstructing gene trees when the gene tree or species tree is so large that TreeFix-DTL becomes prohibitively slow.

Furthermore, while transfer is often reported in prokaryotes, horizontally transferred genes have recently been identified in yeast [101], plants [18, 203], fungi [210], and other eukaryotes [4, 79, 129], and from bacteria to eukaryotes [48] and animals to bacteria [273]. Thus, TreeFix-DTL can aid in gene tree reconstruction within these genomes, as well, likely impacting our understanding of genome evolution across the tree of life.

The fact that accurate gene tree reconstruction has traditionally been difficult in prokaryotes has limited the use of the powerful applications of gene trees for studying prokaryotic evolution. Our work helps open the door to more extensive use of gene trees in evolutionary analyses in prokaryotes and other genomes that have experienced horizontal gene transfer.

3.5 Discussion

57
Species-aware gene tree reconstruction with gene duplication, transfer, and loss: TreeFix-DTL
Chapter 4

Unified reconciliation with gene duplication, loss, and deep coalescence: DLCpar

Accurate gene tree-species tree reconciliation is fundamental to inferring the evolutionary history of a gene family. However, although population-related effects such as incomplete lineage sorting (ILS) can dramatically affect the gene tree, many of the most popular reconciliation methods consider discordance only due to gene duplication and loss (and sometimes horizontal gene transfer). Methods that do model ILS are either highly parameterized or consider a restricted set of histories, thus limiting their applicability and accuracy. To address these challenges, we present a novel algorithm DLCpar for inferring a most parsimonious (MP) history of a gene family in the presence of duplications, losses, and ILS. Our algorithm relies on a new reconciliation structure, the labeled coalescent tree (LCT), that simultaneously describes coalescent and duplication-loss history. We show that the LCT representation enables an exhaustive and efficient search over the space of reconciliations, and, for most gene families, the least common ancestor (LCA) mapping is an optimal solution for the species mapping between the gene tree and species tree in a MP LCT. Applying our algorithm to a variety of clades, including flies, fungi, and primates, as well as to simulated phylogenies, we achieve high accuracy, comparable to sophisticated probabilistic reconciliation methods, at reduced runtime and with far fewer parameters. These properties enable inference of complex evolution of gene families across a broad range of species and large datasets.

The work presented in this chapter has led to the following publication: Wu YC, Rasmussen MD¹, Bansal MS¹, Kellis M. Most parsimonious reconciliation in the presence of gene duplication, loss, and deep coalescence using labeled coalescent trees. Genome Research, in press. (¹ Equal contribution)
Figure 4.1: The three-tree model and the labeled coalescent tree. (A) In the duplication-loss model, incongruence between the gene tree (black) and species tree (blue) can be explained using gene duplications (yellow star) and gene losses (red “x”). (B) In a multispecies coalescent model, incongruence between the gene tree and species tree can be explained due to incomplete lineage sorting (ILS). Because no duplications or losses are allowed, this model is inapplicable to gene families in which multiple gene copies exist in at least one species. (C) The unified model proposed by Rasmussen and Kellis [201] combines the multispecies coalescent and duplication-loss models. In this example, a duplication occurs in one chromosome (note the duplicate’s frequency is initially $p = 1/(2N)$, where $N$ is the effective population size, assuming a diploid genome) and creates a new locus, “locus 2”, in the genome. At locus 2, the Wright-Fisher model dictates how the frequency $p$ of the daughter duplicate (black dots) competes with the null allele (white dots) until it eventually fixates ($p = 1$). A gene tree is a “traceback” in this combined process. Note that the red and yellow trees form an intermediate locus tree (distinct from the gene tree and species tree) that describes how loci are created and destroyed. In this example, the gene tree has the same topology as that in A, but incongruence with the species tree is explained by duplication and deep coalescence. (D) The LCT combines the species tree, locus tree, gene tree, and reconciliations between them into a single structure. Each node of the gene tree is labeled with the species and locus to which it belongs, and gene tree nodes within the same species and locus are totally ordered in time. [Parts of this figure have been adapted with permission from Rasmussen and Kellis [201].]

4.1 Gene tree-species tree reconciliation

In eukaryotic organisms at sufficiently large evolutionary distances, discordance (topological differences) between the gene tree and species tree typically arises due to duplication and loss events (Figure 4.1A). However, at smaller evolutionary distances, an evolutionary phenomenon known as deep coalescence or incomplete lineage sorting (ILS) [159, 262], in which polymorphisms survive several rapid speciations then eventually fix or go extinct in a pattern incongruent to the species tree, can have a potentially prominent effect (Figure 4.1B). The simplest approach to resolving discordance is to seek a most parsimonious reconciliation (MPR) [82, 159, 181], that is, one that minimizes the number of inferred events. It has been shown that the least common ancestor (LCA) mapping solves the MPR problem when minimizing duplications only [86], duplications and losses [86], and deep coalescence [275]. However, these methods do not jointly model duplication, loss and deep coalescence, thus limiting their applicability and accuracy.
4.1.1 Existing work

To address this problem, more sophisticated methods accounting for incongruence due to duplication, loss, and deep coalescence have been proposed. For example, NOTUNG [55, 259] allows users to reconcile a gene tree with a non-binary species tree, with the optimal reconciliation minimizing the duplication-loss cost while allowing for possible deep coalescence at unresolved nodes in the species tree. While this parsimony framework is simple, requiring only a known species tree topology and (user-specified) costs for each evolutionary event, it cannot capture all possible evolutionary histories.

More recently, a three-tree model called DLCoal was proposed that introduced an intermediate locus tree describing how new loci are created and destroyed (Figure 4.1C) [201]. In the model, the reconciliation between the gene tree (or rather the coalescent tree) and the locus tree describes the history of coalescent events, including the presence of ILS, and the reconciliation between the locus tree and the species tree describes the history of duplication and loss events. The associated reconciliation algorithm DLCoalRecon seeks the maximum a posteriori reconciliation and shows dramatic improvement compared to ILS-unaware approaches. However, DLCoalRecon suffers from several drawbacks in practice: (1) it is highly parameterized, requiring divergence times, duplication and loss rates, population sizes, and generation times, all of which are difficult to estimate accurately, and (2) it relies on hill-climbing to search the space of locus trees and reconciliations and therefore may miss the optimal solution due to its limited search. These drawbacks limit the overall applicability, accuracy, and scalability of DLCoalRecon.

4.1.2 Contributions

To address these shortcomings, we present a number of contributions to the reconciliation framework:

- We introduce the concept of a labeled coalescent tree (LCT), which simultaneously describes the species tree, locus tree, and coalescent tree, and the reconciliations between them (Figure 4.1D). In the LCT, each node of the gene tree is labeled with both the species and locus to which it belongs, and gene tree nodes within the same species and locus are totally ordered in time. The LCT effectively combines the three-tree model into a single representation in which we can efficiently search over the space of locus trees and reconciliations.

- We present an algorithm DLCpar for inferring a most parsimonious (MP) LCT, that is, one that minimizes the total cost of inferred duplications, losses, and deep coalescence. This program is available at http://compbio.mit.edu/dlcpar/. As part of the development of DLCpar, we show that, assuming LCA reconciliation between the locus tree and species tree, the LCA mapping is an optimal solution for the species mapping between the gene tree and
species tree in a MP LCT.

We have applied our DLCpar algorithm to both simulated and real datasets and find that it dramatically improves reconciliations compared to ILS-unaware approaches. Furthermore, its accuracy is comparable to DLCoalRecon while requiring far fewer parameters and having a faster runtime.

4.2 A unified model for gene family evolution

In the reconciliation problem, we are given a gene tree and species tree (either inferred using existing phylogenetic algorithms or assumed known) and a leaf mapping that, for each extant gene, defines the extant species from which it was sampled. Note that both trees are full, rooted, and binary, and the leaf mapping indicates only the species, not the locus, to which each extant gene belongs. Our goal is to infer the evolutionary history of the gene family. We make the following assumptions:

1. Any incongruence between the gene tree and species tree topologies can be explained through duplication, loss, and incomplete lineage sorting. Each duplication creates a unique new locus that is unlinked with the original locus, and there is no gene conversion between duplicated loci.

2. Duplication and loss events do not undergo hemiplasy [11]; that is, they do not fix differently in descendant species. Equivalently, all duplications and losses either always go extinct or never go extinct in all descendant lineages. This allows separation of the duplication-loss process from the coalescent process that affect gene family evolution.

3. Each extant species is represented by a single haploid sample; that is, within each gene family, multiple genes from the same extant species are sampled from multiple loci in a single individual (as opposed to being sampled from the same locus across multiple individuals).

Assumption 1 is applicable to evolution within eukaryotic species, and assumption 2 was shown to affect only a small number of gene trees in simulation with biologically realistic parameters [201]. A relaxation of assumption 3 is presented in the discussion.

To motivate the need for models of complex evolutionary histories in which duplication, loss, and ILS can occur, consider the gene history illustrated in Figure 4.1C. Without duplications and losses, a multispecies coalescent process [44, 184, 196, 209, 239] would be sufficient to model the ancestry of genes \(a_1, b_1,\) and \(c_1.\) However, in this example, a duplication event occurs along the branch ancestral to species \(B\) and \(C,\) and this duplicate eventually fixes so that it is present in all descendant extant species, resulting in genes \(b_2\) and \(c_2.\) In the locus tree, such a duplication appears as a bifurcation in
which one child, the mother, continues to evolve at the original locus ("locus 1") and the other child, the daughter, evolves at a new locus ("locus 2"). However, molecular sequences evolve along the gene tree, which is affected by duplications, losses, and ILS. Continuing our example, the duplication has created an additional lineage within the locus 1 tree that must coalesce. Because there exist multiple chromosomes within the population, there likely exists a delay between the duplication event and the time at which the lineage with the duplicate coalesces with another lineage in locus 1. (This is analogous to the delay between a speciation event and the coalescent time of lineages from different species in a coalescent-only model.) In the gene tree, the duplication therefore appears as a locus “change” along a gene tree branch. If ILS occurs, two or more gene lineages may not coalesce at their first opportunity. In the example, despite the duplication event occurring along the branch ancestral to species B and C, the gene lineage with the duplicate does not coalesce with another lineage in locus 1 until after (looking backwards in time) the speciation event that differentiated species A from B and C; this has caused incongruence between the gene tree and locus tree.

Originally proposed by Rasmussen and Kellis [201], this unified three-tree model of gene tree-locus tree-species tree combines the multispecies coalescent and duplication-loss models for gene family evolution. Note that in addition to the gene tree, which represents the history of a set of gene sequences, and the species tree, which represents the history of a set of species, there exists a third tree, the locus tree. This locus tree shows the locus to which each sequence belonged at each point in its history and how these loci are related through duplication events. (While the reconciliation between the locus tree and species tree can additionally reveal “missing” locus tree branches due to loss events, such losses have no effect on the relationships between loci assigned to observable locus tree branches.) Importantly, under the three-tree model, only sequences within the same species and the same locus are allowed to coalesce. Therefore, the gene tree of this model can be viewed as a generalized coalescent tree, as it represents the history of gene sequences as they coalesce within the locus tree. As in the standard coalescent tree from coalescent-only models, internal nodes within the generalized coalescent tree capture coalescences; however, these coalescences can occur in a duplicate locus (for example, locus 2 in Figure 4.1C), and furthermore, branches are allowed to “change” loci (for example, from locus 1 to locus 2 in Figure 4.1C) due to duplication events. For simplicity, throughout the remainder of this manuscript, we will simply use coalescent tree to refer to the generalized coalescent tree. The problem with the term gene tree is that it is used in very different ways in the two underlying models: the gene branches present in one time slice in a species branch in the coalescent model (Figure 4.1B) represent the chromosomes that are ancestral to the extant sequences; however, the same branches in the duplication-loss model (Figure 4.1A) represent the loci that exist within the ancestral genome at that time. Therefore, the “gene tree” from the

4.2 A unified model for gene family evolution
coalescent model corresponds to our coalescent tree, which evolves “inside” the locus tree, and the “gene tree” from the duplication-loss model corresponds to our locus tree, which evolves “inside” the species tree. Indeed, it is this two-step reconciliation that allows us to separate the multispecies coalescent and duplication-loss processes responsible for gene family evolution. Finally, within the three-tree model, molecular sequences evolve along the coalescent tree; however, to be consistent with current phylogenetic literature, throughout this manuscript, we refer to a tree reconstructed using phylogenetic algorithms as a gene tree, with the understanding that this is equivalent to the coalescent tree.

4.2.1 The labeled coalescent tree (LCT)

We now propose a new reconciliation structure, the labeled coalescent tree (LCT, Figure 4.1D), for modeling gene evolutionary histories. Here, we outline the basic structure of the LCT; a formal definition is provided in Section C.1. Given a gene tree, species tree, and leaf mapping, the LCT is specified by the following:

- a **species map** that labels each gene tree node with the species to which it belongs,
- a **locus map** that labels each gene tree node with the locus in which it evolves, and
- a **partial order** that defines a total order on gene tree nodes within the same species and locus.

The LCT implicitly models the locus tree topology and reconciliations between the species, locus, and gene trees; these are precisely the minimal information necessary for inferring evolutionary events. Note also that the species map effectively decomposes the gene tree into disjoint subtrees, one or more of which evolve within each species tree branch. Let a speciation node be any gene tree node that exists at a speciation event, that is, a node found at the “bottom” or “top” of a species tree branch. Some speciation nodes may be hidden in the gene tree due to gene loss or deep coalescence; we call such nodes “implied speciation nodes” and include them in the LCT.

The LCT allows for three sources of incongruence between the gene tree and species tree (Figure 4.2):

- **Duplication** The locus of a gene tree node differs from the locus of its parent node (Figure 4.2A).
- **Loss** A locus that exists within a species is not found within the set of loci at the bottom of the species tree branch (Figure 4.2B). Note that we cannot infer when a locus is lost; thus, we assume that a locus is lost when it no longer exists within the species tree branch.
Figure 4.2: Duplications, losses, and deep coalescence. In A–C, evolutionary events are depicted using the unified model (left), the gene tree evolving “inside” the locus tree (middle), and the LCT (right). (A) Gene duplication was discussed in Figure 4.1. Evolution within a single species tree branch is shown. (B) Similar to duplication, a gene loss starts in one chromosome and drifts until it fixates or goes extinct. Evolution within a single species tree branch is shown. (C) Deep coalescence occurs when two lineages fail to coalesce before either can coalesce with a third lineage. This results in extra lineages at the top of a locus tree branch (top). Within a single species tree branch, multiple lineages can exist at the top of a locus tree branch created by a speciation (middle) or at the top of a locus tree branch created by a duplication (bottom). (D) Events are counted in the LCT of Figure 4.1D.

- **Deep coalescence** Deep coalescence is quantified by counting the number of implied extra lineages (Figure 4.2C, top). As new lineages are created by speciation and duplication events, we must address both these sources:
  
  - ILS at speciations: At a speciation, that is, at the top of a species tree branch, there exist multiple lineages within the same locus (Figure 4.2C, middle).
  
  - ILS at duplications: At a duplication, there exist multiple contemporary lineages that belong to the same locus as the parent locus of the duplication (Figure 4.2C, bottom).

Finally, note that the LCT is a simplified and condensed representation of the three-tree model of gene family evolution (for a comparison, see Section C.4).

### 4.3 The DLCpar algorithm

Using the LCT, we can now develop new methods for gene tree-species tree reconciliation. In this section, we assume that, in addition to the gene tree, species tree, and leaf mapping, we are given
positive costs for duplications, losses, and deep coalescence (extra lineages). Our goal is to infer a most parsimonious (MP) LCT, that is, a (not necessarily unique) LCT with minimum reconciliation cost subject to the condition that the reconciliation between the locus tree and species tree is the LCA mapping. Note that we have constrained the reconciliation between the locus tree and species tree in order to make our inference algorithm more efficient; we find that this assumption holds in the majority of gene trees in simulation with biologically realistic parameters.

We now present our algorithm DLCpar for inferring a MP LCT. Here, we outline the basics of our algorithm; technical details, including pseudo-code and proofs, are provided in Sections C.2 and C.3, respectively. DLCpar adopts a multi-stage approach in which we first infer an optimal species map then use this to infer an optimal locus map and order (Figure 4.3). Our method makes use of the structure of the LCT to search over all possible MP reconciliations and reuse computations.

4.3.1 Inferring an optimal species map

Our first step is to infer an optimal species map. As the leaf mapping of extant genes to extant species is known, our goal is to map each internal gene tree node to a species tree node. Under our assumptions, an optimal species map is the LCA mapping. We can prove this by contradiction: if the species map is not the LCA mapping, moving a gene tree node "down" (towards the leaves of) the species tree incurs a lower cost; thus, the species map cannot be optimal (for full details, see Theorem C.2.1). Once the species map has been determined, implied speciation nodes are added, and the speciation nodes are used to decompose the gene tree into disjoint subtrees that evolve within each species tree branch (Figure 4.3A).

4.3.2 Enumerating locus maps

To find an optimal locus map and order, note that knowledge of the loci at the speciation nodes would allow us to infer evolution within each species tree branch independently of one another. Therefore, rather than inferring the locus map and order for the entire gene tree at once, we consider the subproblem of determining species-specific locus maps and orders. In particular, for each species, a species-specific locus map is defined only on the gene tree nodes mapped to that species. By definition, a partial order for the LCT already separates gene tree nodes by their mapped species.

Our next task is to enumerate the set of species-specific locus maps for each species. We accomplish this through a pre-order traversal of the species tree. To start, we consider the (single) subtree (of the gene tree) that evolves within the root branch of the species tree. We label each gene tree branch within this subtree with a boolean variable that is true if the locus has changed along the branch and false otherwise. Next, we assign the root of the gene tree to an arbitrary
Figure 4.3: The DLCpar algorithm. In this example, we use equal costs for the evolutionary events. Furthermore, for illustrative purposes, the root of the gene tree has been extended so that a duplication may occur along the root branch; in general, this root branch is not necessary. (A) LCA mapping is used to map the gene tree (gray) within the species tree (blue), and implied speciation nodes (*) are added to gene tree branches that span multiple branches of the species tree. (B) Starting at the root branch of the species tree, DLCpar enumerates the locus maps and determines an optimal order and reconciliation cost for each. (In practice, some locus maps are not considered since they are never most parsimonious.) (C) The leaf loci are remapped to relative loci, and for each unique labeling of root loci and leaf loci, an optimal underlying locus map (and associated order) is selected. (D) This is repeated for all branches of the species tree in pre-order traversal, thereby enumerating all locus maps (along with associated optimal orders and reconciliation costs) for this gene tree and species tree. (E) For each species and each assignment of root loci and leaf loci, dynamic programming (DP) is used to determine the minimum total cost along all descendant species tree branches. The DP table is filled by post-order traversal of the species tree (arrows); see text for how these costs are computed. Colors indicate which leaf loci (circles) and which species-specific locus map (squares with colors corresponding to parts C and D) are used. At the root of the species tree, the optimal cost is selected (boxed), and traceback allows assignment of the loci for all speciation nodes (boxed). These can then be used to determine the species-specific locus maps and orders. (F) The inferred LCT is shown.

4.3 The DLCpar algorithm
locus and, for each combination of branch labels, recur down the subtree to determine a possible species-specific locus map. This yields the set of possible species-specific locus maps for the root species (Figure 4.3B). For species that are not the root of the species tree, a similar approach is applied except that multiple subtrees may exist within the species tree branch. Also, instead of assigning the root of the gene tree to an arbitrary locus, we consider all possible “root loci”, that is, all possible locus assignments for the gene tree nodes at the top of the species tree branch. Similarly, “leaf loci” are the locus assignments for the gene tree nodes at the bottom of a species tree branch. Because we have performed a pre-order traversal of the species tree, the set of possible root loci for a species is equal to the set of leaf loci for its parent species, and in turn, the set of leaf loci is determined by the species-specific locus maps.

4.3.3 Inferring and optimal order for each locus map

For each species-specific locus map, we must also determine an associated optimal order. One solution is to choose an order such that the implied duplications are as early in the species tree branch as possible. This is because the order only affects the number of extra lineages due to duplications, and such a selection minimizes this count (for full details, see Proposition C.2.3). In essence, pushing duplications towards the top of a species tree branch effectively enforces LCA reconciliation between the gene tree and locus tree.

4.3.4 Computing relative locus maps

At this point, the reconciliation cost for each species-specific locus map (and associated optimal order) can be computed, but there is one last step to perform within the species tree branch. As previously mentioned, we need only transmit the loci at speciation nodes from one species tree branch to the next. In addition, later in our algorithm, we require a method for comparing the locus assignments at speciation nodes across different locus maps. In particular, even though there are multiple ways of arriving at the same leaf loci, we are only interested in the relative assignments, that is, which nodes are mapped to the same locus and which to different loci. Therefore, we remap the leaf loci to a set of relative loci; this is accomplished by arbitrarily ordering the speciation nodes, assigning the first to an arbitrary locus, then proceeding along the speciation nodes and, based on the species-specific locus map, assigning each to an existing locus or to a new locus. Once this has been performed for each species-specific locus map, we retain, for each unique relative locus map, only its optimal underlying (absolute) locus map (Figure 4.3C). Finally, note that, to remove redundancy, the resulting relative leaf loci are used (instead of the absolute loci) when enumerating species-specific locus maps for the children species.
4.3.5 Inferring an optimal locus map and order

Once locus maps, orders, relative locus maps, and costs have been computed for all species tree branches (Figure 4.3D), we use dynamic programming (DP) to determine optimal relative locus assignments for the speciation nodes (Figure 4.3E). To accomplish this, we perform a post-order traversal of the species tree, and for each species tree branch and each assignment of (relative) root loci and (relative) leaf loci, use DP to determine the minimum total cost along all descendant species tree branches. The cost of assigning a particular leaf loci is either known (for extant species) or computed by adding the respective costs of assigning the same loci as root loci in the children species. The cost of assigning a particular root loci is computed by considering each possible leaf loci and adding the cost of the events within the species tree branch; the minimum among all possible choices is selected. At the root of the species tree, the optimal cost is selected, choosing randomly from among the optima if multiple optimal solutions exist. We then traceback (through a pre-order traversal of the species tree) to assign optimal root loci and leaf loci for each species.

Finally, because we have kept a mapping of relative locus maps to absolute locus maps, we can determine the optimal species-specific locus maps for each species, then look up the associated optimal order for each species-specific locus map. Combining the species-specific locus maps and orders yields an optimal locus map and order for the entire gene tree, and together with our previously inferred optimal species map, this constitutes the MP LCT (Figure 4.3F).

4.4 Benchmarks

4.4.1 Simulated datasets

To evaluate the performance of DLCpar, we used the simulated datasets and metrics of Rasmussen and Kellis [201] and compared DLCpar to the DLCoalRecon algorithm [201] and the usual most parsimonious reconciliation (MPR) algorithm [181]. To effectively model real data, these datasets used known species trees and parameters (divergence times, duplication and loss rates, population sizes, generation times) for two clades of 12 Drosophila species and 17 primates and other mammals (Figure 4.4A,B), then used the DLCoal model to simulate locus trees and gene trees. To reconcile the gene trees to the species trees, we ran DLCpar using the same event costs across all settings and ran DLCoalRecon using the true parameters used in the simulations (MPR requires no parameters).

For most simulation parameters, DLCpar performance either exceeds (fly dataset) or is comparable to (primate dataset) that of DLCoalRecon, with both of these programs showing substantial improvement over MPR (Figures 4.5 and C.4). As an example of the comparative performance of the various methods, for the Drosophila dataset with a duplication-loss rate of
0.0012 events/gene/myr (1× the estimated real rate), a generation time of 0.1 yr, and an effective population of 25 million, our 500 simulated gene trees contain 232 duplications, 216 losses, and 33,182 pairs of orthologous genes. By confusing ILS events as duplications with compensating losses, MPR infers 1241 duplications (80.2% sensitivity, 15.0% precision), 2495 losses (97.7%, 6.0%), and 21,413 ortholog pairs (64.5%, 100%), with 8.4% (locus tree) topological accuracy. In comparison, DLCpar infers far fewer events at increased accuracy, sensitivity, and precision, with 232 duplications (96.6%, 96.6%), 213 losses (98.1%, 99.5%), 33,192 ortholog pairs (99.98%, 99.95%), and 98.0% topological accuracy. Impressively, DLCpar also outperforms DLCoalRecon, with the latter inferring 242 duplications (90.5%, 86.7%), 216 losses (98.6%, 98.6%), 33,285 ortholog pairs (99.7%, 99.4%), with 96.0% topological accuracy. This is despite the advantages that DLCoalRecon has over DLCpar in this simulation study; in particular, the DLCoal model was used to generate the simulated gene trees, and DLCoalRecon both incorporates additional parameters and uses the true parameters used in the simulations.

We also find that with increasing ILS rate (due to increasing population size with a constant generation time), DLCpar performance remains relatively stable whereas DLCoalRecon performance decreases. The decreased performance of DLCoalRecon is likely partially attributable to its heuristic search strategy [201]: on the same simulated Drosophila dataset as above (1× duplication-loss rate, $g = 0.1$ yr, $N = 25$ million), when the search is initialized on the correct locus tree, DLCoalRecon infers 232 duplications (97.4% sensitivity, 97.4% precision), 213 losses (98.6%, 100%), and 33,196 ortholog pairs (100%, 99.96%), with 99.2% locus tree topological accuracy. While this performance is slightly higher than that of DLCpar, it further highlights another advantage of DLCpar over

Figure 4.4: Species and phylogenies used in evaluation. For our evaluation on simulated data, we used (A) 15 primates (plus two outgroup species) [222], (B) 12 Drosophila species [240], as well as simulated phylogenies. (C) For our evaluation on real data, we used 16 fungal species [26].

4 Unified reconciliation with gene duplication, loss, and deep coalescence: DLCpar
Figure 4.5: DLCpar performance on simulated fly and primate gene trees. DLCpar, DLCoalRecon, and MPR were used to reconcile simulated gene trees. Simulated datasets and DLCoalRecon and MPR results were obtained from Rasmussen and Kellis [201], and DLCpar was run with costs of $D = 1$, $L = 1$, $C = 0.5$. Duplications and losses were simulated at rates estimated from real data (flies: 0.0012 events/gene/myr, primates: 0.0017 events/gene/myr), generation times for model organisms were obtained from literature and assumed equal throughout the clade (flies: 0.1 yr, primates: 20 yr), a wide range of effective population sizes was used, and 500 gene trees were simulated per parameter setting. For the fly dataset, DLCpar shows substantial improvement over DLCoalRecon in both the precision of inferring duplications and losses (A,B) as well as the accuracy of reconstructing the locus tree topology (C). For the primate dataset, DLCpar and DLCoalRecon performance is comparable (D,E,F). Both ILS-aware methods dramatically outperform MPR.

4.4 Benchmarks
DLCpar complexity on simulated fly and primate gene trees. For each dataset, gene trees were divided into classes based on the number of extant genes (counts shown as bars). For each gene tree, we determined the total number of reconciliations considered by DLCpar (red), the number of reconciliations if only relative locus maps for speciation nodes are considered (green), and the number of computations to complete the DP table (blue). The mean and 95% confidence interval (mean ± 1.96 × standard error) is shown for each class. Also shown are the number of reconciliations considered by DLCoalRecon (red dots), which was fixed at 1000 search iterations. (DLCoalRecon utilizes a prescreening approach so that 20 × 10^3 locus trees are proposed but only 1000 of these are considered using the full probabilistic model.) Data is shown for flies with simulation parameters of 1× duplication-loss rate, \( g = 0.1 \) yr, \( N = 25 \) million and for primates with simulation parameters of 1× duplication-loss rate, \( g = 20 \) yr, \( N = 25 \) thousand. Note that the number of reconciliations considered by DLCpar increases with increasing gene tree size (red), redundancy in the reconciliation search space is dramatically reduced by considering relative locus maps at speciation nodes (green), and dynamic programming further increases efficiency by reusing subproblems (blue).

DLCoalRecon, namely that DLCpar searches over the entire space of reconciliations compared to the heuristic search approach of DLCoalRecon (Figure 4.6). While increasing the search space of DLCoalRecon could lead to performance increases, the total runtime of DLCoalRecon already far exceeds that of DLCpar; for the dataset above, DLCoalRecon took 5.6h compared to DLCpar at 6.3m, a slow down of 54.2×.

In addition to increased accuracy and reduced runtime, DLCpar holds a major advantage over DLCoalRecon in terms of applicability. In DLCoalRecon, the maximum a posteriori reconciliation is inferred, thus requiring knowledge of speciation times, duplication and loss rates, generation times, and population sizes. For this evaluation, DLCoalRecon used the true parameters used in
the simulation, but in practice, these parameters need to be estimated from genome-wide data and are often difficult, if not impossible, to obtain accurately. In contrast, DLCpar only requires the user to select costs for duplications, losses, and ILS. In this analysis, despite the differences in the number of implanted duplications and losses and varying ILS rate across the different species trees and simulation parameters, we used the same event costs for DLCpar throughout, yet DLCpar performance is still consistently high. Analysis using varying relative costs also shows that DLCpar performance is robust to the choice of these costs (Section C.6, Figure C.5). To demonstrate that DLCpar is applicable to a wide range of datasets, we also conducted additional analyses using simulated species trees and gene trees and find that, almost universally, DLCpar shows dramatic improvement over DLCoalRecon and MPR (Section C.8, Figure C.7).

Finally, we consider the performance of NOTUNG [55, 259], which allows for ILS by reconciling gene trees to non-binary species trees. This either requires species trees with well-studied polytomies or a procedure for collapsing species tree branches; a principled approach for the latter does not exist. Therefore, we ran NOTUNG using species tree topologies with a range of polytomies; these included collapsing the shortest branch or branches within the species tree, collapsing branches that have previously been found to be sources of ILS, and collapsing all branches to create a “star” phylogeny. (Note that the “star” phylogeny is the only one that allows for ILS along all branches of the species tree.) In all cases, we find that DLCpar performs dramatically better than NOTUNG (Figure 4.7). As expected, NOTUNG performs worse than DLCpar and DLCoalRecon but better than MPR in terms of duplication precision, and precision increases with the number of polytomies. That is, by reconciling gene trees to non-binary species trees, NOTUNG is able to correctly identify spurious duplications due to ILS. However, this comes at the cost of increasingly worse loss precision; in many cases, NOTUNG loss precision is worse than that of even MPR. This is likely because losses associated with a polytomy in the species tree can be assigned to multiple candidate gene tree edges depending on how the order of divergence within the polytomy is resolved; such ambiguities are not possible when both the gene tree and species tree are binary. We also find the duplication sensitivity of NOTUNG to be similar to that of MPR (that is, worse than that of DLCpar and DLCoalRecon) and the loss sensitivity of NOTUNG to be worse than that of MPR. These results highlight the shortcomings of NOTUNG. In particular, the NOTUNG model cannot capture all possible evolutionary histories because it does not explicitly model the locus history of the gene family, for example, through a locus tree. Because of this, NOTUNG cannot model deep coalescence of a duplicated lineage with its parent lineage. Furthermore, NOTUNG can only capture pairwise relationships present as bifurcations in the gene tree; this is despite the gene tree being an inaccurate representation of the duplication-loss history of the gene family due to the presence of ILS.
Figure 4.7: Phylogenetic accuracy of NOTUNG on simulated gene trees. In addition to DLCpar (black solid), DLCoalRecon (black dot), and MPR (black dash), NOTUNG (colored) was used to reconcile simulated gene trees. To infer ILS, NOTUNG reconciles binary gene trees to non-binary species trees; the species trees used in reconciliation are shown at right, with multifurcated branches highlighted in red. For the fly dataset, we collapsed the shortest branch (blue), which separates the *Sophophora* and *Drosophila* subgenera; the third shortest branch (green), which groups *D. erecta* and *D. yakuba* as sister species to *D. melanogaster* though large portions of their genomes support alternative phylogenies [189]; the three shortest branches (yellow); and all branches (red). For the primate dataset, we collapsed the third shortest branch (blue), which implies that the relationships between the outgroup species and primates are poorly resolved; the two shortest branches (green), which implies co-divergence of baboon, macaque, and vervet and of human, chimp and bonobo, and gorilla; and all branches (red). For details on the simulation procedure, see Figure 4.5.
Table 4.1: Evaluation on real fungal dataset

<table>
<thead>
<tr>
<th>phylo prog</th>
<th>recon prog</th>
<th>% orths</th>
<th># orths</th>
<th># dups</th>
<th># losses</th>
<th>DCS</th>
<th>% GC</th>
<th>phylo runtime</th>
<th>recon runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhyML</td>
<td>DLCpar</td>
<td>98.4</td>
<td>581,017</td>
<td>4,498</td>
<td>6,699</td>
<td>0.823</td>
<td>91.9</td>
<td>45.3 (20.0) s</td>
<td>74.4 (0.9) s</td>
</tr>
<tr>
<td>PhyML</td>
<td>DLCCoalRecon</td>
<td>97.8</td>
<td>575,374</td>
<td>4,533</td>
<td>6,398</td>
<td>0.883</td>
<td>86.5</td>
<td>45.3 (20.0) s</td>
<td>54.1 (52.2) s</td>
</tr>
<tr>
<td>PhyML</td>
<td>MPR</td>
<td>64.2</td>
<td>464,479</td>
<td>21,264</td>
<td>64,391</td>
<td>0.153</td>
<td>86.5</td>
<td>45.3 (20.0) s</td>
<td>–</td>
</tr>
<tr>
<td>TreeFix</td>
<td>DLCpar</td>
<td>99.1</td>
<td>590,113</td>
<td>4,535</td>
<td>5,535</td>
<td>0.899</td>
<td>100.0</td>
<td>25.7 (8.0) m</td>
<td>11.7 (0.8) s</td>
</tr>
<tr>
<td>TreeFix</td>
<td>DLCCoalRecon</td>
<td>99.0</td>
<td>583,490</td>
<td>4,472</td>
<td>5,378</td>
<td>0.927</td>
<td>100.0</td>
<td>25.7 (8.0) m</td>
<td>47.6 (48.6) s</td>
</tr>
<tr>
<td>TreeFix</td>
<td>MPR</td>
<td>96.4</td>
<td>574,946</td>
<td>6,062</td>
<td>10,981</td>
<td>0.649</td>
<td>97.3</td>
<td>25.7 (8.0) m</td>
<td>–</td>
</tr>
<tr>
<td>SPIMAP</td>
<td>–</td>
<td>96.5</td>
<td>557,981</td>
<td>5,407</td>
<td>10,384</td>
<td>0.650</td>
<td>83.8</td>
<td>21.9 (12.2) m</td>
<td>–</td>
</tr>
<tr>
<td>SYNERGY</td>
<td>–</td>
<td>99.2</td>
<td>595,289</td>
<td>4,604</td>
<td>8,179</td>
<td>0.692</td>
<td>2.7</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

a Several combinations of phylogenetic reconstruction (phylo) programs and reconciliation (recon) programs were evaluated. PhyML+DLCCoal, PhyML+MPR, SPIMAP, and SYNERGY results were obtained from Rasmussen and Kellis [201]. TreeFix+MPR results were obtained from Wu et al. [276], where TreeFix used as input RAXML trees (with 100 bootstraps) and long search parameters. PhyML+DLCpar, TreeFix+DLCpar, and TreeFix+DLCCoalRecon were evaluated using PhyML trees from Rasmussen and Kellis [201] or TreeFix (long) trees from Wu et al. [276] and running DLCpar with costs of $D = L = C = 1$ or DLCCoalRecon with parameters from Rasmussen and Kellis [201].

b Percentage of 183,374 syntenic orthologs recovered.
c Number of pairwise orthologs, duplications, and losses inferred across all gene trees.
d Average duplication consistency score. Scores range from 0 to 1, with a higher score indicating more consistent duplications.
e Percentage of 37 recent gene converted paralogs recovered.
f Average (median) runtime for reconstructing each gene tree. TreeFix runtimes include times for reconstructing initial RAXML trees. Note that depending on program parameterization, shorter runtimes may be possible to achieve similar performance. Since SYNERGY trees were downloaded, no runtime was estimated.
g Average (median) runtime for reconciling each gene tree. MPR runtimes are not included but on average took less than a second. Note that depending on program parameterization, shorter runtimes may be possible to achieve similar performance.

4.4.2 Biological dataset of 16 fungal genomes

To study whether our observed trends in simulated datasets translate to a real dataset, we also assessed the performance of DLCpar on a biological dataset of 5551 gene trees across 16 fungal genomes (Figure 4.4C) [26]; this dataset has been used to evaluate numerous phylogenetic algorithms [200, 201, 267, 276]. As the truth is not known for real data, we used several informative metrics to assess the quality of reconciliations (Table 4.1).

For comparison, we evaluated several combinations of phylogenetic (gene tree) reconstruction programs and reconciliation programs. Phylogenetic programs included some of the best performing methods; we considered sequence-only (PhyML [94]), hybrid (RAxML [230] + TreeFix [276]), and species-tree-aware (SPIMAP [200]) approaches, as well as approaches that integrated other sources of information such as synteny (SYNERGY [267]). Reconciliation programs included DLCpar, DLCCoalRecon, and MPR. However, both SPIMAP and SYNERGY incorporate their own reconciliation model into gene tree reconstruction, and these models are incompatible with an ILS-aware reconciliation model; therefore, we applied DLCpar and DLCCoalRecon only to PhyML and TreeFix gene trees.

The real dataset included some very large gene families: the largest gene family contains 178
extant genes for an average of ~11 genes per extant species, and another family contains 42 genes in an extant species. We therefore bounded the search space of DLCpar by prescreening reconciliations. For the 0.2% (PhyML: 10/5351, TreeFix: 9/5351) of families that were too complicated using the bounded search (memory requirements exceeded 4 GB), we used DLCpar with a hill-climbing search heuristic. Evaluation of these heuristic approaches on the simulated fly and primate genomes shows only minor changes in performance when using biologically realistic simulation parameters (Section C.6, Figure C.6).

First, we assessed the ability to recover syntenic orthologs (one-to-one homologs that are highly likely to be orthologous given their surrounding conserved gene order). We find that when applied to trees reconstructed using the same phylogenetic program, DLCpar recovers slightly more syntenic orthologs than DLCoalRecon (PhyML: 98.4% vs 97.8%, TreeFix: 99.1% vs 99.0%). Compared to reconciliations that do not take into account ILS, this is a dramatic improvement over sequence-only methods (64.2%) and higher than hybrid or species-tree-aware methods (96.4–96.5%). These high sensitivities are also accompanied by more inferred orthologs and fewer inferred duplications and losses. This is, of course, expected, as ILS results in spurious duplications and losses in the gene tree, which are then removed by the ILS-aware reconciliation methods. We find that DLCpar and DLCoalRecon infer similar numbers of orthologs (0.5–0.6% difference), duplications (0.4–0.7%), and losses (1.4–2.3%), and other than SYNERGY, which infers more orthologs and a comparable number of duplications, DLCpar and DLCoalRecon infer at least 16.1% fewer duplications and 35.5% fewer losses than other methods.

Next, we analyzed the duplication consistency score [260], which measures the plausibility of inferred duplications. For each duplication node, this score computes the percentage of species overlap in the two child subtrees; the assumption is that erroneous duplications are often followed by compensating losses and therefore yield a low score. We find that DLCoalRecon slightly outperforms DLCpar, as evidenced by its score distribution (Figure C.8). Compared to other methods, the ILS-aware methods have an average duplication consistency score at least 13.1% higher than the next best performing method (SYNERGY) under this metric.

For our last metric, we evaluated the ability to recover more recent duplications due to gene conversion events. When paired with a sequence-only method, DLCpar recovers slightly more recent gene-converted paralogs than DLCoalRecon (91.9% vs 86.5%), but more impressively, when paired with a hybrid method that accounts for gene tree errors due to statistical uncertainty, DLCpar and DLCoalRecon show 100% recovery. This suggests that even though the alignment is not passed to either DLCpar or DLCoalRecon, and despite neither algorithm modeling gene conversion, the gene tree topology still adequately constrains the underlying locus tree topology so that species tree
information does not override sequence evidence in the reconciliation.

In addition to studying reconstruction quality, we compared the runtime performance of DLCpar and DLCoalRecon. We find that while DLCpar has a longer runtime than DLCoalRecon for PhyML trees (74.4 sec vs 54.1 sec), its average runtime is highly skewed by large families. If we exclude the longest 0.25% of runtimes for either program, DLCpar runtime dramatically decreases (29.4 sec) while DLCoalRecon runtime is unaffected (53.2 sec). (For many gene families, DLCoalRecon runtime could likely be decreased by reducing the number of search iterations. However, reducing the search space yields lower accuracy, and there is no principled way of balancing this trade-off.) Additionally, for TreeFix trees, the average runtimes of DLCpar (DLCoalRecon) reduced by a factor of 6.4 (1.1) × to 11.7 (47.6) sec. Again, the average runtime for DLCpar is highly skewed by large families, as 90.0% of families ran in under one sec. The decreased runtime of DLCpar can likely be attributed to the smaller degree of gene tree-species tree incongruence for the TreeFix trees, which translates to smaller search spaces for the LCT. Finally, we find that the runtime for the full phylogenomic pipeline (including both phylogenetic reconstruction and reconciliation) are comparable for DLCpar (PhyML: 2.0 min, TreeFix: 25.9 min) and DLCoalRecon (PhyML: 1.7 min, TreeFix: 26.5 min). However, the total runtime for pipelines with TreeFix is dominated by the phylogenetic reconstruction and can be greatly reduced. Our TreeFix trees used long search parameters and were built on RAxML gene trees with 100 bootstraps to yield the highest accuracy, but using a smaller search space and no bootstraps would result in only a minor decrease in performance [200, 276] while requiring much less runtime.

4.5 Discussion

In this chapter, we have presented a novel algorithm DLCpar for inferring a most parsimonious gene evolutionary history in the presence of duplication, loss, and ILS. To develop our algorithm, we have also introduced the labeled coalescent tree (LCT) and shown it to be an efficient data structure for representing reconciliations. Our analysis of simulated and biological datasets shows that DLCpar achieves performance comparable to more sophisticated probabilistic reconciliation methods at a fraction of the runtime. In addition, with its parsimony approach, DLCpar is more applicable to a broad range of species and large datasets.

A core novelty of the LCT is that it labels gene tree branches with the species to which it belongs and the locus in which it evolves. This allows the LCT to simultaneously model the duplication-loss history and coalescent history of a gene family. In contrast, current reconciliation structures that map gene tree nodes to species tree nodes are only appropriate for capturing coalescent-only histories (in which case, the reconciliation traces co-existing chromosomes) or duplication-loss histories (in
which case, the reconciliation traces co-existing loci). The only existing reconciliation framework that accounts for duplication, loss, and ILS is the three-tree (DLCoal) model, which relies on a two-step reconciliation structure, with one (the reconciliation between the gene tree and locus tree) describing the coalescent history and the other (the reconciliation between the locus tree and species tree) describing the duplication-loss history. Thus, the LCT bridges the gap between previously disjoint reconciliation structures and allows us to simultaneously capture multiple sources of incongruence. Furthermore, while conceptually identical to the three-tree model, the LCT enables the development of an efficient inference algorithm based on dynamic programming.

Similarly, a major feature of DLCpar is its simple yet powerful model. DLCpar relies on the three-tree model of evolution and the assumption that the most parsimonious reconciliation is the most accurate. This model has a strong biological foundation and captures the most relevant phenomena responsible for eukaryotic evolution, and parsimony assumptions are used in the simplest, most popular reconciliation methods. Currently, only two other methods exist that simultaneously model duplications, losses, and ILS: NOTUNG and DLCoalRecon. However, NOTUNG, while simple, can only capture a subset of the possible reconciliations and therefore has limited accuracy, and DLCoalRecon, while powerful, is highly parameterized, making it difficult to use in many analyses.

Additionally, by using the LCT structure, DLCpar is able to search over the entire reconciliation space. While there exists previous work on exploring and summarizing the reconciliation space, they consider only duplications and losses [8, 49–51] or additionally horizontal gene transfer [218], or, if ILS is addressed, they model only a subset of the evolutionary histories that are possible in our model [259]. That is, this work presents the first approach for fully exploring the reconciliation space while accounting for duplications, losses, and ILS. As we have shown, this is a major advantage over the hill-climbing approach of DLCoalRecon, likely resulting in the latter’s lower performance in our simulations. Furthermore, DLCpar’s dynamic programming approach allows it to reuse computations across different reconciliations. In contrast, DLCoalRecon recomputes the posterior probability for every proposed reconciliation. In addition to the simpler task of computing a parsimony-based reconciliation cost rather than a probability, this likely accounts for the dramatic runtime improvement of DLCpar over DLCoalRecon.

Our results demonstrate that a parsimony approach is capable of distinguishing between gene tree-species tree incongruence due to duplication-loss and ILS. This is in spite of the fact that simple formulas exist for relating the number of duplication-loss events and the number of deep coalescence events [285], which, at first glance, might suggest that parsimony methods would always infer one event over the other (for further discussion, see Section C.5). Furthermore, while we have demonstrated that DLCpar performs well under a variety of species tree parameters,
more study is needed in the “anomaly zone” of species histories in which anomalous gene trees (AGTs) exist, that is, where the gene tree most likely to evolve under a coalescent model differs from the species tree topology [41, 43, 45]. Within an anomaly zone, it has been shown that species trees reconstructed using democratic vote [43], greedy consensus [42], maximum likelihood from concatenated sequences [140], and minimizing-deep-coalescence [243] methods are inconsistent, meaning that the estimated species tree topology is incorrect in the limit as the number of sampled gene trees goes to infinity. However, the risk of AGTs are rarely realized in practice [116] and a Bayesian approach for estimating species trees is consistent [150]. Altogether, this suggests that a more careful treatment of parsimonious reconciliations within this anomaly zone may be required, with particular focus on investigating whether the species tree and gene tree can mislead the locus tree.

In our simulations, we reconciled the simulated gene trees to the species trees, thus assuming that gene trees are accurate and fully resolved. However, in practice, gene tree reconstruction is complicated by the fact that many gene families lack enough information to confidently support a single gene tree topology. This is an important point as phylogenetic analyses depend strongly on the accuracy of reconstructed gene trees [96]. Here, we must distinguish between topological errors due to lack of phylogenetic signal and due to ILS. If we are interested in the duplication-loss (locus) history of a gene family (for example, if the goal is to infer duplications, losses, orthologs, and paralogs), then the tree of interest is the locus tree, not the gene tree. However, to infer the locus tree from molecular sequences involves inferring the gene tree from the sequences, then inferring the locus tree from the gene tree. Phylogenetic signal (or lack or thereof) affects the former, and ILS affects the latter. That is, though ILS often biases reconciliation-based inferences to yield false duplications due to stochastic variations in gene tree topology [288], there could be sufficient phylogenetic signal to accurately infer the gene tree topology, but this gene tree is still an inaccurate representation of the duplication-loss history due to the presence of ILS. Our analysis of the biological dataset demonstrates that while many ILS events can be accounted for by correcting for gene tree errors due to lack of phylogenetic signal, substantial improvement is achieved in locus tree accuracy by accounting for errors due to lack of phylogenetic signal and ILS. To address this need, we suggest a powerful phylogenetic pipeline for gene tree (or more accurately, locus tree) reconstruction and event and homolog inference: reconstruct gene trees using a ML method, correct for topological errors due to statistical uncertainty using TreeFix, and account for ILS using DLCpar. These methods are based on simple models and have few requirements: aside from some user-defined thresholds and costs (to which the algorithms are robust), we require only a sequence alignment and species tree topology, and these are precisely the minimal requirements for any phylogenetic pipeline. Compare
this to other methods that require parameters such as speciation times, duplication and loss rates, generation times, or population sizes, which must be estimated from genome-wide data and are often noisy, and even with the additional information, achieve at most comparable performance.

We have presented a method for reconciling binary gene trees to binary species tree, but in practice, inferring fully-resolved trees is a major issue due to lack of phylogenetic signal or conflicting sources of information. For example, for the PhyML (RAxML) gene trees reconstructed on our biological dataset, 1.4% (0.1%) of gene tree branches are unresolved (have zero branch lengths), and 15.7% (1.7%) of gene trees had at least one unresolved branch. While this might indicate a lack of gene tree polytomies, we also find that for the RAxML gene trees reconstructed on our biological dataset, 29.5% of gene tree branches are poorly supported (have bootstrap < 80%), and 84.3% of gene trees have at least one poorly supported branch. (No estimates for PhyML trees are available, as PhyML trees were reconstructed without bootstrapping.) Accounting for gene tree error resolves some polytomies (for example, TreeFix gene trees have 23.7% of gene tree branches and 55.8% of gene trees poorly supported), but, altogether, these results suggest a need for ILS-aware reconciliation methods capable of handling multifurcating gene trees. One possible approach is to resolve gene tree polytomies by expanding them and selecting the expansion with minimum reconciliation cost; such a method has already been presented for the duplication-loss problem [27, 55, 142]. Future work could also consider reconciliations with multifurcating species trees. NOTUNG [259] addresses this problem by allowing for ILS at unresolved nodes in the species tree, but as we have shown in our analysis on simulated datasets, this approach greatly diminishes loss sensitivity and precision. In addition, more complex models that differentiate between hard and soft polytomies, that is, polytomies that represent simultaneous divergence versus polytomies that replace a binary branching process that could not be fully resolved, may be required.

While we have shown that DLCpar is robust to the choice of reconciliation costs for the different evolutionary events, we suggest that users try multiple event costs, then either take a conservative approach and use only the intersection of the inferred orthologs and events, or alternatively, find the consensus tree among the inferred locus trees, and from this, infer orthologs and events. A possible future direction is to incorporate knowledge of species tree parameters to estimate these costs. For example, under the DLCoal model, short species tree branches should induce fewer duplications and losses and more deep coalescence, resulting in higher duplication and loss costs and a lower deep coalescence cost. Determining an equation that takes into account such species tree information is not straightforward, however, due to the complex relationship between duplications, losses, and deep coalescence; in particular, ILS increases with more duplications and decreases with more losses [201]. Another option is to use an iterative approach to incorporate the inferred counts of duplications,
losses, and extra lineages to assign event costs, but we wished to avoid such circular dependencies in our approach. We note also that using branch-specific costs would require us to search over species maps rather than assuming the LCA mapping.

We also envision the framework presented here to be useful in the future development of methods for understanding gene family evolution. For example, in addition to modeling phylogeny across genomes, the LCT could be extended to capture phylogeny across multiple individuals per genome. This has the benefit of incorporating both population genetic and phylogenomic data into a single analysis. There have also been recent developments on combining models of sequence evolution and duplication-loss [8, 54, 260]; incorporating substitution rate variation [1, 199, 200], synteny [267], or horizontal gene transfer [12, 37, 52, 254]; jointly inferring species trees and gene trees [22] or gene trees and sequence alignments [149]; reconstructing species trees [151]; and estimating population statistics [53, 97, 196]. However, so far, all of these methods rely on the multispecies coalescent or duplication-loss (possibly duplication-transfer-loss) model of evolution, and we believe that incorporating joint models is an exciting area for future study.

In conclusion, as we sequence ever denser clades, deep coalescence will only increase, requiring ILS-aware reconciliation methods that are both accurate and efficient. The LCT structure and DLCpar algorithm presented here enable us for the first time to exhaustively and efficiently search the space of reconciliations and infer gene evolutionary histories on a genome-wide scale without additional knowledge of the species tree beyond its topology. Thus, we recommend that studies use DLCpar in place of the standard MPR reconciliation algorithm, and we believe that its application will many important implications for future investigations of gene evolution.
Chapter 5

Evolution and reconstruction with domain rearrangements: STAR-MP

Although the possibility of gene evolution by domain rearrangements has long been appreciated, current methods for reconstructing and systematically analyzing gene family evolution are limited to events such as duplication, loss, and sometimes, horizontal transfer. However, within the Drosophila clade, we find domain rearrangements occur in 35.9% of gene families, and thus, any comprehensive study of gene evolution in these species will need to account for such events. Here, we present a new computational model and algorithm for reconstructing gene evolution at the domain level. We develop methods for detecting homologous domains between genes for reconstructing maximum parsimony evolutionary histories that include domain generation, duplication, loss, merge (fusion), and split (fission) events. Using this method, we find that genes involved in fusion and fission are enriched in signaling and development, suggesting that domain rearrangements and reuse may be crucial in these processes. We also find that fusion is more abundant than fission and that fusion and fission events occur predominantly alongside duplication, with 92.5% and 34.3% of fusion and fission events retaining ancestral architectures in the duplicated copies. Finally, we provide a catalog of ~9000 genes that undergo domain rearrangement across nine sequenced species, along with possible mechanisms for their formation. These results dramatically expand on evolution at the sub-gene level and offer several insights into how new genes and functions arise between species.

The work presented in this chapter has led to the following publication: Wu YC, Rasmussen MD, Kellis M. 2012. Evolution at the subgene level: domain rearrangements in the Drosophila phylogeny. Molecular Biology and Evolution 29(2):689-705.
5.1 Domain and architecture rearrangements

Despite the sophisticated underlying models in phylogenetic methods, a common assumption is to consider a gene as evolving as a single unit. However, duplications, losses, and other events can occur at the sub-gene level, and it has been suggested that homology inference be applied to domains rather than proteins [190]. Additionally, events such gene fusion and fission challenge the current definition of a gene family, as they can form genes that have varying phylogeny and homology across the gene sequence. These more complicated events could play very important roles in generating novel genes and functions, as they are the primary source of new domain architectures that are thought to be a main source of biological complexity in the human genome and other species [185, 278].

5.1.1 Existing work

Studies of genes at the sub-gene level have often focused on domains, defined as evolutionary, functionally, and structurally conserved units of proteins, and on domain architectures or combinations, defined as the ordered or unordered arrangement of these domains within genes. Multiple large-scale analyses of protein sequences have yielded numerous domain databases based on sequence or structural similarity [7, 16, 34, 88, 89, 110, 114, 160, 170, 179, 217], and this wealth of domain resources has motivated the development of tools for visualizing domain composition and evolution [211, 233, 279]. Furthermore, a number of protein clustering tools address the problem of homology identification in complex multi-domain families [62, 63, 139, 147, 188, 229, 241, 256, 281].

In addition to these computational resources, there are already several experimentally discovered examples of fusion and fission events. For example, jingwei is a chimeric gene found in Drosophila yakuba that arose through the fusion of the two genes yande (involved in nuclear mRNA splicing) and Adh (alcohol dehydrogenase). Although a fusion of genes is likely deleterious, several factors in this case have contributed to jingwei's retention. First, the ancestral functions involved in this fusion event were kept intact, as yande is itself a recent duplicate of yellow-emperor and the Adh portion of jingwei is a retrotransposed copy of Adh [155, 157, 265]. This allowed the jingwei to acquire a novel function in more specific binding for long chain alcohols [219]. Second, jingwei has inherited the promoter sequence of yande, preventing degeneration of the retrotransposed Adh into a pseudogene. Other examples of gene fusion events in Drosophila gave rise to Adh-Twain [124], Adh-Finnegan [123], siren [219], sphinx [263], and Quetzalcoatl [206], which have diverse functions in metabolic processes and male courtship behavior. Fusion and fission events have also been identified within clades such as bacteria [185, 236] and fungi [56], and specific chimeric genes have been studied in humans [35, 252] and plants [266]. However, while intron phase correlations suggest that as many as ~19% of exons in eukaryotic genes might have been formed by exon shuffling [156], large-scale
methods for the systematic identification and reconstruction of domain evolution and gene fusion and fission events are still lacking.

Though they do not reconstruct the history of these events, many directed studies have analyzed domain rearrangements in search of functional or evolutionary insights [20, 168]. Quantitative analyses have shown that fusions are more prevalent to fission [227], that the number of neighbors per domain follows a power law [5, 6] (though this could be attributed to limited coverage [103]), and that specific domain combinations are more conserved than would be expected from random domain shuffling [6]. Also, mechanisms of domain deletions, shufflings, and substitutions have been proposed [269, 270], and protein interaction maps have been generated based on gene fusions [60, 61].

More recently, phylogenomic methods have been developed to infer gene fusion and fission events or domain evolution, with initial approaches discovering domains de novo through sequence similarity [227], and later methods shifting to rely on underlying domain databases. These studies focused on widely divergent species spanning all three domains of life and make three types of simplifying assumptions: (1) only the presence or absence of architectures in complete genomes are considered, with both architecture count and sequence information ignored [70, 87, 141], (2) copy numbers for architectures are considered but domain ordering is ignored, and the models have leaned towards theoretic formulations and only been applied to a limited amount of biological data [17, 192, 272], and (3) domain level events are mapped onto existing gene trees, with agreement between evolutionary events considered only after the independent mappings [72].

5.1.2 Contributions

Our work continues along these recent methods in extending phylogenomics from genes to sub-gene domains. We present the first phylogenomic approach that combines de novo discovery of sub-gene evolutionary units (which we term modules), a general model of gene evolution that captures module gain, loss, duplication, and rearrangement, and a phylogenetic reconstruction algorithm that simultaneously traces the history of all modules while taking into account a common species tree topology. By focusing on modules, we are in many ways looking at how new genes are generated. That is, we can consider gene generation at a very low-level through mutations and insertion/deletions, or at a very high-level through gene duplication and loss. This work proposes a middle perspective that looks at gene generation through the generation of new modules and the duplication, loss, and rearrangement of existing modules.

We present three distinct contributions to sub-gene phylogenomics:

- We present a method for identifying homologous modules for a family of closely related species. Our approach uses sequence similarity to define modules as the basic unit of inheritance, and
Figure 5.1: Relationship between species trees, gene trees, and architecture scenarios.

(A) Gene sequences are compared across species, and a multiple sequence alignment is constructed. Due to the presence of domains or complicated evolutionary mechanisms, these alignments may have a block structure indicating similarity at the sub-gene level. (B) In conventional phylogenetics, genes that descend from a single common ancestor are clustered into a gene family, and the history of gene families are viewed through gene trees (black lines) that evolve inside a species tree (blue area). Duplication (●), loss (●), and speciation (colored sub-gene blocks) events are inferred through the reconciliation of gene trees to species trees. Since each gene can belong to only a single gene family, joint histories that are evident from the architecture structure cannot be captured. (C) In sub-gene phylogenetics as presented in this work, a gene family is generalized to an architecture family in order to capture the relationships between genes with shared modules. This allows the reconstruction of gene histories to be architecture-aware, with an architecture scenario depicting more complicated events such as merges (▼) and splits (not shown). By definition, architecture scenarios use a known species tree, with architectures evolving from a parent species to a child species; thus, no reconciliation is required, and speciation events are not modeled. In this example, the joint histories of the red and teal modules are determined, including their recent merge in the branch leading to species A, corresponding to the formation of chimeric gene a₂. (D) We allow for five types of evolutionary events, two (merge and split) of which are not typically captured in conventional gene phylogenetics. (E) Gene architectures are modeled using directed graphs, with nodes representing modules and edges representing neighboring modules (within the same gene). Rearrangements of these graphs correspond to evolutionary events: adding or removing nodes correspond to generation, duplication, or loss events (not shown), and adding or removing edges correspond to merge or split events.
therefore is not limited to existing domain databases which may be biased towards domains with known structures or domains found in well-studied proteins. We show that the resulting modules are biologically meaningful; in particular, they are frequently produced through exon shuffling events, and, when such annotations are available, they tend to keep functional domains as a single unit.

- We develop a model for gene evolution that captures architecture rearrangements, which we define as module generation, duplication, loss, merge (fusion), and split (fission) events (Figure 5.1D). In contrast to many previous phylogenetic approaches, our model traces gene evolution rather than architecture evolution, allowing us to explicitly capture module duplications and parallel merges and splits.

- We present a maximum parsimony algorithm STAR-MP (Species Tree informed Architecture Reconstruction - Maximum Parsimony) for inferring module architecture evolution based on (reconstructed) module phylogenies, extant module architectures, and a known species tree. Along with our evolutionary model, this algorithm is less restrictive than previous phylogenetic approaches, retaining the advantages of each. In particular, we assume a known species tree, as the added information can improve gene tree reconstruction; we do not rely on a reference gene or domain but instead view modules as the primary unit of genes, allowing us to trace the evolutionary history of genes related through any subsequence within a single reconstruction; we incorporate sequence information for each module captured through phylogenetic reconstruction; and we consider the statistical support of our reconstructions through bootstrapping. The STAR-MP software is available at http://compbio.mit.edu/starmp/.

To demonstrate the sensitivity and robustness of our methods, we consider eukaryotic species that are evolutionary closely related, where a species tree is well-supported and horizontal gene transfer is unlikely and not modeled. We also consider the problem of detecting architecture rearrangements at a smaller timescale, identifying only merge and split events that have occurred in recent history; we focus our analysis on the Drosophila clade (Figure 5.2), as it has a dense phylogeny, a relatively recent (~60 million year old) history [99], and includes both close and distant species. Furthermore, at least 47 putative chimeric genes have been identified within D. melanogaster [205, 289], and it has been estimated that ~30% of the new genes in the D. melanogaster species subgroup are chimeric [289]. We have used our methods to trace the complete history of all genes through their modules in nine Drosophila species and report numerous striking examples of architecture evolution that cannot be captured by traditional gene-level methods.

5.1 Domain and architecture rearrangements
Figure 5.2: Species and phylogeny of the *Drosophila* clade. The phylogeny of 9 *Drosophila* species used in our analysis, as estimated by Tamura et al. [240].

5.2 Architecture-aware phylogenomic pipeline

5.2.1 Definitions

Due to fusion and fission, a gene may contain specific domains (or more generally DNA segments) whose evolutionary history differs from the rest of the gene. Therefore, we introduce several new concepts to describe the possible relationships between such genes. Our primary unit of evolution is the *module*, which is a gene subsequence inherited as a single unit without internal rearrangements or breaks across the species under comparison. Modules discovered from sequence similarity are distinct from structural or functional domains of a protein, though, as we will show, they often agree. Each gene may contain one or more non-overlapping modules. These modules may share homology with other modules present within the same gene or in other genes. We call a cluster of homologous modules a *module family*, defined as the set of modules that descend from a single ancestral module in the last common ancestor (LCA) of all species under consideration.

For each gene, we define its *architecture* as the ordered list of modules it contains. Each species contains a set of genes, which corresponds to a multiset of architectures. We generalize the concept of a gene family to that of a *(gene) architecture family*, which contains the maximal set of genes connected by module homology. While the evolutionary histories of gene families are represented by gene trees, the histories of architecture families are represented by *architecture DAGs* (directed acyclic graphs), which extend gene trees by capturing module generation, fusion, and fission events, in addition to module duplication and loss. Lastly, we define an *architecture scenario* as the multiset of ancestral architectures and evolutionary events mapped onto a known species tree, where each species tree node shows the type and copy number of architectures it contains, and each species tree branch shows the events that have occurred along that branch. In reconstructing architecture scenarios, we will assume a known species tree and infer ancestral architectures and events without requiring a reconciliation mapping. All trees within this work are rooted phylogenetic trees in which
the leaf nodes represent extant evolutionary objects (e.g. extant species or modules in extant species) and the internal nodes represent ancestral objects (e.g. ancestral species or ancestral modules in ancestral species).

5.2.2 Architecture evolution using directed acyclic graphs

Our model for architecture evolution allows for the following evolutionary events: generation, in which a new module is created; duplication, in which an existing module is duplicated; loss, in which an existing module is lost; merge, in which two modules that appeared at the ends of two separate architectures are joined as neighbors in a single gene; and split, in which two modules that appeared as neighbors in a single gene are split and appear at the ends of two separate genes. We also make the further assumption that a module can be generated at most once. This is similar to the assumption used in Dollo parsimony in which a single generation in the last common ancestor followed by (multiple) losses is more likely than multiple independent generation events. We represent an architecture as a DAG capturing module ordering relationships between consecutive modules. Each evolutionary event corresponds to a simple graph operation (Figure 5.1E), and determining architecture rearrangements becomes a matter of graph rearrangements using these operations (Section D.1).

5.2.3 Overview of the pipeline

We present a novel phylogenomic pipeline for the architecture-aware reconstruction of gene evolution (Figure 5.3). The pipeline has three main stages: (1) identifying modules and module families from the genomic sequences, (2) clustering architectures into architecture families, and (3) reconstructing architecture scenarios from the architecture families and the known species tree.

5.2.4 Identifying modules and module families

To identify modules and their boundaries, we ran pairwise all-vs-all BLASTp comparisons [3] between the species' proteomes, discarding any BLAST hit with e-value $> 1 \times 10^{-5}$ or percent identity $< 60\%$. The remaining alignments were extended using LALIGN [117], and the best hit between each query and subject pair was retained. These were re-filtered by e-value and percent identity, and short alignments ($< 50$ aa) and promiscuous hits (genes with $> 80$ hits) were removed. A list of potential module boundaries was then found using the residue correlation matrix as in the ADDA algorithm [109] (resolution = 10 aa, minimum module length = 30 aa), and boundaries within 30 aa of a LALIGN alignment boundary were retained. The resulting module instances were clustered into module families using the Markov cluster algorithm (MCL) [257] with default parameters [63, 147],

5.2 Architecture-aware phylogenomic pipeline
Figure 5.3: **Overview of our phylogenomic pipeline.** At left, the pipeline is separated into three main stages and takes as input the set of all gene sequences across several species and the known species tree relating the species. (A) In the first stage, gene sequences are compared across species, module boundaries are found, and modules are clustered according to similarity, resulting in a set of homologous module families. (B) In the second stage, a module adjacency graph is constructed based on these module families, with an edge between any two module families if at least one module instance from each family are neighbors in the same gene. Connected components of this graph define the module families to be clustered into a single architecture family. Note that B uses as input the module families determined by A, but one can use domains as determined by a database search, e.g. Pfam domains, if desired. (C) In the third stage, architecture scenarios are reconstructed for each architecture family based on a three-step procedure in which the module trees are reconstructed based on multiple sequence alignments of each module family, these module trees are reconciled to determine ancestral module counts, and the module counts, extant architectures, and known species tree are used to reconstruct the ancestral architectures and ancestral events along each branch.

where the nodes represent module instances and edges are weighted by the bitscore of the LALIGN hit multiplied by the relative overlap of the modules.

Note that if desired, these steps can be replaced by matching gene sequences against a database of known structural or functional protein domains to simultaneously detect the domain boundaries and domain families. However, our approach is more general as it defines modules as evolutionarily-conserved units without relying on previous annotations. Thus, we can trace the evolutionary history of clade-specific modules, or modules that are not found in current databases (Section D.2).

As our goal is to study evolutionary events such as gene merge and split events between multiple species, we excluded any module families that appear in only a single species. Also, as in other works [70, 72], to mitigate the effects of short length repeat domains and allow for a more efficient algorithm, we collapsed tandem duplicated modules to a single copy and required that a module family appear at most once within an architecture.
5.2.5 Clustering architectures into architecture families

To determine architecture families, we constructed a module adjacency graph, where each vertex represented a module family, and edges were added between two modules if instances of the modules were neighbors within at least one gene. For each connected component within this graph, we identified the set of genes containing at least one module from the cluster and marked them as an architecture family.

From the module adjacency graph, we discovered several highly promiscuous module families that occur in diverse sets of genes. These module families can complicate analysis by creating very large architecture families composed of many distinct gene clusters that share little in common aside from the promiscuous module family. Therefore, we choose to analyze promiscuous module families in a separate analysis (Section D.3) and excluded them from our reconstructions. Specifically, module families were removed prior to clustering if they had more than 6 neighbors; this removed < 0.21% of all modules and < 0.38% of the modules with neighbors.

In addition, to focus on gene fusions and fissions, we filtered our architecture families to those in which one species has a gene with two neighboring modules and another species is either missing one of these modules or has no gene with these modules as neighbors.

5.2.6 Reconstructing gene architecture histories

For each architecture family, we reconstructed its evolution by producing an architecture scenario. This is complicated by the fact that inferring architectures in ancestral species implicitly requires inferring module counts. Rather than doing these tasks simultaneously, we adopted a three-stage approach to architecture scenario reconstruction, incorporating known rates of evolutionary events where applicable (Section D.4). First, we reconstructed the generation, duplication, and loss history of each module independently of all other modules since these events occur at the module level. Then, we then used these reconstructed module phylogenies to determine ancestral module counts, and finally, we incorporated merge and split events when inferring module groupings into architectures.

In the first stage, we incorporated known rates of evolutionary events to reconstruct the phylogenies of each module family to produce module trees. This was done by taking the peptide sequences of each module family, aligning them with the MUSCLE software package [57], then reverse translating the result into a (codon-aligned) nucleotide alignment. Module trees were then reconstructed from each nucleotide alignment using the SPIMAP program [200] configured with model parameters previously determined for the Drosophila clade [200], 100 pre-screen iterations, and 50 iterations.

In the second stage, we split modules trees into subtrees containing only descendants of a single
common ancestor within or after the root of the species tree (i.e. proper module families). This was achieved by reconciling each module tree to the species tree using maximum parsimonious reconciliation (MPR) [181, 290] and then removing any duplication nodes predating the species tree root (pre-root duplications). Each resulting subtree was then rerooted and reconciled repeatedly using MPR until all pre-root duplications were removed.

In the third stage, we reconstructed architecture scenarios for each architecture family by combining all of its module trees. From the previous steps of the pipeline, we can infer the extant architectures present at the leaves of the species tree, and we can use the reconciled module trees to infer the ancestral module copy numbers. What remains to be reconstructed is how the ancestral modules combine to form ancestral architectures and what events are responsible for their evolution.

5.3 The STAR-MP algorithm

5.3.1 Inferring most parsimonious architecture scenarios

We achieved this reconstruction using a novel maximum parsimony method called STAR-MP (Figure 5.4), which determines the series of events (generation, duplication, loss, merge, and split) with the least total cost that explain the evolution of the given extant architectures. In this work, we used equal costs for each event, therefore minimizing the total number of events in the reconstruction. Analysis of a subset of families showed that reconstructions are robust to these costs (Section D.5).

STAR-MP is a dynamic programming algorithm that first works recursively up the tree to determine the cost of assigning architectures at each node, then works recursively down the tree to assign the most parsimonious architecture at each node as well as the responsible events. In the forward phase, we performed a post-order traversal of the species tree, generating a set of possible architectures for each node by finding all partitions of the available modules, then pruning the resulting list heuristically. For each possible architecture generated, we determined the operations (generation, duplication, loss, merge, split) necessary to transform it into architectures present at the child nodes. Dynamic programming was then used to find the minimum cost-to-go (e.g. minimum total cost along all descendant branches) of assigning the parent architecture. This was repeated until the root of the species tree was reached, at which point the minimum-cost architecture was assigned to the root. In the backward phase, we backtracked down the tree to determine the most parsimonious architectures and events at all the internal nodes and edges, respectively. As the maximum parsimonious reconstruction may not be unique, ties were broken randomly to arrive at a single reconstruction.

To measure uncertainty in our reconstructions, we implemented a bootstrapping procedure for
Input: species tree
leaf architectures
ancestral module counts
Output: architecture scenario

STAR-MP ALGORITHM
2. Termination
Select the minimum cost architecture at the root.

TOTAL COST
traceback
D+M
M
F17
Of
T_M+2S WE S W S

1. Traversal from leaves to root
For each ancestral node, determine the possible architectures and their minimum costs-to-go.

3. Traceback from root to leaves
Assign architectures at the ancestral nodes and events along the branches.

1.1. Generate possible architectures.
Available modules
Possible architectures
(pruned)

1.2. Determine (immediate) cost of architecture rearrangements.
parent
child
D+M 0
S+D S
D+2S

1.3. Determine minimum cost-to-go of assigning parent architecture at node.
D+M
D+2S

Figure 5.4: STAR-MP algorithm. The forward phase of the STAR-MP algorithm performs a post-order traversal of the tree (step 1), considering at each stage a triplet of one parent node and two child nodes. The available parent modules (provided as input and found in our pipeline through reconciled module trees) and the possible children architectures (provided as input if the child is a leaf node or found recursively if the child is an ancestral node) are known. The possible parent architectures are generated through set permutations and pruned heuristically, and the minimum costs-to-go of assigning a parent architecture is determined (see inset at bottom, where the parent node under question is denoted by the ‘?’). The children architectures and events along the branches that led to this minimum cost-to-go is also retained for later traceback. This is repeated until the root of the tree is reached (step 2), at which point the minimum cost architecture is assigned to the root. The backward phase then performs a pre-order traversal of the tree (step 3) to assign the ancestral architectures and events.

STAR-MP. Each module family had 100 module trees reconstructed using SPIMAP on 100 resampled nucleotide alignments. From this set, modules trees were sampled with replacement to be reconciled and analyzed by STAR-MP 100 times, thus generating 100 bootstrapped architecture scenarios.

5.3.2 Reconstructing simulated architecture scenarios

Most methods within our phylogenomic pipeline (e.g. residue correlation matrix, MCL, SPIMAP) have been evaluated in their respective works [63, 109, 200]. To evaluate the last step in this pipeline, our architecture scenario reconstruction algorithm STAR-MP, we simulated module evolution, where simulation parameters were inferred using the MP architecture scenarios reconstructed from real data. Note that this reliance on MP reconstructions means that our simulations underestimate the empirical (and estimated true) event rates.

5.3 The STAR-MP algorithm
Figure 5.5: **Reconstruction accuracy of STAR-MP on simulated datasets.** Event inference using STAR-MP is both sensitive and precise. Error bars show performance loss due to ties in the MP reconstruction, e.g. the MP architecture scenario and the true architecture scenario have equal costs, so events may be missed or extra events may be called in the MP reconstruction.

We started all simulations at the root of the species tree (as was the case for > 82.6% of all MP trees) and for each simulation, generated a root architecture, where the number of module families, the number of modules per module family, and the number of connected modules were simulation parameters. To determine the events along each branch, we assumed a separate geometric distribution for each event type (generation, duplication, loss, merge, split) and each branch. The number and type of events along each branch were sampled from these geometric distributions, and an event was applied uniformly among the available modules (generation/duplication/loss), edges (split), or architectures (merge), and was discarded if it was impossible with the given starting architecture. Despite discarding events, event rates for the simulations were similar to the input rates (< 6% error).

Using rates estimated from the reconstructed architecture scenarios in *Drosophila* (Section 5.4), we simulated 1000 architecture scenarios and found that STAR-MP has ≥ 63.4% sensitivity and ≥ 77.8% precision (Figure 5.5). As in the actual pipeline, the ancestral counts for each module and the architectures at the extant species were provided as input to STAR-MP, accounting for the 100% precision in generation, duplication, and loss events. Evaluation at increased event rates reveals a decrease in sensitivity consistent with a conservative MP algorithm, whereas precision degrades only slightly (Section D.6.3, Figure D.1).

### 5.4 Analysis of 9 *Drosophila* genomes

Analysis was performed on nine species within the *Drosophila* genus: *D. melanogaster*, *D. yakuba*, *D. erecta*, *D. ananassae*, *D. pseudoobscura*, *D. willistoni*, *D. mojavensis*, *D. virilis*, and *D. grimshawi*. Sequences were obtained from FlyBase (May 2009 release), and we analyzed the longest protein sequence per gene and assumed a known species tree [240] (Figure 5.2).
Using our pipeline, we found 22,813 module families combining in 14,418 architecture families, with 70.4% (10,144) of these architecture families containing only a single module, and 28.5% (4107) containing a merge or split. (All module and architecture families are available online.) The large proportion of single-module families despite such a general definition of gene and module evolution is a testament to the high specificity of our approach. The 4107 “merge/split” families consist of 12,324 module families covering 45,282 genes, and involve at least one gene from 35.9% (4457/12,431) of FlyBase gene families.

Architecture scenarios were reconstructed for 3882 families (with 10,448 module families covering 39,476 genes), of which 2818 (72.6%) had unique maximum parsimony reconstructions; the remaining 5.5% of families had many module families per architecture family and/or large ancestral counts from SPIMAP and were too complicated for MP reconstruction. Mean runtime of STAR-MP was 2.37 s with no bootstrapping and 14.40 s for 100 bootstraps. Analysis of architecture scenarios (Sections 5.4.5 and 5.4.6) considered non-bootstrapped reconstructions. Reconstructed scenarios typically had high bootstrap support, with a majority (63.2%) of scenarios having a single reconstruction, e.g. 100% support on all ancestral architectures and events. Furthermore, each event count had a low standard deviation relative to its mean (< 0.035), thus demonstrating the robustness of our reconstruction methodology.

5.4.1 Validation

A significant challenge of reconstructing architecture evolution is dealing with errors in extant genomes, e.g. resulting from sequencing, assembly, or gene model prediction. For example, erroneously connected exons in a gene model or failure to collapse multiple genes into a single gene may cause homologous modules to appear as a single gene in some species but as multiple genes in others. To validate our sequence input, we searched for errors due to gene model or assembly problems. In this section, we provide error rates based on sequence comparison or external evidence; later, in our analysis of architecture scenarios, we will show that these errors have little effect on our biological findings.

In an assembly error, a gene may be separated into multiple scaffolds, or duplicate copies of genes may appear due to undercollapsed scaffolds. In the former case, we would expect a large number of fusion/fission genes to be at the ends of scaffolds. We found 36.2% (1486) of the merge/split families to have at least one gene at the end of its scaffold; however, this large percentage is partly attributable to the presence of several short scaffolds in the sequenced genomes. As an alternative measure, 6.51% (2947) of genes in merge/split families are at the ends of scaffolds compared to 4.85% (6592) overall, meaning that we possibly find inflated counts for the number of merges and splits.

5.4 Analysis of 9 Drosophila genomes
Table 5.1: EST and mRNA-seq evidence in 9 Drosophila genomes

<table>
<thead>
<tr>
<th>species</th>
<th># genes with EST (mRNA-seq)</th>
<th># gene pairs with spanning EST (mRNA-seq)</th>
<th># gene pairs with spanning EST (mRNA-seq)</th>
<th>error rate (%) of EST (mRNA-seq)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmel</td>
<td>14,080 12,640 (12,673)</td>
<td>14,052 11,645 (11,895)</td>
<td>78 (35)</td>
<td>0.67 (0.29)</td>
</tr>
<tr>
<td>dyak</td>
<td>16,077 1,618</td>
<td>15,335 222</td>
<td>4</td>
<td>1.80</td>
</tr>
<tr>
<td>dere</td>
<td>15,044 4,459</td>
<td>14,780 1,556</td>
<td>13</td>
<td>0.84</td>
</tr>
<tr>
<td>dana</td>
<td>15,069 5,022</td>
<td>14,680 1,864</td>
<td>24</td>
<td>1.29</td>
</tr>
<tr>
<td>dpse</td>
<td>16,099 2,851 (13,721)</td>
<td>15,156 699 (12,092)</td>
<td>13 (70)</td>
<td>1.86 (0.58)</td>
</tr>
<tr>
<td>dwil</td>
<td>15,512 4,699</td>
<td>14,442 1,792</td>
<td>17</td>
<td>0.95</td>
</tr>
<tr>
<td>dmoj</td>
<td>14,594 4,910 (13,035)</td>
<td>14,209 1,903 (12,123)</td>
<td>19 (82)</td>
<td>1.00 (0.68)</td>
</tr>
<tr>
<td>dvir</td>
<td>14,491 5,042</td>
<td>14,216 2,052</td>
<td>23</td>
<td>1.12</td>
</tr>
<tr>
<td>dgri</td>
<td>14,982 5,196</td>
<td>13,794 2,133</td>
<td>18</td>
<td>0.84</td>
</tr>
<tr>
<td>total</td>
<td>135,948 46,437 (39,429)</td>
<td>175,882 28,376 (36,110)</td>
<td>262 (187)</td>
<td>0.92 (0.52)</td>
</tr>
</tbody>
</table>

* Two adjacent genes on the same strand.
* Number of adjacent gene pairs in which both genes have EST (mRNA-seq) evidence.
* Number of adjacent gene pairs in which both genes have EST (mRNA-seq) evidence and there exists at least one EST (mRNA-seq) that spans both genes.
* Number of gene pairs with spanning EST (mRNA-seq) evidence over the number of gene pairs with EST (mRNA-seq) evidence.

In the latter case, we would expect near 100% identity in the sequences. Analysis of the sequences using gene spans with 2000 base pairs added upstream and downstream reveals 7.31% (300) of the merge/split families have possibly undercollapsed scaffolds (scaffolds contain undercollapsed genes with ≥ 98% identity, Section D.6.1). Using our rearrangement model, we believe that such families mainly result in double-counting of duplications and losses, with little to no effect on the number of merges or splits.

To check for errors due to faulty gene models, we looked at expressed sequence tag (EST) and mRNA-seq evidence for all pairs of neighboring genes (Table 5.1, Section D.6.2). We found that only 0.92% (0.52%) of EST (mRNA-seq) supported neighboring gene models also had an EST (mRNA-seq) spanning both neighbors, suggesting a low rate of introns misannotated as intergenic modules. The lowest intron annotation error rate was in the well annotated D. melanogaster genome. Larger error rates (e.g. total error rate = 11.53% (EST), 6.66% (mRNA-seq)) occur if we restrict the genes to only those that appear in architecture families (Table D.1), but this is likely attributable to the low number of EST (mRNA-seq) supported neighboring gene models in this set. Finally, note that ESTs (mRNA-seq) only allow us to find introns misannotated as intergenic modules, e.g. spurious gene breaks, not intergenic modules misannotated as introns, e.g. missed gene breaks.

We also investigated transcript evidence (EST and mRNA-seq) at the event and family level, characterizing each event or scenario as consistent if there exists no conflicting evidence, inconsistent if there exists conflicting evidence, or unknown if there exists no evidence (Section D.6.4). We found that 15.1%-16.0% of scenarios are consistent and 1.1%-1.2% inconsistent, and 23.2%-40.9% of...
merge and split events are consistent and 0.6%-1.1% inconsistent (Table 5.2). While this does not conclusively prove that the merges and splits occur, it does suggest that our reconstructed scenarios and events are not a byproduct of poor gene models.

5.4.2 Module boundaries are driven by selection

As our method for finding modules depended solely on sequence similarity rather than relying on previously known structural or functional domain or exon boundaries, we used these two external lines of evidence to study how modules are formed.

Using the curated Pfam-A (Version 23.0) [16] domain definitions as a reference, we found that our module detection algorithm tends to avoid over-fragmentation (Figure 5.6), consistent with the idea of supra-domains [261]. Furthermore, many modules and domains are also similar in size, and many module boundaries are close to domain boundaries. Note that the long tail in Figure 5.6A indicates possible under-fragmentation of domains, which is expected to occur as multiple consecutive domains may have evolved jointly within the ~60 myr Drosophila clade and thus have been collapsed into a single module.

Comparison between modules and exons reveals similar trends (Figure D.2), with many cases of single module-single exon or single module-multiple exons, and a large percentage (33-42%) of modules lying precisely at an exon boundary (peak at zero distance in Figure D.2). To study this effect further, we looked at the number of exon-bordering modules (Table 5.3) and at intron-phase correlations (Table 5.4). We defined an exon-bordering module as a module in which both boundaries are within ±10 residues of an exon boundary. The unusually high number of exon-bordering modules (observed = 100,974; expected = 2138; fold = 47.23; p < 2.23 × 10^{-308}, χ²-test) indicates exon shuffling as a prominent mechanism of module rearrangement.

Exon shuffling is also supported by a high presence of symmetrical intron phases. An intron has phase zero if it falls between two codons, phase one if it falls after the first nucleotide within a codon, phase two if it falls after the second nucleotide within a codon, and a module is labeled with the
Figure 5.6: Correlation of module and domain boundaries. (A) For each module, either the overlap (# aa present in both module and domain/domain length) for modules incompletely covered by domains, or the relative size (module length/domain length) for modules completely covered by domains, was found. 75.6% of modules are equal to or larger than their corresponding domains (relative size ≥ 100%), and 28.4% of modules are of similar size to their corresponding domain (overlap ≥ 75% or relative size ≤ 150%, in gray). Bin size = 10%. (B) For each module boundary, the distance to the closest domain boundary was found, where distance = module boundary – domain boundary, blue represents left module boundaries, and green represents right module boundaries. Thus, a negative distance in blue and a positive distance in green denote that the module boundary extends further than the domain boundary. Module boundaries tend to be close to domain boundaries or extend further than the closest domain boundary. Bin size = 10 aa.

Table 5.3: Correlation of module boundaries and domain boundaries with exon boundaries

<table>
<thead>
<tr>
<th>species</th>
<th># proteins</th>
<th># exons</th>
<th># modules</th>
<th>O</th>
<th>E</th>
<th>O/E</th>
<th># domains</th>
<th>O</th>
<th>E</th>
<th>O/E</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmel</td>
<td>14,080</td>
<td>55,172</td>
<td>19,224</td>
<td>11,810</td>
<td>285</td>
<td>41.49</td>
<td>18,278</td>
<td>1,405</td>
<td>610</td>
<td>2.30</td>
</tr>
<tr>
<td>dyak</td>
<td>16,077</td>
<td>58,629</td>
<td>20,409</td>
<td>13,008</td>
<td>337</td>
<td>28.40</td>
<td>18,349</td>
<td>1,471</td>
<td>604</td>
<td>2.44</td>
</tr>
<tr>
<td>dere</td>
<td>15,944</td>
<td>55,947</td>
<td>19,448</td>
<td>12,306</td>
<td>311</td>
<td>39.76</td>
<td>18,309</td>
<td>1,405</td>
<td>642</td>
<td>2.19</td>
</tr>
<tr>
<td>dana</td>
<td>15,069</td>
<td>56,304</td>
<td>17,002</td>
<td>10,553</td>
<td>205</td>
<td>50.30</td>
<td>18,987</td>
<td>1,542</td>
<td>698</td>
<td>2.21</td>
</tr>
<tr>
<td>dpe</td>
<td>16,099</td>
<td>57,556</td>
<td>16,772</td>
<td>10,323</td>
<td>183</td>
<td>49.85</td>
<td>19,500</td>
<td>1,592</td>
<td>692</td>
<td>2.30</td>
</tr>
<tr>
<td>dwil</td>
<td>15,512</td>
<td>56,273</td>
<td>15,803</td>
<td>9,754</td>
<td>160</td>
<td>61.10</td>
<td>18,995</td>
<td>1,537</td>
<td>669</td>
<td>2.30</td>
</tr>
<tr>
<td>dmoj</td>
<td>14,594</td>
<td>54,664</td>
<td>16,949</td>
<td>10,891</td>
<td>213</td>
<td>51.19</td>
<td>17,529</td>
<td>1,409</td>
<td>576</td>
<td>2.45</td>
</tr>
<tr>
<td>dvir</td>
<td>14,491</td>
<td>54,760</td>
<td>17,258</td>
<td>11,069</td>
<td>222</td>
<td>49.85</td>
<td>18,026</td>
<td>1,468</td>
<td>620</td>
<td>2.37</td>
</tr>
<tr>
<td>dgri</td>
<td>14,982</td>
<td>56,250</td>
<td>17,590</td>
<td>11,260</td>
<td>224</td>
<td>50.31</td>
<td>18,855</td>
<td>1,610</td>
<td>673</td>
<td>2.39</td>
</tr>
<tr>
<td>total</td>
<td>135,948</td>
<td>505,555</td>
<td>160,455</td>
<td>100,974</td>
<td>2,138</td>
<td>47.23</td>
<td>166,828</td>
<td>13,439</td>
<td>5,782</td>
<td>2.32</td>
</tr>
</tbody>
</table>

a The observed and expected number of exon-bordering module (the number of modules in which both boundaries are within ±10 aa of an exon) are shown, as well as the fold percentage. To calculate the expected number of exon-bordering modules, we derived the probability $P$ of an exon border falling onto any amino acid by dividing the total number of exon borders by the total length of proteins. We also determined the total number $T$ of amino acids within ±10 aa of any module boundary. Based on a null hypothesis of randomly distributed exon borders, the product $PT$ gives the expected number of exon borders within ±10 aa of a module boundary; thus, the expected number of exon-bordering modules is $E = (PT/\#\text{exons})^2/\#\text{modules}$. The same analysis is performed exchanging module with domains.

b $P$-values were calculated using a $\chi^2$-distribution (dof = 1); all $p$-values satisfied $p < 2.23 \times 10^{-308}$.

c Column sums may not equal total due to rounding of the expected value.
Table 5.4: Intron phases of exon-bordering modules

<table>
<thead>
<tr>
<th>species</th>
<th>O-0</th>
<th>E</th>
<th>O/E</th>
<th>1-1</th>
<th>O</th>
<th>E</th>
<th>O/E</th>
<th>2-2</th>
<th>O</th>
<th>E</th>
<th>O/E</th>
<th>symmetric</th>
<th>O</th>
<th>E</th>
<th>O/E</th>
<th>non-symmetric</th>
<th>O</th>
<th>E</th>
<th>O/E</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmel</td>
<td>9,560</td>
<td>2,071</td>
<td>4.62</td>
<td>67</td>
<td>1,226</td>
<td>0.05</td>
<td>23</td>
<td>793</td>
<td>0.03</td>
<td>9,650</td>
<td>4,089</td>
<td>2.36</td>
<td>2,160</td>
<td>7,721</td>
<td>0.28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dyak</td>
<td>10,779</td>
<td>2,315</td>
<td>4.66</td>
<td>48</td>
<td>1,340</td>
<td>0.04</td>
<td>23</td>
<td>860</td>
<td>0.03</td>
<td>10,850</td>
<td>4,515</td>
<td>2.40</td>
<td>2,158</td>
<td>8,493</td>
<td>0.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dere</td>
<td>10,168</td>
<td>2,187</td>
<td>4.55</td>
<td>57</td>
<td>1,269</td>
<td>0.05</td>
<td>24</td>
<td>815</td>
<td>0.03</td>
<td>10,248</td>
<td>4,271</td>
<td>2.40</td>
<td>2,058</td>
<td>8,035</td>
<td>0.26</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dana</td>
<td>8,594</td>
<td>1,880</td>
<td>4.57</td>
<td>44</td>
<td>1,077</td>
<td>0.04</td>
<td>26</td>
<td>860</td>
<td>0.03</td>
<td>8,665</td>
<td>3,662</td>
<td>2.37</td>
<td>1,888</td>
<td>6,891</td>
<td>0.27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dpse</td>
<td>8,434</td>
<td>1,807</td>
<td>4.67</td>
<td>49</td>
<td>1,075</td>
<td>0.05</td>
<td>23</td>
<td>692</td>
<td>0.03</td>
<td>8,504</td>
<td>3,574</td>
<td>2.38</td>
<td>1,819</td>
<td>6,749</td>
<td>0.27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dwil</td>
<td>7,876</td>
<td>1,708</td>
<td>4.61</td>
<td>46</td>
<td>1,026</td>
<td>0.04</td>
<td>18</td>
<td>645</td>
<td>0.03</td>
<td>7,940</td>
<td>3,774</td>
<td>2.38</td>
<td>1,841</td>
<td>7,117</td>
<td>0.27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dmoj</td>
<td>8,978</td>
<td>1,929</td>
<td>4.65</td>
<td>52</td>
<td>1,104</td>
<td>0.05</td>
<td>20</td>
<td>740</td>
<td>0.03</td>
<td>9,050</td>
<td>3,774</td>
<td>2.40</td>
<td>1,878</td>
<td>7,236</td>
<td>0.28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dvir</td>
<td>9,110</td>
<td>1,942</td>
<td>4.69</td>
<td>51</td>
<td>1,148</td>
<td>0.05</td>
<td>18</td>
<td>765</td>
<td>0.03</td>
<td>9,191</td>
<td>3,833</td>
<td>2.40</td>
<td>1,878</td>
<td>7,236</td>
<td>0.28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dgri</td>
<td>9,225</td>
<td>1,990</td>
<td>4.64</td>
<td>52</td>
<td>1,158</td>
<td>0.04</td>
<td>19</td>
<td>747</td>
<td>0.03</td>
<td>9,296</td>
<td>3,905</td>
<td>2.38</td>
<td>1,964</td>
<td>7,355</td>
<td>0.27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>82,724</td>
<td>17,829</td>
<td>4.64</td>
<td>476</td>
<td>10,435</td>
<td>0.05</td>
<td>194</td>
<td>6,739</td>
<td>0.03</td>
<td>83,394</td>
<td>35,003</td>
<td>2.38</td>
<td>17,580</td>
<td>65,971</td>
<td>0.27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*For each species, we list the observed and expected numbers of modules with the given intron-phase combinations, where a module is labeled with the phases of its flanking introns.

b Expected numbers were calculated as in [154]. Specifically, assuming that any two introns can flank an module, the expected number of modules with intron-phase \((i,j)\) is given by \(E_{ij} = P_i P_j N\), where \(0 \leq i, j \leq 2\), \(P_i\) is the proportion of intron phase \(i\) actually observed, and \(N\) is the total observed number of intron associations.

c P-values were also calculated based on a \(\chi^2\)-distribution (dof = 1); all p-values satisfied \(p < 2.23 \times 10^{-308}\).

d Column sums may not equal total due to rounding of the expected value.

phases of its flanking introns. The splice-frame rule [186] states that the phases of introns flanking modules tend to match, as this prevents frameshift mutations after exon shuffling events. Similar to previous analyses [126, 145, 152], we found that symmetrical intron phases are enriched (\(O = 83,394; E = 35,003; \text{fold} = 2.38; p < 2.23 \times 10^{-308}, \chi^2\)-test) and non-symmetrical intron phases are depleted (\(O = 17,580; E = 65,971; \text{fold} = 0.27; p < 2.23 \times 10^{-308}, \chi^2\)-test). Furthermore, most of the enrichment in symmetrical intron phases is due to the presence of 0-0 modules; we believe that this enrichment reflects a tendency for exons to be reshuffled at the codon level. Interestingly, though similar trends are seen when comparing Pfam domains and exons (Table D.2), fold enrichments and depletions are dramatically increased for modules (e.g. fold values: exon-bordering domains = 2.32, symmetrical intron phases = 1.79, non-symmetrical phases = 0.58, \(p < 2.23 \times 10^{-308}, \chi^2\)-test), and we found an abundance of 0-0 modules and a lack of 1-1 modules compared to previous analyses. These discrepancies are expected, as previous works used domain definitions produced across many genomes, whereas our modules were detected using data only across the nine Drosophila genomes. Regardless of whether domains or modules were used, these results suggest that modules (and domains) are produced through the shuffling of exons; here, a mutational mechanism is made apparent through module (domain) detection.

An alternative explanation for the correlation between module and domain boundaries could be their common correlation with exon boundaries. Thus, we tested whether module boundaries are depleted within domains, which would suggest that modules tend to maintain domains as a unit moreso than would be expected by exon distributions. We found that 7.1% (29,096/410,463) of

5.4 Analysis of 9 Drosophila genomes
introns are within ±10 residues of any module boundary, whereas within domains, this percentage decreased to 3.0% (4451/146,205), supporting our expectation that module boundaries respect domain boundaries (fold = 2.33, $p < 2.23 \times 10^{-308}$, hypergeometric test).

5.4.3 Gene ontologies associated with rapid architecture evolution reflect adaptation

In this section, we address whether certain functions are more likely to be involved in merge and split events. After correcting for possible biases (Section D.8), we found seven GO terms to be enriched across families with merge/split events compared to families without merge/split events ($p < 0.001$, hypergeometric test, FDR correction, Table 5.5). Interestingly, all enriched GO terms are biological processes, and almost all of them are involved in development.

We hypothesize that although gene fusions and fissions are likely deleterious for most genes, in some cases they may offer an advantage in terms of adaptability. For example, a domain may be a crucial component in several signaling pathways, each of which requires the domain to interact with a different ligand. Rather than generating the same domain multiple times throughout evolution, a species can duplicate the domain and merge it with others that encode different receptors. Such adaptability may be advantageous in signaling and development [19, 187], explaining the enriched GO terms in these categories.

For example, we found an architecture scenario involving the TBP (TATA-binding protein) domain, which associates with different transcription factors to initiate transcription from different RNA polymerases. TBP consists of a highly conserved C-terminal core that binds to the TATA box and interacts with transcription factors and regulatory proteins, and a variable N-terminal module. A study of TBP genes hypothesized that the N- and C-terminal modules may have evolved independently of each other and fused together [237]. Furthermore, TBP is dependent on upstream activators for promoter specificity; however, fusing TBP to a heterologous DNA-binding domain bypasses the need for a transcriptional activation domain, and the recruitment of TBP with an upstream activation domain provides greater flexibility in promoter arrangement [277]. Metazoans may have evolved multiple TBPs to accommodate the vast increase in genes and expression during development and cellular differentiation [193].

5.4.4 Protein-protein interactions suggest fusion and fission of complementary genes

It has been shown that modules that merge or split tend to occur in genes with related functions [60, 61, 161]. This is the basis for the Rosetta Stone model for protein-protein interaction, which suggests
Table 5.5: GO enrichment for genes undergoing module rearrangement

<table>
<thead>
<tr>
<th>rank</th>
<th>GO ID</th>
<th>GO term</th>
<th>k</th>
<th>m</th>
<th>fold</th>
<th>p-value$^a$</th>
<th>p-value$^b$</th>
<th>q-value$^c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GO:0009653</td>
<td>anatomical structure morphogenesis</td>
<td>426</td>
<td>1100</td>
<td>1.36</td>
<td>1.61 x 10^{-14}</td>
<td>2.13 x 10^{-7}</td>
<td>1.08 x 10^{-4}</td>
</tr>
<tr>
<td>2</td>
<td>GO:0048731</td>
<td>system development</td>
<td>499</td>
<td>1304</td>
<td>1.34</td>
<td>8.02 x 10^{-16}</td>
<td>2.34 x 10^{-8}</td>
<td>1.36 x 10^{-5}</td>
</tr>
<tr>
<td>3</td>
<td>GO:0048856</td>
<td>anatomical structure development</td>
<td>557</td>
<td>1465</td>
<td>1.34</td>
<td>5.44 x 10^{-17}</td>
<td>8.18 x 10^{-9}</td>
<td>5.53 x 10^{-6}</td>
</tr>
<tr>
<td>4</td>
<td>GO:0007275</td>
<td>multicellular organisial development</td>
<td>588</td>
<td>1554</td>
<td>1.33</td>
<td>1.97 x 10^{-17}</td>
<td>3.37 x 10^{-9}</td>
<td>3.42 x 10^{-6}</td>
</tr>
<tr>
<td>5</td>
<td>GO:0032502</td>
<td>developmental process</td>
<td>640</td>
<td>1709</td>
<td>1.32</td>
<td>7.95 x 10^{-18}</td>
<td>3.03 x 10^{-9}</td>
<td>3.42 x 10^{-6}</td>
</tr>
<tr>
<td>6</td>
<td>GO:0032601</td>
<td>multicellular organisimal process</td>
<td>711</td>
<td>1903</td>
<td>1.31</td>
<td>1.34 x 10^{-19}</td>
<td>4.23 x 10^{-10}</td>
<td>8.58 x 10^{-7}</td>
</tr>
<tr>
<td>7</td>
<td>GO:0009987</td>
<td>cellular process</td>
<td>804</td>
<td>2218</td>
<td>1.27</td>
<td>3.45 x 10^{-18}</td>
<td>5.56 x 10^{-9}</td>
<td>4.51 x 10^{-6}</td>
</tr>
</tbody>
</table>

$^a$ Computed using the hypergeometric test, which computes the probability of obtaining at least k annotated families for a given GO term among a dataset of size n, using a reference dataset containing m such annotated families out of N families. Here, n = 4107 and N = 14,418.

$^b$ p-values corrected for length bias.

$^c$ p-values corrected for length bias and multiple hypothesis testing (FDR).

that given a Rosetta Stone protein with architecture AB, two proteins with architectures A and B are functionally related and more likely to interact. Possible reasons this trend are that the fusion of neighboring genes allows for tighter co-regulation [20], or a single function has separated into two related genes in the case of fission. Here, we determine whether this is the case within the Drosophila clade. If so, we may be able to propose new functional annotations for genes.

Within D. melanogaster, we identified 1222 gene partners, where a gene partner consists of two genes connected by a Rosetta Stone protein. That is, for each pair of genes, we defined two sets of modules: the first set contains the modules in gene 1 but not in gene 2, and the second set contains the modules in gene 2 but not in gene 1. To be called a gene partner, at least one pair of modules, one from each set, must be found fused in a gene in another species. After removing the GO annotations biological process, cellular component, and molecular function, we found that 138 gene partners have both genes annotated with GO terms, and of these, 114 (82.6%) share at least one GO term. By selecting random gene partners (to control for length bias, these partners were selected from the set of 208 genes that form the 1222 partners), we observed that 61.8% share a GO term on average. This suggests that genes are more likely to have related functions if they have modules that merge or split (fold = 1.34, p < 0.001), though the cause-and-effect may be the reverse.

5.4.5 Trends in architecture scenarios revealed by STAR-MP reconstruction

Our architecture scenarios that involve module merges and splits cover 4107 architecture families, 12,324 module families, and 45,282 genes. However, many of these families have very simple scenarios. Most (2295, 55.9%) contain only two modules (Figure 5.7), and many (1007, 24.5%) contain one gene in each of the nine species. These single gene families frequently consist of distinct

5.4 Analysis of 9 Drosophila genomes
Figure 5.7: **Distribution of architecture family sizes.** (A) The number of sequences per architecture family (20 families with more than 50 sequences not shown), and (B) the number of module families per architecture family (3 families with more than 20 modules not shown) are shown. Color denotes the number of species represented in the architecture family. Many families have simple evolutionary histories, e.g. have a single gene per species or contain only two interacting modules.

Table 5.6: **Inferred evolutionary events across architecture scenarios**

<table>
<thead>
<tr>
<th>dataset</th>
<th>type</th>
<th>G</th>
<th>D</th>
<th>L</th>
<th>M</th>
<th>M_s</th>
<th>S</th>
<th>S_s</th>
<th>D:L</th>
<th>M:S</th>
<th>M_s:S_s</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullb</td>
<td># events</td>
<td>2109</td>
<td>4302</td>
<td>9873</td>
<td>4876</td>
<td>2952</td>
<td>5659</td>
<td>559</td>
<td>1:2.29</td>
<td>0.86:1</td>
<td>5.28:1</td>
</tr>
<tr>
<td></td>
<td># scenarios</td>
<td>1520</td>
<td>1775</td>
<td>2961</td>
<td>2242</td>
<td>955</td>
<td>2880</td>
<td>257</td>
<td>1:1.67</td>
<td>0.78:1</td>
<td>3.71:1</td>
</tr>
<tr>
<td></td>
<td>% scenarios</td>
<td>39.2</td>
<td>45.7</td>
<td>76.3</td>
<td>57.3</td>
<td>24.6</td>
<td>74.4</td>
<td>6.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conservedc</td>
<td># events</td>
<td>1279</td>
<td>1426</td>
<td>5763</td>
<td>2567</td>
<td>1509</td>
<td>2880</td>
<td>235</td>
<td>1:4.04</td>
<td>0.89:1</td>
<td>6.42:1</td>
</tr>
<tr>
<td></td>
<td># scenarios</td>
<td>1015</td>
<td>940</td>
<td>1954</td>
<td>1374</td>
<td>529</td>
<td>1747</td>
<td>81</td>
<td>1:2.08</td>
<td>0.79:1</td>
<td>6.53:1</td>
</tr>
<tr>
<td></td>
<td>% scenarios</td>
<td>40.7</td>
<td>37.7</td>
<td>78.4</td>
<td>55.1</td>
<td>21.2</td>
<td>70.1</td>
<td>3.3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a G: generation, D: duplication, L: loss, M: merge, S: split. M_s and S_s represent simple merges and splits, i.e. merges unaccompanied by generation or duplication events, and splits unaccompanied by duplication or loss events.

b Counts aggregated across all 3882 reconstructed architecture scenarios.

c Counts aggregated across a conservative set of 2506 reconstructed architecture scenarios with limited genome annotation errors.

d Total number of events across all architecture scenarios.

e Number of architecture scenarios with at least one branch having the event type.

subtrees, one with a single module A and another with merged architecture AB, implying a single generation and merge of module B. The second most frequent scenario (767 families, 18.7%) consists of ten sequences across nine species, corresponding to one fused gene in eight species and two fragmented genes in one species. This suggests that fragmented genes (and as we will see, fused genes) may be lineage-specific, an idea we will later revisit.

Using our 3882 reconstructed architecture scenarios, we studied the distributions of each of our events (Table 5.6, Figures 5.8 and D.5).

For generation events, we found that most modules (8339/10,448; or 79.8%) are generated at the species tree root (Figure 5.8) and were therefore inferred to exist prior to the *Drosophila* speciation.
Figure 5.8: Total counts of evolutionary events inferred on the nine Drosophila phylogeny by STAR-MP. Many evolutionary events are inferred along each branch (counts aggregated across 3882 architecture scenarios). The large number of losses is consistent with ancient duplications followed by many compensatory losses. Many merges and splits are located along leaf branches, indicating that many fusion and fission genes may be lineage-specific. Histograms of event counts are shown along each branch, and the number of modules in a species is displayed at each species node, where counts are totaled across all architecture scenarios.
A previous study on the origin of new genes in the *D. melanogaster* species subgroup found that *de novo* gene origination from non-coding sequences accounts for 11.9% of new genes [289], suggesting that partial gene origination may not be rare [154].

For duplication and loss events, we observed that losses occur 2.29 times more than duplications, which is consistent with previous studies at the gene level that found factors of 1.78-3.18 [200]. The large number of duplications relative to losses arises due to paraphyletic modules (modules that appear in an ancestor but do not appear in all descendants of that ancestor), which could require multiple loss events, and also due to modules trees that are incongruent with the species tree so that during reconciliation, a single ancient duplication is compensated for with multiple losses.

Lastly, for merge and split events, a comparison of their counts revealed a 0.86:1 merge-to-split ratio, which at first seemed inconsistent with previous studies suggesting that fusion occurs more often than fission by a factor of 2.6-5.6 [70, 141, 227]. However, one key difference in this analysis is that we measured individual events, as opposed to simply observing the presence of fused and fragmented extant genes, and we measured events over a smaller, higher resolution time scale (the 62 myr *Drosophila* clade vs. all three domains of life diverging over 3.5 billion years). Furthermore, other studies do not indicate how they handle complicated events such as partial gene duplication (architecture AB to architectures AB and A) and partial gene loss (e.g. architecture AB to architecture A). We considered the former example to require a split prior to duplication and the latter to require a split prior to loss, whereas other models may have allowed for the duplication and loss to occur without an accompanying split. Investigation of our reconstructed architecture scenarios showed that many splits are due to such partial duplications and losses; by considering only “simple” merges and splits that are unaccompanied by generation, duplication, or loss events, the merge-to-split ratio became 5.28:1, which is much more comparable to previously determined ratios.

This last observation prompted us to also analyze the co-occurrence of events. The first trend we found is that merge and split events tend to co-occur within module and architecture families. There are 1264 scenarios (32.6% of all reconstructed scenarios, 25.9% of scenarios with merge events, 22.3% of scenarios of split events) with both merge and split events. Furthermore, 2419 module families are involved in both merge and split events (42.9% of the 5645 module families that undergo a merge, 34.3% of the 7049 module families that undergo a split). This suggests that modules that undergo a merge or split event are more likely to undergo further rearrangement (compared to the 22,861 module families in *Drosophila*, fold = 1.39, hypergeometric test, \( p = 1.31 \times 10^{-108} \)).

Another interesting relationship is how merge and splits events co-occur with the other events (Table 5.7). For example, most (74.3%) merges occur between existing (non-generated) modules,
Table 5.7: Retainment of ancestral architectures by merge and split events

<table>
<thead>
<tr>
<th>MERGES</th>
<th>all</th>
<th>without generation with generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of events&lt;sup&gt;a&lt;/sup&gt;</td>
<td>4876</td>
<td>3623 (74.3%)</td>
</tr>
<tr>
<td>retained at least one split architecture&lt;sup&gt;b,c&lt;/sup&gt;</td>
<td>4512 (92.5%)</td>
<td>3437 (94.9%)</td>
</tr>
<tr>
<td>retained both split architectures&lt;sup&gt;c&lt;/sup&gt;</td>
<td>2688 (55.1%)</td>
<td>2688 (74.2%)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SPLITS</th>
<th>all</th>
<th>without loss</th>
<th>with loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of events&lt;sup&gt;a&lt;/sup&gt;</td>
<td>5659</td>
<td>2683 (47.4%)</td>
<td>2976 (52.6%)</td>
</tr>
<tr>
<td>retained merged architecture&lt;sup&gt;d,c&lt;/sup&gt;</td>
<td>1943 (34.3%)</td>
<td>1844 (68.7%)</td>
<td>99 (3.3%)</td>
</tr>
</tbody>
</table>

<sup>a</sup> Total number of merge/split events, as well as whether these events are merges with a newly generated module (e.g. A → AB) or splits that also lose an associated split module (e.g. AB → A). Percentages out of the total number of merge/split events.

<sup>b</sup> Number of merges that retain at least one ancestral split architecture (e.g. A,B → A,AB).

<sup>c</sup> Number of merges that retain both ancestral split architectures (e.g. A,B → A,B,AB).

<sup>d</sup> Number of splits that retain the ancestral merged architecture (e.g. AB → AB,A,B).

<sup>e</sup> Percentages out of the number of events in the top row.

and most (92.5%) retain at least one pre-merge architecture (due to a previous duplication event).

This is similar to cases such as *jingwei* where a duplication and merge has preserved the parental gene forms. In contrast, we found that most (52.6%) split events occur with the loss of a resulting split module, and few (34.3%) retain the pre-split architecture.

### 5.4.6 Genome annotation errors contribute to lineage-specific events

We found that 57.4% of all merge events and 78.9% of all split events occur along a branch leading to an extant species (Table 5.8). This could suggest that merge and split events tend to be lineage-specific, as found in previous studies of *Drosophila* [205, 289], or it could be an artifact of our pipeline arising from poor gene models and architecture annotations. For example, the *D. melanogaster* lineage contains 9.4% of all merge events and 16.3% of all lineage-specific merge events even though its branch accounts for only 2.9% of the total branch length within the species tree and 3.7% of the total leaf branch lengths. This genome also accounts for 14.7% (446) of the 3044 fused genes for which the split form consists of two adjacent genes, compared to an average of 10.7% (295-341) in all other genomes. However, since *D. melanogaster* has the best annotated genome and lowest gene model error rate (Table 5.1), these large percentages could be explained by genes being erroneously called as separate genes in other species and correctly called as a single gene in *D. melanogaster*, leading to a MP reconstruction in which a single merge event has occurred along the *D. melanogaster* branch.

Due to such potential anomalies, we would like a rough estimate of how many architecture families could erroneously contain merge or split events. Though we have previously validated our sequence input, we also decided to consider a highly conservative set of architecture families, which
Table 5.8: Lineage-specific merge and split events

<table>
<thead>
<tr>
<th>species</th>
<th>dist</th>
<th>M</th>
<th>S</th>
<th>genes</th>
<th>% dist</th>
<th>% $M$</th>
<th>% $S$</th>
<th>% dist</th>
<th>% $M_1$</th>
<th>% $S_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmel</td>
<td>11.2</td>
<td>457</td>
<td>233</td>
<td>446</td>
<td>2.9</td>
<td>9.4</td>
<td>4.1</td>
<td>3.7</td>
<td>16.3</td>
<td>5.2</td>
</tr>
<tr>
<td>dyak</td>
<td>8.57</td>
<td>268</td>
<td>634</td>
<td>341</td>
<td>2.2</td>
<td>5.5</td>
<td>11.2</td>
<td>2.8</td>
<td>9.6</td>
<td>14.2</td>
</tr>
<tr>
<td>dere</td>
<td>8.57</td>
<td>233</td>
<td>340</td>
<td>341</td>
<td>2.2</td>
<td>4.8</td>
<td>6.0</td>
<td>2.8</td>
<td>8.3</td>
<td>7.6</td>
</tr>
<tr>
<td>dana</td>
<td>53.4</td>
<td>317</td>
<td>510</td>
<td>335</td>
<td>13.8</td>
<td>6.5</td>
<td>9.0</td>
<td>17.6</td>
<td>11.3</td>
<td>11.4</td>
</tr>
<tr>
<td>dpse</td>
<td>55.8</td>
<td>362</td>
<td>523</td>
<td>319</td>
<td>14.4</td>
<td>7.4</td>
<td>9.2</td>
<td>18.4</td>
<td>12.9</td>
<td>11.7</td>
</tr>
<tr>
<td>dwil</td>
<td>32.7</td>
<td>283</td>
<td>546</td>
<td>295</td>
<td>8.4</td>
<td>6.2</td>
<td>9.7</td>
<td>10.8</td>
<td>10.1</td>
<td>12.2</td>
</tr>
<tr>
<td>dmj</td>
<td>32.7</td>
<td>301</td>
<td>403</td>
<td>335</td>
<td>8.4</td>
<td>6.2</td>
<td>7.1</td>
<td>10.8</td>
<td>10.1</td>
<td>9.0</td>
</tr>
<tr>
<td>dvir</td>
<td>37.1</td>
<td>269</td>
<td>735</td>
<td>314</td>
<td>9.6</td>
<td>5.5</td>
<td>13.0</td>
<td>12.3</td>
<td>9.6</td>
<td>16.5</td>
</tr>
<tr>
<td>dgrl</td>
<td>37.1</td>
<td>269</td>
<td>735</td>
<td>314</td>
<td>9.6</td>
<td>5.5</td>
<td>13.0</td>
<td>12.3</td>
<td>9.6</td>
<td>16.5</td>
</tr>
<tr>
<td>total</td>
<td>302.65</td>
<td>2,799</td>
<td>4,464</td>
<td>3,044</td>
<td>77.9</td>
<td>57.4</td>
<td>78.9</td>
<td>78.9</td>
<td>57.4</td>
<td>78.9</td>
</tr>
</tbody>
</table>

*Number of fused genes for which the split form consists of two adjacent genes.

b Branch length divided by the total branch length.

c Number of merges or splits in this genome divided by the total number of merges or splits, and ratio, in parentheses, of (% M)/(% dist) or (% S)/(% dist).

d Branch length divided by the total leaf branch length.

e Number of merges or splits in this genome divided by the total number of lineage-specific merges or splits, and ratio, in parentheses, of (% M_1)/(% dist_1) or (% S_1)/(% dist_1).

we defined as families in which no genes are neighbors, no genes are at the ends of scaffolds, and no genes have transitive BLAST hits through alternatively spliced forms. This last filter removes possible spurious gene fusions and fissions in which part of the fused gene is found in an alternative transcript but not in the longest transcript.

Filtering the 4107 architecture families involving module merges or splits resulted in a set of 2506 families (61.0% of original set) with 6120 modules (49.7%) covering 21,780 genes (48.0%). This implies that up to 39.0% of the “merge/split” architecture families could be affected by genome annotation errors or alternative transcripts that were not considered. Within the conservative set, 2492 families with 6022 modules covering 21,518 genes had reconstructed architecture scenarios. Note the two-fold decrease in the number of sequences represented. This is expected as our conservative set likely discarded many true examples of gene fusion and fission; for example, all scenarios with adjacent genes merging or a gene splitting into two adjacent genes were removed, despite both of these are being valid potential mechanisms.

This conservative set of families removed 54.7% of lineage-specific merges and 48.8% of lineage-specific splits. However, 49.4% of the remaining (conservative) merge events and 79.3% of the remaining split events are still lineage-specific, and the percentage of merge events in the D. melanogaster lineage was only reduced from 9.4% to 6.8% (percentage of lineage-specific merge events reduced from 16.3% to 13.7%) (Table D.5), suggesting that lineage-specific events are not solely a byproduct of poor gene annotations.

Considering all architecture families, the conservative filter retained 12,408 families (86.1% of original set) with 16,178 modules (70.9%) covering 84,496 genes (75.3%). Though ratios and folds
5.4.7 Phylogenomic pipeline recovers known examples of chimeric genes

Zhou et al. [289] and Rogers et al. [205] previously identified 47 unique chimeric genes in *D. melanogaster*, 21 of which were also identified by our algorithm (Table 5.9), yielding a sensitivity of 44.7%. However, Zhou et al. [289] allowed chimeric genes to arise from a single parental sequence recruiting sequences from other intronic or intergenic sequences or from repetitive elements; this resulted in 32 of their chimeric genes having a single parental gene. Such chimeric genes might not have been detected by our pipeline since a gene subsequence must have had a hit for it to propagate through our module detection algorithm, and our use of protein sequences eliminated any possible hits to intronic or intergenic sequences. By considering only chimeric genes that have two or more parental genes, our sensitivity rises to 60% (9/15). The remaining chimeric genes were not identified due to no hits found (1), no hits found satisfying the percent identity threshold (1), frameshift mutations (1), overlapping alignments (2), or under-clustering of modules into module families (1). The first two reasons are a consequence of the BLAST step in our pipeline, where we chose thresholds consistent with previous studies in phylogenomics [199]. Similarly, regarding the last reason, we chose a clustering threshold for MCL consistent with previous studies [63].

Both Zhou et al. [289] and Rogers et al. [205] used BLASTn to compare CDS sequences, and they used different filters, e.g. they kept only the top hits or used different alignment length and percent identity thresholds. In our pipeline, we used peptide sequences and BLASTp to compare sequences in our pipeline as peptide homology is more sensitive than nucleotide homology. However, our choice to use BLASTp also eliminated our ability to detect frameshift mutations. Investigation of nucleotide alignments suggests that frameshift mutations account for a small percentage (~0.58%) of total alignments and would increase the number of genes participating in merge/split families by <3.15% (Section D.11). Future investigation may incorporate these alignments into our pipeline.

Both cases of overlapping alignments had nearly full overlaps among the three sequences, indicating that the three genes were likely duplicate copies rather than two parental sequences and one chimeric child. Aside from sequence changes in the datasets that could have caused non-overlapping alignments to now appear as overlapping, remember that we also extended our alignments using LALIGN, whereas Zhou et al. [289] and Rogers et al. [205] used BLASTn alignments. Manual inspection of the alignments suggests that the full overlapping alignments are correct, and the two cases correspond to non-chimeric genes.
### Table 5.9: Detection of previously identified chimeric genes

<table>
<thead>
<tr>
<th>chimeric gene</th>
<th>parental genes</th>
<th>source</th>
<th>ref</th>
<th>detected or reason not detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>CG10102</td>
<td>CG12505 (Arc1)</td>
<td>Z</td>
<td>Z</td>
<td>only one hit satisfies %id thr</td>
</tr>
<tr>
<td>CG12592</td>
<td>CG12819 (ele), CG18545</td>
<td>E</td>
<td>R</td>
<td>under-clustered module instances</td>
</tr>
<tr>
<td>CG18853</td>
<td>CG12822, CG11205 (phr)</td>
<td>E</td>
<td>R, Z</td>
<td></td>
</tr>
<tr>
<td>CG18217</td>
<td>CG17286 (spd-2), CG4098</td>
<td>E</td>
<td>R, Z</td>
<td></td>
</tr>
<tr>
<td>CG31332 (unc-115)</td>
<td>CG31352</td>
<td>E</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>CG31687</td>
<td>CG31688, CG2508 (edc23)</td>
<td>E</td>
<td>R, Z</td>
<td></td>
</tr>
<tr>
<td>CG33105 (p24-2)</td>
<td>CG34104 (eca), CG31352</td>
<td>E</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>CG32745</td>
<td>CG34548 (Top303)</td>
<td>E</td>
<td>R, Z</td>
<td>CG32745 became pseudogene</td>
</tr>
<tr>
<td>CG32822</td>
<td>CG32745</td>
<td>E</td>
<td>R, Z</td>
<td>frameshift</td>
</tr>
<tr>
<td>CG340100</td>
<td>CG17706</td>
<td>E</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG35321</td>
<td>CG12824</td>
<td>E</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG32733</td>
<td>CG9821</td>
<td>E</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG12184</td>
<td>CG12179</td>
<td>U</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG17285</td>
<td>CG12825</td>
<td>R</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG14810</td>
<td>CG14811</td>
<td>R</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG32584</td>
<td>CG15645 (cerv)</td>
<td>N</td>
<td>Z</td>
<td>CG32584 split into CG42299/CG42300</td>
</tr>
<tr>
<td>CG32770</td>
<td>CG15818</td>
<td>E</td>
<td>Z</td>
<td>no hits satisfy %id thr</td>
</tr>
<tr>
<td>CG32771 (Acp29AB)</td>
<td>CG17440</td>
<td>E</td>
<td>Z</td>
<td>no hits satisfy %id thr</td>
</tr>
<tr>
<td>CG32772</td>
<td>CG17446</td>
<td>R</td>
<td>Z</td>
<td>no hits</td>
</tr>
<tr>
<td>CG33794</td>
<td>CG33299</td>
<td>N</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG30000</td>
<td>CG30002</td>
<td>R</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG30011</td>
<td>CG30012</td>
<td>U</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG14666 (Tim17a2)</td>
<td>CG14666</td>
<td>U</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG32071</td>
<td>CG12522</td>
<td>R</td>
<td>Z</td>
<td>no hits</td>
</tr>
<tr>
<td>CG14628</td>
<td>CG14628</td>
<td>U</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG14628</td>
<td>CG14628</td>
<td>U</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG33317</td>
<td>CG33317</td>
<td>R</td>
<td>Z</td>
<td>CG33317 became pseudogene</td>
</tr>
<tr>
<td>CG34100 (lectin-24A)</td>
<td>CG34100</td>
<td>U</td>
<td>Z</td>
<td>no hits satisfy %id thr</td>
</tr>
<tr>
<td>CG9902</td>
<td>CG7692</td>
<td>U</td>
<td>Z</td>
<td>no hits satisfy %id thr</td>
</tr>
<tr>
<td>CG10991</td>
<td>CG9360</td>
<td>R</td>
<td>Z</td>
<td>no hits satisfy %id thr</td>
</tr>
<tr>
<td>CG7804</td>
<td>CG10327 (TBPH)</td>
<td>U</td>
<td>Z</td>
<td>only one hit satisfying %id thr</td>
</tr>
<tr>
<td>CG9906</td>
<td>CG11958 (Cnx99A)</td>
<td>R</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG11235</td>
<td>CG11958 (Cnx99A)</td>
<td>N</td>
<td>Z</td>
<td>only one hit</td>
</tr>
<tr>
<td>CG31904</td>
<td>CG17396, CG7216 (Acp1)</td>
<td>E</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>CG30457</td>
<td>CG10953, CG13705</td>
<td>E</td>
<td>R</td>
<td>no hits</td>
</tr>
<tr>
<td>CG17196</td>
<td>CG17197, CG17198</td>
<td>E</td>
<td>R</td>
<td>no hits</td>
</tr>
<tr>
<td>CG11561</td>
<td>CG9416, CG30049</td>
<td>E</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>CG3078 (pnr)</td>
<td>CG9596 (gra), CG10278 (GATAe)</td>
<td>E</td>
<td>R</td>
<td>overlapping alignments</td>
</tr>
<tr>
<td>CG6844 (nAcRo-96Ab)</td>
<td>CG5610 (nAcRo-96Aa), CG11348 (nAcRo-84B)</td>
<td>E</td>
<td>R</td>
<td>overlapping alignments</td>
</tr>
<tr>
<td>CG6653 (Ugt86De)</td>
<td>CG31002 (Gga), CG17200 (Ugt86Dg)</td>
<td>E</td>
<td>R</td>
<td>overlapping alignments</td>
</tr>
<tr>
<td>CG31668</td>
<td>CG33124, CG8451</td>
<td>E</td>
<td>R</td>
<td>overlapping alignments</td>
</tr>
</tbody>
</table>

*a* Chimeric sources are divided into 'E' (from exons of another parental gene), 'N' (from intron or intragenic module), 'R' (simple tandem repeats or repetitive elements) and 'U' (unknown sources).

*b* Reference abbreviations: 'R' [205], 'Z' [289].

*c* If blank, the chimeric gene was detected. Hits refer to LALIGN hits for the chimeric gene, e.g. only one LALIGN hit for the chimeric gene satisfied the percent identity threshold.
5.4.8 Gene fusion and fission events reflect a small number of common mechanisms

In this section, we consider possible mechanisms for generating new architectures that require merges and/or splits (Figure 5.9), show a concrete example of the mechanism, and determine how often each mechanism occurs within *Drosophila* (Section D.12, catalog of genes by mechanism available online).

The first mechanism allows neighboring genes to merge or split, which could occur by mutations that alter start and stop codons. Allowing for the duplication of genes or subsequences before merges or splits, we found that 1681 modules and 6713 genes (16.4% and 17.2% of the modules/genes participating in merge/split events) possibly undergo this mechanism. Of course, such merges and splits are also the most suspect, as they could be caused by poor gene calls. Looking to EST (mRNA-seq) evidence, we found 274 (236) of the above genes are inconsistent with ESTs (mRNA-seqs), 5863 (4534) genes have no ESTs (mRNA-seqs), and 576 (1943) genes are consistent with ESTs (mRNA-seqs). Other more complicated mechanisms may also explain these fusions and fissions. For example, a merged gene that is found between the ancestral split genes (not necessarily as neighbors, example in Figure 5.10) may be the result of large-loop mismatch repair or replication slippage [205]. We found that 32 modules and 19 genes (0.3% and 0.05%) possibly result from these mechanisms.

The second mechanism was introduced with the case of *jingwei* (Figure 5.11), an example which is recovered by our pipeline. Here, a retrotransposed copy of a gene is inserted into another gene and exons are combined to produce a new gene (though a fusion of the transcripts followed by retrotransposition is also possible [2]). Such an event would correspond to a duplication and merge in our algorithm, but duplications and splits are also possible if a partial retrotransposition occurs. We found that 1904 modules and 2023 genes (18.5% and 5.18% of modules/genes participating in merge/split events) potentially result from this mechanism. In comparison, previous studies found that retrotransposition accounts for 12.2% of chimeric genes in *D. melanogaster* [289].

The third mechanism involves segmental duplication followed by differential loss and was observed
Figure 5.10: The inferred evolutionary history of GA28694 in D. pseudoobscura from CG4617 and frizzled 4 (fz4). (A) The MP architecture scenario. Architectures 00821 and 05692-03769 are usually in separate genes corresponding to CG4617 and fz4 orthologs. However, dpse contains the fused architecture 00821-03769 in gene GA28694. (B) Along the dpse branch, the MP reconstruction infers the loss of module 22266 and duplications of modules 00821 and 03769 followed by a merge to form 00821-03769. A (hidden) split is also inferred between 05692 and 03769 due to the sub-architecture duplication of 03769 from fz4 (05692-03769). (C) A genome level view shows that the orthologs of CG4617 and fz4 are adjacent in all species except dpse. In dpse, the new, fused gene GA28694 is located between CG4617 and fz4. (D) The inferred evolutionary history of GA28694 in dpse through duplication and exon shuffling. We postulate that GA28694 was formed by tandem duplication of the chromosomal region from CG4617 to fz4, followed by either large-loop mismatch repair or replication slippage to form the merged gene GA28694. Duplication allows dpse to retain the original gene functions of CG4617 and fz4 and gain a new function with GA28694. Note that while module 05692 is duplicated as part of the tandem duplication, it is subsequently lost in dpse; thus, the MP reconstruction does not infer a duplication of 05692.
Figure 5.11: The inferred evolutionary history of *jingwei* (*jgw*) in *D. yakuba* through retrotransposition and exon shuffling of alcohol dehydrogenase (*ADH*) and *yellow emperor* (*ymp*). (A) The MP architecture scenario. (1) Along the branch leading the dyak, the MP reconstruction infers the duplication of modules 17143 and 00370 followed by their subsequent merge to form *jgw*. (2) Along branch leading the dmel, we see an interaction between modules 17143 and 17258 to form *ymp*. However, there is strong evidence that the gene pairs GE10684-GE10685 and GG12235-GG12236 are actually single gene orthologs of *ymp*. These gene break errors lead to an incorrectly inferred ancestral architecture for *ymp* in which the modules 17143 and 17258 are found in separate genes rather than fused in a single gene. (3) *ADH* consists of a single module 00370, and there are multiple copies of this module in isolated form in many genomes. Multiple cases of fusions with *ADH*-derived modules have also been found experimentally, suggesting that *ADH* may be enriched for fusion events. We find module 00370 fused to the architecture 03541-01876 in dpse, which has not been previously found in literature. It is possible that GA25237 and GA25238 are further examples of *ADH*-derived chimeric genes. (B) A genome level view of *ymp* reveals gene break errors in the *ymp* orthologs. This is supported by exon structure and genome alignment and partially supported by EST evidence: *ymp* in dmel has multiple full ESTs (e.g. ESTs span entire gene), and GE10684 and GE10685 in dyak have multiple spanning ESTs, but none of GG12235 or GG12236 in dere nor GF17267 in dana have ESTs. *ADH* is found on a different chromosome (scaffold). (C) The inferred evolutionary history of *jgw* in dyak. *ymp* is first duplicated to create a second copy *yande* [154], then *ADH* is retrotransposed between the third and fourth exons of *yande* followed by degeneration of the *yande* exons found after the insertion point. Exons in gray represent exons that are not part of the longest transcript.

5.4 Analysis of 9 Drosophila genomes
Figure 5.12: The inferred evolutionary history of GH22519 in D. grimshawi through duplication-degeneration of rhea. (A) The MP architecture scenario. (The full MP architecture scenario is available for download.) Most species have the module 09411 and 04568 fused in a single gene rhea. However, dgri has the two modules in separate genes, with the rhea ortholog containing module 09411 and the GH22519 gene containing module 04568. The MP reconstruction infers a split along the branch leading to dgri. Note that in the full MP architecture scenario, there is a second gene with module 09411 in the (dmel,(dyak,dere)) ancestor, which is caused by the module tree (incorrectly) grouping dmel and dere together. This results in likely spurious duplication, loss, and split events being inferred within the melanogaster subgroup. (B) A genome level view shows that rhea and GH22519 in dgri are found on two scaffolds that alternately contain orthologs to the other eight genomes. (C) The inferred evolutionary history of rhea and GH22519 in dgri through segmental duplication followed by differential degeneration. Instead of losing the entire rhea gene in one of the duplicates, rhea undergoes alternative module loss, with each copy retaining one module of the original rhea gene. This results in two genes that appear fused in the other species and fragmented in dgri.

in the monkey king family [264]. Though we did not find this example in our dataset as the events occur in a sister group of D. melanogaster not included in our nine species, we found that 60 modules and 79 genes (0.6% and 0.2% of the modules/genes participating in merge/split events) result from this mechanism. An example is the evolution of the rhea family (Figure 5.12).
5.5 Discussion

In this chapter, we have presented a novel model of evolution that captures module-level events such as generation, duplication, loss, merge, and split, all of which lead to new module architectures, and we have also introduced a maximum parsimony algorithm STAR-MP for tracing architecture evolution and demonstrated its accuracy in simulation. Furthermore, using our architecture-aware phylogenomic pipeline on a clade of nine *Drosophila* species, we have provided the most complete picture yet of gene and module evolution in a complete genome across multiple species.

Unlike conventional gene tree reconstruction methods, our approach incorporated module architectures and was thus able to model how genes across gene families may be related, as indicated by the presence of similar modules or architectures. Also, unlike most architecture-aware phylogenomic analyses, our approach found gene modules *de novo* rather than relying on external domain models, and our reconstruction pipeline traced gene evolution while incorporating sequence information and providing statistical bootstrapping support.

Our results revealed that merges are more prevalent than splits as reported in previous directed studies. We also showed that merge and split events tend to occur more frequently when duplications have also retained the original architectures, likely allowing new functions to be generated by the newly-formed merged or split gene while retaining the original functions of the ancestral genes. Our approach should enable the systematic study of whether gene merges and splits are enriched in alternatively-spliced genes, and how often an alternatively-spliced form carries the original architecture. We did not focus on this question here, as we only used a single splice form (the longest polypeptide) for each gene in this initial study, and because alternative-splice forms are only well-annotated in *D. melanogaster* and not across the *Drosophila* clade.

In our study, we used SPIMAP for phylogenetic reconstruction of module trees; SPIMAP is a species tree-aware program that can maximize phylogenetic accuracy for small sequences. This is especially important as phylogenetic accuracy is dependent on the length of the sequences compared, which can make sub-gene-level phylogenetic reconstruction (as in our module trees) especially error-prone in absence of a known species tree.

We used equal event costs and ignored branch lengths (both within the known species tree and the reconstructed module phylogenies) when reconstructing architecture scenarios with STAR-MP. This assumes that events are equally likely across all branches regardless of event type or branch lengths. While we could have incorporated the inferred merge-to-split ratios (as reported in Section 5.4.5 when assigning event costs, we wished to avoid such circular dependencies. Future studies may investigate ways to estimate these rates and incorporate them in a probabilistic or weighted parsimony framework.
A major bottleneck of architecture reconstruction algorithms is the enumeration of possible architectures, which can use both the order of modules within architectures and the number of architecture instances within families; thus, the number of possible parent architectures given two children architectures can be intractably large. STAR-MP relied on heuristics to limit the set of parent architectures for increased efficiency, and using a maximum parsimony approach, it was possible to consider a large number of parent architectures since computing the rearrangement cost for each combination of parent and children architectures is relatively fast. However, future work may require a better understanding of architecture rearrangements to better sample the full architecture space. Further analysis, for example looking at how often modules change order, may provide insight into architecture arrangements and help us develop a more biologically relevant model. Similarly, we can examine whether more complicated events such as module inversion are required for accurate architecture reconstruction.

The methods presented here relied on parsimonious reconstructions of evolutionary histories, which allowed us to limit the number of scenarios to consider, resulting in high speed and accuracy. A major challenge going forward is to extend these methods to propagate sequence information across all possible reconstructions, similar to existing Bayesian and maximum likelihood phylogenetic methods, which we believe could better capture the evolutionary history of architecture families. In particular, such probabilistic methods could allow for the modeling of branch lengths within an architecture DAG (rather than being limited to architecture scenarios) and thus place evolutionary events at specific timepoints within the species history. This could also allow the simultaneous modeling of both sequence and architecture evolution, rather than the current approaches of utilizing sequence to reconstruct module trees and then either using architecture to reconstruct architecture scenarios or using reconciliation to determine module insertions and deletions.

Finally, while we have only focused on the Drosophila clade, increasing numbers of complete genomes are becoming commonplace across vertebrates and fungi, especially in mammals and yeast species. Further analysis of such genomes using our methods can reveal many new insights into module neofunctionalization and the emergence of new gene functions through module-level events.
Chapter 6

Conclusion

6.1 Summary

In this thesis, we have presented novel models and algorithms for reconstructing gene histories in the presence of a wide range of evolutionary phenomena, including nucleotide or amino acid substitution, gene duplication and loss, horizontal gene transfer, deep coalescence, and domain rearrangement. Using these tools, we then demonstrated the dramatic impact of phylogenetic accuracy on downstream analysis and reported several striking biological findings that could not have been discovered using previously existing methods. We are confident that this is only the first step: for example, our methods are currently being applied within the mouseENCODE and modENCODE consortia to reconstruct gene trees and identify orthologs, and this has already shown great promise towards improving our understanding of genome evolution within human, mouse, flies, worms, and yeast. Altogether, these results demonstrate the power of computational phylogenetics and comparative genomes in modern biological analysis.

In addition, this work has yielded new software tools and can seen as part of a larger effort to develop practical computational methods for use at the genomic-scale. This enables new biological inferences to be made, but we note that, in developing our methods, we utilized many existing models and algorithms. This included well-established biological models which have been developed and refined over several decades and the latest computational algorithms which have only been introduced in the past two decades, if even that long. Indeed, only by building on the past while taking advantage of new technologies were our successes possible. Therefore, it is our hope that other computational biologists will not only adopt our work in their analyses but also build on our approaches to design new innovations.
6.2 Future directions

Going forward, many of the concepts presented in this thesis can be extended to develop more sophisticated evolutionary models and reconstruction algorithms. Here, we outline some ideas for future work.

Gene evolution

Evolutionary analyses often postulate a gene tree landscape (as in Figure 2.1A); however, like the related fitness landscape [274], the gene tree landscape is often viewed as an abstract concept and is not well-understood. For example, does the landscape consist of hills and valleys, or is it more similar to a plateau? How does the landscape change depending on the gene tree metric under consideration (for example, sequence likelihood versus reconciliation cost)? Here, software tools for visualizing the tree space [112] may prove useful.

One severe drawback of the parsimony-based approaches presented in this thesis is the requirement of user-defined event costs for the different evolutionary phenomena. It is generally understood that reconciliations are sensitive to event costs, but little is understood about the relationship between events costs and solutions and choosing appropriate event costs is a notoriously difficult problem. One approach then is to optimize for event counts rather than total event cost; such an approach has already been considered for the DTL reconciliation problem [28, 253].

Additionally, inference using the joint duplication-loss-deep coalescence (DLC) evolutionary model is currently limited: the probabilistic approach is preferable since it is based directly on the generative evolutionary model and inherently considers statistical support, but it heuristically searches the reconciliation space; on the other hand, the parsimony approach is faster and currently achieves better accuracy due to its exhaustive search of the reconciliation space. One solution then is to combine these approaches, in particular, to seek the max a posteriori reconciliation while utilizing the dynamic programming approach of the parsimony method to exhaustively search the reconciliation space. The challenge of this approach is to determine how the posterior probability can be decomposed so that computations can be used across multiple reconciliations.

Another possibility is to run simulation studies to characterize problematic phylogenetic regions. For example, studies could characterize the overlap between gene tree topological errors due to lack of phylogenetic signal and due to ILS. In particular, both sources of error are associated with short branches in the species tree, making it likely that many observed cases of ILS are due to lack of phylogenetic signal. Correctly distinguishing between the two, or knowing when this is possible, could therefore lead to better understanding of the efficacy of methods that address these problems.
Species evolution

While we have focused in this work on gene tree reconstruction and gene tree-species tree reconciliation, another fundamental problem in phylogenetics is species tree reconstruction. However, whereas we typically reconstruct gene histories independently for each gene family (though approaches do exist that consider dependencies [198]), species tree reconstruction inherently requires the simultaneous analysis of multiple complete genomes, therefore presenting additional challenges.

Gene tree parsimony tackles this problem by taking a collection of gene trees and seeking a species tree with minimum reconciliation cost [159, 182, 216, 224, 268]. While such an approach has not been considered for the DLC evolutionary model, it should be relatively straightforward to adapt existing methods to utilize the locus tree.

Alternatively, we can develop Bayesian approaches to species tree reconstruction that incorporate DLC reconciliation models. For example, the popular *BEAST phylogenetic software [111], widely adopted by the biological community, uses Bayesian Markov Chain Monte Carlo (MCMC) [91, 108, 165] to co-estimate multiple gene trees embedded in a shared species tree along with population parameters for each species. The use of MCMC is notable as it allows estimation of posterior distributions. However, *BEAST considers only population-related evolutionary phenomena and is therefore limited to one-to-one orthologous gene families (that is, those unaffected by duplication). In many analyses, where duplication and loss must also be considered, such gene families form only a small subset of the available genomic data; for example, for a dense clade of 16 fungi species with a relatively recent (180 myr) history, only 36.3% (1943/5351) of gene families are one-to-one orthologous. Integrating a DLC evolutionary model into *BEAST, in place of the current coalescent model, could thus dramatically improve species inference. In addition, such an approach allows for Bayesian estimation of duplication-loss rates compared to the likelihood approaches currently in use. Accurate estimation of these rates is fundamental for understanding the expansion and contraction of gene families [97–99], which, in turn, help us understand the forces that shape genome evolution.

In related work, the DLC evolutionary model currently requires constant population sizes over the lifetime of each species and ignores other potentially useful sources of information, for example, from biogeographic constraints. Relaxing the former constraint or extending the model to incorporate the latter would allow us to reconstruct more complex demography, for example, through population bottlenecks or expansions. Similarly, in the current model, each species is represented by a single haploid sample. This sample may not be representative of the species, and using a single sample limits population inferences to ancestral species. Therefore, a natural extension is to model multiple individuals per species, which would allow us to incorporate variation between individuals and infer population statistics for existing species.

6.2 Future directions
Domain-level phylogenomics

Methods for systematically analyzing domain evolution are still in their infancy; therefore, many of the standard phylogenetic tools available for understanding and summarizing gene-level evolution are not applicable. For example, biologists are often interested in assessing support for varying evolutionary histories. A popular technique for achieving this is bootstrapping, which resamples a multiple sequence alignment, generates a bootstrap tree for each resampling, finds a single optimal tree (either based on the full alignment or by aggregating the bootstrap trees to form a consensus tree), and finally assigns each branch of the optimal tree a bootstrap value that reflects the percentage of bootstrap trees that contain the same bifurcation. However, both the consensus tree and calculation of bootstrap values relies on a tree representation for evolutionary histories. Because domain evolution is better captured with directed acyclic graphs (DAGs), novel methods are needed for summarizing information across multiple DAGs; such a problem is challenging because any representation needs to be computationally efficient to construct and also intuitively understandable.

Another research direction is to extend the current parsimony approach to a probabilistic framework, allowing us to take into account additional sources of information, for example, species tree branch lengths and event evolutionary rates (for preliminary work in this direction, see Section D.14).

6.3 The road ahead

It is clear from past successes in the Human Genome Project \[119, 120, 249, 250, 258\], the International HapMap Project \[248\], the 1000 Genomes Project \[244, 245\], ENCODE \[246, 247\], and modENCODE \[78, 251\] that collaborations among both biological and computational experimentalists and analysts have enabled discoveries that would not otherwise have been possible. Here, experimentalists provided new data for which more sophisticated evolutionary models were needed, and in turn, computational biologists developed novel methods for analyzing the data. This led to the formulation of new hypotheses from which scientists could design future studies. Moving forward, we believe that such integration of methods development and hypothesis-driven investigation will only become more important if we hope to make sense of the explosion of biological data, and, more generally, if we hope to understand the fundamental forces that govern the existence of every living organism.

Finally, we have focused in this thesis on understanding gene family evolution using information from gene sequences and species trees; however, the proteome makes up only 1–5% of the genome. Additionally, consortia projects, such as those previously mentioned, generate not only entire genome sequences but also massive catalogs of functional data. Therefore, new computational
models and algorithms, founded on sound biological principles, are needed for exploring the wealth of information encoded in, for example, non-protein-coding RNA genes, regulatory motifs, and chromatin, or in regulatory pathways and networks. In addition to advancing our understanding of the genome, insights into evolution at multiple levels and across different functional elements and regulatory systems will inform our understanding of biological design principles. We have already seen the impact of such knowledge in the development of evolutionary algorithms (for example, genetic algorithms used in artificial intelligence applications) and in the development of new genetic engineering techniques and drug designs. More broadly, we are in an era of genomics in which rapidly advancing sequencing technologies and computational innovations will enable us to make tremendous discoveries at an ever increasing pace, and we believe that with the continued development of computational evolutionary methods, we can not only address important biological questions and revolutionize our understanding of genome evolution, but also inspire innovative technologies, and ultimately, lead to new insights that will transform society at large.

6.3 The road ahead
Appendix A

Models for gene duplication and loss

A.1 Likelihood-based tests of topologies

In our discussion of hypothesis testing, we said that trees are statistically equivalent if $p \geq \alpha$. However, strictly speaking, failing to reject the null hypothesis does not imply that the null hypothesis is true. For example, it could be that enough variability exists in the sequence information to mask the differences in the statistical support of different topologies. We must therefore also check the statistical power of our test, which is defined as the probability of detecting a true alternative hypothesis given the data. A simple way to increase the power of a test is by increasing the significance criterion $\alpha$, thus increasing the chance of rejecting the null hypothesis, reducing the risk of a Type II error (false acceptance) but increasing the risk of a Type I error (false rejection). In general, tests using slow evolving sequences or longer sequences are also more informative and will have increased statistical power.

In addition, care should be taken when choosing a likelihood test and significance level, as many of the tests are based on comparisons of tree topologies chosen a priori. However, in gene tree error correction, one of the trees under consideration is typically the ML tree, and we are often comparing multiple topologies. Thus, we must account for selection bias and multiple comparisons in our likelihood test. Furthermore, the reported $p$-value may need to be halved (or the significance level doubled) in order to compute a one-tailed (rather than two-tailed) test.
One possibility is to use tests, such as the SH test [220] or the SOWH test [81], that correct for both factors. One correct approach would be to apply the SH test simultaneously to all topologies under consideration. However, by default, TreeFix uses the SH test provided by the RAxML package, which considers alternative topologies consecutively and does not correct for multiple comparisons, thus raising the probability of false rejection. While we could incorporate a correction factor, e.g. based on Bonferroni or false discovery rate, we have chosen not to do so in our current implementation. However, unlike many other uses of p-values, we are interested here in the set of trees that are not significantly different, i.e. for which $p \geq \alpha$. Multiple testing correction would only expand this set and produce less conservative results; thus, by not correcting for multiple comparisons, TreeFix errs towards supporting the sequence information over the species tree topology. For an analysis of statistical errors, see Section A.2.

An alternative approach is to alter existing tests to correct for these errors. For example, it has been shown [81] that for the KH test at a significance level $\alpha$, one can fail to reject the null hypothesis (e.g. the two trees are equivalent) if $p > 2\alpha$. Otherwise, no conclusion can be made. This suits our purposes since it means that we may miss topologies that are statistically equivalent, but we will never allow topologies that are significantly worse.

Users who are worried about violating the assumptions of existing likelihood tests can, of course, implement a basic significance test in their own user-defined statistical module. That is, for each topology $T_x$, resample the given alignment to generate a distribution of $\delta^x$ values, then reject the null hypothesis (where the null hypothesis is that $T_x$ and $T_\text{in}$ are equally good explanations of the data) if $\delta^x$ using the whole alignment falls within the $p$-tail of the distribution.

### A.2 Statistical error rate analysis

To determine the error rate of our hypothesis testing framework, we looked at the number and types of errors made by TreeFix in a simulated fungal dataset (Section 2.4). In a Type I error, we incorrectly reject a true null hypothesis; this prevents the species tree from informing the gene tree (despite the sequence data supporting the proposed topology), so the error-corrected gene trees typically have higher reconciliation costs than they would in the absence of error. On the other hand, in a Type II error, we incorrectly accept a false null hypothesis; this allows the species tree to inform the gene tree (despite the sequence data not supporting the proposed topology), so the error-corrected gene trees typically have lower reconciliation costs than they would in the absence of error.

By design, TreeFix assumes that the true gene tree is statistically equivalent to the maximum likelihood gene tree. However, an analysis of simulated gene trees shows that 99.9% of the true gene
trees were statistically equivalent to the ML (RAxML) tree (at $\alpha = 0.05$), giving us a Type I error rate of 0.1%. Using Bonferroni correction, all true gene trees would be statistically equivalent to the ML tree (i.e. Type I error rate of 0%). This suggests that multiple testing correction has little effect on the error-corrected TreeFix gene tree in most cases.

However, we also found that 1.2% of the TreeFix gene trees have a lower reconciliation cost than the true gene tree (Table A.1). This gives us an indication of the Type II error rate, as in these cases, the statistical test tells us (perhaps incorrectly) that the sequence data supports the TreeFix tree (at $\alpha = 0.05$) even though the reconciliation cost suggests that it overfits to the species tree.

### A.3 Reconciliation cost as a metric of gene tree accuracy

TreeFix utilizes similar concepts as Bayesian inference programs in that both balance sequence information with a prior based on the species tree. However, unlike likelihood and Bayesian inference programs, TreeFix uses hypothesis testing to weigh its estimates of sequence likelihood and topological accuracy (the latter of which is ignored in likelihood approaches due to its assumption of a uniform prior). Because each phylogenetic program optimizes a different criterion, a direct comparison of their gene tree metrics is not possible. However, looking across all search proposals for a gene tree and across 100 gene trees, we found that RAxML likelihood, reconciliation (duplication + loss) cost, and SPIMAP posterior probability are closely correlated (Spearman's $r > 0.705$, Table A.3). In particular, by considering the different terms that contribute to the SPIMAP posterior probability, we found that the reconciliation cost and SPIMAP's prior probability on the gene tree topology (based on the birth-death model of evolution) are highly correlated ($r = 0.978$), suggesting that the reconciliation cost is a good approximation for the gene tree topology prior.

To further evaluate the reconciliation cost, we also analyzed the bootstraps trees in the real fungal dataset (Section 2.4). Bootstrap trees may have different reconciliation costs due to the resampled alignments supporting different gene tree topologies. Thus, we computed the "coefficient of variation" ($CV = \text{standard deviation}/\text{mean}$) of the reconciliation cost for each gene tree, where the standard deviation was calculated over 100 bootstraps and the mean was taken as cost of the TreeFix tree using the full alignment rather than the average over the bootstraps. The mean standard deviation over 1429 trees with a cost of 0 was 0.14, and the mean CV over the other 3922 trees was 1.05, suggesting that the reconciliation cost is a robust metric.

Finally, we looked at the root mean square error (RMSE) of the reconciliation cost on a simulated fungal dataset (Table A.1). As with our other metrics of accuracy (Table 2.4), we found that TreeFix, SPIMAP, and NOTUNG (100) perform the best, with dramatically lower RMSE compared to other phylogenetic methods, in particular showing at least $12.33 \times$ improvement over RAxML. Impressively,
according to this metric, TreeFix (long) surpasses even SPIMAP (2.55× improvement).

A.4 Parameters for gene tree reconstruction and reconciliation

RAxML (v7.0.4) [230] was run with 100 rapid bootstraps, the GTRGAMMA model, and $\epsilon = 0.1$. TreeBest (v1.9.2) [260] was run using default parameters (and internally computes 100 bootstraps). SPIMAP results were taken from [200], which used RY-encoded alignments with 2000 iterations and 1000 prescreens on the real dataset and 5000 iterations and 1000 prescreens on the simulated dataset. All hybrid methods used RAxML trees as input and had duplication and loss costs of 1. NOTUNG (v2.6) [30, 55] was run using bootstrap values as edge weights and three $\theta$ settings of 50%, 90%, 100%. (Note that the suggested $\theta$ setting for NOTUNG is 90%. We chose to test 50% to limit the tree search and 100% to expand the tree search; a threshold of 100% considers all branches with bootstrap support < 100% of the maximum bootstrap as weak.) tt [85] was run using full error correction (e.g. all edges are weak) and three depth settings of 1, 2, 3. (Note that the default depth setting for tt is 1. We chose to test higher depths so that tt considers more distant tree topologies. Depths larger than 3 tended to exceed memory limits.) TreeFix was run using the SH test statistic with RAxML (GTRGAMMA model, $\epsilon = 2$) site-wise log-likelihoods and a variety of settings for $\alpha$, $n_i$, $n_q$, and $f$. In both the main manuscript and the supplement, ‘TreeFix’ corresponds to default search settings of $(n_i, n_q, f) = (100, 50, 0.05)$ and ‘TreeFix (long)’ to search settings of $(1000, 100, 1)$. When bootstrapped, we used default search settings and 100 bootstraps.

Unrooted trees were rooted to minimize the duplication and loss cost. When no reconciliation is provided by the reconstruction algorithm, we used maximum parsimony reconciliation (MPR) [181, 290] as the default reconciliation.

A.5 Search efficiency

To investigate how our search strategy influences reconstruction runtime, we tested a variety of settings for $n_i$, $n_q$, and $f$ (Table A.2). As expected, as $n_i$ and $n_q$ increase (i.e. the search space expands) and $f$ approaches its maximum of 1 (i.e. all gene tree proposals are optimally rooted), accuracy tends to increase. Additionally, although we have implemented TreeFix in Python with very few optimizations, it can drastically improve RAxML trees with only a minor increase in runtime. Even on the worst setting (e.g. lowest valued parameters), TreeFix improves on RAxML by 27.7% and 10.2% in topological and branch accuracy while only increasing average runtime by 3.9 sec (a
2.9% increase on the base RAxML runtime of 2.3 min, though RAxML was run using 100 bootstraps, whereas TreeFix was run on a single input tree). Furthermore, TreeFix can produce performance comparable to SPIMAP with a drastically reduced runtime. For example, running TreeFix with a setting of \( n_i = 1000, n_q = 1, \) and \( f = 0.05 \) increased total runtime to 2.4 min (with TreeFix taking on average 6.2 sec), but showed \( \leq 0.8\% \) reduction in topological and branch accuracy compared to SPIMAP, despite SPIMAP taking 17.3x longer to run.

We also evaluated our subproposal search strategy. The goal of \( n_q \) is to reduce the number of likelihood computations, which in general tend to be more computationally expensive than other TreeFix subroutines. However, in our current implementation, likelihood computation is actually performed in C using RAxML libraries, meaning that much of TreeFix is actually spent in proposing new topologies and computing their costs. We believe that the benefits of our two stage search strategy will become more apparent if TreeFix were implemented completely in C/C++. Looking at how \( n_q \) affects performance, unsurprisingly, increasing \( n_q \) is most beneficial when using small \( n_i \) since as \( n_i \) becomes large, the search space is already sufficiently explored.

To speed up computation, we also evaluated how often a tree proposal needs to be rerooted. We found little performance variation when \( f \) is varied but the iteration parameters are kept constant. These findings are perhaps expected, as rerooting is done on the pool of \( n_q \) proposals. Therefore, as long as \( f \) is sufficiently large, only the best rooted trees (e.g. those with minimum costs) will be at the top of the pool for evaluation by the outer loop controlled by \( n_i \).

## A.6 Scalability with gene family size

We evaluated the influence of gene family size on reconstruction accuracy and runtime (Figure A.3). As our dataset, we simulated more gene trees from the 16 fungal species tree and divided them into six classes based on the their number of extant genes: 5-9, 10-19, 20-29, 30-39, 40-49, and 50-59. Each size class was populated with 100 simulated trees and alignments. We found that, as expected, topological accuracy decreases as family size increases. Furthermore, TreeFix consistently outperforms the other methods, with a more pronounced improvement for large families. Interestingly, except for a few exceptions, branch accuracy and duplication and loss precision remain relatively stable across family size. Finally, the runtime for most methods scales linearly (and at roughly the same rate).
A.7 Effect of species tree divergence and size

We are also interested in how TreeFix performs for a variety of species trees; in particular, how might features of the species tree, such as speciation rate and tree size (number of extant taxa), affect performance? To address this, we simulated species trees using TreeSample [106] with a constant-rate birth-death model and a variety of settings for the speciation rate (0.05–1 events/species/myr) and tree size (5–100 extant species). In all cases, TreeFix shows dramatic improvement over most other programs (Figure A.4). (SPIMAP was not run on these gene trees due to model differences, and NOTUNG performed better than TreeFix in some cases, for reasons discussed below.)

As the speciation rate increases, in effect causing shorter branches in the species tree, we found that the topology and branch accuracy of TreeFix and NOTUNG remain relatively constant whereas the duplication and loss precision decrease. In contrast, all other methods show a decrease in performance across all metrics, which is understandable, as shorter branches result in more similar gene sequences, making it harder for typical phylogenetic methods to accurately reconstruct gene trees. Also note that for high speciation rates, NOTUNG is able to outperform TreeFix if we lower the threshold for uncertain branches. We believe this is because fast speciation rates with a constant tree size result in shorter species tree depths and fewer total gene duplications and losses (under our simulation model), thus producing gene trees that are highly congruent to the species trees. Therefore, ignoring sequence data and reconstructing the gene trees most congruent to the species tree, as in NOTUNG, results in high accuracy.

Furthermore, we found that TreeFix (with long search settings) shows consistent or only minor degradations in performance as the tree size increases, suggesting that high accuracy is achievable by balancing sequence and species tree information and increasing the tree search space. In contrast, the performances of RAxML, NOTUNG, tt, and, to some degree, TreeBest and TreeFix (with quick search settings) decrease dramatically with increasing tree size. This can be attributed to larger species trees resulting in more duplication and loss events per gene tree and thus more incongruence between gene trees and species trees. Note that for large tree sizes, NOTUNG is again able to outperform TreeFix if many rearrangements are allowed. With such a parameter setting, NOTUNG performance, as measured by duplication and loss precision, increases with increasing tree size. We believe this is a result of large species trees generating large gene trees so that many branches in the gene tree reconstruction are uncertain. This allows NOTUNG more freedom in its reconstruction compared to small gene trees, and this larger search space translates to increased accuracy. Finally, note that TreeFix is scalable to many species, and again scales at the same rate as RAxML, whereas one major limitation of Bayesian approaches is their inability to handle large species trees.

As sequencing costs continue to decline, both the net speciation rate and size of species trees will...
only increase. Given its improved accuracy over other methods and its scalability compared to more complex approaches, we believe that TreeFix should be useful for future phylogenetic analyses.
Figure A.1: Cumulative distribution of duplication consistency scores for all duplications inferred on the real fungal dataset using several phylogenetic methods. TreeFix (long), SPIMAP, and NOTUNG perform the best according to this metric (Table 2.1).
Figure A.2: **Sequence support versus tree distance on the real fungal dataset using several phylogenetic methods.** For each output tree per hybrid method, we plot its sequence support, as measured by its p-value using the SH test, against its distance from the input RAxML tree, as measured by the Robinson-Foulds (RF) metric. Note that little rearrangement (low RF score) does not necessarily mean the final tree is supported by the underlying sequence data (p-value ≥ α). TreeFix always return trees that are supported by the sequence, whereas both NOTUNG and tt may return trees that conflict with the sequence information.
Figure A.3: Phylogenetic accuracy and runtime of several phylogenetic methods for gene trees of increasing size for simulated fungal dataset. Gene trees were simulated using an i.i.d. species-specific rate model and divided into six classes based on the number of extant genes: 5-9, 10-19, 20-29, 30-39, 40-49, 50-59. Each size class was populated with 100 simulated trees and alignments. SPIMAP was run with 100 iterations, 100 prescreens, and 100 bootstraps. Note that average runtimes are plotted on a log scale.
Figure A.4: Phylogenetic accuracy and runtime of TreeFix on simulated datasets (with simulated species trees). Species trees were simulated using TreeSample [106] with a constant-rate birth-death model, varying speciation rates $\lambda$ (events/spec/myr) and tree sizes $n$ (number of extant taxa), and extinction rates set to $\mu = 0.9\lambda$. For each setting, we simulated 10 species trees, then simulated 100 gene trees and alignments per species tree using a birth-death process ($\lambda = \mu = 0.0012$ events/gene/myr), relaxed molecular clock ($\Gamma$-distributed substitution rates with $\mu = 0.002$, $\sigma = 0.001$ substitutions/site/myr), and a HKY [107] model of sequence evolution. For each metric, we plot the mean accuracy and 95% confidence interval (mean ± 1.96 x standard error) across all species trees with the same setting. (left), Performance for $\lambda = 0.05, 0.1, 0.2, 0.5, 1.0$ and $n = 20$. (right), Performance for $\lambda = 0.02$ and $n = 5, 10, 20, 50, 100$. For $n = 50, 100$, tt was not run with depth = 3 due to memory overflow.

A.8 Additional figures and tables
<table>
<thead>
<tr>
<th>program</th>
<th># orths$^a$</th>
<th># dups$^a$</th>
<th># losses$^b$</th>
<th>RF$^b$</th>
<th>% fail SH$^c$</th>
<th>% low cost$^d$</th>
<th>RMSE$^e$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TreeFix (long)</td>
<td>118,336</td>
<td>880</td>
<td>799</td>
<td>0.011</td>
<td>*</td>
<td>1.8</td>
<td>0.20</td>
</tr>
<tr>
<td>TreeFix</td>
<td>118,330</td>
<td>915</td>
<td>856</td>
<td>0.014</td>
<td>*</td>
<td>1.2</td>
<td>0.76</td>
</tr>
<tr>
<td>NOTUNG (100)</td>
<td>118,343</td>
<td>918</td>
<td>922</td>
<td>0.021</td>
<td>1.8</td>
<td>2.1</td>
<td>0.92</td>
</tr>
<tr>
<td>NOTUNG (90)</td>
<td>116,746</td>
<td>1,109</td>
<td>1,563</td>
<td>0.037</td>
<td>0.8</td>
<td>1.3</td>
<td>2.57</td>
</tr>
<tr>
<td>NOTUNG (50)</td>
<td>106,752</td>
<td>2,396</td>
<td>5,843</td>
<td>0.153</td>
<td>0.0</td>
<td>0.1</td>
<td>8.46</td>
</tr>
<tr>
<td>tt (3)</td>
<td>114,256</td>
<td>1,207</td>
<td>2,021</td>
<td>0.047</td>
<td>1.2</td>
<td>1.8</td>
<td>3.79</td>
</tr>
<tr>
<td>tt (2)</td>
<td>111,153</td>
<td>1,568</td>
<td>3,196</td>
<td>0.079</td>
<td>0.9</td>
<td>0.9</td>
<td>5.54</td>
</tr>
<tr>
<td>tt</td>
<td>106,752</td>
<td>2,153</td>
<td>5,072</td>
<td>0.126</td>
<td>0.1</td>
<td>0.0</td>
<td>7.97</td>
</tr>
<tr>
<td>SPIMAP</td>
<td>116,822</td>
<td>895</td>
<td>861</td>
<td>0.011</td>
<td>--</td>
<td>1.4</td>
<td>0.51</td>
</tr>
<tr>
<td>TreeBest</td>
<td>98,367</td>
<td>1,540</td>
<td>4,760</td>
<td>0.151</td>
<td>--</td>
<td>0.3</td>
<td>7.76</td>
</tr>
<tr>
<td>RAxML</td>
<td>101,962</td>
<td>3,005</td>
<td>7,914</td>
<td>0.198</td>
<td>--</td>
<td>0.0</td>
<td>11.34</td>
</tr>
</tbody>
</table>

$^a$ Number of pairwise orthologs, duplications, and losses inferred across all gene trees. The true topologies contained a total of 118,385 orthologs, 876 duplications, and 812 losses.

$^b$ Average Robinson-Foulds distance compared to the true trees.

$^c$ For the hybrid methods, percentage of trees that fail the SH test compared to the input RAxML trees at $\alpha = 0.05$.

$^d$ Percentage of trees for which the reconciliation cost of the reconstructed tree was less than the reconciliation cost of the true tree.

$^e$ Average root mean square error of the reconciliation cost compared to the true tree.
Table A.2: Evaluation of accuracy and runtime using several search parameters for TreeFix on simulated fungal datasets

<table>
<thead>
<tr>
<th>program</th>
<th>n_1</th>
<th>n_2</th>
<th>f</th>
<th>% top^a</th>
<th>% branch^b</th>
<th>% searched^c</th>
<th>% changed^d</th>
<th>runtime^e</th>
</tr>
</thead>
<tbody>
<tr>
<td>TreeFix</td>
<td>50</td>
<td>1</td>
<td>0.05</td>
<td>38.9</td>
<td>86.1</td>
<td>39.4</td>
<td>76.7</td>
<td>3.9 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>100</td>
<td>1</td>
<td>0.05</td>
<td>62.9</td>
<td>91.9</td>
<td>64.1</td>
<td>85.3</td>
<td>4.0 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>500</td>
<td>1</td>
<td>0.05</td>
<td>90.0</td>
<td>97.5</td>
<td>92.6</td>
<td>88.8</td>
<td>5.3 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>1000</td>
<td>1</td>
<td>0.05</td>
<td>93.4</td>
<td>98.4</td>
<td>95.5</td>
<td>88.8</td>
<td>6.2 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>50</td>
<td>1</td>
<td>0.5</td>
<td>38.8</td>
<td>86.1</td>
<td>39.1</td>
<td>75.1</td>
<td>3.9 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>100</td>
<td>1</td>
<td>0.5</td>
<td>62.7</td>
<td>91.8</td>
<td>63.2</td>
<td>86.5</td>
<td>4.5 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>500</td>
<td>1</td>
<td>0.5</td>
<td>91.9</td>
<td>97.9</td>
<td>93.8</td>
<td>88.8</td>
<td>7.1 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>1000</td>
<td>1</td>
<td>0.5</td>
<td>93.3</td>
<td>98.2</td>
<td>95.3</td>
<td>88.8</td>
<td>8.9 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>50</td>
<td>1</td>
<td>1</td>
<td>40.1</td>
<td>86.5</td>
<td>40.3</td>
<td>78.5</td>
<td>4.3 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>100</td>
<td>1</td>
<td>1</td>
<td>62.1</td>
<td>91.8</td>
<td>62.7</td>
<td>86.2</td>
<td>5.4 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>500</td>
<td>1</td>
<td>1</td>
<td>92.2</td>
<td>97.8</td>
<td>93.7</td>
<td>88.8</td>
<td>9.1 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>1000</td>
<td>1</td>
<td>1</td>
<td>93.9</td>
<td>98.5</td>
<td>95.8</td>
<td>88.8</td>
<td>12.3 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>50</td>
<td>50</td>
<td>0.05</td>
<td>92.0</td>
<td>98.0</td>
<td>95.1</td>
<td>88.7</td>
<td>10.3 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>100</td>
<td>50</td>
<td>0.05</td>
<td>94.0</td>
<td>98.5</td>
<td>96.3</td>
<td>88.8</td>
<td>13.9 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>500</td>
<td>50</td>
<td>0.05</td>
<td>95.3</td>
<td>99.0</td>
<td>97.6</td>
<td>88.8</td>
<td>34.8 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>1000</td>
<td>50</td>
<td>0.05</td>
<td>95.8</td>
<td>99.1</td>
<td>97.8</td>
<td>88.8</td>
<td>56.4 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>50</td>
<td>50</td>
<td>1</td>
<td>94.2</td>
<td>98.5</td>
<td>95.7</td>
<td>88.8</td>
<td>35.6 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>100</td>
<td>50</td>
<td>1</td>
<td>94.6</td>
<td>98.6</td>
<td>96.6</td>
<td>88.8</td>
<td>55.0 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>500</td>
<td>50</td>
<td>1</td>
<td>95.7</td>
<td>99.2</td>
<td>98.3</td>
<td>88.8</td>
<td>2.7 min</td>
</tr>
<tr>
<td>TreeFix</td>
<td>1000</td>
<td>50</td>
<td>1</td>
<td>95.2</td>
<td>98.9</td>
<td>97.3</td>
<td>88.8</td>
<td>4.4 min</td>
</tr>
<tr>
<td>TreeFix</td>
<td>50</td>
<td>100</td>
<td>0.05</td>
<td>92.0</td>
<td>98.2</td>
<td>95.4</td>
<td>88.8</td>
<td>17.7 sec</td>
</tr>
<tr>
<td>TreeFix</td>
<td>100</td>
<td>100</td>
<td>0.05</td>
<td>96.0</td>
<td>99.1</td>
<td>98.4</td>
<td>88.8</td>
<td>2.6 min</td>
</tr>
<tr>
<td>TreeFix</td>
<td>500</td>
<td>100</td>
<td>0.05</td>
<td>93.6</td>
<td>98.2</td>
<td>95.2</td>
<td>88.8</td>
<td>1.4 min</td>
</tr>
<tr>
<td>TreeFix</td>
<td>1000</td>
<td>100</td>
<td>0.05</td>
<td>96.0</td>
<td>99.1</td>
<td>98.2</td>
<td>88.8</td>
<td>15.5 min</td>
</tr>
</tbody>
</table>

NOTUNG (100)  -  -  -   90.1  97.8  -  -  86.4   0.4 sec
NOTUNG (90)   -  -  -   76.6  95.6  -  -  83.4   0.4 sec
NOTUNG (50)   -  -  -   18.2  80.9  -  -  44.4   0.4 sec
tt (3)        -  -  -   73.6  93.7  -  -  88.8   2.1 sec
tt (2)        -  -  -   54.6  89.4  -  -  88.8   1.3 sec
tt (1)        -  -  -   32.5  83.3  -  -  88.7   0.1 sec

SPIMAP^f     -  -  -  94.2  98.8  -  -  41.0 min
SPIMAP (short, boot)^f,g -  -  -  88.6  97.5  -  -  21.4 min
SPIMAP (short)^f -  -  -  72.7  93.7  -  -  11.1 sec
TreeBest^g   -  -  -  40.7  81.2  -  -  23.8 sec
RAxML^g      -  -  -  11.2  75.9  -  -  2.3 min

^a Percentage of correctly inferred full gene tree topologies.
^b Percentage of accurately reconstructed branches.
^c Percentage of gene trees in which the search contained the correct tree topology.
^d Percentage of gene trees whose topologies differ from that of the input gene tree (for TreeFix, NOTUNG, and tt). Note that 888 of the 1000 (input) RAxML trees have incorrect topologies.
^e Average runtime for reconstructing gene trees.
^f SPIMAP was run with many iterations (5000 iterations, 1000 prescreens, corresponding to Figure 2.4 in the main text) and with few iterations (50 iterations, 100 prescreens).
^g These methods were run with 100 bootstraps.
Table A.3: Average Spearman correlation coefficient of different cost metrics on the simulated fungal dataset

<table>
<thead>
<tr>
<th></th>
<th>RAxML</th>
<th>dup-loss</th>
<th>SPIMAP</th>
<th>SPIMAP top</th>
<th>SPIMAP branch</th>
<th>SPIMAP seq lik</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAxML</td>
<td></td>
<td></td>
<td>0.705</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dup-loss</td>
<td>0.705</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIMAP</td>
<td>0.876</td>
<td>0.766</td>
<td></td>
<td></td>
<td>0.755</td>
<td></td>
</tr>
<tr>
<td>SPIMAP top</td>
<td>0.682</td>
<td>0.978</td>
<td>0.550</td>
<td>0.686</td>
<td>0.731</td>
<td>0.674</td>
</tr>
<tr>
<td>SPIMAP branch</td>
<td>0.550</td>
<td>0.686</td>
<td>0.731</td>
<td>0.686</td>
<td>0.731</td>
<td></td>
</tr>
<tr>
<td>SPIMAP seq lik</td>
<td>0.884</td>
<td>0.669</td>
<td>0.969</td>
<td>0.657</td>
<td>0.669</td>
<td>0.590</td>
</tr>
</tbody>
</table>

*a For 100 gene trees in the simulated fungal dataset, all TreeFix proposals were recorded. Different costs were then calculated for each proposal, and the Spearman correlation coefficient and associated p-value was determined for each pair of cost criterion using all proposals of a gene family. Shown are the average correlations across all 100 gene families. Using Fisher's combined probability test, all aggregate p-values were below machine precision of $2.26 \times 10^{-308}$.

*b The change in sequence likelihood compared to the RAxML gene tree, calculated using the GTRGAMMA model of evolution.

*c The total number of duplications and losses.

*d The SPIMAP posterior probability (negated to give positive correlations). SPIMAP probabilities are unnormalized; that is, they reflect the joint probability (sequence likelihood times gene tree prior) and are not conditioned on the sequence data.

*e The SPIMAP branch prior probability, calculated using a prior of gene and species-specific branch distributions (negated).

*f The SPIMAP topology prior probability, calculated using a birth-death process of gene evolution (negated).

*g The SPIMAP sequence likelihood, calculated using the HKY model of evolution (negated).
Appendix B

Models for gene duplication, transfer, and loss

B.1 The duplication-transfer-loss (DTL) reconciliation model

The DTL-reconciliation model allows for the reconciliation of a given rooted gene tree with a given rooted species tree by postulating duplication, transfer, and loss events. The problem of reconciling a gene tree with a species tree under the DTL-reconciliation model is referred to as the DTL-reconciliation problem and has been extensively studied [12, 28, 33, 36, 37, 52, 83, 148, 163, 164, 180, 207, 253, 254]. The DTL-reconciliation problem is typically solved in a parsimony framework, where costs are assigned to duplication, transfer and loss events, and the goal is to find a reconciliation with minimum total cost. DTL-reconciliations can sometimes violate temporal constraints; that is, the inferred transfers may induce contradictory constraints on the dates for the internal nodes of the species tree. Such paradoxical reconciliations are called time-inconsistent [52].

Several formulations of the DTL-reconciliation problem exist: these include formulations where (i) the input species tree is undated and the goal is to find an optimal (minimum cost) time-consistent DTL-reconciliation, (ii) the input species tree is fully dated and the goal is to find an optimal time-consistent DTL-reconciliation, and (iii) the input species tree is undated and the goal is to find an optimal (not necessarily time-consistent) DTL-reconciliation. The first, undated formulation is known to be NP-hard [180, 254] and is therefore unlikely to be efficiently solvable. The second, fully dated formulation is known to be polynomially solvable [148], and the fastest known algorithms have
a time complexity of $O(mn^2)$ [52], where $m$ and $n$ denote the number of leaves in the gene tree and species tree, respectively. However, this formulation requires the use of a fully dated species tree, and accurately dating the internal nodes of a species tree is a notoriously difficult problem [212]. In contrast, the third formulation does not require a dated species tree and is known to be efficiently solvable in $O(mn)$ time [12], where $m$ and $n$ denote the number of leaves in the gene tree and species tree, respectively. Thus, due to its wider applicability and efficient solvability, we make use of formulation (iii) for the DTL-reconciliation problem, which assumes that the input species tree is undated and seeks an optimal (not necessarily time-consistent) DTL-reconciliation. We note, however, that MowgliNNI uses formulation (ii) of the problem and requires the input species tree to be fully-dated.

The DTL-reconciliation model can be easily extended to work with unrooted gene trees by considering all possible rootings of the gene tree and picking one that minimizes the reconciliation cost. Remarkably, even with the additional burden of evaluating all possible rootings, the time complexity of formulation (iii) of the DTL-reconciliation problem remains $O(mn)$ [12].

### B.2 Complexity analysis of TreeFix-DTL

Let $s$ denote the number of search iterations performed by the algorithm. Each of these iterations involves evaluating a constant number (100 in the current implementation) of tree topologies. The time spent on each tree is dominated by two additive components: first, the time complexity of performing the SH-test and of computing the likelihood for a given tree topology, and second, the time required to perform DTL-reconciliation. Let $m$ and $n$ denote the sizes of the gene tree and species tree, respectively, and let $a$ denote the alignment length. In our method, the calculations for the first component are performed heuristically by RAxML; we denote this cost as $r(a, m)$, since it depends on the alignment length and on the size of the input gene tree. The second component has a time complexity of $O(mn)$. Therefore, the total time complexity of our algorithm is $O(s(r(a, m) + mn))$.

### B.3 Parameters for simulated datasets

The low-DTL, medium-DTL and high-DTL gene trees had, on average, 52.3, 70.4, and 91.3 leaf nodes, 1.2, 2.8, and 5.0 duplications, 2.2, 5.5, and 9.9 transfers, and 2.1, 2.3, and 2.9 losses, respectively. We generated these datasets using a probabilistic model of gene duplication, transfer, and loss [253, 254] with duplication, transfer, and loss rates of 0.1, 0.2, and 0.2, respectively, for the low-DTL datasets, 0.2, 0.4, and 0.2, respectively, for the medium-DTL datasets, and 0.3, 0.6, and 0.2,
respectively for the high-DTL datasets. These rates represent the probability of a particular event type occurring per gene lineage per unit branch length on the species tree, and our 50-taxon species trees had an average branch length of 11.24 per tree (and 0.115 per edge). The ratio of duplications, transfers, and losses that we used was based on our analysis of a 4736 gene tree, 100 species dataset from David and Alm [37], which consists of predominantly prokaryotic species sampled broadly from across the tree of life. In general, we chose the duplication, transfer, and loss rates for which the reconstructed RAxML trees yielded duplication, transfer, and loss counts in a ratio roughly similar to those observed on that dataset. Specifically, the reconstructed RAxML trees for our low-DTL, medium-DTL, and high-DTL datasets showed, on average, 1.25, 2.91, and 4.89 duplications, 5.88, 10.04, and 15.49 transfers, and 4.88, 5.86, and 7.10 losses, respectively.

The choice of 173 for the amino acid alignment length is based on an analysis of the same biological dataset, where we observed that the median alignment length was 173, and the choice of the other alignment length, 333, is motivated by the fact that the typical prokaryotic gene length is approximately 1000 base pairs [138].

Our mutation rates are meant to span the range from slow evolving (or less diverged) gene families to fast evolving (or highly diverged) gene families. The average branch lengths for our low-DTL datasets, with mutation rates 1, 3, 5, and 10, were 0.11, 0.33, 0.55, and 1.1, respectively (in terms of average number of mutations per site). The corresponding mutation rates for the medium-DTL datasets were 0.10, 0.3, 0.5, and 1.0, respectively, and for the high-DTL datasets they were 0.095, 0.29, 0.48, and 0.95, respectively. The average branch length in the biological dataset was 0.264.

B.4 Parameters for gene tree reconstruction

TreeFix-DTL, AnGST, and MowgliNNI all require as input the costs for duplication, transfer, and loss events to be used with the DTL-reconciliation model. Based on existing literature [37], we set these costs to 2, 3, and 1 respectively for duplication, transfer, and loss.

Each of the species tree aware methods used in this study, TreeFix-DTL, AnGST, MowgliNNI, NOTUNG, and TreeFix, require as input a gene tree reconstructed using sequence-only methods. We used the RAxML tree as the input gene tree for these methods. We used extremely thorough search settings when building the RAxML trees, picking the best tree out of 10 full RAxML searches, and with 100 rapid bootstraps.

NOTUNG and MowgliNNI require the input gene tree to be labeled with bootstrap supports at each edge. We inferred these bootstrap support values based on the 100 rapid bootstraps from the RAxML runs. Both NOTUNG and MowgliNNI also require a bootstrap cutoff percentage which specifies the edges in the gene tree can be modified; for NOTUNG we used the default value of
90% of the maximum bootstrap, and for MowgliNNI we used a cutoff of 80% based on the author recommendation [172]. In addition, MowgliNNI requires that the input gene tree be rooted and that the species tree be fully dated. Our simulated species trees were fully dated and were used as input for MowgliNNI, and we ran MowgliNNI on all possible rootings of the input gene tree. AnGST requires as input a set of bootstrap replicates for the gene tree. While the authors of AnGST originally suggested that using more than 10 bootstraps provided limited additional improvements, we discovered that performance was significantly improved by using 100 bootstraps instead; thus, we used the 100 rapid bootstraps from RAxML as input to AnGST. TreeFix was run using a thorough search setting ("long" version as defined in Wu et al. [276]) that corresponds to the (default) search setting used for TreeFix-DTL.

B.5 Scalability and speed

To study the scalability and performance of the methods on larger datasets (with more taxa), we created datasets with 100- and 200-taxon species trees using the same methodology used to create the 50-taxon datasets. Gene trees were simulated using medium-DTL event rates and, for the 100-[200]-taxon datasets, had, on average, 144.3 [290.7] leaf nodes, 5.3 [10.1] duplications, 11.3 [21.3] transfers, and 4.3 [9.1] losses per gene tree. Sequences for these larger datasets were simulated for mutation rates 1 and 5 and sequence length 333.

We observed that the error rates of the TreeFix-DTL trees on these 100- and 200-taxon datasets (Figure B.3) are generally similar to those observed on the corresponding 50-taxon datasets (Figure 3.2), which suggests that the performance of the method does not deteriorate as the number of taxa in the input trees increases. Furthermore, TreeFix-DTL is fast enough to be easily applied to gene trees and species trees with hundreds of leaves (Table B.1). For instance, on the 100- and 200-taxon datasets, TreeFix-DTL required 13.8 and 43.5 hours per tree, respectively, on average (including the time to build the initial RAxML tree), when executed on a compute cluster with each node consisting of an 800 MHz AMD Opteron processor and 4 GB of RAM. This compares favorably to the 4.1 and 15.0 hours of average runtime required to build just the initial RAxML trees themselves (using the thorough search settings as previously described). Thus, gene tree reconstruction using TreeFix-DTL is only about three times as slow as doing a thorough sequence-only reconstruction using RAxML. AnGST is very efficient in general (assuming the bootstrap trees have already been computed), especially for the smaller (50- and 100-taxon) input instances, but, due to excessive memory requirements, we found it hard to run AnGST on datasets with more than 200 taxa (on a computer with 4 GB of RAM) (Table B.1). Runtimes for MowgliNNI were roughly twice as high as those for TreeFix-DTL.

138

B Models for gene duplication, transfer, and loss
TreeFix-DTL relies on a local search strategy to find more accurate gene trees, and its performance depends on the number of local search steps allowed during the search. By default, TreeFix-DTL executes 1000 local search steps per run, providing a reasonable trade-off between running time and accuracy for a wide range of tree sizes. To study the impact of using more local search steps, we also ran TreeFix-DTL with 5000 local search steps per run on the 100- and 200-taxon datasets. As expected, we observed that accuracy improved with the use of more local search steps, with the more exhaustive version having, on average, a 15% smaller error rate than the default parameterization of TreeFix-DTL. This additional increase in accuracy, however, comes at the cost of a five fold increase in running time. Nonetheless, the number of local search steps can be increased to obtain even better accuracy whenever accuracy is paramount, or when the number of leaves in the gene trees or species trees involved exceeds a few hundred.

B.6 Additional simulated datasets

B.6.1 Robustness to duplication, transfer, and loss ratios

Since the datasets considered above in the basic experimental setup have more transfers than duplications (as learnt from the biological dataset of David and Alm [37]), we also tested the performance of AnGST and TreeFix-DTL on datasets in which there were more duplications than transfers. These gene trees contained, on average, 5.2 duplications, 4.7 transfers, and 3.9 losses, and we simulated sequences for mutation rates 1 and 5 and sequence length 333. We observed that TreeFix-DTL is just as effective at inferring gene trees on these datasets as in our previous experiments, decreasing the error rate of the RAxML trees by 70.5% (Figure B.4A).

B.6.2 Robustness to short gene alignments

We observed that over 10% of the 4736 gene trees in the biological dataset of [37] had a multiple sequence alignment length of less than 75 amino acids. Thus, to test the ability of the methods to reconstruct accurate gene trees on short alignments, we created datasets, using the 50-taxon species trees and the same basic experimental setup, but with sequence length 75 (for medium-DTL and mutation rates 1 and 5). On these datasets with very short alignments, the error rates of the reconstructed gene trees were, unsurprisingly, substantially higher than for gene trees reconstructed using the length 173 or 333 alignments (Figure B.4B). For example, for the dataset with mutation rate 1, RAxML, AnGST, and TreeFix-DTL had normalized RF distances 0.185, 0.075, and 0.064, respectively, but in spite of these higher absolute error rates, the average error rate for the TreeFix-DTL trees (8.2%) is much smaller than the average error rate for the RAxML trees (21.2%).
B.6.3 Robustness to very high rates of duplication, transfer, and loss

In general, the absolute reconstruction accuracy of TreeFix-DTL and AnGST tends to decrease as the rate of duplication, transfer, and loss increases (Figure 3.2). To study the performance of the methods on datasets with very high rates of duplication, transfer, and loss, we created gene trees with very high rates of duplication, transfer, and loss on the 50-taxon species trees from the basic simulation setup. These gene trees contained, on average, 109 leaf nodes, 10.0 duplications, 20.6 transfers, and 6.9 losses, and we simulated sequences for mutation rates 1 and 5 and sequence length 333. Even with higher event rates, we observed that TreeFix-DTL was still quite effective at accurately reconstructing gene trees (Figure B.4C). For comparison, the average normalized RF distances for RAxML, AnGST, and TreeFix-DTL on these datasets were 0.065, 0.047, and 0.040 respectively.

B.6.4 Robustness to branch-specific mutation rate variation

The gene trees used in our simulation study were all generated using a process that assumes a uniform rate of mutation along the entire tree; thus, all the simulated gene trees are ultrametric. To study the performance of TreeFix-DTL on datasets with branch-specific mutation rate variation, we simulated additional 50-taxon datasets: for the medium-DTL gene trees from the basic simulation setup, we assigned new branch lengths randomly between 0.02 and 0.2, thus creating highly non-ultrametric gene trees. We then used these modified gene trees to generate sequence alignments of length 333 and with mutation rates 1 and 5. As with the basic simulation setup, we noticed that TreeFix-DTL decreases the error rate of the RAxML trees by over 75% on the non-ultrametric gene trees, suggesting that the performance of TreeFix-DTL is robust to branch-specific mutation rate variation.
B.7 Additional figures and tables

Figure B.1: Reconstruction error on simulated datasets of 50 taxa. Error rates in terms of the normalized Robinson-Foulds (RF) distance are shown for the gene trees reconstructed using RAxML, NOTUNG, TreeFix, MowgliNNI, AnGST, and TreeFix-DTL on the 24 simulated datasets.
Figure B.2: Effect of different event costs on reconstruction error for simulated datasets of 50 taxa. Error rates in terms of the normalized Robinson-Foulds (RF) distance are shown for the gene trees reconstructed using RAxML, TreeFix-DTL, and TreeFix-DTL-EqualCosts (the variant of TreeFix-DTL with costs for duplication, transfer, and loss events set to 1) on the 24 simulated datasets of 50 taxa.
Figure B.3: **Reconstruction error on simulated datasets of 100 and 200 taxa.** Error rates in terms of the normalized Robinson-Foulds (RF) distance are shown.

Figure B.4: **Reconstruction error on additional simulated datasets.** Error rates in terms of the normalized Robinson-Foulds (RF) distance are shown.

**B.7 Additional figures and tables**
Figure B.5: Number of inferred evolutionary events on simulated datasets of 50 taxa. The total number of duplications, transfers, and losses inferred (using DTL-reconciliation) are shown for gene trees reconstructed using RAxML, AnGST, and TreeFix-DTL on the 24 simulated datasets. Counts are for a specific rate of duplication, transfer and loss (low-, medium-, or high-DTL), a specific sequence alignment length (173 or 333 amino acids), and averaged over the four chosen rates of mutation (rates 1, 3, 5, and 10). The true (implanted) counts are also shown.
Table B.1: Average runtimes for gene tree reconstruction

<table>
<thead>
<tr>
<th>dataset</th>
<th>program</th>
<th>runtime $^a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>50-taxon, low DTL, sequence length 173</td>
<td>RAxML</td>
<td>0.8 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + AnGST</td>
<td>0.8 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + TreeFix-DTL</td>
<td>2.7 hr</td>
</tr>
<tr>
<td>50-taxon, medium DTL, sequence length 173</td>
<td>RAxML</td>
<td>1.3 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + AnGST</td>
<td>1.3 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + TreeFix-DTL</td>
<td>4.9 hr</td>
</tr>
<tr>
<td>50-taxon, high DTL, sequence length 173</td>
<td>RAxML</td>
<td>2.0 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + AnGST</td>
<td>2.0 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + TreeFix-DTL</td>
<td>7.9 hr</td>
</tr>
<tr>
<td>50-taxon, low DTL, sequence length 333</td>
<td>RAxML</td>
<td>1.5 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + AnGST</td>
<td>1.6 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + TreeFix-DTL</td>
<td>3.6 hr</td>
</tr>
<tr>
<td>50-taxon, medium DTL, sequence length 333</td>
<td>RAxML</td>
<td>2.1 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + AnGST</td>
<td>2.1 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + TreeFix-DTL</td>
<td>5.8 hr</td>
</tr>
<tr>
<td>50-taxon, high DTL, sequence length 333</td>
<td>RAxML</td>
<td>3.1 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + AnGST</td>
<td>3.1 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + TreeFix-DTL</td>
<td>10.1 hr</td>
</tr>
<tr>
<td>100-taxon, medium DTL, sequence length 333</td>
<td>RAxML</td>
<td>4.1 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + AnGST</td>
<td>4.1 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + TreeFix-DTL</td>
<td>13.8 hr</td>
</tr>
<tr>
<td>200-taxon, medium DTL, sequence length 333</td>
<td>RAxML</td>
<td>15.0 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + AnGST</td>
<td>16.0 hr</td>
</tr>
<tr>
<td></td>
<td>RAxML + TreeFix-DTL</td>
<td>43.5 hr</td>
</tr>
</tbody>
</table>

$^a$ RAxML was run with the command line option that produces 100 rapid bootstraps and executes the search heuristic 10 times. AnGST was run with 100 bootstrap trees as input, and TreeFix-DTL was run using its default settings. RAxML and TreeFix-DTL were run on a compute cluster (each run on a single core), and AnGST was run on a desktop computer with a 3.2 GHz processor and 4 GB of RAM.

$^b$ For the 50-taxon datasets, runtimes are averaged over the four datasets (each with 100 gene-tree/species-tree pairs) with mutation rates 1, 3, 5, and 10.

$^c$ For the 100-taxon and 200-taxon datasets, runtimes are averaged over the two datasets (each with 100 gene-tree/species-tree pairs) with mutation rates 1 and 5.
Appendix C

Models for gene duplication, loss, and deep coalescence

Within this appendix, all proofs are provided in Section C.3.

C.1 The labeled coalescent tree

Throughout this work, the term tree refers to a rooted binary tree. Let $T = (V(T), E(T))$ be a tree with a set $V(T)$ of nodes and a set $E(T)$ of directed branches $(u, v)$. Let $L(T) \subset V(T)$ denote the set of leaves of $T$, $I(T) = V(T) \setminus L(T)$ denote the set of internal nodes, and $r(T) \in I(T)$ denote the root node. For node $v$, let $c(v)$ denote its set of children, $p(v)$ denote its parent, $e(v)$ denote the branch $(p(v), v)$, and $T(v)$ denote the (maximal) subtree of $T$ rooted at $v$. Define $\leq_T$ to be the partial order on $V(T)$, where $u \leq_T v$ if $u$ is a node on the path between $r(T)$ and $v$ (inclusive), and define $\geq_T$ analogously, that is, $u \geq_T v$ if $v$ is a node on the path between $r(T)$ and $u$ (inclusive). For $u \neq v$, we say that $u$ is an ancestor of $v$, or $v$ is a descendant of $u$, if $u \leq_T v$. Given a non-empty set of nodes $\mathcal{U} \subset V(T)$, let $lca_T(\mathcal{U})$ denote the shared ancestor of $\mathcal{U}$ that is located farthest from the root. Finally, for $u, v \in V(T)$, let $d_T(u, v)$ denote the number of edges on the unique path from $u$ to $v$ in $T$.

Assume that we are given a gene tree (topology) $G$, a species tree (topology) $S$, and a leaf mapping $L_e : L(G) \to L(S)$.

**Definition C.1.1.** The LCT for $G$, $S$, $L_e$ is a tuple $(\mathcal{M}, \mathcal{L}, \mathcal{L}, \mathcal{O})$, where

- $\mathcal{M} : V(G) \to V(S)$ defines a **species map** that maps each node of $G$ to a node of $S$,
- $\mathcal{L}$ defines a **locus set**, that is, a set of loci that have evolved within the gene family,
\( \mathcal{L}: V(G) \rightarrow L \) defines a **locus map** that maps each node of \( G \) to a locus in \( L \), and

- \( \mathcal{O} \) defines a **partial order** on \( V(G) \). For \( s \in V(S) \), let \( \text{parent.loci}(s) \subset L \) denote the set of loci that yield a new locus in species \( s \). Then for \( s \in V(S) \) and \( l \in \text{parent.loci}(s) \), \( \mathcal{O} \) enforces a total order on the set of nodes \( \mathcal{O}(s, l) = \{ g : g \in V(G); g \neq r(G); M(g) = s; \mathcal{L}(p(g)) = l \} \).

For \( g, g' \in \mathcal{O}(s, l): g \neq g' \), define \( g \prec_\mathcal{O} g' \) if \( g \) precedes \( g' \), subject to the following constraints:

1. If \( g \in L(G) \), then \( M(g) = L_\text{e}(g) \).
2. If \( g \in I(G) \), then for \( g' \in c(g) \), \( M(g) \leq_\mathcal{S} M(g') \).
3. For \( g, g' \in L(G): g \neq g' \), if \( M(g) = M(g') \), then \( \mathcal{L}(g) \neq \mathcal{L}(g') \).
4. For \( l \in L \), \( \exists g \in V(G): \mathcal{L}(g) = l \).
5. For \( l \in L \), let \( N(l) = \{ g : g \in V(G); g \neq r(G); \mathcal{L}(g) = l; \mathcal{L}(p(g)) \neq l \} \). Then \( |N(l)| \leq 1 \), where equality holds everywhere except for \( l = \mathcal{L}(r(G)) \).
6. For \( s \in V(S), l \in \text{parent.loci}(s), g, g' \in \mathcal{O}(s, l): g \neq g' \), if \( g \prec_\mathcal{O} g' \), then \( g \not\mathcal{L}_G g' \).

Constraint 1 above ensures that \( M \) is consistent with the known leaf mapping, and constraint 2 imposes on \( M \) the temporal constraints implied by \( S \). Constraint 3 imposes on \( \mathcal{L} \) that genes within the same extant species must belong to different loci, constraint 4 restricts \( L \) to the range of \( L \), and constraint 5 imposes that for every locus, the nodes that map to that locus must belong to a single component so that each locus is created only once. Finally, constraint 6 imposes on \( \mathcal{O} \) the temporal constraints implied by \( G \).

Before we develop the LCT further, note that from constraint 4, the locus set \( L \) is defined by the locus map \( \mathcal{L} \); therefore, in the main manuscript and in the remainder of this work, we reduce the LCT notation to a tuple \( \langle M, \mathcal{L}, \mathcal{O} \rangle \). In addition, as mentioned in the main manuscript, the LCT also includes implied speciation nodes (Section C.1.1).

### C.1.1 Implied speciation nodes

For \( g \in I(G) \), let \( \text{spec}(g) \) be a boolean variable that denotes whether \( g \) is a speciation node. That is \( \text{spec}(g) \) is true if \( \forall g' \in c(g), M(g) \neq M(g') \) and false otherwise. In this section, we discuss how to add speciation nodes that are hidden in the gene tree; such implied speciation nodes are added to \( G \) and included in \( M, \mathcal{L}, \) and \( \mathcal{O} \).
Implied speciation nodes are necessary when branches of the gene tree span multiple branches of the species tree. To break up gene tree branches, locate all $g \in V(G) \setminus \{r(G)\}$ such that (a) $p(M(g)) \neq M(p(g))$ or (b) $\neg spec(p(g))$ and $M(g) \neq M(p(g))$. Replace edge $(p(g), g)$ with a new node $v$ and two new edges $(p(g), v)$ and $(v, g)$, and set $M(v) = p(M(g))$. Repeat until there exist no $g$ exist that satisfy the above conditions.

In addition, by definition, if a speciation node has multiple children, then these children must map to different species. To account for this, locate all $g \in I(G)$ such that $spec(g)$ and $M(g') = M(g'')$, where $g'$ and $g''$ denote the children of $g$. (Note that no such $g$ exist if $M$ is the LCA reconciliation. This is because in the LCA reconciliation, if $M(g') = M(g'') = s$, then $M(g) = s$ so that $spec(g)$ is false; that is, the two conditions are mutually exclusive.) For $g' \in c(g)$, replace edge $(g, g')$ with a new node $v$ and two new edges $(g, v)$ and $(v, g')$, and set $M(v) = M(g)$.

**Delayed speciation nodes**

A delayed speciation node is a special type of implied speciation node. Unlike previous implied speciation nodes, delayed speciation nodes are not hidden as a result of gene loss or deep coalescence. Rather, these are necessary to model delays between speciation and coalescence. That is, while gene tree nodes are found within a species tree branch, $M$ maps $V(G)$ to $V(S)$ rather than $E(S)$. In effect, the LCT rounds speciation nodes to the breakpoint between species (Figure C.2A). To correct for this, the LCT includes delayed speciation nodes (Figure C.2B). To add these, locate all $g \in I(G)$ such that $spec(g)$ and $|c(g)| > 1$. For $g' \in c(g)$, replace edge $(g, g')$ with a new node $v$ and two new edges $(g, v)$ and $(v, g')$, and set $M(v) = M(g)$. By default, delayed speciation nodes are not included in the LCT when inferring a MP LCT using DLCpar. In the remainder of this section, we discuss how this affects evolutionary events and our reasoning behind this choice.

**Definition C.1.2.** Within each species, consider two lineages that exist to the speciation event such that, looking backwards in time, the two lineages have coalesced in a locus in this species. A post-coalescence duplication is a duplication has occurred in the locus in the time between the speciation event and the coalescence event. Formally, for $g \in I(G), g', g'' \in c(g): g' \neq g''; M(g) = M(g') = M(g''); spec(g') \land spec(g''); |c(g')| = 1 \land |c(g'')| = 1$, then $(g, g')$ and $(g, g'')$ are two lineages that have existed to the speciation event and coalesced at $g$ in locus $l = L(g)$ in species $s = M(g)$. A post-coalescence duplication is one that has occurred along $(g, g')$ so that $L(g') \neq l$ or along $(g, g'')$ so that $L(g'') \neq l$.

**Remark.** To understand the last requirement $|c(g')| = 1 \land |c(g'')| = 1$, note that if $|c(g')| > 1$ or $|c(g'')| > 1$, then it is the lineages in $s$ immediately following $g'$ or $g''$, respectively, that exist to the speciation event.

### C.1 The labeled coalescent tree
We now consider the relationship between delayed speciation nodes and post-coalescence duplications (Figure C.2C).

**Proposition C.1.1.** Delayed speciation nodes are required if and only if post-coalescence duplications exist.

Finally, the concept of a post-coalescence duplication is not intuitive; therefore, we relate it to LCA reconciliation between the locus tree and species tree.

**Proposition C.1.2.** Assume that losses incur a positive cost. If the reconciliation between the locus tree and species tree is the LCA mapping, then there are no post-coalescence duplications.

In simulation, we found LCA reconciliation between the locus tree and the species tree occurred in at least 99.6% (94.2%) of gene families when duplications and losses are simulated at $1 \times$ ($4 \times$) the rate estimated from real data. Using Proposition C.1.2, this is a lower bound for the number of gene families with no post-coalescence duplications.

### C.1.2 Inferring evolutionary events in the LCT

For reasons that will become clear later, we consider evolutionary events separately for each species tree branch, but first, we define some useful functions for relating how gene tree nodes map within species tree branches. (To reduce notational complexity, in the remainder of this work, unless otherwise noted, the dependency on $G$, $S$, $M$, $L$, and $O$ are implicit in our function definitions.)

For $s \in V(S)$, let $nodes(s)$ be the set of gene tree nodes mapped to $s$ and $leaves(s)$ and $roots(s)$ be the set of gene tree nodes mapped to the bottom and top of species tree branch $e(s)$, respectively. That is, $nodes(s) = \{ g : g \in V(G); M(g) = s \}$, $leaves(s) = \{ g : g \in nodes(s); \ (g \in L(G) \land \forall g' \in c(g), g' \notin nodes(s)) \}$, and $roots(s) = leaves(p(s))$ if $s \neq r(S)$ and $roots(s) = \emptyset$ otherwise. (Note that under these definitions, $nodes(s)$ does not include $roots(s)$.) Finally, for a set of gene tree nodes $\emptyset \subset V(G)$, let $loci(\emptyset) = \{ l : l = L(g); \ g \in \emptyset \}$ be the set of associated loci.

**Definition C.1.3.** For $G$, $S$, $Le$ and LCT $\alpha = \langle M, L, O \rangle$, the following evolutionary events are defined:

- **Duplication** For $s \in V(S)$, let $D(s) = \{ g : g \in nodes(s); \ g \neq r(G); \ L(g) \neq L(p(g)) \}$ be the set of gene tree nodes within $e(s)$ that are the result of a duplication. Then $nD(s) = |D(s)|$ and $nD_\alpha = \sum_{s \in V(S)} nD(s)$ are the number of duplications in a species and in the LCT, respectively.

- **Loss** For $s \in V(S)$, let $lost.loci(s) = \{ l : l \in loci(roots(s) \cup nodes(s)) \setminus loci(leaves(s)) \}$ be
Algorithm S1 Counting extra lineages due to duplications

**Input:** $G, s \in V(S), l \in parent\_loci(s), (\mathcal{M}, \mathcal{L}, \mathcal{O})$

**Output:** $nCD(s, l)$

1. Let $\hat{O}(s, l)$ be the ordered set of nodes such that the set of nodes in $\hat{O}(s, l)$ is equal to $O(s, l)$ and the order of nodes in $\hat{O}(s, l)$ satisfies $\mathcal{O}$. That is, for $g, g' \in \hat{O}(s, l)$: $g \neq g'$, $g$ precedes $g'$ if and only if $g <_\mathcal{O} g'$.

2. Let $\text{start\_lineages}(s, l) = \{(g, g') : (g, g') \in E(G); (g \in D(s) \lor g = \tau(G) \lor g \in \text{roots}(s)); g' \in \text{nodes}(s); \mathcal{L}(g) = l\}$ be the set of starting lineages.

3. Initialize $\text{CD}(s, l) = \text{start\_lineages}(s, l)$ and $nCD(s, l) = 0$.

4. for $g' \in \hat{O}(s, l)$ do {traverse nodes using the partial order $\mathcal{O}$}

5. Set $l' = \mathcal{L}(g')$.

6. if not $g' \in L(G) \land l' = l$ then {node is not an extant gene in the same locus}

7. Remove $(p(g'), g')$ from CD$(s, l)$.

8. if $l' = l$ then {node is mapped to the same locus}

9. $\forall g'' \in c(g'),$ add $(g', g'')$ to CD$(s, l)$.

10. else {node is mapped to a new locus, that is, a duplication has occurred}

11. Add extra$(\text{CD}(s, l))$ to nCD$(s, l)$.

the set of loci lost within $e(s)$. Then $nL(s) = |\text{lost\_loci}(s)|$ and $nL_\alpha = \sum_{s \in V(S)} nL(s)$ are the number of losses in a species and in the LCT, respectively.

- **Deep coalescence** In the following, for a set $\mathcal{X}$ of lineages, define the number of extra lineages as $\text{extra}(\mathcal{X}) = \max(0, |\mathcal{X}| - 1)$.

  - ILS at to speciations: At the top of species tree branch $e(s)$ for $s \in V(S)$, let the lineages in locus $l \in \text{loci}(\text{roots}(s))$ be given by $\text{CS}(s, l) = \{(g, g') : (g, g') \in E(G); g \in \text{roots}(s); \mathcal{L}(g) = l\}$. Then $nCS(s) = \sum_{l \in \text{loci}(\text{roots}(s))} \text{extra}(\text{CS}(s, l))$ and $nCS_\alpha = \sum_{s \in V(S)} nCS(s)$ are the number of extra lineages induced by speciations in a species and in the LCT, respectively.

  - ILS at to duplications: For $s \in V(S)$, let $parent\_loci(s) = \{l : l = \mathcal{L}(p(g)); g \in D(s)\}$. Then, for $s \in V(S)$, $l \in parent\_loci(s)$, the number $nCD(s, l)$ of contemporary lineages with locus $l$ at the time of a duplication, summed over all duplications in $s$ and $l$, can be determined by Algorithm S1, and $nCD(s) = \sum_{l \in parent\_loci(s)} nCD(s, l)$ and $nCD_\alpha = \sum_{s \in V(S)} nCD(s)$ are the number of extra lineages induced by duplications in a species and in the LCT, respectively.

Finally, $nC(s) = nCS(s) + nCD(s)$ and $nC_\alpha = nCS_\alpha + nCD_\alpha = \sum_{s \in V(S)} nCS(s) + nCD(s)$ are the number of extra lineages in a species and in the LCT, respectively.

C.1 The labeled coalescent tree

151
C.2 The DLCpar algorithm

C.2.1 Problem statement

Definition C.2.1. Given $G$, $S$, $L_e$, $D$, $L$, $C$, the reconciliation cost for LCT $\alpha = (M, L, O)$ is $R_\alpha = D \cdot nD_\alpha + L \cdot nL_\alpha + C \cdot nC_\alpha$.

Problem C.2.1. Given $G$, $S$, $L_e$, $D$, $L$, $C$, the MP LCT is $\alpha^* = (M^*, L^*, O^*) = \arg\min_\alpha R_\alpha$.

Problem C.2.2. Given $G$, $S$, $L_e$, $D$, $L$, $C$, the restricted MP LCT is $\alpha^* = (M^*, L^*, O^*) = \arg\min_\alpha R_\alpha$ subject to the condition that the reconciliation between the locus tree and species tree is the LCA mapping.

For Problems C.2.1 and C.2.2, $\alpha^*$ is not necessarily unique. Furthermore, Problem C.2.2 enforces an additional constraint on the MP LCT, in particular, that the reconciliation between the locus tree and the species tree is the LCA mapping. To understand this restriction, recall that the LCT is a simplified representation of the three-tree DLCoal model. Thus, inferring a LCT is equivalent to inferring three components: a locus tree (with annotated daughter lineages), a reconciliation between the gene tree and locus tree, and a reconciliation between the locus tree and species tree. Naively, in a MP LCT, we could enforce that the two reconciliations must be the LCA mapping, as we know that the LCA mapping between the gene tree and locus tree minimizes the number of extra lineages, and the LCA mapping between the locus tree and species tree minimizes the number of duplications and losses. However, these two reconciliations are not independent; in particular, there exist reconciliations that violate at least one of the LCT constraints. Therefore, to make our algorithm more efficient, we constrain the reconciliation between the locus tree and species tree so that rather than inferring a global MP LCT (Problem C.2.1), we infer a restricted MP LCT (Problem C.2.2). Determining the gap between the global and restricted MP solutions is non-trivial, so instead, we used simulations to ask how often the true LCT satisfies the additional constraint: we found that the reconciliation between the locus tree and species tree is the LCA mapping in at least 99.6% (94.2%) of gene families when duplications and losses are simulated at $1 \times (4 \times)$ the rate estimated from real data. For simplicity, in the main manuscript and in the remainder of this work, we have omitted the term restricted and understand that a MP LCT always refers to a restricted MP LCT.

C.2.2 Main algorithm

In the remainder of this section, to keep track of the current LCT $\alpha = (M, L, O)$, dependencies on $\alpha$ (or parts of $\alpha$) are explicit in our function definitions and will be denoted using subscripts.
Algorithm S2 DLCPAR

Input: $G$, $S$, $L_e$, $D$, $L$, $C$
Output: $M^*$, $L^*$, $O^*$
1: Use LCA to find $M^*$.
2: Prune the species tree to $S = S(M^*(r(G)))$.
3: Add implied speciation nodes and factor $G$ (Figure 4.3A).
4: for $s \in \text{preorder}(\hat{S})$ do
5: Use $\text{ENUMERATELOCi}$ to find the set $L(s)$ of locus maps for nodes in this species tree branch. That is, each item of $L$ is one choice for $L*$: $\text{roots}_{M^*}(s) \cup \text{nodes}_{M^*}(s) \rightarrow L$ (Figure 4.3B, top).
6: for $L* \in L(s)$ do
7: Use $\text{FINDORDER}$ to find an optimal order (associated with this locus map) $O^*_L$.
8: Compute the reconciliation cost for this species. For $\alpha = \langle M^*, L*, O^*_L \rangle$, $R\alpha(s) = D \cdot nD_{\alpha}(s) + L \cdot nL_{\alpha}(s) + C \cdot (nCS_{\alpha}(s) + nCD_{\alpha}(s))$ (Figure 4.3B, bottom).
9: Use $\text{REMAPLocI}$ to map the set $\mathcal{L}(s)$ of absolute locus maps to a set $\mathcal{A}(s)$ of relative locus maps. Each item of $\mathcal{A}(s)$ is one choice for $\mathcal{RL}_*: \text{leaves}_{M^*}(s) \rightarrow \mathcal{L}$. This also returns $\mathcal{RA}_*: \mathcal{A}(s) \rightarrow L(s)$, which maps each relative locus map to an optimum underlying absolute locus map (Figure 4.3C), and a cost function $C_\alpha(l, l')$, which denotes the minimum cost of assigning $l'$ and $l$ to roots$_{M^*}(s)$ and leaves$_{M^*}(s)$.
10: Use $\text{FINDOPTIMUM}$ to find optimal relative locus maps $\mathcal{RL}^*_L \forall s \in V(\hat{S})$ (Figure 4.3E).
11: For $s \in V(\hat{S})$, find $L^*_s$ using $\mathcal{RL}^*_L$ and $\mathcal{RA}_s$. Then lookup $O^*_L$.
12: Put together $L^*_s$ and $O^*_L \forall s \in V(\hat{S})$ to determine $L^*$ and $O^*$.

The main algorithm is shown in Algorithm S2. Throughout, if multiple optima exist, one is chosen randomly.

C.2.3 Inferring an optimal species map

Without a species map, we cannot determine how the gene tree fits within the species tree. Thus, in this step of the algorithm, all internal nodes $g \in I(G)$ of the gene tree have more than one child; that is, the LCT does not contain implied speciation nodes. As the leaf mapping $L_e$ of extant genes to extant species is known, our goal is to map each internal gene tree node $g \in I(G)$ to a species tree node $s \in V(S)$.

Theorem C.2.1. If $\alpha^* = \langle M^*, L^*, O^* \rangle$ is a MP LCT, then an optimal species map $M^*$ is the LCA mapping.

Theorem C.2.1 allows us to infer an optimal species map $M^*$ using the LCA mapping; for completeness, we reproduce the LCA algorithm (Algorithm S3). Once this is done, implied speciation nodes are added, and the speciation nodes are used to factor the gene tree. That is, for $s \in V(S)$, we decompose $G$ into disjoint subtrees $\text{subtrees}(s)$ that evolve within the species tree branch $e(s)$; these subtrees can be partitioned from $G$ using roots$(s)$ and leaves$(s)$. In addition, for the remainder of the algorithm, we work with the pruned species tree $\hat{S} = S(M^*(r(G)))$ as the gene tree evolves.

C.2 The DLCpar algorithm
Algorithm S3 LCA

Input: $G$, $S$, $L_e$
Output: $M^*$

1: for $g \in \text{postorder}(G)$ do
2:  if $g \in L(G)$ then
3:    Set $M^*(g) = L_e(g)$.
4:  else
5:    Set $M^*(g) = \text{lca}_S(\{M^*(g'), M^*(g'')\})$, where $g'$ and $g''$ denote the children of $g$.

C.2.4 Enumerating locus maps

Let us start by considering the space of locus maps that can yield MP LCTs:

**Proposition C.2.2.** There always exists a MP LCT that satisfies the following properties:

- The locus changes at most once along any branch of the gene tree. (This constraint is implicit in the LCT definition.)
- The locus changes in at most one child for nodes internal to a species tree branch.
- The number of loci is at most one more than the minimum number of inferred duplications under the duplication-loss model.

For each gene tree branch $e \in E(G)$, let $\text{changed}(e)$ denote whether the locus changed along $e$; that is, if $e = (g, g')$, then $\text{changed}(e)$ is true if $L(g) \neq L(g')$ and false otherwise. Then one option for enumerating locus maps is to consider all possible labels $\text{changed}(e)$ for each gene tree branch $e \in E(G)$, where these labels must satisfy Proposition C.2.2. We can then assign $r(G)$ to an arbitrary locus and recur down $G$ to determine the locus assignments for all $g \in V(G)$. However, this brute-force approach neglects to reuse information across locus maps. In particular, knowledge of the loci at the speciation nodes would allow us to infer the loci within each species tree branch independently of one another; thus, rather than considering the entire locus map at once, we focus separately on each species tree branch.

For $s \in V(S)$, let $L(s)$ denote the set of locus maps for the nodes in species tree branch $e(s)$; that is, each item of $L(s)$ is a species-specific locus map $L_s : \text{roots}(s) \cup \text{nodes}(s) \rightarrow L$ defined only on the set of gene tree nodes within $e(s)$ (inclusive of the root nodes). We can determine $L(s)$ using $\text{subtrees}(s)$ and $\text{changed}(e)$. In particular, we start by assigning $r(G)$ to an arbitrary locus. Then for each species tree node $s$ in pre-order traversal, we consider all labels $\text{changed}(e)$ for $e \in E(T), T \in \text{subtrees}(s)$, where these labels must satisfy Proposition C.2.2. Next, if $s = r(S)$,
Algorithm S4 ENUMERATELOCIS

**Input:** G, M*, s ∈ V(Ś), RŁ(p(s)) if s ≠ r(Ś)

**Output:** L(s)

1. Initialize L(s) = ∅.
2. for T ∈ subtreesM*(s) do
3. Assign all combinations C(T) of changed(e)∀e ∈ E(T) subject to the constraints of Proposition C.2.2.
4. if s = r(Ś) then
5. Initialize RL'(r(G)) = l0.
6. Set RŁp to be the set (of size 1) that includes RL'.
7. else
8. Set RŁp = RŁ(p(s)). (each item of RŁp is a choice for RL': leavesM*(p(s)) → Ł)
9. for RL' ∈ RŁp do
10. if not s ∈ L(Ś) ∧ ∃g, g' ∈ leavesM*(s): g ≠ g' and Ls(g) = Ls(g') then {check Constraint 3 of the LCT definition}
11. if not L0 defined on rootsM*(s) using RL'. Else set Ls defined on r(G) using RL'.
12. Find Ls defined on nodesM*(s) using changed(e)∀T ∈ subtreesM*(s), e ∈ E(T). If changed(e) is true, the new locus must not currently exist in order to satisfy Constraint 5 of the LCT definition.
13. Add Ls to L(s).

we combine changed with knowledge of the locus at r(G) to find L(s). If s ≠ r(S), we iterate over the locus maps L(p(s)) for the parent species. Note that each locus map L' ∈ L(p(s)) defines a locus assignment for leaves(p(s)), which equals the locus assignment for roots(s). Thus, combining changed with L(p(s)) determines L(s). While this implementation is currently equivalent to the brute-force approach, we shall see later how L(p(s)) can be replaced by a set of relative locus maps RŁ(p(s)) to reduce redundancy.

The pseudo-code for this algorithm differs slightly from our description here as we have used RŁ(p(s)) in place of L(p(s)) for the input (Algorithm S4). Furthermore, for clarity, we do not include the third locus map constraint (on the maximum number of loci in a MP LCT) in our algorithm; to include this constraint, we must keep track of the cumulative (minimum) number of duplications for each locus map as we traverse down the species tree.

C.2.5 Inferring an optimal order for each locus map

At this point, an optimal species map M* and the set L(s) of species-specific locus maps for each s ∈ V(S) is known. What remains to be determined is an optimal order O* for each locus map Ls ∈ L(s).

Proposition C.2.3. Let α* = (M*, L*, O*) be a MP LCT. If M* and L* are known, then an optimal associated order O* = arg minO R(M*, L*, O) is one such that for s ∈ V(S), l ∈ parent. loci(s),
Algorithm S5 \textsc{FindOrder}

\textbf{Input:} $G, \mathcal{M}^*, s \in V(\tilde{S}), L_s$

\textbf{Output:} $O^*_{L_s}$

1: for $l \in \text{parent}\_\text{loci}_{\mathcal{M}^*, L_s}(s)$ do
2: \hspace{1em} Initialize $O^*_{L_s}(s, l)$ to an empty ordered set.
3: \hspace{1em} Find $\text{start}\_\text{lineages}_{\mathcal{M}^*, L_s}(s, l)$.
4: \hspace{1em} Initialize the set of nodes that require ordering $X = \text{start}\_\text{lineages}_{\mathcal{M}^*, L_s}(s, l)$.
5: \hspace{1em} Initialize the set of current lineages $\mathcal{X} = \text{start}\_\text{lineages}_{\mathcal{M}^*, L_s}(s, l)$.
6: For $g \in X$, let $d(g) = d_G(r, g)$, where $e(r) \in \mathcal{X}$ and $r \leq_G g$, be the length of the path to a current lineage.
7: while $\exists g \in X: L_s(g) \neq l$ do \{extra lineages can be incurred due to duplications\}
8: \hspace{1em} Select $g$ from among the set \{$g : g = \arg \min_{g' \in X : L_s(g') \neq l} d(g')$\} of nodes mapped to a different locus with shortest path to a current lineage.
9: \hspace{1em} if $d_G(r, g) > 1$ then \{$g$ cannot be immediately added\}
10: \hspace{1em} for $g'$ along the path from $r$ to $g$ (non-inclusive) do
11: \hspace{2em} Add $g'$ to $O^*_{L_s}(s, l)$. Remove $g'$ from $X$. Remove $(p(g'), g')$ from $\mathcal{X}$, and $\forall g'' \in c(g')$, add $(g', g'')$ to $X$.
12: \hspace{1em} Add $g$ to $O^*_{L_s}(s, l)$. Remove $g$ from $X$. Remove $(p(g), g)$ from $\mathcal{X}$. ($g$ maps to a different locus so $c(g)$ do not affect $\mathcal{X}$.)
13: \hspace{1em} while $|\mathcal{X}| > 0$ do \{there remain nodes to be ordered\}
14: \hspace{1em} Select $g$ from among the set \{$g : g \in X; d(g) = 1$\} of immediate descendants of current lineages.
15: \hspace{1em} Add $g$ to $O^*_{L_s}(s, l)$. Remove $g$ from $X$. Remove $(p(g), g)$ from $\mathcal{X}$, and $\forall g' \in c(g)$, add $(g, g')$ to $\mathcal{X}$.
16: Combine $O^*_{L_s}(s, l) \forall l \in \text{parent}\_\text{loci}_{\mathcal{M}^*, L_s}(s)$ to find an optimal order $O^*_{L_s}$.

$O^*$ enforces a total order on $O(s, l)$ such that duplications are as early in the species tree branch as possible.

Thus, for each locus map $L_s \in \mathcal{L}(s)$, rather than using brute-force to enumerate all orders then choosing an optimal $O^*_{L_s}$, Proposition C.2.3 allows us to intelligently infer $O^*_{L_s}$ (Algorithm S5).

C.2.6 Computing relative locus maps

For $s \in V(S)$ and $L_s \in \mathcal{L}(s)$, the reconciliation cost of $\langle \mathcal{M}^*, L_s, O^*_{L_s} \rangle$ can finally be computed. For the last step within the species tree branch, let $\mathcal{R}\mathcal{L}_s \colon \text{leaves}(s) \rightarrow \overline{L}$ denote a species-specific relative locus map defined on the leaf nodes of species $s$. For $s \in V(S)$, we remap the set $\mathcal{L}(s)$ of absolute locus maps to a set $\mathcal{R}\mathcal{L}(s)$ of relative locus maps. Additionally, we keep a mapping $\mathcal{R}\mathcal{A}_s \colon \mathcal{R}\mathcal{L}(s) \rightarrow \mathcal{L}(s)$ of each relative locus map to its optimal underlying (absolute) locus map and a lookup table $C_s(l_t, l_b)$ that denotes the minimum cost of assigning $l_t$ and $l_b$ to $\text{roots}_{\mathcal{M}^*}(s)$ and $\text{leaves}_{\mathcal{M}^*}(s)$, respectively (Algorithm S6).
Algorithm S6 RECALI

Input: $G$, $M^*$, $s \in V(\hat{S})$, $L(s)$, $R_{(M^*,L^*,O^*_L)}(s) \forall L_s \in L(s)$
Output: $RL(s)$, $RA_s$, $C_s$

1: Initialize $RL(s) = \emptyset$.
2: Initialize the locus count $n = 0$.
3: Arbitrarily order $\text{leaves}_{M^*}(s)$, for example, each $g \in \text{leaves}_{M^*}(s)$ is ordered by its index in a preorder traversal of $G$.
4: for $L_s \in L(s)$ do
5: Set $g$ to be the first node in $\text{leaves}_{M^*}(s)$.
6: Set $RL_s(g) = \text{in}$.
7: Set $\text{map}(L_s(g)) = 1$.
8: Increment $n$.
9: for each successive $g \in \text{leaves}_{M^*}(s)$ do
10: if $L_s(g)$ is not already defined in $\text{map}$ then
11: Set $\text{map}(L_s(g)) = n$.
12: Increment $n$.
13: Set $RL_s(g) = \text{map}(L_s(g))$.
14: Add $RL_s$ to $RL(s)$.
15: Set $\text{cost}(L_s, RL_s) = R_{(M^*,L^*,O^*_L)}(s)$.
16: for $RL_s \in RL(s)$ do
17: Find $\hat{L}_s = \arg \min_{L_s} \text{cost}(L_s, RL_s)$. Set $RA_s(RL_s) = \hat{L}_s$.
18: Find a relative locus map $RL: \text{roots}_{M^*}(s) \rightarrow \hat{L}$ from $\hat{L}_s(g) \forall g \in \text{roots}_{M^*}(s)$.
19: Set $l_t = RL(g) \forall g \in \text{roots}_{M^*}(s)$, $l_b = RL_s(g) \forall g \in \text{leaves}_{M^*}(s)$, $C_s(l_t, l_b) = \text{cost}(\hat{L}_s, RL_s)$.

C.2.7 Inferring an optimal locus map and order

At this point, the set of relative locus maps are known for all speciation nodes, allowing us to use dynamic programming to determine an optimal relative locus map for the speciation nodes (Algorithm S7). For $s \in V(S)$, let $F(s, l, p)$ denote the cost-to-go of assigning relative loci $l$ at the top ($p = t$) or bottom ($p = b$) of species tree branch $e(s)$, where the cost-to-go is defined as the minimum total cost along all descendant species tree branches. In the forward phase, we perform a post-order traversal of the species tree, where at each node $s \in V(S)$, we first determine the cost-to-go of assigning $l_t$ at $\text{leaves}(s)$, then determine the cost-to-go of assigning $l_t$ at $\text{roots}(s)$, for all possible choices of relative loci $l_t$ and $l_b$. At the root of the species tree, the minimum cost solution is selected. Then, in the backward phase, we perform a pre-order traversal to traceback down the species tree, where at each node $s \in V(S)$, we assign an optimal relative locus map $RL^*_s$.

Finally, as stated in the main text, for each $s \in V(\hat{S})$, we can use $RL^*_s$ in conjunction with $RA_s$ to look up $L^*_s$, which in turn can be used to look up $O^*_L$. These are the subparts of $L^*$ and $O^*$, and together with our previously inferred optimal species map $M^*$, constitutes the MP LCT.
Algorithm S7 FINDOPTIMUM

Input: \( G, \hat{S}, \mathcal{RL}(s) \forall s \in V(\hat{S}), C_s \forall s \in V(\hat{S}) \)

Output: \( RL^*_s \forall s \in V(\hat{S}) \)

1: for \( s \in \text{postorder}(\hat{S}) \) do
2:   for \( RL_s \in \mathcal{RL}(s) \) do
3:     Set \( l = RL_s(g) \forall g \in \text{leaves}_{M^*}(s) \).
4:     if \( s \in L(\hat{S}) \) then
5:       Set \( F(s, l, b) = 0 \).
6:     else
7:       Set \( F(s, l, b) = F(s', l, t) + F(s'', l, t) \), where \( s' \) and \( s'' \) denote the children of \( s \).
8:     end if
9:   end for
10: end for
11: Set \( \mathcal{RL}_p = \mathcal{RL}(p(s)) \) if \( s \neq r(\hat{S}) \). Else set \( \mathcal{RL}_p \) to be the set (of size 1) with element \( l_t \) such that \( l_t \) is empty (that is, at the root of the species tree, \( \text{roots}_{M^*}(s) \) is empty so no loci are assigned).
12: for \( RL' \in \mathcal{RL}_p \) do
13:   Set \( l_t = RL'(g) \forall g \in \text{leaves}_{M^*}(p(s)) \) if \( s \neq r(\hat{S}) \). Else set \( l_t \) to be empty.
14:   Find \( l_b = \text{arg min}_{l_b \in \mathcal{RL}(s)} F(s, l_b', b) + C_s(l_t, l_b') \).
15:   Set \( F(s, l_t, t) = F(s, l_b, b) + C_s(l_t, l_b) \).
16:   Set \( TB(s, l_t) = l_b \). \{traceback pointer\}
17: end for
18: for \( s \in \text{preorder}(\hat{S}) \) do
19:   if \( s = r(\hat{S}) \) then
20:     Set \( l_t \) to be empty. \{locus map at top of species tree root is known\}
21:   else
22:     Set \( l_t \) using \( RL^*_p(s) \). \{locus map at top of species is equal to locus map at bottom of parent species\}
23:   end if
24:   Set \( RL^*_s \) using \( l_b = TB(s, l_t) \). \{use traceback to assign locus map at bottom of species\}

C Models for gene duplication, loss, and deep coalescence
C.3 Proofs

Proof of Proposition C.1.1

Proof. We first prove that if post-coalescence duplications exist, then delayed speciation nodes are required. This follows directly from Definition C.1.2 by observing that \( g' \) and \( g'' \) must be delayed speciation nodes.

Next we prove that if delayed speciation nodes are required, then post-coalescence duplications exist. This is equivalent to stating that if post-coalescence duplications do not exist, then delayed speciation nodes are not required. Intuitively, this should be true because an absence of post-coalescence duplications means that nothing has occurred in the locus between the speciation event and the coalescence event; thus, we do not need to model the delay between speciation and coalescence. Formally, since \( g' \) and \( g'' \) are delayed speciation nodes, each has a single child \( cg' \) and \( cg'' \), respectively. Furthermore, if there are no post-coalescence duplications, then \( L(g) = L(g') = L(g'') \). These two observations, in conjunction with \( M(g) = M(g') = M(g'') \), allows us to replace node \( g' \) and edges \( (g,g'), (g',cg') \) with a single edge \( (g,cg') \) and replace node \( g'' \) and edges \( (g,g''), (g'',cg'') \) with a single edge \( (g,cg'') \), thus removing the delayed speciation nodes \( g', g'' \).

Remark. Note that while node \( g \) has been rounded to the point of speciation, the actual coalescence at \( g \) may still occur prior to the speciation. This is an important point as the time of coalescence may affect the number of extra lineages. The discussion here simply shows that, assuming that post-coalescence duplications are not allowed, we do not need to add delayed speciation nodes to explicitly model the delay between speciation and coalescence.

Proof of Proposition C.1.2

Proof. Under the assumption that losses incur a positive cost, for a given locus tree and species tree, LCA reconciliation is the unique solution that minimizes the total duplication and loss cost [86]. Thus, the proposition is equivalent to the following: for a LCT, consider the associated locus tree and reconciliation between the locus tree and the species tree (both of these are implicit in a LCT); if the reconciliation has minimum duplication and loss cost, then there are no post-coalescence duplications. We prove by contrapositive: if there are post-coalescence duplications, then the reconciliation between the locus tree and species tree (implied by the LCT) does not have minimum duplication and loss cost. Note that for the LCT, we must fix the implied locus tree topology \( T^{L} \); otherwise, there could exist a different locus tree topology with lower minimum duplication and loss.
cost, but this has no bearing on the whether the reconciliation between locus tree $T^L$ and species tree $S$ is the LCA mapping.

For $g, g', g'', cg', cg''$ as previously defined, assume that a post-coalescence duplication has occurred along $(g, g')$ so that $L(g) \neq L(g')$. Let $l = L(g)$ denote the original locus, $l' = L(g')$ denote the new locus, and let $s' = M(cg')$ and $s''$ be the child of $s$ that is not $s'$. Because $|c(g')| = 1$ and a locus can be created only once in the LCT, $l'$ cannot exist in $s''$, that is, $l'$ is lost in $s''$. Now consider alternative scenarios in which we remove the duplication along $(g, g')$. Such an operation has no effect on the LCT beyond $g, g', cg'$, in particular, the resulting LCT is valid. (1) If $L(cg') \neq l'$, then the duplication along $(g, g')$ was followed by a duplication along $(g', cg')$, and removing the duplication along $(g, g')$ would not affect the locus tree topology but would reduce the number of duplications by one and remove the loss of $l'$ in $s''$, thus reducing the number of losses by one. (2) If $L(cg') = l'$ then “pushing” the duplication from $(g, g')$ to $(g', cg')$ would not affect the locus tree topology nor the number of duplications but would remove the loss of $l'$ in $s''$, thus reducing the number of losses by one. In either alternative scenario, the number of extra lineages may also be affected. However, what is important is that the original scenario, with the post-coalescence duplication, has the same locus tree topology but a locus tree-species tree reconciliation that does not minimize the number of duplications and losses, thus completing our proof.

Proof of Theorem C.2.1

Strictly speaking, we do not require LCA reconciliation between the locus tree and species tree. Rather, we require the following weaker condition: post-coalescence duplications have not occurred.

We have chosen to phrase Theorem C.2.1 in terms of the LCA reconciliation between the locus tree and species tree since the concept of post-coalescence duplications is not intuitive and the former implies the latter (Proposition C.1.2).

Consider the problem of mapping explicit nodes in the coalescent tree to a node in the species tree. The outline of our proof is as follows:

1. Let $\alpha^* = (M^*, L^*, O^*)$ be a MP LCT in which $M^*$ is not the LCA reconciliation.
2. There exists an explicit node $g \in I(G)$ such that $M^*(g)$ is not the LCA mapping.
3. There exists an alternative LCT $\bar{\alpha} = (M, L, O)$ in which $g$ is mapped “down” the species tree and $\bar{\alpha}$ implies the same number of duplications and losses as $\alpha$ but implies one fewer extra lineage.
4. The reconciliation cost of the alternative LCT is lower than than of the original LCT, that is, $R_{\bar{\alpha}} < R_{\alpha^*}$.
5. $\alpha^*$, in which $M^*$ is not the LCA reconciliation, cannot be a MP LCT.
Obviously, 1 implies 2, 3 implies 4, and 4 implies 5; thus we only need to prove that 2 implies 3. For notational simplicity, we remove the asterisks and consider \( \alpha = (\mathcal{M}, \mathcal{L}, \mathcal{O}) \). Figure C.3 may also be useful in following this proof. Finally, note that this proof is conceptually simple but requires a large amount of setup and mathematical notation to be precise.

If \( \mathcal{M} \) is not a MP LCT, let \( g \in I(G) \) denote an explicit node that is not mapped to the LCA, and let \( g' \) and \( g'' \) denote the children of \( g \). By definition, \( g \) can be mapped down the species tree, meaning that \( g' \) and \( g'' \) must evolve down the same child species. That is, \( g' \) and \( g'' \) must map to the same child species or to the descendants of the same child species. Without loss of generality, we assume the former; else, nodes can be added along \((g, g')\) and \((g, g'')\) until the new children of \( g \) map to the same child species. Formally, let \( s \in I(S) \) be the species such that \( \mathcal{M}(g) = s \), let \( s' \in c(s) \) be the child species such that \( \mathcal{M}(g') = \mathcal{M}(g'') = s' \), and let \( s'' \) be the child species that is not \( s' \). Also, let \( l = \mathcal{L}(g) \).

For our proof, we must consider how the implied evolutionary events change between this original scenario \( \alpha = (\mathcal{M}, \mathcal{L}, \mathcal{O}) \) and an alternative scenario \( \bar{\alpha} = (\bar{\mathcal{M}}, \bar{\mathcal{L}}, \bar{\mathcal{O}}) \) in which \( g \) is mapped down the species tree to \( s' \). Recall that we have assumed \( \alpha \) is a MP LCT; that is, \( \mathcal{L} \) and \( \mathcal{O} \) are an optimal locus map and order for the species map \( \mathcal{M} \). However, in the alternative scenario, we have full control over \( \bar{\mathcal{L}} \) and \( \bar{\mathcal{O}} \). In particular, these do not have to be an optimal locus map and order for the species map \( \bar{\mathcal{M}} \); we must only show that there exists a \( \bar{\mathcal{L}} \) and \( \bar{\mathcal{O}} \) for \( \bar{\mathcal{M}} \) such that \( \bar{\alpha} \) implies the same number of duplications and losses as \( \alpha \) but fewer extra lineages.

Now let us understand how the gene tree maps within the species tree as implied by \( \mathcal{M} \) and \( \bar{\mathcal{M}} \). Because we have assumed no post-coalescence duplications, delayed speciation nodes are unnecessary (Proposition C.1.1). Therefore, for \( \alpha \), \( g \) is mapped to the bottom of species tree branch \( e(s) \) and lineages \((g, g')\) and \((g, g'')\) are found at the top of species tree branch \( e(s') \). However, for \( \bar{\alpha} \), an implied speciation node is necessary so as not to force the original parent node of \( g \) to map to the bottom of species tree branch \( e(s) \). That is, let \( pg = p(g) \) denote the original parent node of \( g \); we break up \((pg, g)\) by adding implied speciation node \( sg \) (and setting \( \bar{\mathcal{M}}(sg) = s \)); this creates branches \((pg, sg), (sg, g)\) with lineage \((sg, g)\) at the top of species tree branch \( e(s') \).

Finally, consider possible cases for \( \alpha \). Recall that we have assumed no post-coalescence duplications, that is, duplications are not allowed along either child branch of the explicit node \( s \), so \( L(g') = L(g'') = l \). Therefore, we must consider two cases:

1. No duplication has occurred along the branch leading to the explicit node \( g \) (Figure C.3A, top): \( L(p(g)) = l \). For \( \bar{\alpha} \), choose \( \bar{\mathcal{L}} \) to “match” \( \mathcal{L} \) (Figure C.3A, bottom): \( \bar{\mathcal{L}}(sg) = l \), and for all other nodes, \( \bar{\mathcal{L}}(v) = L(v) \) (where we have used \( v \) so as not to confuse a node with \( g \)). Also, choose \( \bar{\mathcal{O}} \) to “match” \( \mathcal{O} \): all \( v \) remain in the same position except \( g \) is replaced by \( sg \).
for \( \tilde{O}(s,l) \) and \( g \) is moved to the first position in \( \tilde{O}(s',l) \), assuming that \( l \) is a parent locus of a duplication in \( s \) and \( s' \), respectively. Then, comparing \( \alpha \) and \( \tilde{\alpha} \), the only difference is in the number of lineages that exist at the species breakpoint between \( s \) and \( s' \); thus, the numbers of duplications, losses, and extra lineages due to duplications remain unchanged. To consider how the number of extra lineages due to speciations is affected, we determine the set \( \mathfrak{X} \) of lineages in locus \( l \) at the top of \( e(s') \) as all other species and loci are unaffected. For \( \alpha \), \( \mathfrak{X} \) includes lineages \( (g,g'), (g,g'') \), and possibly \( k \) other lineages, thus incurring \( k + 1 \) extra lineages. In comparison, for \( \tilde{\alpha} \), \( \mathfrak{X} \) includes lineage \( (sg,g) \) and \( k \) other lineages, thus incurring \( k \) extra lineages. That is, \( \tilde{\alpha} \) implies one fewer extra lineage (due to speciation) than \( \alpha \).

2. A duplication has occurred along the branch leading to the the explicit node \( g \) (Figure C.3B, top): \( L(p(g)) \neq l \). For \( \tilde{\alpha} \), as with the previous case, we choose \( \tilde{L} \) and \( \tilde{O} \) to “match” \( L \) and \( O \), respectively, with the difference that the duplication along \( (p(g),g) \) in \( \alpha \) remains in species \( s \) but occurs along \( (pg,sg) \) in \( \tilde{\alpha} \) (Figure C.3B, bottom). Then, a comparison of implied evolutionary events for \( \alpha \) and \( \tilde{\alpha} \) is identical to the previous case, that is, the numbers of duplications, losses, and extra lineages due to duplications remain unchanged but the number of extra lineages due to speciations decreases by one.

**Remark.** We now consider the implications of allowing post-coalescence duplications. If a post-coalescence duplication has occurred, delayed speciation nodes are necessary (Proposition C.1.1). That is, for \( \alpha \), we break up \( (g,g') \) and \( (g,g'') \) by adding implied speciation nodes \( sg' \) and \( sg'' \), respectively, thus creating branches \( (g,sg'), (sg',g'), (g,sg''), (sg'',g'') \) with \( M(sg') = M(sg'') = s \) and with lineages \( (sg',g'), (sg'',g'') \) at the top of species tree branch \( e(s') \). Furthermore, the locus of at least one node \( sg' \) or \( sg'' \) is different from the locus of \( g \); assume that \( L(sg'') \neq l \) (Figure C.3C). Then shifting \( g \) in the alternative scenario \( \tilde{\alpha} \) may induce more events. In particular, \( sg' \) and \( sg'' \) do not exist in \( \tilde{\alpha} \), but setting \( \tilde{M}(g) = s' \) would require us to determine where to place the duplication originally found along \( (g,sg'') \). If the duplication remains in species \( s \) (that is, \( \tilde{L}(pg) \neq \tilde{L}(sg) \)), then \( \tilde{L}(g') = \tilde{L}(g'') \) so that \( \tilde{\alpha} \) may no longer be a valid LCT. Adding another duplication along \( (g,g'') \) to force \( \tilde{L}(g') \neq \tilde{L}(g'') \) would, of course, go against our efforts by incurring an additional duplication. Otherwise, if the duplication is “pushed” to occur along \( (g,g'') \) in species \( s' \) (that is, \( \tilde{L}(g) \neq \tilde{L}(g'') \)), then \( \tilde{\alpha} \) may incur more extra lineages due to duplication compared to \( \alpha \). This is because the duplication now occurs later in the gene evolutionary history; therefore, bifurcations in the gene tree in locus \( l \) between the time of the old duplication event in \( \alpha \) and the new duplication event in \( \tilde{\alpha} \) could result in additional lineages in locus \( l \) that are contemporaneous to the duplication along \( (g,g'') \) in \( \tilde{\alpha} \).
Proof of Proposition C.2.2

We prove each component separately:

1. Allowing multiple changes would incur additional duplications and losses, and possibly additional extra lineages, and therefore be strictly less parsimonious.

2. Consider a gene tree node that is internal to a species tree branch and has two children that map to distinct loci that are both different from the locus of their shared parental node (ex. Figure 4.3B, map 6); such a scenario would correspond to a locus duplicating twice to create two daughter loci, possibly followed, as in the figure, by the original mother locus becoming lost. An alternative explanation is for the original locus to duplicate, then for the new locus to duplicate, again possibly followed by the original locus becoming lost (ex. Figure 4.3B, map 5). These two evolutionary histories are identical in terms of locus assignments after the double round of duplications, and the latter is always at least as parsimonious as the former. (In particular, extra lineages can be incurred in the former if there exist additional lineages in the same locus. In the figure, an additional red lineage in maps 5 and 6 would result in one extra lineage at the root of the species tree branch in both maps, but map 6 would also incur an extra lineage at the first duplication.)

3. Incorporating ILS allows a decrease in the number of inferred duplications. If extra lineages have a low cost, very few duplications can be inferred, in particular fewer duplications than in a duplication-loss model. As this cost increases, reconciliation reverts to a duplication-loss model, thus giving an upper bound on the maximum number of duplications, and consequently, the maximum number of loci, in the MP LCT.

Proof of Proposition C.2.3

Recall that \(O\) defines ordered sets \(O(s, l) \forall s \in V(S), l \in parent.loci(s)\). Furthermore, \(O(s, l)\) only affects the number of extra lineages due to duplications, and each \(O(s, l)\) affects this number independently. This observation allows to infer an optimal ordered set \(O^*(s, l)\) independently for each \(s \in V(S), l \in parent.loci(s)\). In particular, to find \(O^*(s, l)\), we begin with \(start.lineages(s, l)\) and intelligently choose the next node \(g \in O(s, l)\) until all nodes have been exhausted. To optimally select \(g\), realize that the lineage count in \(l\) decreases by one every time \(L^*(g) \neq l\) (due to a duplication). Otherwise, assuming \(L^*(g) = l\), if \(g \in leaves(s)\), then the lineage count either is unchanged (if \(|c(g)| \leq 1\); note that if \(|c(g)| = 0\), then \(s = L(S)\) but extant genes exist to present day and are not removed from the set of lineages) or increases by one (if \(|c(g)| = 2\), and if \(g \in nodes(s) \setminus leaves(s)\), then the lineage count increases by one (since \(|c(g)| = 2\). That is,
choosing the next node $g: \mathcal{L}^*(g) \neq l$ decreases the lineage count, and choosing $g: \mathcal{L}^*(g) = l$ either has no effect or increases the lineage count. Thus, $\hat{\mathcal{O}}^*(s, l)$ should have all duplications as early in the species tree branch as possible.

*Remark.* Note that we have not shown that all $\mathcal{O}^* = \arg\min_{\mathcal{O}} R(M, \mathcal{L}, \mathcal{O})$ satisfy this property. This theorem simply states that this particular $\mathcal{O}^*$ is an optimal solution.

### C.4 Comparison with the three-tree model

The idea for separating the locus history and coalescent history of a gene family was initially proposed in the three-tree model of DLCoal [201]. There, the locus tree was introduced to trace the history of a pool or set of sequences, namely all of the sequences in a population that belong to the same species and the same locus. This is important as coalescence is restricted to sequences within the same pool. In addition, each daughter edge (defined in the locus tree as the child lineage of a duplication that has evolved in the new locus) has complete coalescence, that is, only one gene lineage is present at the top of each daughter edge.

For a given gene tree and species tree, provided that the underlying evolutionary model used in phylogenetic reconstruction is compatible with the three-tree model, the reconciliation problem is to infer both the “hidden” locus tree and the reconciliations between the three trees. DLCoal [201] separates the locus history and coalescent history of a gene family by explicitly representing the locus tree and two-step reconciliations. Therefore, DLCoalRecon should ideally search over the space of locus trees and reconciliations. However, the number of distinct locus tree topologies for a given set of extant sequences is exponential in the number of extant sequences, making a full search infeasible. Furthermore, for a gene tree, locus tree (with annotated daughter nodes), and species tree, only certain reconciliations are valid; in particular, despite the LCA reconciliation being the optimal solution for minimizing deep coalescence and minimizing duplications and losses, LCA reconciliation between the gene tree and locus tree and between the locus tree and species tree may violate the requirement of complete coalescence in a daughter edge.

The LCT overcomes these challenges by collapsing the locus tree and reconciliations into the gene tree. We accomplish this by tracking the locus directly within the gene tree. In addition, the LCT is a simplified representation of the three-tree model. In particular, inferring the duplication-loss history of a gene family requires a locus tree topology and its reconciliation to the species tree. That is, the dates of the ancestral nodes in the locus tree are unknown. To overcome this problem, DLCoal models fully dated locus trees, and DLCoalRecon integrates over many samples of duplication times (as speciation times are fixed by the species tree) to yield the *max a posteriori* locus tree topology.

164
In contrast, the LCT models only the locus tree topology. In addition, though loci change along gene tree branches, the LCT locus map labels gene tree nodes. In effect, the LCT rounds locus changes to the nearest node: for \( g \in V(G): g \neq r(G), \) if \( L(g) \neq L(p(g)) \), then the locus changes immediately before \( g \). Despite these simplifications, the LCT can model the same evolutionary histories captured by the three-tree model; this includes the popular and simple duplication-loss only (Figure C.1B) or coalescent-only models (Figure C.1C) for gene evolution, as well as more complicated histories involving duplications, losses, and ILS (Figure C.1A).

For the reconciliation problem, the LCT and DLCpar provide further advantages. Motivated by the need for simple and efficient ILS-aware reconciliation algorithms, we chose to adopt a maximum parsimony approach for DLCpar. This means fewer assumptions on the evolutionary model. As an example, unlike DLCoalRecon, we do not assume that the locus tree evolves within the species tree according to a birth-death process [1, 8, 54, 200]. Furthermore, we can directly take into account the restrictions placed on a MP LCT by the gene tree and species tree. This allows us to infer how the gene tree maps to the species tree (bypassing the locus tree) as well as enumerate over the space of valid reconciliations. That is, we have a priori restricted the reconciliation search space. In comparison, since it is impractical to search through the space of all possible reconciliations using the three-tree model, DLCoalRecon uses a search strategy in which reconciliations are proposed then checked for validity.

Finally, due to the small number of trees that have post-coalescence duplications, we have chosen to make DLCpar more efficient by disallowing post-coalescence duplications; thus, delayed speciation nodes are not required (Proposition C.1.1). However, note that no post-coalescence duplications does not imply LCA reconciliation between the locus tree and species tree. This is an additional advantage of DLCpar over DLCoalRecon; namely, while DLCoalRecon only searches over the space of locus trees and reconciliations such that the reconciliation between the locus tree and the species tree is the LCA mapping, DLCpar actually allows non-LCA reconciliation between the locus tree and species tree.

C.5 Relating duplication, loss, and deep coalescence in a parsimony framework

Here, we discuss the implications of Zhang [285] on our work. In particular, for gene tree \( G \) and species tree \( S \), Zhang [285] presented a simple formula relating the number \( n_c(G, S) \) of extra lineages (induced by deep coalescence) to the numbers \( n_D(G, S) \) and \( n_L(G, S) \) of duplications and losses; however, his work applied the duplication-loss and deep coalescence parsimony scores separately to
the reconciliation between gene tree and species tree. We highlight that our work jointly models duplication-loss and ILS. This is an important distinction as it means that, for many gene families (in particular, those affected by duplication-loss and ILS), the results of Zhang [285] do not necessarily apply.

What about distinguishing between gene trees that are affected only by duplication-loss (DL) or deep coalescence (DC)? Recall that we have assumed that genes that map to the same species must be paralogous (as they belong to different loci), and contrast this with coalescent theory that traces orthologous genes. This means that if multiple gene copies exist within any species, the gene tree must include duplications, and distinguishing between DL and DC is moot. So let us restrict our analysis to uniquely-labeled gene trees. For simplicity, we first consider equal costs for duplication, loss, and deep coalescence. In this case, applying the results of Zhang [285], the DC cost is always less than the DL cost:

$$DC \text{ cost} = n_G = n_L - 2 \cdot n_D < n_D + n_L = DL \text{ cost}$$

where we have hidden the dependencies on $G$ and $S$ for notational simplicity. This suggests that, if the gene tree is incongruent to the species tree, we would always infer deep coalescence over duplication-loss. We argue that this is acceptable as one-to-one gene families often assume that all genes are orthologous to one another, so inferring deep coalescence is the “correct” solution. If the costs for duplication, loss, and deep coalescence are unequal, then raising the cost $C$ of deep coalescence (or lowering the costs $D$ and $L$ of duplications and losses) could result in the DC cost being greater than the DL cost:

$$DC \text{ cost} = C \cdot n_G = C \cdot (n_L - 2 \cdot n_D) > D \cdot n_D + L \cdot n_L = DL \text{ cost}$$

This is, of course, exactly what we would want, since, choosing such costs implies that we prefer inferring DL over DC.

### C.6 Duplication, loss, and coalescence costs

As the locus tree is used to infer duplications, losses, orthologs, and paralogs, we assessed the robustness of DLCpar to different event costs by analyzing the similarity of inferred locus trees. Using the parameter setting $D = L = 1, C = 0.5$ as a reference, the Robinson-Foulds (RF) distances [204] of locus trees reconstructed under varying costs were $< 0.040$ for fly population sizes $\leq 100$ million (max RF $= 0.125$ for $N = 500$ million, $4\times$ rate) and $< 0.022$ for all primate
datasets. We then analyzed whether this similarity in locus tree topology translates to accurate event inference (Figure C.5). While we have simulated a wide range of population sizes in order to analyze reconciliation performance for datasets with a large amount of ILS, a biologically realistic range of population sizes is on the order of 1–50 million for Drosophila and 10–50 thousand for primates [29]. In this population range, with a duplication-loss rate equal to the estimated real rate, DLCpar performance as measured by topological accuracy varies by 0.2–4.2% to a low of 94.8%, duplication precision by 0.2–9.6% to a low of 87.6%, and loss precision by 0.4–17.2% to a low of 82.8%, with other metrics varying up to 9.6% to a low of 87.6%. If we consider only the primates datasets for which biologically reasonable population sizes change by 5 x, compared to the 50 million x difference in Drosophila, performance variation across all metrics is < 4.0%, with the lowest metric at 94.6%. For larger population sizes or higher duplication-loss rates, more care must be taken in choosing event costs, but as expected, as population size, and consequently ILS rate, increases, a smaller coalescence cost (relative to duplication and loss costs) yields better performance.

C.7 Search heuristics

Some gene trees may be too complex for DLCpar to reconcile. In particular, large gene trees or gene trees highly incongruent to the species tree contain many implied gene tree nodes, thus increasing the LCT search space. To address this problem, we introduce two heuristics to the DLCpar algorithm.

The first heuristic bounds the LCT search space by limiting the maximum number of duplications, losses, or loci in each species tree branch and by prescreening proposed locus maps. For the prescreen, when enumerating locus maps, we keep track of the cost-to-go of varying loci assignments at the bottom of each species tree branch, where the cost-to-go is the minimum total cost along all ancestor branches. If the minimum cost-to-go $c_{\text{min}}$ across all loci assignments exceeds a threshold $p_{\text{min}}$, then loci assignments with cost-to-go exceeding $p_{\text{factor}} \times c_{\text{min}}$ are pruned. Based on analysis of the simulated datasets, we chose default parameters of 4 duplications and 4 losses per species tree branch (no restriction on the maximum number of loci) and $p_{\text{min}} = 5$, $p_{\text{factor}} = 2$. We find this heuristic to have limited effect on algorithm performance (Figure C.6). Compared to DLCpar, RF distances for DLCpar-bound are < 0.008 for fly population sizes < 100 million (max RF = 0.032 for $N = 500$ million, 4x rate) and < 0.004 for all primate datasets.

The second heuristic further limits the LCT search space by using the three-tree model and the hill-climbing search strategy of DLCoalRecon [201]. We chose default search parameters of 1000 iterations and 20 prescreens. Compared to DLCpar, RF distances for DLCpar-search are < 0.057 for fly population sizes ≤ 100 million (max RF = 0.142 for $N = 500$ million, 4x rate) and < 0.031 for all primate datasets.
C.8 Simulated species trees

We also assessed how DLCpar performance changes with varying speciation rates and species tree sizes. Here, we used the simulated species trees of Wu et al. [276], which were generated by TreeSample [106] with a constant-rate birth-death model, a variety of settings for the speciation rate (0.05–1 events/species/myr) and tree size (5–100 extant species), and a fixed speciation rate-extinction rate ratio ($\mu = 0.9\lambda$). For each species tree, we used the DLCoal model, with parameters from the Drosophila clade, to simulate gene trees, then used DLCpar, DLCoalRecon, and MPR to reconcile these gene trees with the species tree. As before, DLCpar used the same event costs across all settings whereas DLCoalRecon was tuned to each simulation setting. As with the simulated gene trees reconciled to real species trees, almost universally, DLCpar shows dramatic improvement over other reconciliation programs (Figure C.7).

As the speciation rate increases, resulting in shorter branches in the species tree and therefore fewer duplications and losses and more deep coalescence, a number of trends emerge. MPR performs consistently low, and across all metrics, performance decreases with increasing speciation rate. The latter observation suggests that the reconciliation problem becomes more difficult as the speciation rate increases and that the increased ILS rate overcomes the lower number of implanted duplications and losses to result in more gene tree-species tree incongruence. Of particular note is the dramatic drop in branch accuracy of the locus tree, which was shown to be a robust metric in the gene tree reconstruction problem [200, 276]. In contrast, with increasing speciation rate, DLCpar and DLCoalRecon branch accuracies remain consistently high, but more interestingly, topological accuracies increase. This is the reverse of our observation in the previous simulation study, in which we found locus tree topological accuracy to be stable or slightly decrease with increasing deep coalescence. The difference is that previously, increased population sizes resulted in increased ILS rates without affecting the number of implanted duplications and losses, whereas here, decreased speciation times result in increased ILS rates and fewer implanted duplications and losses. Finally, for the last metrics, duplication and loss precision of DLCpar and DLCoalRecon decrease with increasing speciation rate, with DLCpar slightly outperforming DLCoalRecon in duplication precision but DLCoalRecon holding a slight (but insignificant) edge in loss precision (where significance is measured by overlap in the 95% confidence intervals). Overall, the robust performance of DLCpar, and to a lesser extent, DLCoalRecon, suggests that varying speciation rates have limited effect on the accuracy of inferred reconciliations when an ILS-aware method is used.

As the species tree size increases, larger numbers of duplication, loss, and coalescent events lead to more gene tree-species tree incongruence and thus a more difficult inference problem. This is
evident as all three reconciliation programs show decreased locus tree topological accuracy with increasing tree size. Across the other metrics, MPR performance is consistently low regardless of tree size, and except for the relatively robust metric of branch accuracy, DLCoalRecon performance decreases rapidly with increasing tree size. In contrast, DLCpar performance is highly robust to tree size. The noticeable exception is that DLCpar loss precision is lower than that of DLCoalRecon for small species trees, but at these sizes, very few (single digit) duplications and losses are simulated, meaning that small counts are likely the source of low precision and explaining the larger variance in precision at small tree sizes. For larger tree sizes, DLCpar performs substantially better than DLCoalRecon despite the average run time of DLCoalRecon being 16.8 (9.8)× that of DLCpar for 50 (100) extant species. This is again likely attributable to the heuristic search strategy and limited search space of DLCoalRecon. In evaluation, DLCoalRecon searched over the same number of reconciliations regardless of species tree size, whereas DLCpar, by virtue of its algorithm, searched over increasingly larger spaces as the species tree size increased.

C.8 Simulated species trees
C.9 Additional figures and tables

Figure C.1: Models for gene family evolution. Evolution of a gene family is shown using the three-tree model (left) and the labeled coalescent tree (right). (A) In the three-tree model, the (coalescent) gene tree reconciles to the locus tree, allowing inference of ILS, and the locus tree reconciles to the species tree, allowing inference of duplications and losses. (B) In a duplication-loss model, ILS is assumed not to occur; thus, the gene tree and locus tree are congruent. Therefore, (1) effectively, the daughter lineage of a duplication coalesces immediately with its parent lineage and (2) within each species, at most one lineage exists per locus. In the LCT, this translates to (1) loci changes occur at nodes and (2) all coexisting lineages within the same species tree branch belong to different loci. (C) In a multispecies coalescent model, duplications and losses are assumed not to occur; thus, the locus tree and species tree are congruent. Therefore, there exists a single gene per extant species, and all lineages evolve within the same locus. [Parts of this figure have been adapted with permission from Rasmussen and Kellis [201].]
Figure C.2: **Delay between speciation and coalescence.** *(A)* Explicit (−) and implicit (*) speciation nodes separate the gene tree into disjoint subtrees that evolve within the species tree branches. Note that speciation nodes are rounded to the breakpoint between species. *(B)* Delay between speciation and coalescence causes explicit speciation nodes to be found earlier in the tree and adds delayed speciation nodes (+). It is now the delayed and implicit speciation nodes that separate the gene tree. (In particular, previously defined explicit speciation nodes are no longer speciation nodes.) Note that this models the same three trees as before (albeit with different timings), so delayed speciation nodes are unnecessary. *(C)* Delayed speciation nodes are necessary when post-coalescence duplications have occurred (new locus in dark blue). These duplications could instead be inferred in the child species, which would reduce the number of losses without impacting the locus tree topology. Thus, such duplications are not possible if we assume LCA reconciliation between the locus tree and species tree, as the LCA in this instance minimizes the number of duplications + losses. In this example, duplication in the ancestral species of species B and species C is followed by loss of the dark blue locus in species B and loss of the yellow locus in species C. Inferring the duplication in species C instead would reduce the number of losses by one since the dark blue locus is no longer lost in species B.
Figure C.3: Reducing the number of inferred evolutionary events by shifting gene tree nodes down the species tree. Explicit nodes in the gene tree are mapped to a species (break between species tree branches shown as blue line). Assume that there exists an explicit node (square) that is not mapped to the LCA; thus, it can be mapped “down” the species tree. Such a shift is only possible if the children of the explicit node evolve down the same child species; thus the other child branch in the species tree is not shown. Lineages with an unknown locus mapping are depicted in gray. Note that other lineages may exist; we have restricted the figure to a single triplet (parent with two children) for clarity. (A) No duplication has occurred along any branch incident to the explicit node (top left), so delayed speciation nodes are unnecessary (top right). Shifting the explicit node incurs one fewer extra lineage due to speciation (bottom). In this example, in a non-LCA mapping, two red lineages at the species break implies one extra lineage (top), whereas, by shifting the explicit node, one red lineage at the species break implies no extra lineages (bottom). The numbers of implied duplications, losses, and extra lineages due to duplications are unchanged. (B) A duplication has occurred along the parent branch of the explicit node (top); if the explicit node is shifted to the child species but the duplication remains in the original species (bottom), this is identical to case A. (C) A duplication has occurred along a child branch of the explicit node, between the time of coalescence and the speciation event; such a duplication is a post-coalescence duplication and is not allowed if we assume LCA reconciliation between the locus tree and species tree. That is, the lineage with the new locus (yellow) evolves down a single child species, so the duplication can be shifted to occur in the child species, immediately after the speciation, which would incur one fewer loss without impacting the locus tree topology. Note that if the lineage with the new locus evolved down multiple children species (not shown), the duplication could not be shifted because the speciation node with the new locus would have two children, making it impossible to shift the duplication down a single lineage.
Figure C.4: Phylogenetic accuracy of DLCpar on simulated fly and primate gene trees. DLCpar (solid), DLCoalRecon (dot), and MPR (dash) were used to reconcile simulated gene trees. Duplication and losses were simulated at rates that were the same as (1x, red), twice (2x, green), and four times (4x, blue) the rate estimated in real data. See also Figure 4.5, which corresponds to simulation at 1x rate.
Figure C.5: Phylogenetic accuracy of DLCpar on simulated gene trees using various event costs. DLCpar was run on simulated fly and primate datasets using $D = L = 1$, and a range of costs $C$ for extra lineages. The difference in precision or accuracy compared to DLCpar with $C = 0.5$ is shown for $C = 0.25$ (solid), $C = 0.75$ (dash), $C = 1$ (dot). A comparison to DLCoalRecon is also shown (lighter colors, long dash), and the gray line indicates no change. For details on the simulation procedure, see Figure 4.5 and C.4.
Figure C.6: **Phylogenetic accuracy of DLCpar on simulated gene trees using search heuristics.** DLCpar was run on simulated fly and primate datasets using $D = L = C = 1$ and two search heuristics. The difference in precision or accuracy compared to DLCpar (without search heuristics) is shown, with the gray line indicating no change. In DLCpar-bound (solid), the search space was limited by pre-screening locus maps and enforcing upper bounds on the maximum number of duplications and losses per species. In DLCpar-search (dash), the three-tree model rather than the LCT model was used, and a local hill-climbing strategy was employed to search the space of locus trees and reconciliations (as in DLCoal). For details on the simulation procedure, see Figure 4.5 and C.4.
Figure C.7: Phylogenetic accuracy and runtime of various reconciliation methods on simulated species trees. Simulated species trees were obtained from Wu et al. [276], which used TreeSample [106] with a constant-rate birth-death model, varying speciation rates $\lambda$ (events/spec/myr) and tree sizes $n$ (number of extant taxa), and extinction rates set to $\mu = 0.9\lambda$, with 10 species trees simulated per setting. For each species tree, 100 locus trees and coalescent trees were simulated using the DLCoal model [201], with parameters from the fly clade ($\lambda = \mu = 0.0012$ events/gene/myr, $N = 10$ million, $g = 0.1$ yr). Finally, DLCpar ($D = L = C = 1$, red solid), DLCoalRecon (green dot), and MPR (blue dash) were used to reconcile the simulated (coalescent) gene trees. Performance is measured in terms of mean accuracy/precision and runtime, with bars indicating the 95% confidence interval (mean $\pm 1.96 \times$ standard error) across all species trees with the same setting. (left) Performance for $\lambda = 0.05, 0.1, 0.2, 0.5, 1.0$ and $n = 20$. (right) Performance for $\lambda = 0.2$ and $n = 5, 10, 20, 50, 100$. 

C Models for gene duplication, loss, and deep coalescence
Figure C.8: Cumulative distribution of duplication consistency scores for all duplications inferred on the real fungal dataset using several phylogenetic methods. Each program was used genome-wide to infer the duplications present in 16 fungal species. For each duplication, we computed the duplication consistency score, defined as $|L \cap R| / |L \cup R|$, where $L$ and $R$ are the sets of species present in descendants left and right of the duplication node, respectively.
C Models for gene duplication, loss, and deep coalescence
Appendix D

Models for domain rearrangements

In this appendix, let architectures refer to both a single architecture, e.g. $a$, $ab$, and a multiset (unordered list) of architectures, e.g. $\{a\}$, $\{ab\}$, $\{a, ab\}$, $\{a, a, ab\}$. Architectures can therefore be assigned to both genes (using the single architecture definition) and to entire species (using the multiset of architectures definition).

D.1 Model of architecture evolution

Consider the problem of transforming a parent architecture to a child architecture. We represent these architectures as directed acyclic graphs, where the in-degree and out-degree of a node are each at most one. A module is indicated by a node, and neighboring modules within the same gene are joined with a directed edge. The problem is now equivalent to transforming from the parent graph to the child graph, where each allowable graph operation corresponds to an evolutionary event (Figure 5.1E):

- generation: Add a node (corresponding to a module that does not currently exist).
- duplication: Duplicate a node.
- loss: Remove a node.
- merge: Add an edge between two (existing) nodes.
- split: Remove an edge between two (existing) nodes.

Note that these definitions require that generation, duplication, and loss occur at the module level. For example, generating a sequence of multiple modules is only possible through generation of the component modules. This assumption treats modules as the basic building blocks of a gene and
implies that the generation, duplication, and loss of larger sequences (as measured by the number of modules) incur a higher cost.

Evolutionary events are applied in the following order: generation, duplication, loss, split, merge. There are also some caveats to account for the architecture representation. For example, we allow for the duplication and loss of entire architectures or sub-architectures, where the cost of duplication/loss corresponds to the number of modules duplicated/lost. Furthermore, the duplicated (sub-)architecture retains all the edges of the original, and edges within a lost (sub-)architecture can be removed without penalty. Duplication/loss of a sub-architecture may also incur one or two (hidden) splits, depending on whether the left and right end of the sub-architecture were connected to another module in the parent architecture; this prevents parental sub-architectures from appearing in isolation in the child without penalty. Finally, merges are free between two generated modules (but not between a generated module and an existing module); this allows for the generation of architectures in addition to the generation of modules.

Note that with this representation, there is a one-to-one mapping between nodes in the parent and child graphs (using the "null" node as the parent of a generated node and the child of a lost node). Tracing this mapping for each module in a reconstructed architecture scenario reveals the series of generations, duplications, and losses that have led to the extant module counts, and tracing the addition or removal of edges across the mappings reveals the series of merges and splits that have led to the extant architectures.

To demonstrate the robustness of our architecture model, consider the case of a duplicated sub-architecture. For example, rearranging parent architecture $A = \{abc\}$ to form child architecture $B = \{ab, abc\}$ would infer a duplication of module $a$, a duplication of module $b$, and a (hidden) split between modules $b$ and $c$. Note that (1) the original parent architecture $abc$ is retained in $B$ (with no inferred events), (2) no merge event is required between $a$ and $b$ since the multi-module sub-architecture $ab$ arose from $abc$, and (3) a (hidden) split is inferred to account for the missing edge between $b$ and $c$ when $ab$ is duplicated from $abc$.

D.2 Domain database annotations

As noted in the main text, Section 5.2.4 may be replaced by a database search against existing domains (e.g. Pfam [16], SCOP [170], SMART [217], CDD [160], etc). We have chosen to use a de novo approach to module identification rather than using a domain database search for a number of reasons.

Our main reason is that we wished to make no a priori assumptions about the identity or boundaries of the modules. As mentioned in the main text, domain databases are often biased, for
example, towards domains with known structures or function. However, our definition of modules is evolutionarily-based and depends solely on sequence conservation.

An analysis of genome coverage (excluding singleton domains or modules) also revealed that only 62% of *Drosophila* genes have Pfam annotations compared to 82% of genes with module annotations. If we include singleton domains/modules, the change in coverage for Pfam annotations is negligible while the coverage for module annotations increases to 85%, with the remaining 15% of genes lacking BLAST hits that pass our filters. We believe that this difference in coverage is because our approach captures both known and unknown domains; in particular, it captures domains that are evolutionarily (rather structurally or functionally) conserved.

In addition, domain definitions are compiled using genomes across the three domains of life, meaning that domain families may be overclustered when looking at a small subset of genomes such as the ~60 myr Drosophila clade. (Recall that gene and domain families are defined as the set of genes/domains that descend from a single gene/domain in the most recent common ancestor of all species under consideration. Therefore, restricting the genomes to a small subset will break the original families into many smaller clusters.) A major benefit of our approach is that it can be used at multiple timescales: we can look across the three domains of life as in domain databases, or we can find novel clade-specific domain families that may be missing from domain databases, as in our analysis of *Drosophila*. For comparison, ADDA found a number of novel domain families missing from Pfam and SMART, with the majority of these new families specific to a single domain of life. Such novel domain families may also be present within the *Drosophila* phylogeny (perhaps to confer clade-specific biological functions), and a such using domain definitions compiled across all three domains of life may lack the power to detect such recently evolved families.

Our approach can also capture known and unknown domains and neutral evolutionary events. In particular, we can identify modules linked to a protein function but associated with an unknown domain. This is important, as we are also interested in analyzing genome-wide event rates (or counts), and if we focus on the subset of genes in which the merged or split domain has a known function, these rates (counts) may be biased.

Finally, our approach has a higher power than a database search, and moreover, it can be applied to newly sequences genomes to discover new modules. In particular, we can analyze a group of closely related genomes that are together distant from other genomes. As mentioned previously, our method will find novel domain families that have evolved solely within the newly sequenced clade without requiring these families to be defined in domain databases.

D.2 Domain database annotations
D.3 Promiscuous modules

Analysis in our main text (Section 5.4) excluded promiscuous modules in our pipeline. Including these in our analysis decreased the number of architecture families to 14,156 (1.8% decrease), with 4201 families containing more than one module (0.03% decrease in ratio of # families with ≥ 2 modules/# families) and 4037 families containing a fusion or fission (<0.01% decrease in ratio). These “fusion/fission” families consist of 12,567 module families (0.2% increase) covering 46,100 sequences (0.2% increase) and involve at least one gene from 4533 (36.8%) of gene families (0.6% increase). As expected, the distribution of architecture families also shifted; for example, there would be 11 families with more than 20 modules and 22 families with more than 50 sequences (compare to Figure 5.7). Such increases are particularly problematic for our STAR-MP architecture reconstruction algorithm and would likely result in increased runtime or be too complicated for MP reconstruction.

D.4 Known rates of evolutionary events

We have incorporated known tendencies in event costs where applicable in our pipeline; for example, we used known estimated duplication and loss rates in Drosophila [99] to reconstruct module phylogenies with SPIMAP. However, systematic studies of merge and split events [70, 141, 227] have only determined total counts or merge-to-split ratios, and these are neither specific to the Drosophila clade nor do they incorporate architecture counts. The few studies on fusion and fission in Drosophila [205, 289] focus on a subset of species and on genes with significant experimental evidence. In contrast, it takes a systematic, genome-wide approach to determine event rates that are unbiased and reflective of the entire genome.

D.5 Cost of evolutionary events

We analyzed a subset of 200 families using varying event costs. Note that the cost of generation does not affect STAR-MP reconstructions since each module is assumed to have been generated only once at the most recent common ancestor of all species that contain the module. In addition, in our current implementation, duplication and loss costs also have limited effect since ancestral counts are inferred using the reconstructed module phylogenies, and a module that is duplicated and subsequently lost along the same branch does not incur any cost.

We tested six settings for the event costs: one in which all events were equal (as in the main text), four in which a single event (D,L,M,S) had twice the cost of the others, and one in which
merges and splits were twice the cost of generations, duplications and losses. For each setting, we summed the number of inferred evolutionary events of each type, then computed its deviation from the inferred counts under equal costs. All deviations were less than 3.7% except for four cases: when merges had double the cost, the number of merges decreased by 23.9% and the number of splits increased by 20.3%, and when splits had double the cost, the number of merges increased by 36.6% and the number of splits decreased by 26.8%. Furthermore, while the number of inferred regions are the same across all settings (since we have used the same input module trees), the number of genes for a doubled merge cost decreased by 2.4% and for a doubled split cost increased by 2.9%. This is consistent with our expectations, as a higher merge cost should result in a larger number of merged ancestral genes (e.g. fewer genes given the same number of modules) so that fewer merges and more splits are inferred. Similarly, a higher split cost should result in a larger number of split ancestral genes (e.g. more genes given the same number of modules) so that more merges and fewer splits are inferred.

However, in this smaller set of families, each reconstruction contributes a larger portion to the total number of events; thus, many deviations could be attributed to the small number of families that have multiple maximum parsimonious reconstructions. (Remember that ties are broken randomly.) If we consider the 143 families for which only a single maximum parsimonious reconstruction exists for every setting, almost all deviations drop two-fold or more. For this filtered set, all deviations were less than 1.1% except for two cases: when splits have double the cost, the number of merges increased by 22.0% and the number of splits decreased by 9.1%. Note that a doubled merge cost negligibly affects the inferred evolutionary events, and that the deviations in merge and split counts for a doubled split cost have dropped. We believe that many of the deviations for the doubled split cost are due to cases in which an architecture is partially lost. Here, the high split cost causes STAR- MP to infer a split ancestral gene when other parameter settings would infer a merged ancestral form. (See also Section D.6.3.)

Our analysis shows that in almost all cases, the balance of inferred events is maintained since these events are constrained by the evolutionary evidence. Significant deviations may be seen if a higher split cost is used, but then, a larger number of merges and a lower number of splits will be inferred, which would further support our findings that merges are more prevalent than splits (Section 5.4.5).

D.5 Cost of evolutionary events
D.6 Validation

D.6.1 Detection of undercollapsed scaffolds

The BLASTp hits of a species versus itself were filtered to retain hits between genes located on different scaffolds and with percent identity ≥ 95%. These were run through LALIGN using nucleotide sequences extended to ± 2000 nt upstream and downstream, and hits with percent identity ≥ 98% were retained. An architecture family is said to contain possibly undercollapsed scaffolds if at least two genes within the family have a hit in this final list.

D.6.2 EST and mRNA-seq evidence

ESTs were obtained from GenBank and compared against the protein sequences using BLASTx. Only hits with long ESTs (≥ 250 nt), e-value ≤ 1 × 10^{-5}, percent identity ≥ 96%, and alignment length ≥ 50 aa were retained. If a single EST aligned to the same sequence in multiple places, the alignment with the highest percent identity was retained. An EST is said to span two genes if it aligns with both genes and the alignments are in the same direction and overlap by ≤ 15 aa.

mRNA-seqs at 36 and 75 nt resolution were obtained from modENCODE (http://www.modencode.org). Briefly, this protocol used polyA RNA extracted from D. melanogaster, D. pseudoobscura, and D. mojavensis male and female heads, with sequences aligned to the genome with Bowtie allowing for up to two mismatches. We mapped the mRNA to genes based on genome location, and an mRNA-seq was said to span two genes if it aligns with both genes and the alignments are in the same direction and do not overlap.

D.6.3 Simulations under various event rates

Keeping the generation rate constant at 1X the estimated true rate, we set the duplication, loss, merge, and split rates at 1X, 2X, and 4X the estimated true rates. We tested five different settings, simulating 1000 architecture scenarios for each setting (Figure D.1).

In general, STAR-MP has higher precision than sensitivity for any given event, and performance tends to degrade as the event rates increase and the true architecture scenarios become more complicated. Indeed, part of the decrease in sensitivity can be attributed to trying to explain more complex architecture scenarios with a conservative MP algorithm. STAR-MP also tends to have higher generation, duplication, and loss performance than merge and split performance, and as expected, generation, duplication, and loss performance is consistent across various merge and split rates. Interestingly, merge performance is typically higher than split performance. Further investigation showed that low split performance can be attributed to cases in which an architecture
is partially lost. Here, the true reconstruction is a merged parent architecture undergoing a split and loss to result in the surviving sub-architecture; however, STAR-MP tended to reconstruct a split parent architecture so that only a loss is needed to produce the surviving sub-architecture.

D.6.4 Support for fusion and fission events using transcript evidence

We excluded from this analysis the 6.1% (249/4107) of merge/split scenarios have no merge or split events. This occurred since we determined merge/split families based solely on the clustering of architectures into architecture families. However, all merge and split events in the family may have occurred prior to the species tree root; such a case can only be determined after the architecture scenario reconstruction.

Consider a merge or split event between two modules, as found by our architecture scenario reconstruction algorithm. Each of these events bifurcates the leaves, with one subset containing the leaves belonging in the subtree descended from the event, and the other subset consisting of the the rest of the leaves. We denoted these sets as the merged genes or split genes, respectively, if we were considering a merge event, or vice-versa if we were considering a split event. A merged gene was classified as consistent if there existed at least one EST/mRNA-seq that covers the boundary between the modules, and as unknown otherwise. (Note that we did not allow a merged gene to be inconsistent with the evidence.) A split gene was classified as inconsistent if there existed at least one spanning EST (mRNA-seq) (Section D.6.2) for the gene, as consistent if there existed an EST but no spanning EST (mRNA-seq), and as unknown otherwise. Once the genes were classified, an event was classified as consistent if there existed at least one consistent merged gene and at least one consistent split gene, as inconsistent if there existed at least one inconsistent split gene, and as unknown otherwise. Finally, the events for each scenario were pooled, and each scenario was classified as inconsistent if any event was inconsistent, as consistent if all events were consistent, and as unknown otherwise.

D.7 No substitution rate bias in merge/split families

Analysis of Adh-derived chimeric genes previously revealed elevated rates of amino acid substitution after merge events [123, 124, 155]. It was speculated that the new function of the chimeric gene no longer required strong conservation, or that amino acids along the merge boundary rapidly evolved to repair any possible damage incurred by the merge event. To examine whether this bias occurs at a systematic level, we computed the substitution rate for each module family (Figure D.4) using the SPIMAP model of assigning a gene-specific (here module-specific) rate and species-specific rate to
each tree [199, 200]. Notice that while the distributions are significantly different, the effect size is small (fold of median rates = 0.970-0.975). Furthermore, contrary to the results with Adh, we found that modules in merge/split families tend to have lower substitution rates than average. We believe that this discrepancy may be attributed to limitations in our model. For example, we computed rates across entire module trees, but it may be more appropriate to compute separate rates for portions of the tree affected and not affected by the merge/split event. We also did not consider differences in composition across species, which may confound the gene-specific and species-specific rate. Finally, we question whether an elevated substitution rate is indeed expected, since it is also plausible that modules should be more conserved after merge/split events in order to maintain functionality; that is, modules that undergo many substitutions may lose functionality and degenerate into pseudogenes.

D.8 GO term and Pfam domain enrichment/depletion

Enrichment/depletion values were computed using GOseq [282] to correct for possible length (Figure D.3) and substitution bias (Section D.7, Figure D.4) in the data. Briefly, GOseq determines a probability weighting function that quantifies how the probability of a gene selected out of the reference set changes as a function of some external variable such as transcript length. It then resamples the genes many times, weighting the probability of choosing a gene using this function, and uses the resulting sampling distribution to calculate a p-value. Alternatively, GOseq uses the Wallenius non-central hypergeometric distribution to approximate p-values; we use this approximation in our analysis. p-values were corrected separately for length bias and substitution bias; correction for substitution bias did not change the set of enriched/depleted terms, so only the correction for length bias is shown in the main text.

For gene functions, we looked at gene ontology (GO) annotations for D. melanogaster, as the other species have little to no GO annotation. Only GO terms with experimental evidence were retained, and a gene with a GO term was expanded to also include all parent GO terms. D. melanogaster genes contain 4524 unique GO terms, 3327 of which are found in genes that participate in architecture families with merges or splits. Enrichment/depletion values were computed separately for each of the three ontologies biological process, cellular component, and molecular function. For domains, we looked at Pfam domains for all species. The Drosophila clade contains 3204 unique Pfam domains, 1510 of which are found in genes that participate in architecture families with merges or splits. After correcting for length bias, no Pfam domains were significantly enriched or depleted (hypergeometric test, \( p < 0.001 \), FDR correction).

D Models for domain rearrangements
D.9 Functional complementarity using DroID database

Rather than looking at shared GO terms, we searched for gene partners against the Drosophila interactions database (DroID) [283] (April 2010 release). Of the 1222 gene partners, 589 are those in which both genes have at least one known PPI (but not necessary with each other) and 135 (22.9% of 589) are those in which the genes are known to interact with each other. This is compared to 0.3% random (fold = 75.74, p < 0.001).

Using a set of high confidence interactions in which we retain only PPI with experimental evidence (discarding those detected through homology), these numbers reduced to 57 gene partners in which both genes have at least one known PPI, 24 (42.1%) in which the genes are known to interact with each other, and 0.38% random (fold = 110.53, p < 0.001). However, low counts means that we must take care in making any biological statements.

D.10 PPI in conservative set of architecture families

Using the conservative set of architecture families, we identified 222 gene partners within D. melanogaster. Of these, 33 gene partners have both genes annotated with GO terms, and 30 (90.9% of 33) share at least one GO term, compared to 61.8% random (fold = 1.47, p < 0.001). Using the DroID database, 140 gene partners are those in which both genes have at least one known PPI and 48 (34.3%) are those in which the genes are known to interact with each other, compared to 0.3% random (fold = 107.05, p < 0.001). Using the set of high confidence interactions, these numbers reduced to 16 gene partners in which both genes have at least one known PPI, 5 (31.3%) in which the genes are known to interact with each other, and 0.37% random (fold = 83.48, p < 0.001).

D.11 Detection of frameshift mutations

To investigate how often nucleotide similarity is not propagated to peptide similarity, we ran pairwise all-vs-all tBLASTx between the species, then post-processed the alignments with LALIGN and filters (e-value $\leq 1 \times 10^{-5}$ and percent identity $\geq 80$; a higher threshold for percent identity was used to account for the smaller nucleotide alphabet). We found 2740 hits at the nucleotide level but not at the peptide level, 130,422 hits at both the nucleotide and peptide level, and 345,554 hits at the peptide level but not at the nucleotide level. This translated to 1429 additional genes that may participate in a merge/split architecture family; note that the actual number of additional genes that participate in merge/split families is likely smaller than this count. For example, some of the genes may be part of families in which each gene consists of a single module, and the module is
simply frameshifted in some genes, or some of the frameshift mutations may be a result of frameshift sequencing errors.

D.12 Systematic detection of gene fusion and fission by mechanism

D.12.1 Fusion/fission of adjacent genes

Two modules that merge or split were tagged if they were found in neighboring genes and the modules would be adjacent if the neighboring genes were considered as a single gene. This list was expanded to genes by looking for all genes with these tagged modules and including only those genes that are neighboring or are found in species descended from the most ancestral branch with a merge or split of the modules. Note that this list of genes includes both parental and children genes (e.g. pre- and post-merge/split genes), as this allows for ambiguities that may have arisen from ties in the MP reconstruction (e.g. one MP reconstruction finds split genes at the root followed by a merge along one branch, whereas another reconstruction finds a merged gene at the root followed by a split along the other branch). We also tested for experimental support for each gene by looking at EST and mRNA-seq data. Using our previously determined set of spanning ESTs (mRNA-seqs) (Section D.6.2), we called a gene consistent if all of the associated fragmented genes had EST (mRNA-seq) evidence but none were part of a spanning EST (mRNA-seq), inconsistent if at least one of the associated fragmented genes was part of a spanning EST (mRNA-seq), and unknown otherwise.

Genes that may have resulted from large-loop mismatch repair and replication slippage were detected by looking for merged genes flanked by (but not necessarily next to) two genes, with one gene containing one of the merged modules and the other gene containing the other merged module.

D.12.2 Retrotransposition and exon shuffling

While retrotransposition and exon shuffling are two separate mechanisms for novel gene formation, it has been suggested that retrotransposition is a driving mechanism for mediating exon shuffling [80], and exon shuffling by retrotransposition is one method for conferring novel gene functions to the resulting chimeric gene rather than allowing the retrosequence to degenerate into a pseudogene [153]. To find retrotransposed-mediated exon shuffling events, we searched for modules that undergo a merge or split and that have multiple exons in at least one gene with merged form and a single exon in at least one gene with split form. This ignores possible retrotranspositions of single-exon
genes which cannot be distinguished in our analysis from simple duplication. Exon comparison was performed at the module level to allow for (1) a chimeric gene to result from a multi-exon parent fusing with a second retrotransposed parent, and (2) partial gene retrotransposition. Furthermore, each merge or split formed a bifurcation of the species tree, and the merged form and split form must belong to different subtrees formed from this bifurcation. Only genes where the module has a single exon were included in the final count.

D.12.3 Duplication-degeneration

We constructed syntenic modules by defining a syntenic block to be at least three genes within 200 kb of each other with no other blocks in between. Two genes are the result of duplication-degeneration if (1) they result from a split, and (2) they belong to different syntenic blocks but have hits to the same syntenic block. False detection could occur due to faulty syntenic blocks. For example, a missed connection between two scaffolds may result in one syntenic block being separated into two blocks. As a simple test, we checked whether both genes are located at the ends of their respective scaffolds; 36 modules and 50 genes remain. Note that this method of detection does not take into account the case of undercollapsed assemblies where genes are located within the scaffolds.

D.13 Additional figures and tables

<table>
<thead>
<tr>
<th>Input rate multipliers</th>
<th>actual rate multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>D L M S</td>
<td>D L M S</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>0.98 0.94 0.98 0.97</td>
</tr>
<tr>
<td>1 1 2 2</td>
<td>0.98 0.97 1.40 1.39</td>
</tr>
<tr>
<td>1 1 4 4</td>
<td>0.98 0.98 1.97 1.97</td>
</tr>
<tr>
<td>2 2 2 2</td>
<td>1.74 1.57 1.23 1.27</td>
</tr>
<tr>
<td>4 4 4 4</td>
<td>3.23 2.61 1.66 1.74</td>
</tr>
</tbody>
</table>

Figure D.1: Sensitivity and precision of STAR-MP in simulation under various event rates. See Figure 5.5 for details. Sensitivity decays dramatically as event rates increase, as is consistent with a conservative MP reconstruction. In contrast, precision is robust to event rates. Event rate multipliers were obtained from the simulations and differ from the input rate multipliers of N: (1,1,1,1), MS2: (1,1,2,2), MS4: (1,1,4,4), all2: (2,2,2,2), and all4: (4,4,4,4) due to our approach of discarding events that were impossible with the given starting architecture.

D.13 Additional figures and tables
Figure D.2: **Correlation of module and exon boundaries.** See Figure 5.6 for details.

Figure D.3: **Length bias.** Genes in merge/split families tend to be longer than average. *(Left)* The distribution of gene lengths, bin size = 50 aa. Median lengths for all genes and for genes in merge/split families were 361 aa and 492 aa, respectively; the distributions differed significantly (Mann-Whitney U = 3.22 × 10⁹, p < 2.23 × 10⁻³⁰⁸, one-sided). *(Right)* The distribution of median gene lengths per architecture family, bin size = 50 aa. Median lengths for all families and for merge/split families were 391 aa and 524 aa, respectively; the distributions differed significantly (Mann-Whitney U = 3.19 × 10⁷, p = 9.23 × 10⁻¹¹⁶, one-sided).
Figure D.4: **No substitution rate bias.** Genes in merge/split families have similar substitution rates compared to average. *(Left)*, The distribution of substitution rates across module families, bin size = 0.1 sub/site. Median rates for all modules and for modules in merge/split families were 0.723 sub/site and 0.701 sub/site, respectively; the distributions differed significantly (Mann-Whitney $U = 1.35 \times 10^8$, $p = 5.23 \times 10^{-10}$, one-sided). *(Right)*, The distribution of median substitution rates per architecture family, bin size = 50 aa. Median rates for all families and for merge/split families were 0.744 sub/site and 0.725 sub/site, respectively; the distributions differed significantly (Mann-Whitney $U = 2.85 \times 10^7$, $p = 1.73 \times 10^{-4}$, one-sided).
Figure D.5: Distribution of inferred evolutionary events across architecture families. Event counts are not dominated by a subset of families. Most families have at most one event per type along each branch.
Figure D.6: Total counts of evolutionary events inferred across a conservative set of architecture families. See Figure 5.8 for details.
Figure D.7: Distribution of inferred evolutionary events across a conservative set of architecture families. See Figure D.5 for details.
Table D.1: EST evidence for genes in merge/split families

<table>
<thead>
<tr>
<th>species</th>
<th># genes with EST (mRNA-seq)</th>
<th># gene pairs with EST (mRNA-seq)</th>
<th># gene pairs with spanning EST (mRNA-seq)</th>
<th>error rate (%) of EST (mRNA-seq)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmel</td>
<td>4,769</td>
<td>4,439</td>
<td>420</td>
<td>5.17 (3.00)</td>
</tr>
<tr>
<td>dyak</td>
<td>5,604</td>
<td>429</td>
<td>806</td>
<td>22.22</td>
</tr>
<tr>
<td>dere</td>
<td>5,169</td>
<td>1,697</td>
<td>715</td>
<td>11.86</td>
</tr>
<tr>
<td>dana</td>
<td>4,988</td>
<td>1,804</td>
<td>655</td>
<td>11.96</td>
</tr>
<tr>
<td>dpse</td>
<td>5,092</td>
<td>935 (4,587)</td>
<td>618</td>
<td>20.00 (3.23)</td>
</tr>
<tr>
<td>dwil</td>
<td>4,827</td>
<td>1,592</td>
<td>507</td>
<td>10.14</td>
</tr>
<tr>
<td>dmoj</td>
<td>4,797</td>
<td>1,713 (4,507)</td>
<td>640</td>
<td>16.13 (6.26)</td>
</tr>
<tr>
<td>dvir</td>
<td>4,809</td>
<td>1,847</td>
<td>654</td>
<td>14.81</td>
</tr>
<tr>
<td>total</td>
<td>45,282</td>
<td>16,233 (13,549)</td>
<td>5,666</td>
<td>9.35 (4.42)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>species</th>
<th># families with EST (mRNA-seq)</th>
<th># families with spanning EST (mRNA-seq)</th>
<th>error rate (%) of EST (mRNA-seq)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmel</td>
<td>284 (296)</td>
<td>18 (10)</td>
<td>6.34 (3.38)</td>
</tr>
<tr>
<td>dyak</td>
<td>17</td>
<td>4</td>
<td>23.53</td>
</tr>
<tr>
<td>dere</td>
<td>55</td>
<td>7</td>
<td>12.73</td>
</tr>
<tr>
<td>dana</td>
<td>77</td>
<td>11</td>
<td>14.29</td>
</tr>
<tr>
<td>dpse</td>
<td>23 (413)</td>
<td>6 (16)</td>
<td>26.09 (3.87)</td>
</tr>
<tr>
<td>dwil</td>
<td>58</td>
<td>7</td>
<td>12.07</td>
</tr>
<tr>
<td>dmoj</td>
<td>55 (465)</td>
<td>10 (36)</td>
<td>18.18 (7.74)</td>
</tr>
<tr>
<td>dvir</td>
<td>77</td>
<td>12</td>
<td>15.58</td>
</tr>
<tr>
<td>dgri</td>
<td>64</td>
<td>3</td>
<td>4.69</td>
</tr>
<tr>
<td>total</td>
<td>451 (766)</td>
<td>52 (51)</td>
<td>11.53 (6.66)</td>
</tr>
</tbody>
</table>

*See Table 5.1 for details. Similar statistics are also provided across architecture families, where at least one gene pair must be present in the architecture family for it to be included in the count. Large error rates can be attributed to low counts.

Table D.2: Intron phases of exon-bordering domains

<table>
<thead>
<tr>
<th>species</th>
<th>0-0</th>
<th>1-1</th>
<th>2-2</th>
<th>symmetric</th>
<th>non-symmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>O</td>
<td>E</td>
<td>O/E</td>
<td>O</td>
<td>E</td>
</tr>
<tr>
<td>dmel</td>
<td>665</td>
<td>246</td>
<td>2.71</td>
<td>149</td>
<td>146</td>
</tr>
<tr>
<td>dyak</td>
<td>745</td>
<td>262</td>
<td>2.85</td>
<td>143</td>
<td>152</td>
</tr>
<tr>
<td>dere</td>
<td>693</td>
<td>250</td>
<td>2.78</td>
<td>137</td>
<td>145</td>
</tr>
<tr>
<td>dana</td>
<td>770</td>
<td>275</td>
<td>2.80</td>
<td>139</td>
<td>157</td>
</tr>
<tr>
<td>dpse</td>
<td>784</td>
<td>279</td>
<td>2.81</td>
<td>164</td>
<td>166</td>
</tr>
<tr>
<td>dwil</td>
<td>750</td>
<td>269</td>
<td>2.79</td>
<td>149</td>
<td>162</td>
</tr>
<tr>
<td>dmoj</td>
<td>703</td>
<td>250</td>
<td>2.82</td>
<td>147</td>
<td>143</td>
</tr>
<tr>
<td>dvir</td>
<td>734</td>
<td>258</td>
<td>2.85</td>
<td>139</td>
<td>152</td>
</tr>
<tr>
<td>dgri</td>
<td>809</td>
<td>284</td>
<td>2.84</td>
<td>165</td>
<td>167</td>
</tr>
<tr>
<td>total</td>
<td>6,656</td>
<td>2,372</td>
<td>2.81</td>
<td>1,332</td>
<td>1,389</td>
</tr>
</tbody>
</table>

*See Table 5.4 for details.

*All p-values satisfied p < 1 × 10^-5 except for the 1-1 domains.

D.13 Additional figures and tables
Table D.3: GO enrichment for genes undergoing module rearrangement in conservative set of architecture families

<table>
<thead>
<tr>
<th>rank</th>
<th>GO ID</th>
<th>GO term</th>
<th>k</th>
<th>m</th>
<th>fold</th>
<th>p-value</th>
<th>q-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GO:0007275</td>
<td>multicellular organismal development</td>
<td>323</td>
<td>119</td>
<td>1.43</td>
<td>2.24 × 10^{-13}</td>
<td>2.44 × 10^{-6}</td>
</tr>
<tr>
<td>2</td>
<td>GO:0032502</td>
<td>developmental process</td>
<td>355</td>
<td>1253</td>
<td>1.40</td>
<td>2.02 × 10^{-13}</td>
<td>3.41 × 10^{-6}</td>
</tr>
<tr>
<td>3</td>
<td>GO:0032501</td>
<td>multicellular organismal process</td>
<td>382</td>
<td>1358</td>
<td>1.39</td>
<td>6.30 × 10^{-14}</td>
<td>2.79 × 10^{-6}</td>
</tr>
</tbody>
</table>

See Table 5.5 for details.

Here, \( n = 2506 \) and \( N = 12,408 \), and we used a p-value cutoff of \( p < 0.01 \).

Table D.4: Retainment of ancestral architectures by merge and split events in conservative set of architecture families

<table>
<thead>
<tr>
<th></th>
<th>all</th>
<th>without generation</th>
<th>with generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MERGES</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>number of events</td>
<td>2559</td>
<td>1676 (65.5%)</td>
<td>883 (34.5%)</td>
</tr>
<tr>
<td>retained at least one split architecture</td>
<td>2392 (93.5%)</td>
<td>1620 (96.7%)</td>
<td>772 (87.4%)</td>
</tr>
<tr>
<td>retained both split architectures</td>
<td>1413 (55.2%)</td>
<td>1413 (84.3%)</td>
<td>n/a</td>
</tr>
<tr>
<td>SPLITS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>number of events</td>
<td>2446</td>
<td>670 (27.4%)</td>
<td>1776 (72.6%)</td>
</tr>
<tr>
<td>retained merged architecture</td>
<td>465 (19.0%)</td>
<td>443 (66.1%)</td>
<td>22 (1.2%)</td>
</tr>
</tbody>
</table>

See Table 5.7 for details.

Table D.5: Lineage-specific merge and split events in conservative set of architecture families

<table>
<thead>
<tr>
<th>species</th>
<th>dist</th>
<th>M</th>
<th>S</th>
<th>% dist</th>
<th>M</th>
<th>S</th>
<th>% dist</th>
<th>M</th>
<th>S</th>
<th>% dist</th>
<th>M</th>
<th>S</th>
<th>% dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>dmel</td>
<td>11.23</td>
<td>174</td>
<td>122</td>
<td>2.9</td>
<td>6.8</td>
<td>2.3</td>
<td>4.2</td>
<td>4.2</td>
<td>1.4</td>
<td>3.7</td>
<td>13.7</td>
<td>3.7</td>
<td>5.2</td>
</tr>
<tr>
<td>dyuk</td>
<td>8.57</td>
<td>110</td>
<td>309</td>
<td>2.2</td>
<td>4.3</td>
<td>1.9</td>
<td>10.7</td>
<td>4.9</td>
<td>2.8</td>
<td>2.8</td>
<td>8.7</td>
<td>3.1</td>
<td>13.5</td>
</tr>
<tr>
<td>dere</td>
<td>8.57</td>
<td>106</td>
<td>169</td>
<td>2.2</td>
<td>4.1</td>
<td>1.9</td>
<td>5.9</td>
<td>2.7</td>
<td>2.8</td>
<td>8.4</td>
<td>3.0</td>
<td>7.4</td>
<td>2.6</td>
</tr>
<tr>
<td>dana</td>
<td>53.40</td>
<td>146</td>
<td>271</td>
<td>13.8</td>
<td>5.7</td>
<td>0.4</td>
<td>9.4</td>
<td>0.7</td>
<td>17.6</td>
<td>11.5</td>
<td>0.7</td>
<td>11.9</td>
<td>0.7</td>
</tr>
<tr>
<td>dpec</td>
<td>55.80</td>
<td>170</td>
<td>248</td>
<td>14.4</td>
<td>6.6</td>
<td>0.5</td>
<td>8.6</td>
<td>0.6</td>
<td>18.4</td>
<td>13.4</td>
<td>0.7</td>
<td>10.9</td>
<td>0.6</td>
</tr>
<tr>
<td>dwil</td>
<td>62.49</td>
<td>146</td>
<td>293</td>
<td>16.1</td>
<td>5.7</td>
<td>0.4</td>
<td>10.2</td>
<td>0.6</td>
<td>20.6</td>
<td>11.5</td>
<td>0.6</td>
<td>12.8</td>
<td>0.6</td>
</tr>
<tr>
<td>dmoj</td>
<td>32.74</td>
<td>152</td>
<td>282</td>
<td>8.4</td>
<td>5.9</td>
<td>0.7</td>
<td>9.8</td>
<td>1.2</td>
<td>10.8</td>
<td>12.0</td>
<td>1.1</td>
<td>12.3</td>
<td>1.1</td>
</tr>
<tr>
<td>dvir</td>
<td>32.74</td>
<td>137</td>
<td>204</td>
<td>8.4</td>
<td>5.3</td>
<td>0.6</td>
<td>7.1</td>
<td>0.8</td>
<td>10.8</td>
<td>10.8</td>
<td>1.0</td>
<td>8.9</td>
<td>0.8</td>
</tr>
<tr>
<td>dgri</td>
<td>37.11</td>
<td>127</td>
<td>386</td>
<td>9.6</td>
<td>4.9</td>
<td>0.5</td>
<td>13.4</td>
<td>1.4</td>
<td>12.3</td>
<td>10.0</td>
<td>0.8</td>
<td>16.9</td>
<td>1.4</td>
</tr>
</tbody>
</table>

See Table 5.8 for details.
D.14 Bayesian reconstruction of domain rearrangements

The methods presented in this thesis (Chapter 5) relied on the parsimonious reconstruction of evolutionary histories, which allowed us to limit the number of scenarios to consider, resulting in high speed and accuracy. A major challenge going forward is to extend these methods to propagate sequence information across all possible reconstructions, similar to existing Bayesian and maximum likelihood phylogenetic methods, which we believe could better capture the evolutionary history of architecture families. In particular, such probabilistic methods could allow for the modeling of branch lengths within an architecture DAG (rather than being limited to architecture scenarios) and thus place evolutionary events at specific timepoints within the species history. This could also allow the simultaneous modeling of both sequence and architecture evolution, rather than the current approaches of utilizing sequence to reconstruct module trees and then either using architecture to reconstruct architecture scenarios or using reconciliation to determine module insertions and deletions. We outline such an approach in the remainder of this appendix.

D.14.1 The plexus model

Definition

The plexus model for architecture evolution [271, 272] is an extension of the tree model for gene evolution and describes how a set of genes have changed through gene and subgene evolutionary events over time. Here, we outline the basic structure of the plexus; more details are provided in Section D.14.5.

Recall that in an architecture scenario, ancestral architectures and evolutionary events are mapped onto a known species tree such that each node of the species tree depicts the type and copy number of architectures within the species, and each branch of the species tree depicts the events that have occurred along that branch (Figure D.8A). In comparison to this summary view, an architecture plexus is based directly on the concept of gene architecture histories as directed acyclic graphs, in which nodes of the graph represent genes (and their architectures), and edges of the graph trace the history of underlying modules (Figure D.8B).

Formally, let $T = (V, E)$ be a rooted tree with vertices (nodes) $V$ and directed edges (branches) $E$, and let $F = \{T\}$ be a forest of such trees. Then $P = (F, B, A)$ is a plexus with forest $F$, blocks $B$, and directed arcs $A$ if the following conditions are satisfied: (1) $P$ is a directed acyclic graph (DAG) with vertices $B$ and edges $A$, (2) $B$ is a partition over $\bigcup_{T \in F} V(T)$ such that vertices within any block $b \in B$ are ordered and are not related (where not related means that $\forall b \in B$ and $v_1, v_2 \in b : v_1 \neq v_2, v_1 \text{ and } v_2 \text{ are in different trees}$), and (3) $(b_1, b_2) \in A$ if and only if $\exists v_1 \in b_1, \exists v_2 \in b_2$. 
Figure D.8: Architecture histories. (A) An architecture scenario depicts the multiset of architectures and evolutionary events mapped onto a known species, with architectures evolving from a parent species to a child species. Thus, no reconciliation is required, and speciation events are not modeled. (B) An architecture plexus overlays blocks (representing genes) onto module phylogenies to show how modules have rearranged over time.

\[ v_2 \in b_2 \text{ such that } (v_1, v_2) \in \bigcup_{T \in P} E(T) \] (that is, the arcs are fully determined by the underlying forest of trees).

The plexus\(^1\) structure lends itself well to modeling architecture evolution, with each tree representing a module phylogeny, each block a gene, and each arc an inheritance relationship.

**Architecture evolution in a plexus**

For a plexus to represent architecture evolution, its blocks and arcs must represent evolutionary events. We allow the following events (Figure D.9):

- **generation:** Either a new block consisting of a single (root) node is generated, or a parent block yields a child block with an additional root node generated at its start or end.

- **duplication:** A parent block duplicates to form two child blocks whose node structures are identical to the parent block.

- **loss:** Either a parent block consisting of a single node is lost, or a parent block yields a child block in which one of the nodes in the parent block has been lost.

- **merge:** Two parent blocks merge to form a child block. In the child block, all child nodes of one of the parent blocks must precede the child nodes of the other parent block.

\(^1\)A plexus is a network of branching and rejoining nerves, in which the nerve fibers travel together but electrical signals remain separate.

**D Models for domain rearrangements**
split: A parent block splits to form two child blocks. In the parent block, all parent nodes of one of the child blocks must precede the parent nodes of the other child block.

In all cases, the nodes of the parent block(s) retain their relative order in the child block(s). Furthermore, note that these events differ from those defined in Section D.1 and used in Chapter 5. In particular, we now allow existing architectures to be extended; for example, merges are free between a generated module and an existing module, and duplications now occur at the gene-level rather than the module-level. Simple revisions to either model would make the sets of evolutionary events consistent.

**D.14.2 Generative model of architecture evolution**

We now present a generative model of architecture evolution: Given a species with a specified topology and speciation times, (1) we generate initial architectures at the species root, then generate an architecture plexus topology, along with event times, by repeated used of a Markov process, (2) we then generate substitution rates from gene and species-specific distributions, and (3) we use these rates to generate molecular sequences according to a continuous-time Markov process. This generative model is based heavily on the SPIMAP model for gene tree evolution [200], with the only difference being the first step, in which we generate an architecture plexus topology rather than a gene tree topology. We will therefore focus here on generating the topology and only review the substitution rate and sequence evolution model.

The parameters in our model are $\theta = (S, t, \theta_t, \theta_b)$, where $S$ and $t$ are the species tree topology and branch lengths, $\theta_t = (\pi, \kappa, \lambda, \mu, \nu, \xi)$ are the plexus topology parameters, and $\theta_b = (\alpha_G, \beta_G, \alpha, \beta)$ are the branch length parameters.

**Generating topology and divergence times**

The birth-death (BD) process has been widely used to model gene duplication and loss [8, 9, 195, 200], and its extension to a birth, death, and innovation model (BDIM) [127, 128] has been used to explain

---

**Figure D.9: Evolutionary events in a plexus.**
observed distributions of domain family sizes. In this work, we build on these models and extend the simple BD process to a generalized Markov process that also captures module generation, fusion, and fission, which we call the innovation-birth-death-merge-split (IBDMS) process.

The IBDMS process generates a plexus according to a constant rate $\kappa$ of lineage innovation (representing module or gene generation), $\lambda$ of lineage bifurcation (representing gene duplication), $\mu$ of (partial) lineage termination (representing module or gene loss), $\nu$ of lineage consolidation (representing gene fusion), and $\xi$ of lineage division (representing gene fission). When choices are possible (for example, where to place a generated node, which block comes first in a merge, where to split a block), we assume that each possibility occurs with equal probability.

In this model, the initial distribution of architectures at the species root is given by $\pi$, and we apply the IBDMS process repeatedly along each species tree branch, effectively evolving the architecture plexus "inside" the species tree to generate an architecture plexus topology and arc lengths (in units of time). In detail, (1) we initialize the architecture plexus topology by sampling from $\pi$ and set the reconciliation of these blocks $B$ to the root of $S$, that is, $\forall b \in B, R(b) = \text{root}(S)$. (2) For each set of blocks $B$ at the top of species branch $b(u)$ of length time $t(u)$, we generate a plexus according to the IBDMS process for $t(u)$ units of time, marking blocks along the way by their evolutionary event. (3) For each newly created block $b$, we record its reconciliation as $R(b) = u$. (4) For each $b$ that survives across that species branch, we mark it as an extant gene if if $u$ is a leaf species, otherwise we mark it as a speciation event. (5) We recursively apply steps 2–4 until all speciation nodes have been processed. (6) As a post-processing step, we prune all doomed lineages.

This topology model has parameters $\Theta = (\pi, \kappa, \lambda, \mu, \nu, \xi)$, where $\pi$ is the probability distribution of architectures at the species root, and $(\kappa, \lambda, \mu, \nu, \xi)$ correspond to the generation, duplication, loss, merge, and split rates, respectively.

**Generating substitution rates**

We use a relaxed clock model, in which substitution rates are allowed to vary between lineages [200]. Note that in our extension from a gene tree to an architecture plexus, rates for the plexus are generated independently for each branch of the underlying module trees. Future investigations may consider a model in which substitution rates are generated for each arc in the plexus, and these rates are propagated to each underlying branch.

For a node $v$ in a tree, let $l(v)$ denote the length of the branch $\text{parent}(v), v)$. Furthermore, let $l$ denote the vector of all branch lengths in a tree. Then for each tree in a plexus, each branch has a length $l(v)$ (substitutions/site), which is a product of a duration of time $t(v)$ (myr) and a substitution rate $r(v)$ (substitutions/site/myr). The times are given by the Markov model, and the
substitution rate is a product of a module-specific rate $g_j$ and a species-specific rate $s_k$, where

$$P(G_j = g_j | \beta_G) = \text{InvGamma}(g_j | \alpha_G = \beta_G + 1, \beta_G)$$

$$P(S_k = s_k | \alpha_i, \beta_i) = \text{Gamma}(s_k | \alpha_i, \beta_i)$$

and $(\alpha_G, \beta_G)$ are module-specific branch parameters and $(\alpha_i, \beta_i)$ are species-specific branch parameters that depend on the reconciled species $u_i = R(v_k)$.\(^1\) Putting everything together, the branch length $l(v_k)$ of module tree $j$ can be expressed as $l(v_k) = r(v_k) \times t(v_k) = g_j \times s_k \times t(v_k)$. In total, this rate model has parameters $\theta_b = (\beta_G, \alpha, \beta)$, where $\alpha = (\alpha_1, \ldots, \alpha_m)$, $\beta = (\beta_1, \ldots, \beta_m)$, and $m$ is the number of species branches $|E(S)|$.

**Generating sequence**

Molecular sequences are evolved down each tree using a continuous-time Markov chain to model sequence substitution. In particular, we use the HKY model \cite{HKY} to generate nucleotide sequences. Note that only sequences on the leaves of each tree are emitted, that each site evolves independently, and that sequence insertion and deletion are not modeled, that is, gaps in the sequence alignment are treated as missing data.

**D.14.3 Maximum a posteriori reconstruction of architecture evolution**

Given the sequence data $D$ and model parameters $\theta$, our goal is to find the MAP architecture plexus defined by the branch lengths $l$, the topology $T$, and the reconciliation $R$. As in SPIMAP \cite{SPIMAP},

$$\hat{l}, \hat{T}, \hat{R} = \arg\max_{l, T, R} P(l, T, R | D, \theta)$$

$$= \arg\max_{l, T, R} P(D | l, T, R, \theta) P(l | T, R, \theta) P(T, R | \theta) / P(D | \theta)$$

$$= \arg\max_{l, T, R} P(D | l, T) P(l | T, R, \theta) P(T, R | \theta).$$

In the final equation, the first term corresponds to the sequence likelihood and is defined by the sequence evolution model, the second term corresponds to the branch length prior and is defined by the substitution rate model, and the third term corresponds to the topology prior and is defined by the Markov model. As computation of the first and second terms have previously been described \cite{SPIMAP}, what remains to be completed are efficient methods for (1) searching the space of

\(^1\)In other words, the module-specific rate is represented as an inverse-gamma-distributed random variable $G_j$ with shape and scale parameters $(\alpha_G, \beta_G)$, and without loss of generality, $G_j$ is constrained to a mean value of one across all module families, that is, $\alpha_G = \beta_G + 1, \alpha_G > 1$. The species-specific rate is represented as a gamma-distributed random variable $S_k$ with shape and scale parameters $(\alpha_i, \beta_i)$. Furthermore, we assume that each $S_k$ is independent of the others and of the gene rate $G_j$.

**D.14 Bayesian reconstruction of domain rearrangements**

201
architecture topologies, (2) computing the topology prior, and (3) estimating the model parameters.

D.14.4 Differences in plexus formulation

Here, we outline differences between our plexus definition and its original formulation [271, 272]. Whereas the original plexus model ignored node order within a block, thus tracing the evolution of module composition, we trace architecture evolution and thus consider node order. Furthermore, in the original plexus model, a vertex was related to another if it was an ancestor or descendant of the other. We make a stronger assumption and require that the pair of vertices belong to separate trees; this is done to mitigate the effects of short length repeat domains. Finally, evolutionary events have also been defined in a slightly different way in order to incorporate event timings, and the repeat event has been removed and the split event explicitly modeled (in the original formulation, a split was modeled using a sequence of duplication and loss events or a sequence of duplication and merge events).

D.14.5 Formal plexus definition

To avoid confusion, we use math script to denote elements of a tree and refer to these as nodes and edges (or branches), and we use calligraphy script to denote elements of a plexus and refer to these as blocks and arcs.

We will use several functions to indicate tree and plexus relationships. For a tree $T$, let $\text{root}(T)$ represent its root node and $\text{leaves}(T)$ represent its set of leaf nodes, and similarly, for a plexus $P$, let $\text{roots}(P)$ represent its set of root blocks and $\text{leaves}(P)$ represents its set of leaf blocks. For a node $v$, let $\text{parent}(v)$ represent its parent node, $\text{children}(v)$ represent its set of children nodes, $\text{tree}(v)$ represent its tree, $\text{block}(v)$ represent its block, and $\text{rank}(v)$ represent its rank (index) in its block. For a block $b$, let $|b|$ represent the number of nodes within $b$, $\text{parents}(b)$ represent its set of parent blocks, and $\text{children}(b)$ represent its set of children blocks. Finally, for two blocks $b_1$ and $b_2$ such that $(b_1, b_2) \in A$,

1. let $b_1 \prec b_2$ if and only if $\forall v_1, v_2 \in b_1, v_3, v_4 \in b_2$ where $\text{tree}(v_1) = \text{tree}(v_3)$ and $\text{tree}(v_2) = \text{tree}(v_4)$, if $\text{rank}(v_1) < \text{rank}(v_2)$, then $\text{rank}(v_3) < \text{rank}(v_4)$, and

2. let $b_1 \vdash b_2$ if and only if $b_1 \prec b_2$ and $\bigcup_{v \in b_1} \text{children}(v) = \bigcup_{v \in b_2} v$, or equivalently, $\bigcup_{v \in b_1} v = \bigcup_{v \in b_2} \text{parent}(v)$.

In simple terms, the first relationship states that nodes within the same tree maintain the same order within a pair of parent-child blocks, and the second relationship states that a parent block yields a child block with no changes in the underlying nodes.
To model architecture evolution, we enforce a set of rules for the plexus. Let block $b$ have parent blocks $P$ and children blocks $C$. Then, for $(P, b, C)$ to represent an evolutionary event, we must have (a) $\forall p \in P, p \prec b$, (b) $\forall c \in C, b \prec c$, and (c) one of the following conditions must be satisfied:

- **generation:** $C = \{c\}$, $b \prec c$, and either (1) $P = \emptyset$, $|b| = 1$, or (2) $P = \{p\}$, $\bigcup_{v \in b} \text{children}(v) \subseteq \bigcup_{v \in c} \text{children}(v)$, $|p| + 1 = |b|$, and for $n = \{v : v \in b, v = \text{root}(\text{tree}(v))\}$ and $O = \{v : v \in b, v \neq \text{root}(\text{tree}(v))\}$, either (a) $\text{rank}(n) = 0$ and $\forall v \in O, \text{rank}(v) = \text{rank}(\text{parent}(v)) + 1$, or (b) $\text{rank}(n) = |b| - 1$ and $\forall v \in O, \text{rank}(v) = \text{rank}(\text{parent}(v))$

- **duplication:** $P = \{p\}, C = \{c_1, c_2\}, p \vdash b$, $\bigcup_{v \in b} \text{parent}(v) = \bigcup_{v \in c_1} \text{parent}(v) = \bigcup_{v \in c_2} \text{parent}(v)$

- **loss:** $P = \{p\}, p \vdash b$, and either (1) $C = \emptyset$, $|b| = 1$, or (2) $C = \{c\}, \bigcup_{v \in c} \text{parent}(v) \subseteq \bigcup_{v \in b} \text{parent}(v)$, $|b| - 1 = |c|$

- **merge:** $P = \{p_1, p_2\}, C = \{c\}, b \vdash c$, $\bigcup_{v \in p_1 \cup p_2} v = \bigcup_{v \in c} \text{parent}(v)$, $p_1 \prec b$, $p_2 \prec b$, and $\forall v_1 \in p_1, v_2 \in p_2, c_1 \in \text{children}(v_1) : c_1 \in b, c_2 \in \text{children}(v_2) : c_2 \in b$, either (1) $\text{rank}(c_1) < \text{rank}(c_2)$ in all instances or (2) $\text{rank}(c_1) > \text{rank}(c_2)$ in all instances

- **split:** $P = \{p\}, C = \{c_1, c_2\}, p \vdash b$, $\bigcup_{v \in b} v = \bigcup_{v \in c_1 \cup c_2} \text{parent}(v)$, $b \prec c_1$, $b \prec c_2$, and $\forall v_1 \in c_1, v_2 \in c_2, p_1 = \text{parent}(v_1), p_2 = \text{parent}(v_2)$, either (1) $\text{rank}(p_1) < \text{rank}(p_2)$ in all instances or (2) $\text{rank}(p_1) > \text{rank}(p_2)$ in all instances
Bibliography


