The Dissertation Committee for Stacia Kathleen Wyman
certifies that this is the approved version of the following dissertation:

Algorithms for the Analysis of Whole Genomes

Committee:

Benjamin Kuipers, Supervisor

Robert K. Jansen, Supervisor

Jeffrey L. Boore

David M. Hillis

C. Gregory Plaxton

Algorithms for the Analysis of Whole Genomes

by

Stacia Kathleen Wyman, B.A., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2004

Acknowledgments

I would like to thank my committee for making it possible for me to do interdisciplinary work in Computational Biology. I could not have done it without the steadfast support and encouragement of Bob Jansen. I would like to thank all the people in the Jansen and Theriot lab for being supportive over the past years and, in particular, Andrew Alverson for his excellent proofreading skills, humor and moral support. I would like to thank Katherine St. John for tirelessly proofreading drafts of this document and for being an excellent mentor and friend.

Stacia Kathleen Wyman

The University of Texas at Austin
August 2004

Algorithms for the Analysis of Whole Genomes

Publication No. _____

Stacia Kathleen Wyman, Ph.D.
The University of Texas at Austin, 2004

Supervisors: Benjamin Kuipers, Robert K. Jansen

With the advent of whole genome sequencing, we now have an abundance of whole genomes which have been sequenced and we have entered an era when algorithms can address problems at the whole genome level. In the past, sequencing efforts often focused on a single gene, and therefore, existing algorithms are at the scale of a single gene. With whole genome sequencing, we have access to sequence data for the entire genome of an organism or an organelle and algorithms are needed for whole genome analysis. In this research, we have addressed new computational problems that have arisen out of the availability and abundance of whole genome data.

In genome annotation, all of the genes of a genome are located and identified in preparation for publication of the complete genome sequence. We address the problem of genome annotation with a software package that allows researchers to locate and identify all the genes in a genome and prepare the genome for direct submission to GenBank. A difficult problem that arises in the annotation of organellar genomes is the identification of animal mitochondrial transfer RNA genes. We present an experimental evaluation a set of methods (including our own) for

identifying tRNAs.

The problem of reconstructing phylogenies from gene order data involves recreating the evolutionary history of a set of organisms based on the order and direction of the genes in the genomes. This can give insight into mechanisms of large-scale evolutionary events. We present a new method for gene order phylogeny reconstruction, as well as improvements to an existing method, and evaluate the results on both real and simulated datasets.

Finally, we address the problem of identification of regulatory elements. Understanding gene expression is one of the most pressing unsolved problems in molecular biology today because gene expression controls all of the metabolic and developmental processes in an organism. We present a new method which uses a comparative genomics approach which is made possible now that we have access to the complete DNA sequences of many sets of related organisms.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the advent of whole genome sequencing, we now have an abundance of whole genomes which have been sequenced, and we have entered an era when algorithms can address problems at the whole genome level. In the past, sequencing efforts often focused on a single gene, and efforts would be made to sequence that gene (and perhaps the surrounding sequence) and to understand the function and evolution of that particular gene. With whole genome sequencing, we have access to the complete genome sequence of an organism or an organelle. With the new data, genome analysis allows us to identify genes, discover their function, understand how they are regulated, and to reconstruct evolutionary histories on the scale of the whole genome. This has such far-reaching implications as understanding causes of diseases and developing drugs for them, or understanding large-scale mechanisms of evolution which can give us insight into the origin of HIV [47] or predict the next strain of influenza [20]. Currently, genome sequencing technology far exceeds the capabilities of genome analysis algorithms and new approaches must be developed. In this research, we have addressed new computational problems that have arisen out of the availability and abundance of whole genome data.

In the first several chapters, we address the problem of reconstructing phy-

logenies from gene order data. Given a whole genome sequence, we can infer the ordering of the genes along with the directionality of the genes, thus representing the genome by an ordering of signed genes. We can then use the gene orders of the genomes as input for reconstructing an evolutionary tree which has the minimum number of evolutionary events. Recreating the evolutionary history of a set of organisms based on their gene order can give insight into mechanisms of large-scale evolutionary events. Here we present a new method for gene order phylogeny reconstruction, as well as improvements on an existing method, and evaluate the results on both real and simulated datasets. Because this data is so newly available, this work represents some of the first significant inroads into the gene order phylogeny problem.

Chapters 5 and 6 focus on problems in genome annotation. Once a genome has been sequenced, the next step in the analysis of the genome is annotation. In genome annotation, all of the genes of a genome are located and identified in preparation for publication of the complete genome sequence. When a large number of previously annotated genomes exist for a set of closely related organisms, those previously-annotated genomes can be used to annotate the newly sequenced genomes. This is the case for a set of plant chloroplast and animal mitochondria organellar genomes. In Chapter 6, we present the organellar genome annotation package (`DOGMA`). We developed this package to allow biologists to annotate animal mitochondrial and chloroplast genomes using a web-based graphical user interface and custom databases. It allows researchers to view their own sequence as well as those for closely related organisms in order to locate and identify the genes and prepare the genome for direct submission to GenBank, the National Institutes of Health genomic sequence database.

A difficult problem that arises in the annotation of organellar genomes (specifically animal mitochondrial genomes) is the identification of transfer RNA genes.

Animal mitochondrial tRNAs are notoriously difficult to identify because their primary (DNA nucleotide) sequence is not conserved from one organism to the next, but they remain conserved in the basepairing of the secondary structure, and they tend to evolve through compensatory mutations in the secondary structure. Algorithms are therefore needed to identify these tRNAs based on secondary structure conservation. Another difficulty is that although most tRNAs have a cloverleaf-like secondary structure, animal mitochondrial tRNAs often have a non-canonical shape to their secondary structure. In Chapter 5, we evaluate a set of methods (including our own) for identifying tRNAs. The best-performing method has been incorporated into our annotation package, `DOGMA`.

Finally, the last problem we address is the identification of regulatory elements. Understanding gene expression is one of the most pressing unsolved problems in molecular biology today because gene expression controls all of the metabolic and developmental processes is organisms. We use a comparative approach called Phylogenetic Footprinting which is made possible now that we have access to the complete DNA sequences of so many organisms. This method explicitly incorporates an accepted phylogeny and models of DNA sequence evolution into the method. Our approach uses maximum likelihood to evaluate sets of regulatory elements with respect to both functional and non-functional models of evolution.

Chapter 2

Whole Genome Evolution: Reconstructing Phylogenies from Gene Order Data

2.1 Introduction

The genomes of some organisms have a single chromosome or contain organelles with a single chromosome (such as mitochondria or chloroplasts) whose evolution is largely independent of the evolution of the nuclear genome for these organisms. Many single-chromosome organisms and organelles have circular chromosomes. Given a particular strand from a single chromosome, whether linear or circular, we can infer the ordering of the genes, along with directionality of the genes, thus representing each chromosome by an ordering (linear or circular) of signed genes. Note that picking the complementary strand produces a different ordering, in which the genes appear in the reverse direction and reverse order. The evolutionary process that operates on the chromosome can thus be seen as a transformation of signed orderings of genes.

The first heuristic for reconstructing phylogenetic trees from gene order data was introduced by Blanchette *et al.* in [10] and was implemented in the program `BPAnalysis`. It sought to reconstruct the *breakpoint phylogeny* and was applied to a variety of datasets [11, 98].

A different technique for reconstructing phylogenies from gene order data was introduced by Cosner in [27]. We have modified her technique so that it requires less biological input. Our approach can also be described as a heuristic for the breakpoint phylogeny, although it is quite different in its technique from `BPAnalysis`. We call our approach *Maximum Parsimony on Binary Encodings (MPBE)*. The MPBE method first encodes a set of genomes as binary sequences and then constructs maximum-parsimony trees for these sequences.

We describe MPBE and compare it with two other methods (`BPAnalysis`, the heuristic designed and implemented by Blanchette *et al.* [10], and the polynomial-time, distance-based method neighbor joining [95]) on both real and synthetic data. We find that, when the rates of evolution are sufficiently low, all methods recover very good estimates of the evolutionary tree (although `BPAnalysis` is much slower than MPBE). However, when the rates of evolution are high, all methods recover poor estimates of the evolutionary tree.

2.2 Definitions

We assume a fixed set of genes $\{g_1, g_2, \ldots, g_n\}$. Each genome is then an ordering (circular or linear) of some multi-subset of these genes, each gene given with an orientation that is either positive (g_i) or negative ($-g_i$). The multi-subset formulation allows for deletions or duplications of a gene. A linear genome is then simply a permutation on this multi-subset, while a circular genome can be represented in the same way under the implicit assumption that the permutation closes back on itself. For example, the circular genome on gene set $\mathcal{G} = \{g_1, g_2, \ldots, g_6\}$ given by

$g_1, g_2, -g_3, g_4, g_6, g_2$ has one duplication of the gene g_2, has a deletion of the gene g_5, and has a reversal of the gene g_3. That same circular genome could be represented by several different linear orderings, each given by rotating the linear ordering above. Furthermore, the ordering $g_1, g_2, \ldots, g_n$, whether linear or circular, is considered equivalent to that obtained by considering the complementary strand, i.e., to the ordering $-g_n, -g_{n-1}, \ldots, -g_1$.

In tracing the evolutionary history of a collection of genomes with a single chromosome, we use inversions, transpositions and inverted transpositions because these events only rearrange gene orders. A more complex set of structural changes has been considered, e.g., in [27].

Let G be the genome with signed ordering (linear or circular) $g_1, g_2, \ldots, g_n$. An *inversion* between indices i and j, for $i < j$, produces the genome with linear ordering:

$$g_1, g_2, \ldots, g_{i-1}, -g_j, -g_{j-1}, \ldots, -g_i, g_{j+1}, \ldots, g_n.$$

If we have $j < i$, we can still apply an inversion to a circular (but not linear) genome by simply rotating the circular ordering until the two indices are in the proper relationship—recall that we consider all rotations of the complete circular ordering of a circular genome as equivalent.

A *transposition* on the (linear or circular) ordering G acts on three indices, i, j, k, with $i < j$ and $k \notin [i, j]$, and operates by picking up the interval $g_i, g_{i+1}, \ldots, g_j$ and inserting it immediately after g_k. Thus the genome G above (with the additional assumption of $k > j$) is replaced by:

$$g_1, \ldots, g_{i-1}, g_{j+1}, \ldots, g_k, g_i, g_{i+1}, \ldots, g_j, g_{k+1}, \ldots, g_n.$$

Once again, if we have $j > i$, we can still apply the transposition to a circular (but not linear) genome by first rotating it to establish the desired index relationship.

An *edit sequence* describes how one genome evolves into another through

a sequence of these evolutionary events. For example, let G be a genome and let $p_1, p_2, \ldots, p_k$ be a sequence of evolutionary events operating on G. Then G' is defined by $p_1, p_2, \ldots, p_k(G)$. When each operation is associated with a cost, then the *minimum edit distance* between two genomes G and G' is defined to be the minimum cost of any edit sequence transforming G into G'. As long as the cost of each operation is finite, any two genomes have finite edit distance.

The *inversion distance* between two genomes is the minimum number of inversions needed to transform one genome into another. The inversion distance between two signed genomes is computable in polynomial time [44, 56]; this algorithm is available in software (`signed_dist`), which we modified for use in our experiments to compute only distances (and thus run very much faster). We let I refer to the inversion distance. The *transposition distance* between two genomes is the minimum number of transpositions needed to transform one genome into the other. Computing the transposition distance is of unknown computational complexity.

When inversions, transpositions, and inverted transpositions are allowed, nothing is known about the computational complexity or approximability of computing edit distances; however, heuristics have been developed that can estimate the minimum edit distance for weighted sums of inversions, transpositions, and inverted transpositions. One such heuristic, `derange2` [9], is available in software; both we and Sankoff *et al.* [100] have used it for tree estimation purposes. We will refer to the distances computed by `derange2` as ITT distances.

Another distance between genomes that is not directly an evolutionary metric is the *breakpoint distance*. Given two genomes G and G' on the same set of genes, a breakpoint in G is defined as an ordered pair of genes (g_i, g_j) such that g_i and g_j appear consecutively in that order in G, but neither (g_i, g_j) nor $(-g_j, -g_i)$ appear consecutively in that order in G'. For instance, if $G = g_1, g_2, -g_4, -g_3$ and $G' = g_1, g_2, g_3, g_4$, then there are exactly two breakpoints in G: $(g_2, -g_4)$, and $(-g_3, g_1)$;

the pair $(-g_4, -g_3)$ is not a breakpoint in G' since (g_3, g_4) appear consecutively and in that order in G'. The *breakpoint distance* is the number of breakpoints in G relative to G' (or vice-versa, since the measure is symmetric).

An *evolutionary tree* (or *phylogeny*) for a set S of genomes is a binary tree with $|S|$ leaves, each leaf labeled by a distinct element of S. A putative evolutionary tree is "correct" as long as this leaf-labeled topology is identical to the true evolutionary tree. The true phylogeny is unknown for real data. Synthetic data can be created with simulations using a given model tree; for such data, the "true" tree is the model tree and is thus known. Studies using such synthetic data are standard in the phylogenetics literature because they enable one to test the reliability of different methods. The process of tree reconstruction generally involves the inference of additional aspects of the tree. For example, a given method may infer weights on the edges (also called "branch lengths"), genomes at internal nodes (i.e., "ancestral" genomes), or probabilities for each type of evolutionary event on each edge of the tree. These topologies and additional parameters are estimated in order to optimize some objective criterion; the three basic optimization criteria in use by biologists are *Maximum Parsimony*, *Distance-Based Methods*, and *Maximum Likelihood*. We briefly review the first two criteria. We now describe two approaches currently in favor for genome phylogeny reconstruction.

Maximum parsimony: Assume that we are given a tree in which each node is labeled by a genome. We define the cost of the tree to be the sum of the costs of its edges, where the cost of an edge is one of the edit distances between the two genomes that label the endpoints of the edge. Finding the tree of minimum cost for a given set of genomes and a given definition of the edit distance is the problem of *Maximum Parsimony for Rearranged Genomes (MPRG)*; the optimal trees are called the maximum-parsimony trees. (The MPRG problem is related to the more usual maximum-parsimony problem for biomolecular sequences, defined later.)

Distance-based methods: Distance-based methods for tree reconstruction operate by first computing all pairwise distances between the taxa in the dataset, thus computing a representation of the input data as a distance matrix d. In the context of genome evolution, this calculation of distances is done by computing breakpoint (BP) distances, or minimum edit (e.g. I or ITT) distances. Given the distance matrix d, the method computes an edge-weighted tree whose leaf-to-leaf distances closely fit the distance matrix. The most frequently used distance-based methods are polynomial-time methods such as neighbor joining [95]. These methods do not explicitly seek to optimize any criterion, but can have good performance in empirical studies. In particular, neighbor joining has shown excellent performance in studies based upon simulating biomolecular sequence evolution and is probably the most popular distance-based method.

2.3 Previous Phylogenetic Methods

2.3.1 Distance-Based Methods

There has been little use of distance-based methods for reconstructing phylogenies from gene order data. Blanchette *et al.* [11] recently evaluated two of the most popular polynomial-time distance-based methods for phylogenetic reconstruction, neighbor joining and Fitch-Margoliash [38], for the problem of reconstructing the phylogeny of metazoans. They calculated a breakpoint distance matrix for inferring the metazoan phylogeny from mitochondrial gene order data. They found the trees obtained by these methods unacceptable because they violated assumptions about metazoan evolutionary history. Later, they examined a different dataset and found the result to be acceptable with respect to evolutionary assumptions about that dataset [99].

2.3.2 Computational Complexity of MPRG

MPRG seems to be the optimization criterion of choice; indeed, most approaches to reconstructing phylogenetic trees from gene order data have explicitly sought to find the maximum-parsimony tree with respect to some definition of genomic distances (inversion distances or a weighted sum of inversions, transpositions, and inverted transpositions). However, all these problems are NP-hard or of unknown computational complexity. Even the fundamental problem of computing optimal labels (genomes) for the internal nodes is very difficult: when only inversions are allowed, it is NP-hard, even for the case where there are only three leaves [22].

2.3.3 Breakpoint Phylogeny

Blanchette *et al.* [11] recently proposed a new optimization problem for phylogeny reconstruction on gene order data. In this problem, the tree sought is that with the minimum number of breakpoints rather than that with the minimum number of evolutionary events. It has long been known that the breakpoint distance is at most twice the inversion distance for any two genomes [44]. For some datasets, however, there can be a close-to-linear relationship between the breakpoint distance and either the inversion distance or the weighted sum of inversions and transpositions. When a linear relationship exists, the tree with the minimum number of breakpoints is also the tree with the minimum number of evolutionary events. Consequently, when a close-to-linear relationship exists, the tree with the minimum number of breakpoints may be close to optimal with respect to the number of evolutionary events. Blanchette *et al.* [11] observed such a close-to-linear relationship in a group of metazoan genomes (we computed the correlation coefficient between the two measures for their set and obtained a very high value of 0.9815) and went on to develop a heuristic for finding the breakpoint phylogeny.

Computing the breakpoint phylogeny is NP-hard for the case of just three

linear signed genomes [89], a special case known as the *Median Problem for Breakpoints (MPB)*. Blanchette *et al.* showed that the MPB reduces to the Traveling Salesman Problem (TSP) [45] and designed special heuristics for the resulting instances of TSP. Their overall heuristic for the breakpoint phylogeny considers each tree topology in turn. For each tree, it fills in internal nodes by computing medians of triplets of genomes iteratively (until no change occurs) using the TSP reduction, then scores the resulting tree. The best tree is returned at the end of the procedure. This heuristic is computationally intensive on several levels. First, the number of unrooted binary trees on n leaves is exponential in n (specifically it is $(2n-5)\cdot(2n-7)\cdot\ldots\cdot 3$), so that the outer loop is exponential in the number of genomes. Secondly, the inner loop itself is computationally intensive, since computing the median of three genomes is NP-hard [89] and because the technique used by Blanchette *et al.* involves solving many instances of TSP in a reduction where the number of cities equals the number of genes in the input. Finally, the number of instances of TSP can be quite large, since the procedure iterates until no further change of labeling occurs within the tree. Thus the computational complexity of the entire algorithm is exponential in *each* of the number of genomes and the number of genes.

The accuracy of `BPAnalysis` for the breakpoint phylogeny problem depends upon the accuracy of its component heuristics. While it evaluates every tree, the labeling given to each tree is only locally optimal: although it solves TSP exactly at each node, it labels nodes with an iterative method that can easily be trapped at a local optimum. In our experiments, we have found that `BPAnalysis` often needed to be run on several different random starting points in order to score a given tree accurately. This is typical of hill-climbing heuristics, but will affect the running time proportionally.

2.4 Our New Method: Maximum Parsimony on Binary Encodings of Genomes (MPBE)

In this section, we describe a new approach to reconstructing phylogenies from gene order data. This new method is derived from an earlier method developed by Cosner in [27]. Like Cosner's technique, our method encodes the genome data as binary sequences and seeks a maximum-parsimony tree for these sequences, although our encoding is very simple and uses no biological assumptions. However, our method has a second phase, in which we select, from the maximum-parsimony trees we find, the trees that have minimum length with respect to some evolutionary metric (such as the inversion distance or the ITT distance). We now describe the two phases of the MPBE approach.

2.4.1 Phase I: Solving Maximum Parsimony on Binary Encodings of Genomes

We begin by defining the binary encoding. We note all ordered pairs of signed genes (g_i, g_j) that appear consecutively in at least one of the genomes. Each such pair defines a position in the sequences (the choice of index is arbitrary). If (g_i, g_j) *or* $(-g_j, -g_i)$ appear consecutively in a genome, then that genome has a 1 in the position for this ordered pair, and otherwise it has a 0. These "characters" can also be weighted. (In this study, we did not weight any characters; however, in the study reported in [27], character weighting was used, along with other characters such as gene segment insertions and deletions, duplications of inverted repeats, etc. Thus, the method can be extended to allow for evolutionary events more complex than gene order changes.)

Let $H(e)$ be the Hamming distance between the sequences labeling the endpoints of the edge e—the Hamming distance between two sequences is the number

of positions in which they differ. We define the *Binary Sequence Maximum Parsimony (BSMP)* problem as follows: the input consists of a set S of binary sequences, each of length L; the output is a tree T with leaves labeled by S and internal nodes labeled by additional binary sequences of length L in such a way as to minimize $\Sigma H(e)$ as e ranges over the edges of the tree. The trees with the minimum score are called maximum-parsimony trees.

Our first phase then operates as follows. First, each genome is replaced by a binary sequence. The BSMP problem is then solved exactly or approximately, depending upon the dataset size. BSMP is NP-hard [39], but fast heuristics exist that are widely available in standard phylogeny software packages, such as `PAUP*` [107]. Although no study has been published on the accuracy of these heuristics on large datasets, it is generally believed that these heuristics usually work well on datasets of size up to about 40 genomes. Moreover, exact solutions on datasets of up to about 20 genomes can be obtained through branch-and-bound techniques in reasonable amounts of time; consequently, BSMP has been solved exactly in some cases.

2.4.2 Phase II: Screening The Maximum Parsimony Trees

Once the maximum-parsimony trees are obtained, the internal nodes are then relabeled by circular signed gene orders (recall that the labeling of internal nodes obtained in the first phase of MPBE is with binary sequences, not with circular signed genomes). The relabeling is obtained by giving the maximum-parsimony tree as a constraint to `BPAnalysis`, thus producing a labeling of each internal node with circular signed gene orders which (hopefully) minimizes the breakpoint distance of the tree. The labeling also allows us to score each tree for the I or ITT distance. The tree that minimizes the total cost is then returned.

2.4.3 Running Time of MPBE

The computational complexity of MPBE, while less than that of `BPAnalysis`, remains high. The maximum-parsimony evaluation of a single tree in the search space takes polynomial time (the precise time is $\Theta(nk)$, where n is the number of genomes and k is the number of genes in each genome). Thus, the first phase is exponential in the number of genomes, but polynomial in the number of gene segments. However, we have the option of doing hill-climbing through tree space (rather than exhaustive search) and thus can reduce the computational effort by comparison to the exhaustive search strategy of `BPAnalysis`. In the second phase, we give the maximum-parsimony trees to `BPAnalysis` as constraint trees. Thus we also call `BPAnalysis`, (which is exponential in both the number of genomes and the number of gene segments), but only on a (typically small) subset of the possible trees. Finally, we compute the cost of each node-labeled tree with respect to I or ITT distances. Computing ITT distances is fast, although `derange2` can be inexact. Computing inversion distances with the original `signed_dist` is fairly slow because the program also returns inversions, but fast when it is modified to compute only distances. Overall, Phase II is more computationally expensive than Phase I.

2.4.4 MPBE as a Heuristic for the Breakpoint Phylogeny

Suppose T is the breakpoint phylogeny for the set $G_1, G_2, \ldots, G_n$ of genomes. Each node in T is labeled by a circular ordering of signed genes and the number of breakpoints in the tree is minimized. If each node in the tree is replaced by its binary encoding, the parsimony length of the tree is exactly twice the number of breakpoints in the tree. Thus, seeking a tree with the minimum number of breakpoints is exactly the same as seeking a tree (based upon binary encodings) with the minimum parsimony length, *provided that* each binary sequence can be realized by a circular ordering of signed genes.

This last point is significant, because not all binary sequences are derivable from signed circular orderings on genomes. In other words, it is possible for the MPBE tree (that is, the tree with minimal parsimony length for the binary sequence encodings of the genomes) to have internal nodes whose binary sequence encodings cannot be realized by circular orderings of signed genes. If the sequences in the internal nodes of an MPBE correspond to signed circular orderings, then the tree will be a breakpoint phylogeny. If they do not, then the MPBE trees and the breakpoint phylogenies may be disjoint.

Consider rephrasing the breakpoint phylogeny problem as follows. We say that a binary sequence is a "circular genome sequence" if it is the binary encoding of a circular genome under a given representation method. The breakpoint phylogeny problem is to find the tree of minimum parsimony length, with leaves labeled by the binary encodings of the circular genomes and internal nodes labeled by "circular genome sequences." Since MPBE does not restrict the labels of internal nodes to circular genome sequences, it searches through a larger space for the labels of internal nodes and thus may assign labels to nodes that are not circular genome sequences. When this happens, MPBE will fail to find feasible solutions to the breakpoint phylogeny problem.

MPBE is thus a heuristic for breakpoint phylogeny, but it produces labelings of the internal nodes that are binary sequences; as we discussed, these may not correspond to circular orderings of signed gene segments. Therefore we must relabel the internal nodes by circular genome sequences (using `BPAnalysis` or other such techniques) so that the breakpoint distance of the trees can be computed. This is why we have included Phase II in our method.

Since each of the problems we solve (maximum parsimony on binary sequences, the median problem for breakpoints, and the ITT) is either known or conjectured to be NP-hard, the accuracy of the heuristics will determine whether

we find globally optimal or only locally optimal solutions.

2.5 Chloroplast Data Analysis

Chloroplast DNA is generally highly conserved in nucleotide sequence, gene order and content, and genome size [85]. The genomes contain approximately 120 genes involved in photosynthesis, transcription, translation, and replication. Major changes in gene order, such as inversions, gene or intron (introns are pieces of non-coding sequence within genes which get spliced out) losses, and loss of one copy of the inverted repeat, are rare. These genes are very useful as phylogenetic markers because they are easily polarized and exhibit very little homoplasy when properly characterized [33]. In groups in which more than one gene order change has been detected, the order of events is usually readily determined (e.g., [49, 58]). Chloroplast DNA gene order changes have been useful in phylogenetic reconstruction in many plant groups (see [33]). These changes have considerable potential to resolve phylogenetic relationships and provide valuable insights into the mechanisms of cpDNA evolution.

2.5.1 The Campanulaceae cpDNA Dataset

We have used the chloroplast genomes of the flowering plant family Campanulaceae for a test case of our technique. In earlier work [27], Cosner obtained detailed restriction site and gene maps for 18 genera of the Campanulaceae and the outgroup *Nicotiana*. (An "outgroup" is a taxon selected so that any two other members of the set are more closely related than either is to the outgroup; the use of outgroups in phylogenetic analysis allows us to root the tree). She then used a variant of the MPBE analysis described above to obtain a phylogenetic analysis of these genera. We analyzed the same dataset, but, in order to apply the MPBE method, had to remove two incompletely mapped genera from the dataset. We also removed the repeated regions, causing certain pairs of genera (which differ only in terms

of insertions and deletions of gene segments or expansions and contractions of the inverted repeat) to become indistinguishable, reducing our dataset to 13 genera from the original 19.

2.5.2 Data Analysis

We used gene maps to encode each of the 13 genera as a circular ordering of signed gene segments. The result is shown in Figure 2.1.

We used these 13 circular orderings as input to `BPAnalysis`. The program spent over 43 hours of computation time without completing. We also encoded these orderings with our binary encoding technique and conducted an analysis of the resulting binary sequences under maximum parsimony using the branch-and-bound procedure of `PAUP*`. We obtained four maximum-parsimony trees from this dataset. We inferred circular orderings of signed gene segments for each internal node by giving each binary tree as a constraint tree to `BPAnalysis`. This produces a tree in which each node (internal and leaf) is represented by circular signed orderings on genes, potentially minimizing the number of breakpoints in the tree. (An actual minimization is not guaranteed, because `BPAnalysis` uses hill-climbing on each fixed-tree and thus may find only a local minimum.) We then scored each tree for the number of breakpoints. Interestingly, the labeling of internal nodes obtained by `BPAnalysis` produced the same number of breakpoints on all four trees, namely 89.

We note that the best breakpoint score obtained in 43 hours of computation by `BPAnalysis` from the original orderings was 96—much larger than the breakpoint score obtained by our parsimony analysis of binary sequences.

We then scored each tree (using the labels assigned by `BPAnalysis`) for the I distance using our modified `signed_dist` and for the ITT distance using `derange2` with relative weights of 2.1 for transpositions and inverted transpositions versus 1 for inversions. Using this weighting scheme, the first tree has a total

Trachelium
(1–15)(76–56)(53–49)(37–40)(35–26)(44–41)(45–48)(−36)(25–16)(90–84)
(77–83)(91–96)(55–54)(105–97)
Campanula
(1–15)(76–49)(39–37)(40)(35–26)(44–41)(45–48)(−36)(25–16)(90–84)(77–83)
(91–96)(105–97)
Adenophora
(1–15)(76–49)(39–37)(28–35)(40)(26–27)(44–41)(45–48)(−36)(25–16)(90–84)
(77–83)(91–96)(105–97)
Symphyandra
(1–15)(76–56)(39–37)(49–53)(40)(35–26)(44–41)(45–48)(−36)(25–16)(90–84)
(77–83)(91–96)(55–54)(105–97)
Legousia
(1–15)(76–56)(27–26)(44–41)(45–48)(36–35)(25–16)(90–84)(77–83)
(91–96)(55–53)(105–98)(28–34)(40–37)(49–52)(−97)
Asyneuma
(1–15)(76–61)(56–53)(60–57)(27–26)(44–41)(45–48)(36–35)(25–16)
(89–84)(77–83)(90–96)(105–98)(28–34)(40–37)(49–52)(−97)
Triodanis
(1–15)(76–56)(27–26)(44–41)(45–48)(36–35)(25–16)(89–84)(77–83)
(90–96)(55–53)(105–98)(28–34)(40–37)(49–52)(−97)
Wahlenbergia
(1–11)(60–49)(37–40)(35–28)(12–15)(76–61)(27–26)(44–41)(45–48)
(−36)(25–16)(90–84)(77–83)(91–96)(105–97)
Merciera
(1–10)(49–53)(28–35)(40–37)(60–56)(11–15)(76–61)(27–26)(44–41)
(45–48)(−36)(54)(25–16)(90–85)(77–84)(91–96)(−55)(105–97)
Codonopsis
(1–8)(36–18)(15–9)(40)(56–60)(37–39)(44–41)(45–53)(16–17)(54–55)
(61–76)(96–77)(105–97)
Cyananthus
(1–8)(29)(36–26)(40)(56–60)(37–39)(25–9)(44–41)(45–48)(55–49)(61–96)
(105–97)
Platycodon
(1)(8)(2–5)(29–36)(56–50)(28–26)(9)(49–45)(41–44)(37–40)(16–25)
(10–15)(57–59)(6–7)(60–96)(105–97)
Nicotiana
(1–105)

Figure 2.1: 12 genera of Campanulaceae and the outgroup *Nicotiana*, as circular orderings of signed gene segments. We represent each circular ordering as a linear ordering, beginning at gene segment 1.

of 40 inversions and 12 transpositions/inverted transpositions; the second has 48 inversions and 18 transpositions/inverted transpositions; the third has 40 inversions and 12 transpositions/inverted transpositions; and the fourth has 67 inversions and 43 transpositions/inverted transpositions. Thus, the first and third trees are superior (under this analysis) to the second and fourth. We then evaluated the first and third trees with respect to the inversion distance, given the labeling on internal nodes obtained by `BPAnalysis`: the first tree has a total number of 68 inversions, while the third has 67. Both trees have zero-length edges (i.e., the endpoints of some edges have the same gene orderings). When these edges are contracted, the two trees are identical. The contracted tree is shown in Figure 2.2. Interestingly, that tree is also a contraction of each of the trees obtained by the Cosner analysis [27] on the original 19 genera, and then restricted to the subset of 13 genera. Thus our restricted subset of characters is compatible with the more biologically rich analysis performed by Cosner, in which insertions, deletions, duplications, contractions/expansions of the inverted repeat, etc., were also used.

We computed neighbor-joining trees (using `Phylip` [36]) on three different distance matrices: the I matrix computed using our modified `signed_dist`, the ITT matrix computed with `derange2` with relative weights of 1, 2.1, and 2.1, and the breakpoint matrix computed using `BPAnalysis`. We show the `derange2` distance matrix in Table 2.1.

The three neighbor-joining trees have identical topologies, differing only in their edge weights, while the MPBE trees differ from the NJ trees by at most 2 edges; see Table 2.2. The similarity between all reconstructed trees indicates a high level of confidence in the the accuracy of the common features of the phylogenetic reconstructions.

The conditions under which these genomes evolved (low rates of evolution and a large number of gene segments) are probably responsible for this high level of

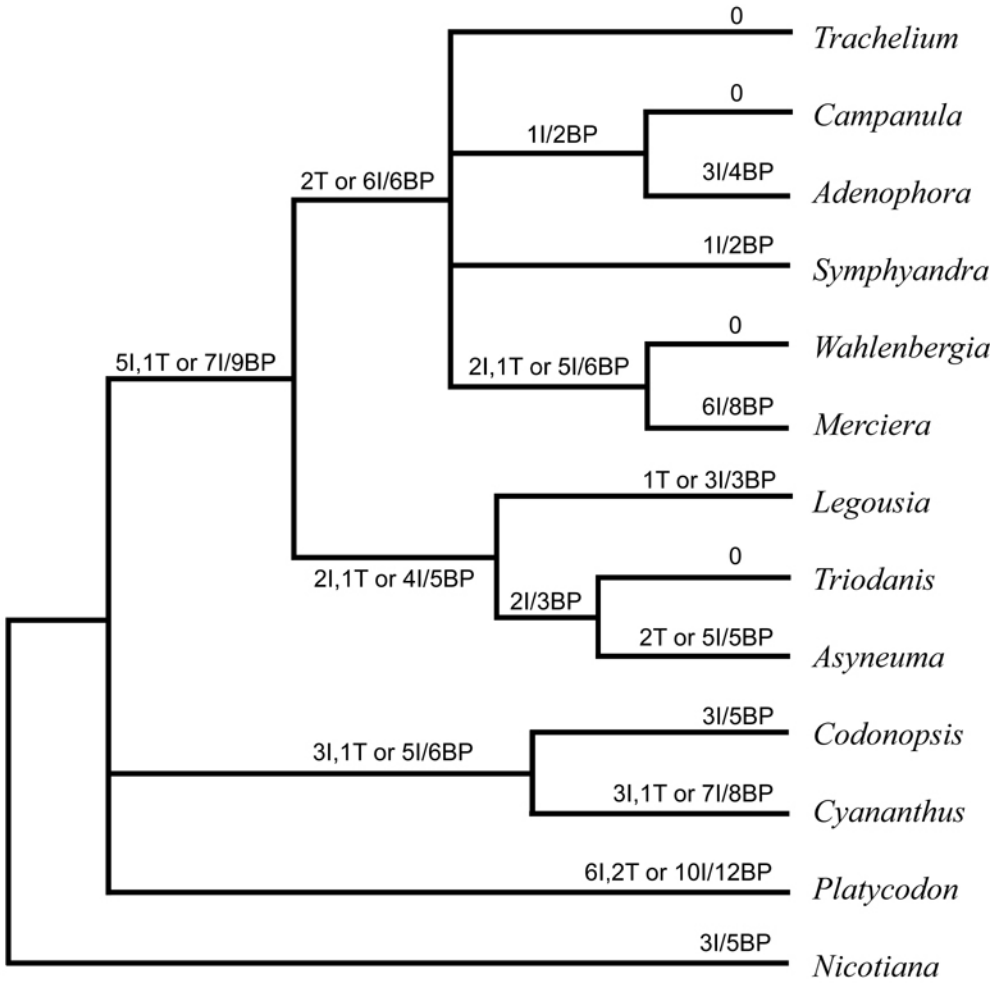


Figure 2.2: The reconstructed phylogeny of 12 genera of Campanulaceae and the outgroup *Nicotiana* based upon an MPBE analysis of 185 binary characters. Above each edge are given the number of inversions and transpositions/inverted transpositions, the number of inversions in an inversion-only scenario, and the number of breakpoints.

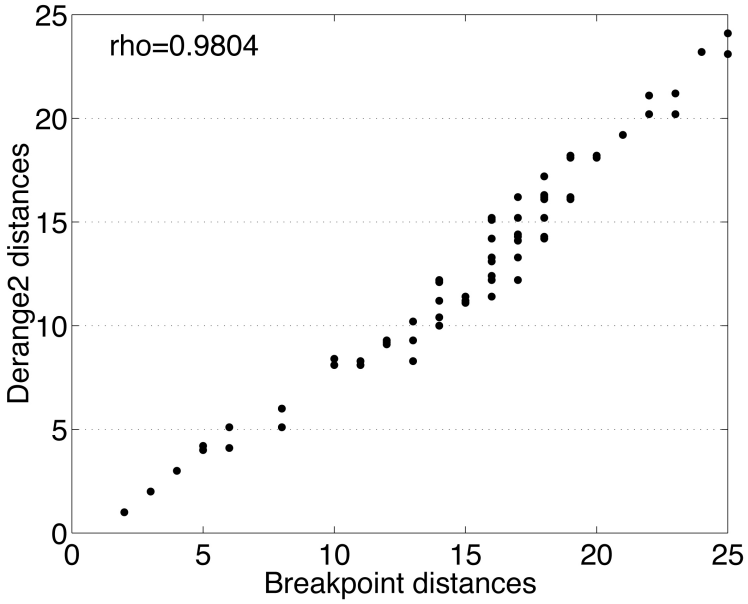


Figure 2.3: Comparison of distance calculations on the Campanulaceae dataset.

similarity, which is observable at various levels. For instance, the breakpoint distance and the ITT distance (using relative costs of 1, 2.1, and 2.1) are very closely related, as illustrated in Figure 2.3. (The high correlation coefficient ρ indicates that the two distances stand in a nearly linear relationship to each other.) These observations suggest that this dataset forms an easy case for phylogeny reconstruction. We therefore began an experimental investigation into the performance of methods for phylogenetic reconstruction from gene order data to determine whether all methods continue to perform well under a larger range of model conditions and whether there are model conditions under which some methods consistently outperform others.

	Tra	Cam	Ade	Sym	Leg	Asy	Tri	Wah	Mer	Cod	Cya	Pla	Tob
Tra	0.0	1.0	4.0	1.0	8.3	10.4	8.3	4.1	8.1	15.2	14.1	19.2	10.0
Cam	1.0	0.0	3.0	2.0	9.3	11.4	9.3	5.1	9.2	15.1	15.2	20.2	11.2
Ade	4.0	3.0	0.0	5.1	12.1	14.3	12.1	8.1	11.2	16.2	15.2	20.2	13.1
Sym	1.0	2.0	5.1	0.0	9.2	11.4	9.3	5.1	9.1	14.2	13.3	20.2	11.1
Leg	8.3	9.3	12.1	9.2	0.0	8.4	4.1	12.2	14.3	18.1	16.1	23.2	14.2
Asy	10.4	11.4	14.3	11.4	8.4	0.0	4.2	12.4	16.2	18.2	16.2	21.1	12.2
Tri	8.3	9.3	12.1	9.3	4.1	4.2	0.0	12.2	14.4	18.2	15.2	21.2	12.2
Wah	4.1	5.1	8.1	5.1	12.2	12.4	12.2	0.0	6.0	18.1	16.2	23.1	14.2
Mer	8.1	9.2	11.2	9.1	14.3	16.2	14.4	6.0	0.0	17.2	16.3	24.1	16.1
Cod	15.2	15.1	16.2	14.2	18.1	18.2	18.2	18.1	17.2	0.0	8.3	18.2	10.2
Cya	14.1	15.2	15.2	13.3	16.1	16.2	15.2	16.2	16.3	8.3	0.0	16.3	10.2
Pla	19.2	20.2	20.2	20.2	23.2	21.1	21.2	23.1	24.1	18.2	16.3	0.0	13.3
Tob	10.0	11.2	13.1	11.1	14.2	12.2	12.2	14.2	16.1	10.2	10.2	13.3	0.0

Table 2.1: The ITT distance matrix for the Campanulaceae dataset, computed using `derange2` and a 2.1 weight ratio.

2.6 Our Experimental Investigation

We developed a simple simulator that, given a model tree and parameters, mimics the evolutionary history of a genome and produces a set of genomes. Using both actual and synthetic model trees, we then reconstruct the putative phylogeny using the various methods proposed to date as well as our new method (only through Phase I). These putative phylogenies are then compared to the model tree.

We computed breakpoint distances (BP) with our own code, I distances using our modified `signed_dist`, and ITT distances using `derange2`. Since we generate the synthetic data ourselves, we can observe the actual process that happens during the simulation. In particular, we can note when no evolutionary event (inversion, transposition, or inverted transposition) takes place on an edge, enabling us to derive a better estimate of the quality of a reconstruction, since no reconstruction method can recover an edge (other than by guessing) when no evolutionary event happens on it.

	NJ	MPBE1	MPBE2	MPBE3	MPBE4
NJ	0	1	2	1	2
MPBE1	1	0	1	1	2
MPBE2	2	1	0	2	1
MPBE3	1	1	2	0	1
MPBE4	2	2	1	1	0

Table 2.2: The number of missing edges for various reconstruction methods (with respect to the other methods) on the Campanulaceae data of Figure 2.1. MPBE1 through MPBE4 are the four most parsimonious trees by the first phase of the MPBE method. NJ refers to the tree obtained by neighbor joining on the three distance matrices (these three trees were identical).

2.6.1 Terminology

Let T be a tree leaf-labeled by the set S. Given an edge e in T, the deletion of the edge from T produces a bipartition π_e of G into two sets. The set $C(T) = \{\pi_e : e \in E(T)\}$ uniquely defines the tree T; this characterization is called the *character encoding* of T. Given a collection of trees $T_1, T_2, \ldots, T_k$, each leaf-labeled by S, we define the *strict consensus* of the trees to be that unique tree T_{sc} defined by $C(T_{sc}) = C(T_1) \cap C(T_1) \cap ... \cap C(T_n)$. This is the maximally resolved tree which is a common contraction of each tree T_i. Character encodings are used to compare trees and to evaluate the performance of a phylogenetic reconstruction method. Let T be the "true" tree and let T' be the estimate of T. Then the *false negatives* of T' with respect to T are those edges e that obey $\pi_e \in C(T) - C(T')$, i.e., edges in the true tree that the method fails to infer. The *false positives* of T' with respect to T are those edges e that obey $\pi_e \in C(T') - C(T)$, i.e., edges in the inferred tree that do not exist in the true tree and should not have been inferred. Note that every trivial bipartition (induced by the edge incident to a leaf) exists in every tree. Consequently, false positives and false negatives are calculated only with respect to the internal edges of the tree. These are sometimes expressed as a percentage of the number of internal edges.

2.6.2 Experimental Setup

The simulator: The *Nadeau-Taylor* model [81] is the standard model of genome evolution; it assumes that only inversions occur during the evolutionary history of a set of genomes, that all inversions are equally likely, and that the number of inversions on each edge obeys a Poisson distribution. We designed a simulator to enable us to generate gene orders under the Nadeau-Taylor model, as well as under more complex models in which transpositions and inverted transpositions also occur. The input to the simulator is the topology of a rooted tree T (which determines the number of genomes), the number k of genes in the genomes, the expected number λ_e of inversions on each edge e, and a constant C denoting the relative cost of inversions to transpositions and inverted transpositions. The expected number of events per edge determines the rate of evolution for the dataset. More commonly (i.e. in nucleotide data), the rate of evolution is defined as the expected number of mutations over a branch per unit time, however, for gene order data, the rate of evolution is the number of evolutionary events per branch. In the simulator, the number of each of these events is a random variable obeying a Poisson distribution. Thus, we generate a random leaf-labeled tree, randomly assign lengths (chosen uniformly from various ranges) to each edge to represent the expected number of inversions per edge, and feed the result to the simulator.

The simulator generates signed circular orderings of the genes as follows. The root is assigned the identity gene ordering $g_1, g_2, \ldots, g_k$. When traversing an edge e with expected number of inversions λ_e, three random numbers are generated. The first determines the actual number of inversions on that edge; the second the actual number of transpositions; and the third the actual number of inverted transpositions. Once the number of each event is determined, the order of these events is randomly selected. This process produces a set of circular signed gene orders for each genome at the leaves of the model tree. The simulator also produces other information for

use in performance studies: the gene orders computed at each internal and leaf node, the actual number of inversions, transpositions, and inverted transpositions that occurred during that run of the simulator on each edge, and the "true distance matrix" D between every pair of leaves in the tree. (Given the actual number of inversions, transpositions, and inverted transpositions that occur on each leaf-to-leaf path, the distance between the two leaves is the number of inversions plus the weighted cost of the transpositions and inverted transpositions.) Note that this matrix defines the model tree, with each edge weighted according to the weighted cost of the events on that edge. As long as every edge has at least one event, standard distance methods (such as neighbor joining [95]), when applied to the matrix D are *guaranteed* to recover the true tree topology (see [112]).

Phylogenetic methods: For each dataset generated by the simulator, we computed the BP distance and at least one of the I or ITT distances. We computed neighbor-joining trees (as implemented in `Phylip` [36]) on these distance matrices. We denote the neighbor-joining trees for the different distance matrices by NJ(BP), NJ(ITT), and NJ(I). The MPBE heuristic is only computed through Phase I, so that we return the strict consensus of all maximum-parsimony trees we compute and do not perform any additional screening.

We wrote software to obtain binary sequence representations of the signed circular gene orderings. We solved maximum parsimony exactly on datasets of up to 20 taxa using the branch-and-bound program of `PAUP*` and heuristically for larger datasets; naturally, when we use a heuristic to "solve" maximum parsimony, we are not guaranteed to find globally optimal solutions, only locally optimal ones. We used the TBR (tree-bisection-reconnection) branch-swapping heuristic of `PAUP*`, with 100 initial starting points (trees obtained by optimizing the sequential placement of taxa, randomly ordered, into the tree). We kept up to 10,000 trees in memory and included auto-increment in the analysis. As these searches often returned hundreds

model tree	Avg. Number (Avg. %) of False Negatives NJ(BP) *vs.* model	NJ(ITT) *vs.* model
T_A	14.84 (87.29%)	15.34 (90.24%)
T_B	8.22 (48.35%)	7.70 (45.29%)
T_C	2.24 (13.18%)	1.90 (11.18%)

Table 2.3: Average false negatives of the NJ trees from the matrices BP and ITT. Values in parentheses are the percentages over the 17 nontrivial bipartitions in each model tree.

or thousands of local optima, we computed the strict consensus and majority consensus trees of the local optima. In the following, we denote these trees by MPBE, "maximum-parsimony tree(s) for the binary encoding of the genome data."

We labeled internal nodes of each tree with circular orderings of signed genes using `BPAnalysis`, and scored the resultant node-labeled trees under breakpoint distances (ourselves), I distances (using `signed_dist`) and ITT distances (using `derange2`).

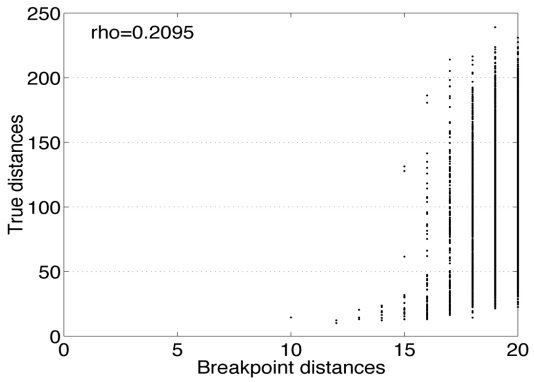
We were unable to run `BPAnalysis` to completion on our datasets because of its computational complexity; however, we did use `BPAnalysis` in a restricted search, by providing it with the strict consensus of the trees we obtained using our other techniques as a "constraint" tree. This way of using `BPAnalysis` makes it evaluate all binary trees that resolve the constraint tree. Since all trees we found using other methods will be in the set of refinements of the constraint tree, this strategy enables `BPAnalysis` to evaluate these trees and to find other, potentially better, trees.

2.6.3 Experiment 1: Neighbor-Joining on Synthetic Data

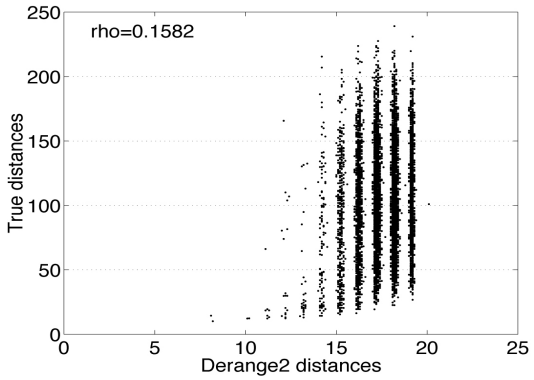
The first round of experiments focused on the performance of neighbor joining under a variety of model conditions. We generated three random model trees. T_A had 20 genomes and 20 genes, with high rates of change (3 to 10 inversions per edge

on average), T_B had 20 genomes and 20 genes, but low rates of change (1 to 3 inversions per edge on average), and T_C had 20 genomes and 105 genes, with low rates of change (1 to 3 inversions per edge on average). In each of the 50 runs of this experiment, we ran our simulator on each random tree with relative costs of 1, 2.1, and 2.1 for inversions, transpositions, and inverted transpositions. This simulation generated gene orders for the 20 genomes at the leaves. Each run thus gives rise to three matrices: D, BP, and ITT (true distances, breakpoint distances, and ITT distances). The matrix D is determined during the simulation, the matrix BP can be calculated exactly in linear time, but the matrix ITT is estimated using `derange2`, perhaps with significant errors. We constructed the neighbor-joining trees on the BP and ITT matrices, thus producing trees NJ(BP) and NJ(ITT) (see our earlier discussion). These were then compared with the model tree, scoring the comparison in terms of false negatives (since all trees are binary, false positive and false negative rates are identical). Note that, on trees with low rates of evolution (T_B and T_C), slightly more than 3 edges per tree have no changes; in these cases, a false negative rate of around 3 would indicate complete success, so that all false negative rates should be scaled down accordingly. (3 edges represents 18% of the interior edges of T_B and T_C; thus false error rates should be decreased by about 12% to make up for the zero-length edges and the expected accuracy of a guessed resolution of an unresolved tree.) The results are summarized in Table 2.3.

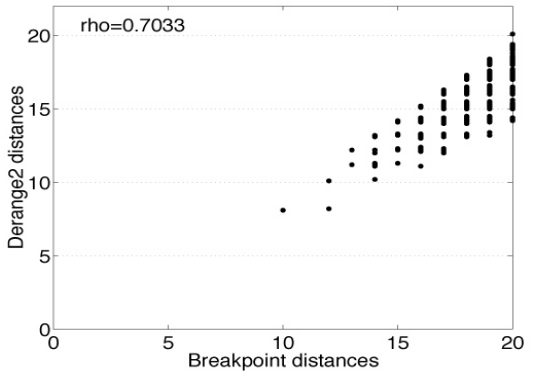
In Figures 2.4, 2.5, and 2.6, we compare the distances BP and ITT to the true distances, on trees T_A, T_B, and T_C, respectively. We also give the correlation coefficient between the two measurements in each figure—a statistical measure of the degree to which the two distances are linearly related. Note how closely correlated the breakpoint and ITT distances are in the second and third cases (and, to a lesser extent, in the first case), indicating a linear or nearly linear relationship. In contrast, the true distance shows no particular correlation to the other two distances in the



(a) breakpoint vs. true distances

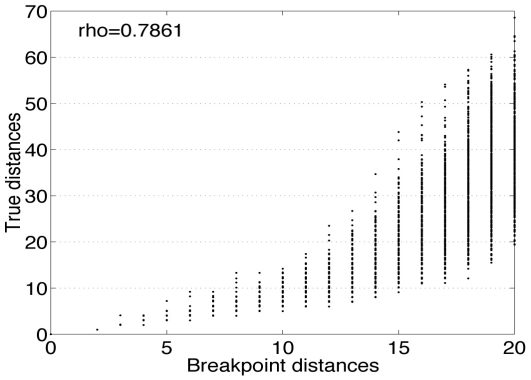


(b) ITT vs. true distances

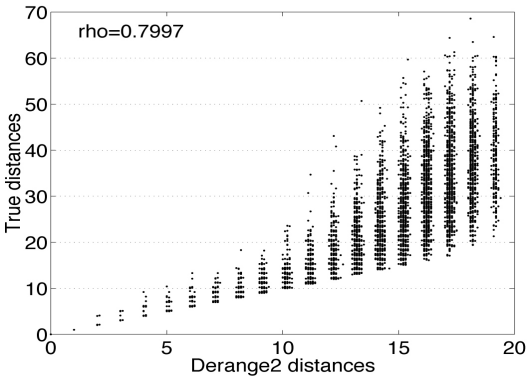


(c) breakpoint vs. ITT distances

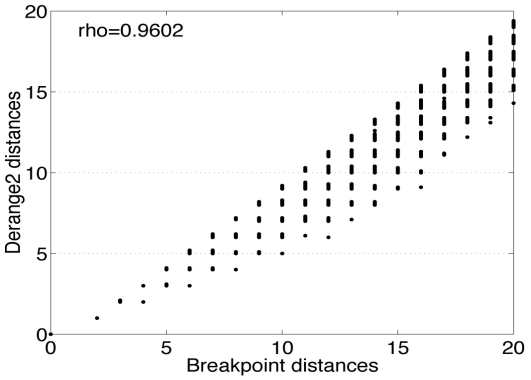
Figure 2.4: Comparison of distances on model tree T_A.



(a) breakpoint vs. true distances

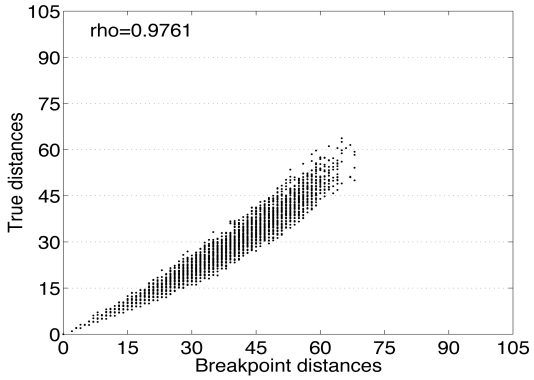


(b) ITT vs. true distances

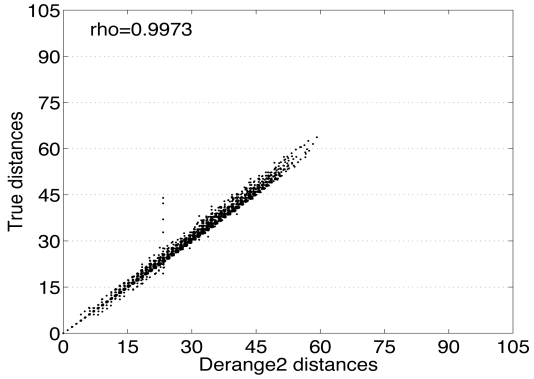


(c) breakpoint vs. ITT distances

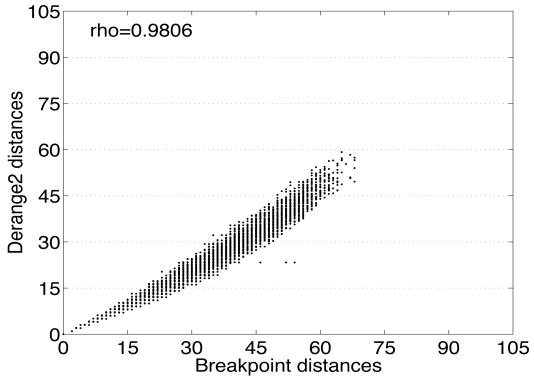
Figure 2.5: Comparison of distances on model tree T_B.



(a) breakpoint vs. true distances



(b) ITT vs. true distances



(c) breakpoint vs. ITT distances

Figure 2.6: Comparison of distances on model tree T_C.

first two trees. In tree T_C, all three distances are closely correlated, reflecting the relative lack of evolution and overall simplicity of that tree.

Neighbor-joining does quite well on the third tree T_C, but poorly on T_B and very poorly on T_A. Furthermore, its performance does not appear to depend upon the choice of edit distance, but it does correlate well with the accuracy of the edit distance calculation (BP or ITT) with respect to the true distance D. This accuracy in turn seems to depend upon the rate of evolution relative to the number of genes in the genomes.

2.6.4 Experiment 2: All Methods on Synthetic Data

In this experiment, we simulated only inversion events and so used the Nadeau-Taylor model of evolution. We varied the number of genomes, the number of genes, and rates of evolution. We computed BP distances, ITT distances, and I distances and then calculated neighbor-joining trees NJ(BP), NJ(ITT) (and sometimes NJ(I)) for these distance matrices. We computed the strict consensus of the trees obtained during Phase I of the MPBE method; in some cases we also computed trees using `BPAnalysis` with the strict consensus of various recovered trees given as a constraint tree (see the discussion above). We compared each tree to the model tree and computed false negatives and false positives. Our results are summarized in Table 2.4. As the model trees and neighbor-joining trees are always binary, we only report false negative rates for neighbor-joining trees. On the other hand, we report both false negatives and false positives for the MPBE strict consensus trees.

These results indicate that the various methods (neighbor joining on BP and ITT distances and maximum parsimony on binary encodings of gene order data) have the same qualitative performance on all model conditions we examined. That is, we cannot as yet identify a model condition under which one method will outperform the others. However, one other trend is clear: all methods do well when

the rate of change on an edge is low relative to the number of genes, while their performance decreases as this rate increases. What is surprising is that the rate at which their performance decreases appears to be the same.

We then examined the performance of `BPAnalysis` with respect to solving the breakpoint phylogeny problem. We were also interested in determining whether the model tree is one of the breakpoint phylogenies (and hence determine whether solving the breakpoint phylogeny is a good approach to reconstructing trees from gene order data). However, our results for `BPAnalysis` are limited, because of the extreme slowness of the program; we found that the trees obtained by `BPAnalysis` were almost always the same trees found by using Phase I of the MPBE method, provided that we let `BPAnalysis` run long enough. Therefore, `BPAnalysis` seems to be doing a reasonably effective job at solving the breakpoint phylogeny problem.

It seems that the breakpoint phylogeny may not always be a good estimate of the model tree. In our experiments, the breakpoint phylogeny is a good estimate of the model tree only when the rates of evolution on each edge are low relative to the number of genes. In these cases, the model tree is one of the breakpoint phylogenies or is close to optimal. In other cases, the breakpoint score of the model tree is significantly larger than the breakpoint scores found by either MPBE or `BPAnalysis`. This discrepancy suggests that, for model conditions in which the rates of evolution are high, breakpoint phylogenies are unlikely to be accurate estimates of the true evolutionary tree.

2.7 Software Issues

Running time is always important in comparing phylogenetic methods. While neighbor joining runs in polynomial time, neither MPBE nor `BPAnalysis` does. Even calculating the distance matrix between every pair of signed circular genomes in a large set is computationally challenging. `derange2` is fast, but inexact: because it

heuristically computes the distance between two genomes by using inversions, transpositions, and inverted transpositions using a greedy strategy, it only allows an operation if that operation decreases the breakpoint distance between the two genomes. Consequently, it can miss minimal edit sequences, as we observed in our tests. Hannenhalli's software `signed_dist` for pairwise distances runs in slow polynomial-time ($\Theta(k^4)$ to compute distances between a pair of genomes on k genes); in order to compute all pairwise distances, it requires $\Theta(n^2k^4)$ time. For our datasets, k was as large as 105 and n as large as 60; we found the running time to be considerable for any $k \geq 40$, and even for smaller k if n was large.

We timed each method on the Campanulaceae dataset, using a Sun E5500 with 2GB of memory running Solaris 2.7. The first phase of MPBE took 0.15 seconds to complete on the Campanulaceae dataset (finding the four maximum-parsimony trees with `PAUP*` took 0.15 seconds on a Macintosh G4). The second phase took somewhat longer. Labeling the internal nodes with `BPAnalysis` took 0.38 seconds for each tree. Computing inversion distances on each edge using our modified `signed_dist` took 0.02 seconds and computing ITT distances on each edge using `derange2` took 0.01 seconds. The second phase of MPBE thus took about 4.5 seconds in all. Hence the complete MPBE analysis ran in under 5 seconds.

We also attempted to time `BPAnalysis` on the Campanulaceae dataset, but it did not complete its search, so we had to estimate the amount of time it took per tree and extrapolate. Our experiments suggest that `BPAnalysis` evaluates 120 trees a minute; at this rate, since the number of trees on 13 leaves is 13,749,310,575, `BPAnalysis` would take well over 200 years to complete its search of tree space for our problem. Blanchette *et al.* did complete their analysis of the metazoan dataset, which has 11 genomes on a set of 37 genes. This is a much easier problem, as there are far fewer trees to examine (only 2,027,025) and as scoring each tree involves solving a smaller number of TSP instances on a much smaller number of cities (37

rather than 105). Overall, it is clear that datasets of sizes such as ours are currently too large to be fully analyzed by `BPAnalysis`.

In view of these observations, our new method stands as a good compromise between speed and accuracy. Neighbor-joining is faster (guaranteed polynomial-time), but returns only one tree and thus tells us little about the space of near-optimal trees, while `BPAnalysis` is quite slow. Furthermore, our results confirm that our new method returns results as good as any of the other methods and does so within very reasonable times, even on datasets on which `BPAnalysis` cannot run to completion.

2.8 Conclusions

Our initial study on real and synthetic data containing a single chromosome suggests that, for some conditions (when the rate of inversions per edge is low relative to the number of genes), many of the proposed methods for reconstructing small phylogenetic trees from gene order data *can recover* highly accurate tree topologies. Further, under model conditions with low evolutionary rates, the breakpoint phylogeny seems to be a good candidate for the true evolutionary tree. Consequently, under these conditions, methods that seek the breakpoint phylogeny offer real promise. However, the methods can be distinguished in terms of the computational effort involved, in which respect the MPBE method is a significant improvement over `BPAnalysis` for at least some moderate to large datasets.

Our results suggest that all of the methods we evaluated have unacceptable levels of errors on trees in which the inversion rate on the edges is high relative to the number of genes. Thus, new methods need to be developed for these types of genome evolution problems and current approaches to phylogenetic analyses based upon gene orderings should be restricted to cases with low rates of evolution. These findings apply to neighbor joining based upon various ways of calculating genome

distances, maximum-parsimony analyses of binary sequences derived from genome data, and breakpoint phylogenies. Indeed, it may be that any approach for solving the breakpoint phylogeny will perform poorly in the presence of high evolutionary rates relative to the number of genes. In such cases, approaches that explicitly seek to minimize the total number of evolutionary events may be required, but no such method currently exists.

2.9 Future Work and Recommendations

Faster methods are needed for solving the breakpoint phylogeny problem, as well as to score trees with respect to evolutionary distances (ITT and I). Since MPBE depends upon `BPAnalysis` in order to label internal nodes with circular genomes, and upon `derange2` and `signed_dist` to score these trees for ITT and I distances, a first step should be to speed up `BPAnalysis` and `signed_dist`, and improve the accuracy of `BPAnalysis` and `derange2` (since these find local optima but not necessarily global optima). More effective implementations of the basic concept in `BPAnalysis`, such as hill-climbing or branch-and-bound through the tree space and abandoning strict optimality in solving the TSP instances in favor of a fast and reliable heuristic (such heuristics abound in the TSP literature), could make the method run fast enough to be applicable to datasets comparable to ours.

Since the utility of the breakpoint phylogeny for reconstructing phylogenies from gene order data seems limited to low evolutionary rate conditions, we will also investigate methods which explicitly seek to minimize the number of mutations (inversions, transpositions, inverted transpositions, as well as insertions, deletions, and duplications of gene segments).

We note that in our studies the polynomial-time method of neighbor joining has performed as well as MPBE in terms of topological accuracy, bringing into question whether the more computationally intensive approaches deserve considera-

tion. One clear advantage of both MPBE and `BPAnalysis` is that they tell us more about the space of optimal and near-optimal trees than neighbor joining does and hence help us identify alternative hypotheses. The task remains to identify regions of the parameter space in which MPBE or `BPAnalysis` outperform neighbor joining in topological accuracy. We conjecture that such regions do exist (as other studies based upon biomolecular sequence evolution show [50, 94]).

Given the rapid increase in the availability of complete genome sequences, the current limitation in reconstructing phylogenies from gene order data for datasets containing many genomes or genes is of major concern. Until improved methods are developed, we recommend that phylogenetic analyses of gene order data seek to obtain the breakpoint phylogenies and that these breakpoint phylogenies then be scored under ITT distances, for some appropriate weighting of the events. We also recommend that MPBE be used, until `BPAnalysis` can be made competitively fast.

Table 2.4: The false negative rates (in %) with respect to the true tree of various reconstruction methods for various model trees and rates of evolution.

Genomes	Genes	Inv./Edge[a]	NJ(BP)[b]	NJ(ITT)[c]	NJ(I)[d]	MPBE[e]
10	105	9–11	0	0	0	0 / 0
25	105	1–5	9.09	4.55		9.09/ 4.55[f]
25	105	4–6	0	0		0 / 0
25	105	1–10	9.09	0		4.55/ 4.55
40	105	1–5	13.51	10.81		10.81/ 2.70[f,g]
40	105	1–10	16.22	0		2.70/ 2.70[g]
25	37	1–5	22.73	9.09	4.55	27.27/ 9.09[f]
25	37	1–10	9.09	13.64	13.64	31.82/13.63[f]
40	37	1–5	37.84	10.81	18.92	35.14/ 2.70[f,g]
40	37	1–10	32.43	32.43	32.43	48.65/24.32[f,g]
20	20	3–10	49.41	60.00	60.00	65.88/20.00[f]
60	20	3–5	66.66	68.42		75.43/57.89[f,g]

[a] the expected number of inversions per edge

[b] neighbor joining on the breakpoint distance matrix

[c] neighbor joining on the ITT distance matrix computed by `derange2`

[d] neighbor joining on the inversion distance matrix computed by `signed_dist`

[e] maximum parsimony on the binary encoding of the genomes; includes both false negative and false positive rates

[f] the strict consensus of all maximum-parsimony trees

[g] dataset too large for branch-and-bound parsimony, heuristic used instead

Chapter 3

Gene Order Phylogenetics: A Detailed Study of Breakpoint Analysis

3.1 Introduction

Some organisms have a single chromosome or contain single-chromosome organelles (mitochondria or chloroplasts), the evolution of which is mostly independent of the evolution of the nuclear genome. Given a particular strand from a single chromosome (whether linear or circular), we can infer the ordering of the genes along with the directionality of the genes, thus representing each chromosome by an ordering of oriented genes. The evolutionary process that operates on the chromosome may include inversions and transpositions, which change the order in which genes occur in the genome as well as their orientation. Other events, such as insertions, deletions, or duplications, change the number of times and the positions in which a gene occurs. Gene order, orientation, and number represent a new source of data for phylogeny reconstruction. Appropriate tools for analyzing such data may help resolve some

difficult phylogenetic reconstruction problems; indeed, this new source of data has been embraced by many biologists in their phylogenetic work [83, 85, 93].

A natural optimization problem for phylogeny reconstruction from this type of data explicitly attempts to reconstruct an evolutionary scenario with a minimum number of permitted evolutionary events (e.g., duplications, insertions, deletions, inversions, and transpositions) on the tree. Such approaches are computationally very intensive (all are known or conjectured to be NP-hard); worse, to date, no automated tools exist for solving such problems. Another approach is first to estimate leaf-to-leaf distances (based upon some metric) between all genomes, and then to use a standard distance-based method such as neighbor joining[95] to construct the tree. Such approaches are quite fast and may prove valuable in reconstructing the underlying tree, but cannot recover the ancestral gene orders.

Blanchette and Sankoff developed a technique, breakpoint phylogeny, for the special case in which the genomes all have the same set of genes, and each gene appears once. Our earlier simulation study suggests that the approach works well for certain datasets (i.e., it obtains trees that are close to the model tree), but that the implementation, the `BPAnalysis` software, is too slow to be used on anything other than small datasets with a few genes (as described in Chapter 2). In this chapter, we describe our reimplementation of `BPAnalysis` and how we have obtained speedups of 2 to 3 orders of magnitude.

3.2 Definitions

When each genome has the same set of genes and each gene appears exactly once, a genome can be described by an ordering (circular or linear) of these genes, each gene given with an orientation that is either positive (g_i) or negative ($-g_i$). Given two genomes G and G' on the same set of genes, a *breakpoint* in G is defined as an ordered pair of genes, (g_i, g_j), such that g_i and g_j appear consecutively in that

order in G, but neither (g_i, g_j) nor $(-g_j, -g_i)$ appears consecutively in that order in G'. The breakpoint distance between two genomes is the number of breakpoints between that pair of genomes. The breakpoint score of a tree in which each node is labeled by a signed ordering of genes is then the sum of the breakpoint distances along the edges of the tree. Given three genomes, we define their *median* to be a fourth genome that minimizes the sum of the breakpoint distances between it and the other three. The *Median Problem for Breakpoints* (MPB) is to construct such a median and is NP-hard [89]. Sankoff and Blanchette developed a reduction from MPB to the Traveling Salesman Problem (TSP), perhaps the most well-studied of all optimization problems [55]. Their reduction produces an undirected instance of the TSP from the directed instance of MPB by representing each gene by a pair of cities connected by an edge that must be included in any solution.

3.3 BPAnalysis

`BPAnalysis` (see Figure 3.1) is the method developed by Blanchette and Sankoff to solve the breakpoint phylogeny. Within a framework that enumerates all trees, it is an iterative heuristic to label the internal nodes with signed gene orders. This procedure is computationally very intensive. The outer loop enumerates all $(2n-5)!!$ leaf-labeled trees on n leaves, an exponentially large value. The inner loop runs an unknown number of iterations (until convergence), with each iteration solving an instance of the TSP (with a number of cities equal to twice the number of genes) at each internal node. The computational complexity of the entire algorithm is thus exponential in each of the number of genomes and the number of genes, with significant coefficients. The procedure nevertheless remains a heuristic: even though all trees are examined and each MPB problem solved exactly, the tree-labeling phase does not ensure optimality unless the tree has only three leaves.

```
Initially label all internal nodes with gene orders
Repeat
    For each internal node v, with neighbors A, B, and C, do
        Solve the MPB on A, B, C to yield label m
        If relabeling v with m improves the score of T, then do it
Until no internal  node can be relabeled
```

Figure 3.1: High-level description of the main loop of `BPAnalysis`.

3.4 Study Objectives

Our earlier experiments with techniques for reconstructing phylogenies from gene order data suggested that Sankoff and Blanchette's implementation of `BPAnalysis` is much too slow. On a collection of Campanulaceae with 13 genomes of 105 gene segments, we estimated that Sankoff and Blanchette's `BPAnalysis` would take well over 200 years to complete—an estimate based on the average number of trees processed by the code per unit time and extended to the 13,749,310,575 tree topologies on 13 leaves. Although an exhaustive search of tree topologies is likely impossible for more than 15 genomes (there are $0.2\text{x}10^{15}$ trees on 16 genomes), even selective exploration of tree space requires very fast labeling of the internal nodes of a tree.

Our objective was therefore to develop a much faster implementation of `BPAnalysis`, prior to modifying the method used for searching tree space. Our three major goals were flexibility (e.g., the ability to change TSP solvers or to change distance measures between genomes), the introduction of approximate TSP solvers (which are required for large number of genes), and overall speed. To achieve the last goal, we used a process termed algorithmic engineering [73, 76]—a combination of low-level algorithmic changes, data structures changes, and coding strategies that combine to eliminate bottlenecks in the code, balance its computational tasks, and make it cache-sensitive.

3.5 Algorithmic Aspects of Our Implementation

3.5.1 Tree Generation

Exploring tree space, whether exhaustively or selectively, requires the efficient generation of tree topologies. We need a generating mechanism that is interruptible and restartable at any point. We chose to generate a preorder encoding of the tree, then to produce the topology from the encoding. Generating the next tree in the ordering takes amortized constant time. This enables us to produce only trees that are refinements of a given constraint tree, as well as to generate only every kth tree for a given stepping value k. The stepping value is a crucial feature for sampling-based exploration. By generating every kth tree, we investigate a wide range of tree topologies for large datasets. Without the step, we are limited to a fraction of very similar tree topologies. Detailed profiling shows that the time taken by tree generation does not rise above the noise level in our time measurements.

3.5.2 Tree Labeling

Labeling the internal nodes of a tree is the most challenging part of the problem—indeed, no algorithm is known that would produce an optimal solution for more than three leaves. Although the problem is NP-hard even for a three-leaf tree, it is possible to produce an optimal solution for many realistic problems on three leaves by using the TSP reduction. The approach used by Sankoff and Blanchette to label an entire tree is to do a postorder traversal of the tree; at each internal node, use the labels of its three neighbors to define an instance of the MPB, solve that problem, and assign the new label to the node if the number of breakpoints is thereby lowered; and repeat the entire process until no change occurs. This process is rather wasteful—an NP-hard problem must be solved at each internal tree node, over and over, with most solutions discarded because they do not bring about any

improvement. Our implementation only generates an MPB problem for nodes that saw at least one of their three neighbors relabeled over the last pass. We also score the tree incrementally in constant time after each relabeling (whereas `BPAnalysis` calls a tree-scoring routine that requires linear time to run) and do so only if the label has changed—a relatively rare occurrence. These changes brought about a speed-up on the Campanulaceae on the order of 1.5.

3.5.3 Condensation

Sets of closely related genomes will inevitably share a number of adjacencies even when not closely related. In those cases when all genomes in a set contain shared adjacencies, we condense the shared adjacencies by redefining gene fragments to consist of the longest shared subsequences and replace the original instance by one given in terms of the new gene fragments. Such condensation does not affect labeling or any of the rearrangement based distance measures (breakpoint, inversion, transposition). Condensation can be implemented (and reversed) efficiently and saves time by producing significantly smaller numbers of genes in the genomes—and hence smaller TSP instances. We use both an initial condensing of the entire dataset and a dynamic condensing of each triple of genomes when computing the median. In our Campanulaceae dataset, the 13 genomes have sufficient runs in common that they can be initially condensed from 105 down to just 36 gene segments. When only three genomes are considered at a time (as in the TSP instances), condensing can have an even greater impact. Combining initial and dynamic condensation on the Campanulaceae dataset results in a speed-up by a factor of 6.

3.5.4 Approximate TSP Solvers

Each MPB problem is solved through reduction to a TSP instance; the instance produced has two cities for each gene in the genome. The number of such instances

solved in the analysis of a dataset is very large—and, of course, the TSP problem is itself NP-hard. Therefore we spent most of our design effort on the TSP solver. We used the Concorde library [3] for two of our three solvers—the chained and the simple versions of the well-known Lin-Kernighan heuristic [65]. These heuristics typically come within a few percent of optimal for the simple version and even closer for the slower chained version, at least for the geometric TSP instances used in most testbeds [55]. Unfortunately, the LK solvers are quite slow—even the simple LK solver takes cubic time and suffers from significant coefficients.

We also implemented the standard greedy algorithm for TSP (also known as "coalesced simple paths" [78]). This algorithm successively adds the next available edge of least cost, subject to not creating a short cycle nor a vertex of degree three. For our instances of the TSP, this method can be implemented to run in very fast linear time, but tends to yield significantly poorer solutions than the LK solvers.

3.5.5 Our Exact TSP Solver

We implemented a standard include-exclude backtracking search with pruning—the most basic technique for exhaustive search of a state space—along the lines of the `BPAnalysis` code. This approach orders the edges by cost, then recursively attempts first to include, then to exclude each edge in turn, the inclusions subject again to not creating a short cycle nor a vertex of degree three. (In effect, the greedy method described earlier is simply the first probe sequence of this search method.) The recursion stops when a solution is obtained, when it runs out of edges, or when a lower bound computation indicates that no tree can be found to improve upon the current upper bound (the value of the current best solution).

In comparison with `BPAnalysis`, we used more streamlined data structures, better bounding, and some features tailored to the special nature of the instances generated in the reduction. Of the $\Theta(n^2)$ edges of an instance produced by the

reduction to TSP, at most $3n$ are nontrivial—those that correspond to adjacent genes segments in the three genomes. Our exact TSP solver considers only nontrivial edges, treating the others as an undifferentiated pool—a refinement that allows each step in the search to run in linear rather than quadratic time. Our lower bound is computed with as much information as can easily be maintained in linear time—excluded from the computation are not just edges that have already been considered (as in `BPAnalysis`), but also any edges that would create a short cycle or a vertex of degree three. Finally, we provide the solver with what often proves a very tight upper bound by determining which of the current label and its three neighbors would provide the best median, and initializing the solver with this solution. Our lower and upper bounds prove tight enough that over two thirds of the calls to our TSP solver are pruned immediately, without a single attempt to include or exclude an edge. This combination of algorithmic changes accounted for a speed-up factor of 10 on the Campanulaceae dataset.

3.5.6 Initial Labeling

Since the labeling procedure is iterative, assigning initial labels can make a big difference in performance. Sankoff and Blanchette proposed several initializations. We implemented all but one of them (their last heuristic, based on an *ad hoc* solution method for a set of linear equations to define the parameters of each TSP instance, is extremely slow), along with several of our own devising. The choice of an initialization procedure is crucial, because little to no relabeling is done after a good initialization. All but one of the methods run in linear time (one of the two methods described by Sankoff and Blanchette as "onerous" takes quadratic time), with the exception of the cost of the calls to the exact TSP solver. Some of the methods make no such call, some make one, while the more demanding methods make one such call at each internal node. After much experimentation with these methods,

we settled on one of those proposed by Sankoff and Blanchette as the best compromise of accuracy and speed—but our code provides another 6 methods. The chosen method sets up a TSP instance for each internal node by using the closest already labeled neighbor along each of the three directions out of that internal node; the computed median is assigned to the internal node, which is then considered to be labeled.

3.6 Coding Aspects of Our Implementation

Algorithmic engineering suggests a refinement cycle in which the behavior of the current implementation is studied in order to identify problem areas which can include excessive resource consumption or poor results. We used extensive profiling and testing throughout our development cycle, which allowed us to identify and eliminate a number of such problems. For instance, converting the MPB into a TSP instance dominates the running time whenever the TSP instances are not too hard to solve. We cut the running time of that routine down by a factor of at least six—and thereby nearly tripled the speed of the overall code. We focused equal attention on distance computations and on the computation of the lower bound, with similar results. These steps provided a speed-up by a factor of 6–8 on the Campanulaceae dataset.

The original `BPAnalysis` is written in C++ and uses a space-intensive full distance matrix, as well as many other data structures. It has a significant memory footprint (over 60MB when running on the Campanulaceae dataset) and poor locality (a working set size of about 12MB). Our implementation has a tiny memory footprint (1.8MB on the Campanulaceae dataset) and mostly good locality (nearly all of our storage is in arrays preallocated in the main routine), which enables it to run almost completely in cache (the working set size is 600KB). Cache locality can be improved by returning to a FORTRAN-style of programming, in which records

(structures/classes) are avoided in favor of separate arrays, in which simple iterative loops that traverse an array linearly are preferred over pointer dereferencing, in which code is replicated to process each array separately, etc. While we cannot measure exactly how much we gain from this approach, studies of cache-aware algorithms [61, 118] indicate that the gain is likely to be substantial—factors of anywhere from 2 to 40 have been reported. Many new memory hierarchies show differences in speed between cache and main memory that exceed two orders of magnitude.

3.7 Experimental Procedure

We had two objectives for our experiments. We wanted to compare the raw running times of our versions and of `BPAnalysis` and investigate their dependency on the number of genomes, the number of genes, and the rate of evolution in the model. We also wanted to study the impact on the quality of solutions of using an approximation algorithm for TSP.

We used a simulator we developed to create datasets for our experiments. First, a random tree topology is generated and then evolution is simulated on the tree using inversions and transpositions as the evolutionary events. We examined different parameter settings by varying the expected number of inversions and transpositions per edge, as well as varying the number of genes and the number of genomes. We generated datasets with each of 2, 4 or 8 events per edge to simulate different rates of evolution. The numbers of genes in each dataset were 10, 20, 40, 80, 160, and 320. In order to obtain statistically significant data, we used runs of trials. Each trial is one dataset, while a run is composed of a number of independent trials. One then retains only the median value for the entire run and repeats the process with additional runs, with each run yielding a single value. These values are then examined for consistency. This method produces robust results even when, as in our case, the enormous size of the sample space precludes any fair sampling.

3.8 Experiments

In order to test our TSP solvers, we ran a number of tests with just 3 genomes. These problems have only one tree topology and only one internal node to label, so that only one call is made to the TSP solver. We used 6 different numbers of genes per genome and up to three rates of evolution. In some cases, the rate $r = 8$ was too high a rate of evolution for the exact solvers—so we do not present data for those cases. We ran the 3-genome problem for five different methods: `BPAnalysis` itself, our exact solver, the greedy algorithm, the basic Lin-Kernighan, and the chained Lin-Kernighan.

In our first experiment, we tested the different solvers under the different rates of evolution. In our second experiment, we used the same data to compare the relative running times of the 5 solvers. In our third experiment, we computed the percent over optimal that the approximate solvers obtained. Our fourth experiment explored tree processing rates for the various solvers. We used both real and synthetic datasets for this experiment, including our Campanulaceae dataset. Most datasets did not run to completion, but we ran them long enough to obtain an accurate count of the number of trees processed per unit time and thus to be able to predict the running time to completion (in the case of our code, we also used the sampling option to ensure that the estimates were not biased towards the small fraction of trees explored in a large problem).

3.9 Results and Discussion

3.9.1 Experiments on 3-Genome Problems

Figure 3.2 shows the running times of our four versions and of `BPAnalysis` on 3-genome problems of increasing sizes and at different rates of evolution. The rates $r = 2$, 4, and, 8 reflect the expected number of inversions on each tree edge. Higher

rates of evolution clearly induce much harder instances of the TSP, so that the two exact solvers suffer; `BPAnalysis` could not solve any of the larger instances with $r = 8$ within 20 minutes of computation.

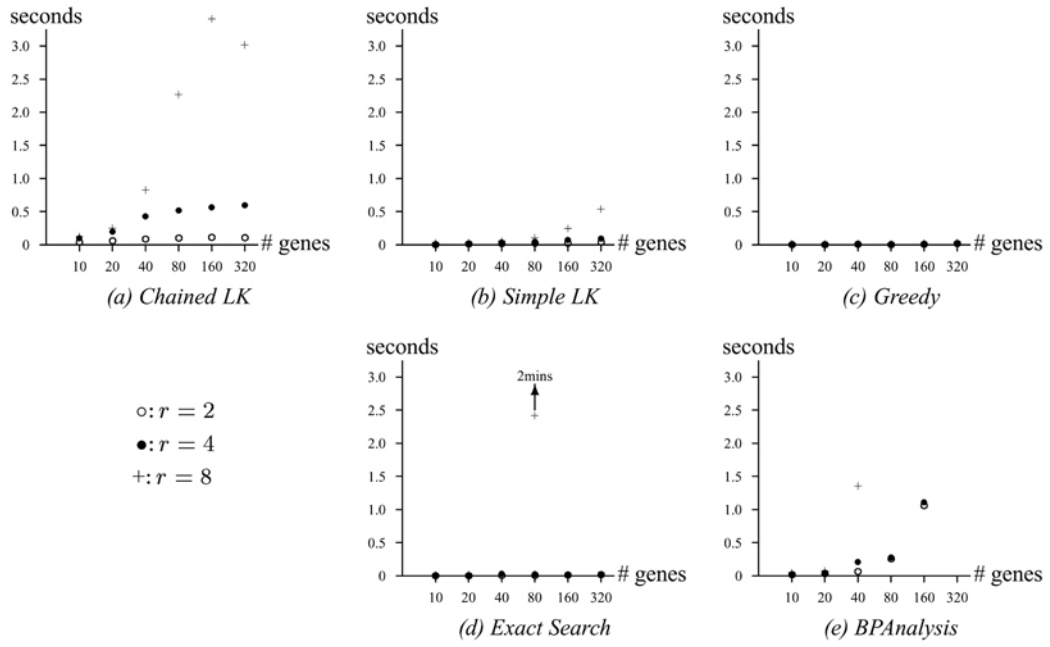


Figure 3.2: Speed of the five solvers on various 3-genome problems under 3 different rates of evolution.

Figure 3.3 presents the same data, this time per evolutionary rate so as to facilitate comparisons of running times. The greedy solver is so fast that all of its times fall on the horizontal axis, as do most of the times of the exact solver for $r = 2$ and $r = 4$. The greedy solver runs (in our special case) in linear time, the two LK solvers in roughly cubic time, and the exact solvers in exponential time. Our figures clearly demonstrate the exponential behavior of `BPAnalysis`, but our exact solver stays in a flat part of its exponential curve all the way to 320 genes for $r = 2$ and $r = 4$. The two LK solvers show at least quadratic behavior, while the greedy solver give us nearly flat values (below noise level) for the entire range. Because high rates of evolution induce numerous breakpoints, the resulting instances of the TSP have

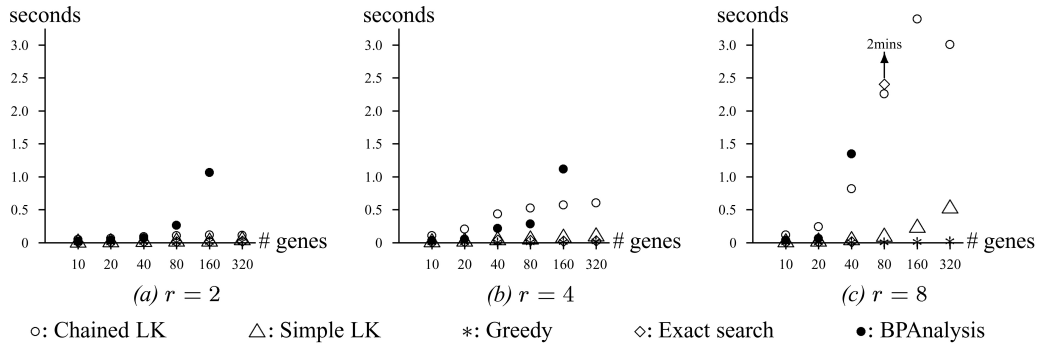


Figure 3.3: Relative running times of the five methods on 3-genome problems of various sizes.

relatively undifferentiated edges costs: thus most of the edges have maximal or near maximal cost, making it difficult to find an optimal solution quickly, so that the exact solvers suffer—indeed, the problems rapidly become intractable. In contrast, the Chained LK solver slows down a bit and the simple LK and greedy solvers not at all.

3.9.2 Quality of Approximation of LK Heuristics

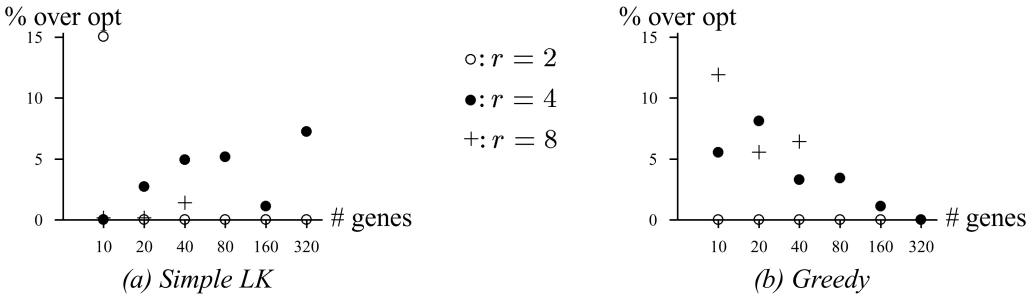


Figure 3.4: Percentage excess over optimal for LK and greedy solvers.

Figure 3.4 shows the percentage by which the solutions returned by the simple LK solver exceed the optimal. In contrast, the Chained LK solver returned

method	13/105/-	8/105/-	10/20/2	10/20/4	10/20/8	10/160/2	60/20/2	80/100/2
Greedy baseline in trees/s	23,100	73,400	15,300	15,200	13,550	5,250	2,400	975
Simple LK	68	66	31	33	31	40	45	100
Chained LK	280	220	225	310	300	210	250	330
Exact solver	3.5	1.1	3.4	4.3	3.6	1.7	2.6	2.6
`BPAnalysis`	2,000	3,820	220	250	225	840	350	500

Table 3.1: Ratios of tree-processing rates of 5 methods to the rate of the greedy method on various datasets.

optimal solutions for all of these test instances. The error percentage can be artificially larger for smaller evolutionary rates because, for these lower rates, the number of breakpoints in the optimal solution is quite small, thus magnifying percentages. Rather surprisingly, the fast and unsophisticated greedy algorithm holds its own against the simple LK algorithm—not a situation encountered on typical TSP benchmark[55].

3.9.3 Tree Processing Rates

In order to estimate the raw speed of each method, we ran all given methods on real and synthetic datasets, letting them process thousands of trees until a fixed time bound was attained. We then normalized the results and, since the greedy algorithm was always the fastest, computed ratios of processing rates for the other four methods against the rate of the greedy method. Table 3.1 shows the results. In the table, $n/N/r$ denotes a problem with n genomes, N genes, and r inversions per model tree edge. The first two data sets are the full Campanulaceae dataset and its first 8 members, respectively. The figure shown for the greedy method is the actual processing rate of that method, in trees processed per second. The high processing rate of our exact solver (we have observed rates from 70 to 5,000 times faster than `BPAnalysis`) makes it possible to solve problems with 10–12 genomes on a single processor. Chained LK is much too slow to be of use, and even simple LK, while often faster than `BPAnalysis`, is far slower than our exact solver. On the

other hand, the greedy algorithm, while much faster than the exact solver, tends to yield worse solutions than the exact solver and should be reserved for difficult instances (large numbers of genes and high rates of evolution).

3.10 Conclusions and Future Work

We have presented a new implementation of breakpoint analysis that improves on the original `BPAnalysis` by 2 to 3 orders of magnitude. Our implementation makes it possible to analyze much larger datasets; it can be combined with massive parallelism, reducing the running time for the Campanulaceae dataset from two centuries down to a day when run on a 512-processor supercluster. Breakpoint scores may not be the measure of choice; our latest implementation includes a fast linear-time computation of inversion distances, allowing us to minimize either measure and compare their relative use. These and other improvements pale against the main drawback of the approach: enumerating all tree topologies is impossible for 16 or more genomes. An implicit exploration of tree space is the next step.

Chapter 4

Advances in Gene Order Phylogenetics

4.1 Introduction

Biologists can infer the ordering and strandedness of genes on a chromosome, and thus represent each chromosome by an ordering of signed genes (where the sign indicates the strand). These gene orders can be rearranged by evolutionary events such as inversions and transpositions and, because they evolve slowly, give us an important new source of data for phylogeny reconstruction. Developing such tools is thus an important area of research—indeed, the DCAF symposium [31] was devoted to this topic, as was a workshop at DIMACS [32].

A natural optimization problem for phylogeny reconstruction from gene order data is to reconstruct an evolutionary scenario with a minimum number of the permitted evolutionary events on the tree. This problem is NP-hard for most criteria—even the very simple problem of computing the median of *three* genomes under such models is NP-hard in the number of genes[22, 89]. All approaches to phylogeny reconstruction for such data must therefore find ways of handling the

significant computational difficulties. Moreover, because suboptimal solutions can yield very different evolutionary reconstructions, exact solutions are strongly preferred over approximate solutions (see [106]).

For some datasets (e.g., chloroplast genomes of land plants), biologists conjecture that the only rearrangement events that occur are *inversions*. In other datasets, transpositions and inverted transpositions are viewed as possible, but their relative preponderance with respect to inversions is unknown, so that it is difficult to define a suitable distance measure based on these three events. Researchers have used breakpoint distance as an independent measure of distance between genomes and the *breakpoint phylogeny*, proposed by Blanchette *et al.* [10], is the most parsimonious tree with respect to breakpoint distances.

4.2 Prior Results

We build upon several major prior results.

BPAnalysis: Blanchette *et al.* [10] proposed the breakpoint phylogeny (which seeks the tree with the fewest breakpoints) and developed a reconstruction method, `BPAnalysis` [100], for that purpose. Their method examines every possible tree topology in turn and for each topology, it generates a set of ancestral genomes so as to minimize the total breakpoint distance in the tree. This method returns good results, but takes exponential time: the number of topologies is exponential and generating a set of ancestral genomes is achieved through an unbounded iterative process that must solve an instance of the Traveling Salesman Problem (TSP) for each internal node at each iteration. And hence, the total running time is exponential in *both* the number of genes and the number of genomes.

MPBE: We developed an alternate method, based on a binary encoding of breakpoints, to take advantage of existing parsimony software [29, 28]. This method, called *Maximum Parsimony on Binary Encodings* (MPBE), is exponential only in the number of genomes (because the parsimony problem is NP-hard), runs very fast in practice, but returns only candidate tree topologies and so must make use of the labeling phase of `BPAnalysis` in order to return ancestral genomes. (Similar approaches based on neighbor joining suffer from the same problem.)

GRAPPA: We reimplemented `BPAnalysis` in order to analyze our larger datasets and also to experiment with alternate approaches. Our program, called `GRAPPA` [79], includes all of the features of `BPAnalysis`, but runs about three orders of magnitude faster. As part of the development of `GRAPPA`, we designed a new and very fast linear-time algorithm for computing inversion distances [6], which has enabled us to extend our work on breakpoint phylogeny to the inversion phylogeny.

4.3 New Results

We present several new results in this chapter:

- A simulation study examining the relationship between topological accuracy and two definitions of tree length: the number of breakpoints on the tree and the number of inversions on the tree. We find that both definitions for tree length are correlated with topological accuracy, but that the correlation is weakest for genomes on 37 genes (the mitochondrial genome case), especially when the dataset is close to saturation.
- A detailed study of the efficacy of using a simple lower bound on the inversion length of a candidate phylogeny. We observe that this simple bound can quickly eliminate close to 100% of the candidate trees when the evolutionary rates are sufficiently low.

- A successful analysis of the Campanulaceae dataset using a combination of these techniques, resulting in a *million-fold* speedup over previous approaches.

Our research combines the development of mathematical techniques with extensive experimental performance studies. We present a cross-section of the results of the experimental study we conducted to characterize and validate our approaches. We used a large variety of simulated datasets as well as several real datasets (chloroplast and mitochondrial genomes) and tested speed (in both sequential and parallel implementations), efficacy (for our new bounding technique), and accuracy (for reconstruction and distance estimation).

4.4 Basic material

4.4.1 Evolutionary Events

When each genome has the same set of genes and each gene appears exactly once, a genome can be described by an ordering (circular or linear) of these genes, each gene given with an orientation that is either positive (g_i) or negative ($-g_i$). Let G be the genome with signed ordering $g_1, g_2, \ldots, g_n$. An *inversion* between indices i and j, for $i \leq j$, produces the genome with linear ordering:

$$g_1, g_2, \ldots, g_{i-1}, -g_j, -g_{j-1}, \ldots, -g_i, g_{j+1}, \ldots, g_n.$$

A *transposition* on the (linear or circular) ordering G acts on three indices, i, j, k, with $i \leq j$ and $k \notin [i, j]$, picking up the interval $g_i, g_{i+1}, \ldots, g_j$ and inserting it immediately after g_k. Thus the genome G above (with the additional assumption of $k > j$) is replaced by:

$$g_1, \ldots, g_{i-1}, g_{j+1}, \ldots, g_k, g_i, g_{i+1}, \ldots, g_j, g_{k+1}, \ldots, g_n.$$

An *inverted transposition* is an inversion composed with a transposition. The *distance* between two gene orders is the minimum number of inversions, transpositions,

and inverted transpositions needed to transform one gene order into the other; when only one type of event is used in the model, we speak of inversion distance or transposition distance.

A dataset of genomes is said to be *saturated* if it contains a pair of genomes whose inversion distance is as large as the expected distance between two completely unrelated genomes. This saturation value depends upon the number of genes and is bounded from above by n, the maximum distance between any two genomes on n genes [74]. Reconstructing trees from saturated datasets is difficult because of the seeming randomness in the data—this is well understood for gene sequences [50, 51, 106], so it is no surprise that it applies to gene rearrangements as well.

Given two genomes G and G' on the same set of genes, a *breakpoint* in G is defined as an ordered pair of genes (g_i, g_j) such that g_i and g_j appear consecutively in that order in G, but neither (g_i, g_j) nor $(-g_j, -g_i)$ appear consecutively in that order in G'. The number of breakpoints in G relative to G' is the *breakpoint distance* between G and G' (and is symmetric).

4.4.2 The Nadeau-Taylor Model

The *Nadeau-Taylor* model [81] of genome evolution uses only genome rearrangement events. This results in all genomes having equal gene content. The model assumes that each of the three types of events obeys a Poisson distribution on each edge—with the three means for the three types of events in some fixed ratio.

4.4.3 Model Trees: Simulating Evolution

A model tree is a rooted binary tree in which each edge e has an associated non-negative real number, λ_e, denoting the expected number of events on e. The model tree also has a weight parameter, which defines the probability that a rearrangement event is an inversion, transposition, or inverted transposition. We used weights equal

to 1:0:0 (inversions only), 0:1:0 (transpositions only), and 1:1:1 (all three events are equally probable).

In our experimental studies, we used random leaf-labeled trees as the topologies and assigned uniform edge lengths to these trees. The simulator generates signed circular orderings of the genes as follows. The root is assigned the identity gene ordering $g_1, g_2, \ldots, g_k$. When traversing an edge e with expected number of events λ_e, three random numbers are generated according to the model weight: the first determines the actual number of inversions on that edge, the second that of transpositions, and the third that of inverted transpositions. Once the number of events of each type is determined, the order of these events is randomly selected, as are the indices on which these events operate. This process produces a set of circular, signed gene orders for each genome at the leaves of the model tree.

4.4.4 Labeling Internal Nodes

The approach proposed by Sankoff and Blanchette to derive ancestral genomes for the internal nodes of a tree is iterative, using a local optimization strategy. After initial labels have been assigned in some way, their procedure repeatedly traverses the tree, computing the breakpoint median of its three neighbors for each node, and using it as the new label if this change improves the overall breakpoint score. The median-of-three subproblems are transformed into instances of the Traveling Salesman Problem (TSP) and solved optimally. The overall procedure is a heuristic without any approximation guarantees, but does very well in practice on datasets with a small number of genomes.

`GRAPPA` uses the same overall iterative strategy and also solves the median-of-three problem in its TSP formulation to get a potential label for the internal nodes. `GRAPPA`, however, has the option of accepting a relabeling of an internal node based on either the breakpoint score (as in `BPAnalysis`) or on the inversion score of the

tree. In addition, GRAPPA can substitute approximate TSP solvers (greedy and variations of Lin-Kernighan [55]) for the exact one whenever the exact solver gets bogged down by a TSP instance.

4.4.5 Performance Criteria

Let T be a tree leaf-labeled by the set S. Deleting some edge e from T produces a bipartition π_e of G into two sets. Let T be the true tree and let T' be an estimate of T. Then the *false negatives* of T' with respect to T are those bipartitions that appear in T that do not appear in T'. This number is normalized by dividing by the number of non-trivial bipartitions of T. Note that, if this rate is 0, then T' equals T or refines it.

4.5 Better Analyses

4.5.1 Neighbor-Joining Performance

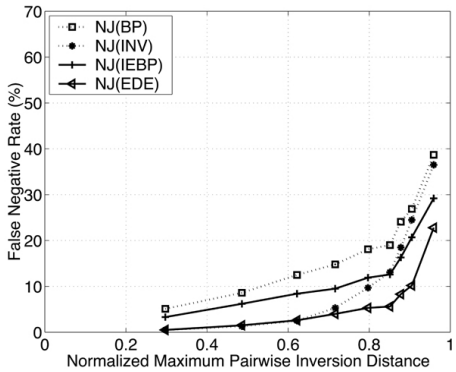
We conducted a simulation study to compare the performance of NJ using four different distances: BP, INV, and two corrected distances: IEBP [110], and EDE [111]. Figure 4.1 shows our findings. This figure shows false negative rates for a best case—inversion-only scenarios—and for a nearly worst-case—scenarios with both transpositions and inverted transpositions. Note that NJ with the corrected distance EDE is remarkably robust: even though it was engineered for inversions only, it handles datasets with a large number of transpositions and inverted transpositions almost as well. NJ with EDE can recover 90% of the edges even for the close-to-saturation datasets where the maximum pairwise inversion distance is close to 90% of the maximum value. NJ with EDE is even competitive with the more computationally intensive MPBE (Maximum Parsimony on Binary Encoding) method (data not shown).

4.5.2 Parsimony Improves Accuracy

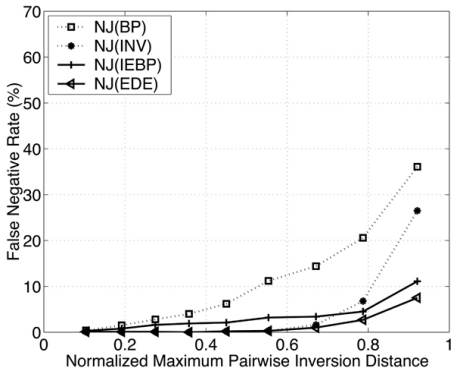
Since the main goal of phylogeny reconstruction is producing the correct tree topology, the parsimony approach we have taken needs to be evaluated in terms of the topological accuracy of the trees produced. We ran a large series of tests on model trees to test the hypothesis that reducing the total breakpoint or inversion length of trees would yield more topologically accurate trees.

We ran a total of 209 tests of NJ with inversion and breakpoint distances with at least 12 data points, on sets of up to 40 genomes. We used two genome sizes (37 and 120 genes, representative of mitochondrial and chloroplast genomes, respectively) and various ratios of inversions to transpositions and inverted transpositions, as well as various evolutionary rates. For each dataset, we computed the total distances and compared their values with the percentage of errors (measured as false negatives). We used the nonparametric Cox-Stuart test [26] for detecting trends—i.e., for testing whether reducing breakpoint or inversion distance consistently reduced topological errors. Using a 95% confidence level, we found that over 97% of the datasets with inversion distance and over 96% of those with breakpoint distance exhibited such a trend. Indeed, even at the 99.9% confidence level, over 82% of the datasets still exhibited such a trend.

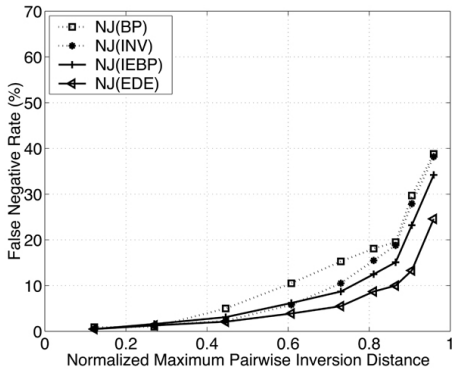
The false negative rates of the four distances are shown in Figures 4.1 and 4.2. Figure 4.2 shows the results of scoring the different NJ trees under the two optimization criteria: breakpoint score and inversion length. Only the inversion-only scenario is shown here, since other evolutionary settings produce similar behavior. In general, the relative ordering and trend of the curves agree with the curves of Figure 4.1, suggesting that decreasing the number of inversions or breakpoints should lead to an improvement in topological accuracy. This correlation is strongest for the 120-gene case and somewhat weaker for the 37-gene case. Finally, this trend still holds under the other evolutionary models (such as when only transpositions occur).



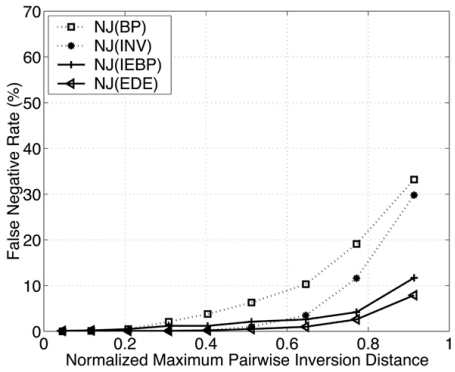
(a) inversion only, 37 genes



(b) inversion only, 120 genes

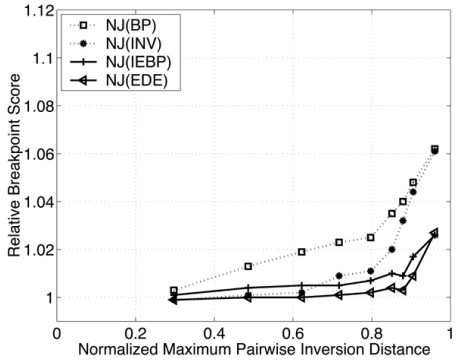


(c) three classes equally likely, 37 genes

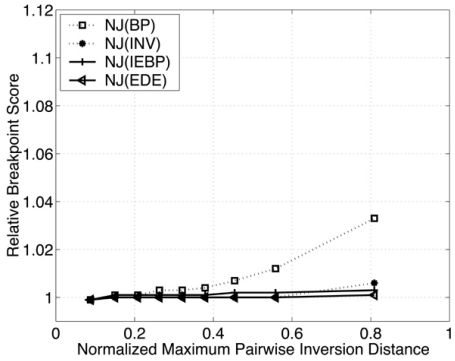


(d) three classes equally likely, 120 genes

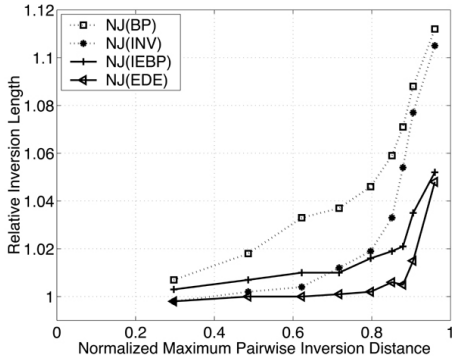
Figure 4.1: False negative rates of NJ methods under various distance estimators as a function of the maximum pairwise inversion distance, for 10, 20, 40, 80, and 160 genomes. Model weight settings are 1:0:0 (inversion only) and 1:1:1 (equally likely events).



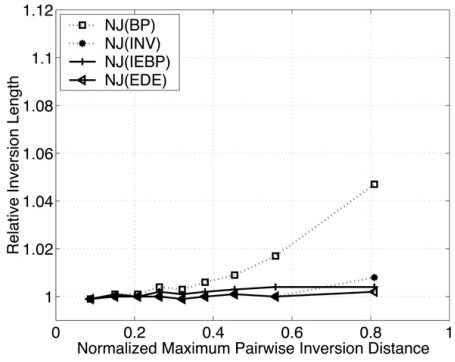
(a) breakpoint score, 37 genes



(b) breakpoint score, 120 genes



(c) inversion length, 37 genes



(d) inversion length, 120 genes

Figure 4.2: Scoring NJ methods under various distance estimators as a function of the maximum pairwise inversion distance for 10, 20, and 40 genomes. Plotted is the ratio of the NJ tree score to the model tree score (breakpoint or inversion) on an inversion-only model tree.

These experiments support the conjecture that improving the inversion length or breakpoint length should lead to improved topological accuracy, at least for the case of chloroplast genomes (which have many genes) and of mitochondrial genomes where the rates of evolution are sufficiently low to keep the dataset below saturation.

4.5.3 A Lower Bound Using Circular Orderings

The following theorem is well known:

Theorem 1 *Let d be a $n \times n$ matrix of pairwise distances between the taxa in a set S; let T be a tree leaf-labeled by the taxa in S; and let w be an edge-weighting on T, so that we have $w_{ij} = \sum_{e \in P_{ij}} w(e) \geq d_{ij}$. Set $w(T) = \sum_{e \in E(T)} w(e)$. If $1, 2, \ldots, n$ is a circular ordering of the leaves of T, under some planar embedding of T, then we have $2w(T) \geq d_{1,2} + d_{2,3} + \ldots + d_{n,1}$.*

And this corollary immediately follows:

Corollary 1 *Let d be the matrix of minimum inversion distances between every pair of genomes in a set S, let T be a fixed tree on S, and let $1, 2, \ldots, n$ be the circular ordering of leaves in T. Then the inversion length of T is at least $\frac{1}{2}(d_{1,2} + d_{2,3} + \ldots + d_{n,1})$.*

(This corollary forms the basis of the old "twice around the tree" heuristic for the TSP based on minimum spanning trees [45].)

We use these bounds to help search tree space in the obvious way. First, we obtain a good upper bound on the minimum achievable inversion length by using a polynomial-time technique (NJ with EDE distances [111]); we update this upper bound every time the search finds a better tree. For each tree, we quickly compute the circular lower bound of Corollary 1. If that lower bound exceeds the upper bound, the tree can be discarded without being scored. Since scoring a tree involves solving numerous TSP instances, such bounding can dramatically reduce the running time.

r value	$r = 2$					$r = 4$				$r = 8$			
# genomes	10	20	40	80	160	10	20	40	80	10	20	40	80
# genes													
10	0	0	0	1	1	0	0	0	1	0	0	0	1
20	0	80	91	1	1	0	0	0	0	0	0	0	0
40	91	100	100	100	100	0	0	0	0	0	0	0	0
80	99	100	100	100	100	65	72	100	0	0	0	0	0
160	100	100	100	100	100	98	100	100	100	0	0	0	0
320	100	100	100	100	100	99	100	100	100	71	90	100	0

Table 4.1: Percentage of trees eliminated through bounding.

4.5.4 The Lower Bound In Practice

We ran three different experiments to quantify various aspects of our bounding techniques. Our first experiment measures the percentage of trees that are pruned through bounding (and thus not scored) as a function of the three model parameters: number of genomes, number of genes, and number of inversions per edge. We used an inversion-only scenario, as well as one with approximately half inversions and half transpositions or inverted transpositions. Our data consisted of two collections of 10 datasets each for a combination of parameters. The numbers of genomes were 10, 20, 40, 80, and 160, the numbers of genes were 10, 20, 40, 80, 160, and 320, and the rate of evolution varied from 2 to 8 events per tree edge, for a total of 90 parameter combinations and thus 1,800 datasets. For each dataset, we generated 1,000 random circular orderings, scored their pairwise circular order inversion distance, and compared these scores to the upper bound. Table 4.1 shows the percentage of trees pruned away by the circular lower bound. The lower bound is surprisingly effective for certain datasets, but for many others would not eliminate any trees. For low rates of evolution and datasets with up to 160 genomes, we found that most random circular orderings were eliminated—a very encouraging result for chloroplast genomes (which can contain several hundred genes) and quite sufficient for the analysis of mitochondrial datasets (where the number of genes is 37), since

many have low evolutionary rates. However, note that at higher rates of evolution, the bound is not effective until the number of genes gets into the range of 80 to 160, while the bound is simply ineffective at truly high rates of evolution.

Our second experiment used our real dataset of 13 chloroplast genomes, 12 from the family Campanulaceae and with *Nicotiana* as an outgroup. Each of the 13 genomes has 105 gene segments and, though highly rearranged, has what we consider to be a low rate of evolution. We ran `GRAPPA` on this dataset both with and without the lower bound and computed the percentage of trees eliminated using the bound. After running for 12 hours (on a 300MHz Pentium II workstation) and processing well over 50 million trees, the code using bounding had eliminated 85% of the trees from further computation.

Because computing the circular bound entails its own cost (linear in the number of genomes), we were interested in what kind of running time speedup `GRAPPA` would gain through this bounding technique. Our third experiment ran our code on the Campanulaceae dataset for 12 hours each with and without bounding: the version with bounding processed nearly 10 times as many trees. Thus the speedup over `BPAnalysis` reported in [79] is now increased by another factor of 5–10, to a value of over 5,000.

The speedup obtained by bounding depends upon two factors: the percentage of trees that can be eliminated by the bounding and the difficulty of the TSP instances avoided by using the bounds. As Table 4.1 shows, when the rate of evolution is not too high, close to 100% of the trees can be eliminated by using the bounds. However, the TSP instances solved in `GRAPPA` can be quite small when the evolutionary rate is low, due to how we compress data (as described in [79]). Consequently, the speedup will also depend upon the rate of evolution, with lower rates of evolution producing easier TSP instances and thus smaller speedups. The Campanulaceae dataset is a good example of a dataset that is quite easy for `GRAPPA`,

in the sense that it produces easy TSP instances—but even in this case, a significant speedup results. More generally, the speedup increases with larger numbers of genomes and, to a point, with higher rates of evolution. When one is forced to exhaustively search tree space, these speedups represent substantial savings in time.

Our last experiment used a combined heuristic. We analyzed the Campanulaceae dataset using NJ and MPBE and then took the strict consensus tree (the maximally resolved tree that is a common contraction) of the 8 trees returned by these procedures. We gave this tree as a constraint tree to `GRAPPA`; this makes `GRAPPA` search the space of all refinements of the constraint tree for the minimum inversion tree. The search space contained only 10,395 trees, which we can run to completion in much less than a minute (though not because of the bounding technique, since it did not eliminate any tree, an expected occurrence when all trees examined have a good topology). The search returned 216 optimal trees, with an inversion score of 67 and a breakpoint score of 84. Since earlier attempts to analyze this dataset found only four trees with an inversion score of 67 [28], this represents a significant advance.

4.6 High-Performance Computing

Even the best algorithms for phylogeny reconstruction are likely to take exponential time in many cases, so that we should take advantage of high-performance tools whenever possible. We used the best precepts of algorithm engineering [76] to improve the running time of our `GRAPPA` software, eventually achieving a 2,000-fold speedup, as reported in [79]. More recently, we parallelized our software (an easy task, since it offers "embarrassing parallelism") and used the 512-processor Los Lobos supercluster at the University of New Mexico to run a complete analysis of the Campanulaceae dataset discussed in [28]. This analysis took only 1.5 hours instead of the several centuries estimated in [28], for a million-fold speedup [5].

We expect to effect similar speedups (by several orders of magnitude) in a future reimplementation of parsimony searches (both local, using TBR techniques, and global, using branch-and-bound searches), based on the same principles of high-performance algorithm engineering and parallel algorithm development. Although even a million-fold speedup will allow us to increase the number of taxa by only a few when using an exponential-time algorithm, the same speedup applied to a polynomial-time algorithm will represent the difference between solving a problem today or waiting a few generations. We have also produced the first ever linear parallel speedups for complex combinatorial problems [4], using shared-memory machines (SMPs). Branch-and-bound falls in this category of problems, so that we can now expect to see respectable parallel speedups in parsimony searches and other related optimization problems when using our newly developed parallel techniques [77].

4.7 Conclusions and Future Work

We have described new theoretical and experimental results that have enabled us to analyze significant datasets in terms of inversion events and that also extend to models incorporating transpositions. The work described here is part of an ongoing project to develop fast and robust techniques for reconstructing phylogenies from gene order data. Our current software suffers from limitations that we need to address; most limiting is the fact that it explicitly searches all of (constrained) tree space. However, the bounds we have described can be used in conjunction with branch-and-bound (based upon either inserting leaves into subtrees or extending circular orderings), as well as in heuristic techniques for searching through tree space. Our immediate work will implement these extensions to the software. In the long term, we plan to extend the techniques to solving the IT (inversion plus transposition) phylogeny problem, enable analysis of genomes with unequal gene sets, and handle multiple chromosomes.

Chapter 5

Identifying Transfer RNAs

5.1 Introduction

Genome annotation is a critical aspect of whole genome analysis. With the advent of high throughput sequencing, we are presented with a wealth of whole genomes which must be analyzed, but methods are out-of-date and do not scale to the problems at hand. Annotating whole genomes involves identification of protein coding genes (for which excellent methods exist based on search by sequence similarity [2, 88]), as well as ribosomal RNAs (rRNAs) and transfer RNAs (tRNAs). Here, we evaluate the effectiveness of existing methods for identifying animal mitochondrial (mt) tRNAs.

In animal mitochondrial genomes, tRNAs make up 22 of the 37 genes and yet no program exists which can automate the identification process. Methods developed for protein coding genes (which are based on conservation of nucleotide sequence) are not suitable for animal mt tRNAs (or most other tRNAs) because selection operates on functional tRNAs based on maintenance of base-pairing (secondary) structure rather than conservation of nucleotide sequence. Transfer RNAs sharing the same function may appear unrelated based on primary sequence similarity, but the close relationship becomes apparent once their secondary structure is

known.

Many general-purpose programs have been developed for identifying RNA molecules [37, 40, 68, 87] but they often focus on features to accommodate identification of general RNA molecules (like pseudoknots, which create complex secondary structures) or are based on a combination of secondary structure and primary sequence searching techniques. Because animal mitochondrial tRNAs have almost no conservation of sequence at the nucleotide level, methods must focus on covariation of basepairing in the secondary structure.

Here we present an analysis of the performance of four existing methods in identifying animal mt tRNAs as well as an in-depth exploration of ways to improve the best-performing method. The methods were rigorously tested by running each program on the 300 complete animal mt genomes in GenBank [41] containing 6,600 tRNAs. The programs which we tested were: COVE [34], tRNAscan-SE [68], RNAMotif [69], and our own method. Eddy and Durbin's COVE software is based on probabilistic covariation models constructed from aligned sequences. This method has a solid theoretical framework and is quite tractable for the small size of animal mt genomes (approx. 15000 nucleotides). Lowe and Eddy's tRNAscan-SE, probably the most popular program which exists for identifying tRNAs, is a hierarchical method based on a combination of three methods. Macke *et al.*'s RNAMotif (a descendent of RNAMOT) uses descriptors of structural motifs, but differs from RNAMOT in its more powerful descriptor language for capturing secondary structure information as well as its new global scoring mechanism. Finally, we tested our own method which implements a custom animal mitochondrial tRNA scoring scheme integrated into a structural motif-based search algorithm.

5.2 Biological Background

Transfer RNAs (see Söll and RajBhandary [103]) are approximately 70 nucleotides in length and are necessary components to a cell's protein synthesis machinery. They fold into a complex shape, including both single-stranded regions and helices based on internal nucleotide pairings. This can be represented in schematic form as a cloverleaf with four stems. These parts are illustrated in Figure 5.1.

Each tRNA is enzymatically charged with one particular amino acid according to features internal to the tRNA. The amino acid is chemically linked to the discriminator nucleotide. The tRNA then delivers this amino acid to the growing peptide chain on the ribosome. The order of entry of tRNAs into the ribosome is specified by the messenger RNA (mRNA). The mRNA is a chain of nucleotides (generally hundreds or thousands of nucleotides in length) that threads through the ribosome by triplets, with each triplet (i.e. "codon") binding to the three complementary nucleotides at the base of the tRNA (the anticodon; in the case of Figure 5.1 these are `TAC`). Thus, the order of nucleotides in the mRNA specifies the sequencing of tRNAs into the ribosome and, consequently, the order of amino acids in the growing peptide chain.

These tRNAs are encoded by genes, which are commonly identified by the potential of their sequences to form these cloverleaf-like structures and by certain well-conserved nucleotide positions. This can be challenging to do reliably, and attempts fail both by missing tRNA genes that are poorly conserved or aberrantly structured as well as by generating false positives. These problems are especially acute for the tRNAs that are encoded by animal mitochondrial genomes, which are especially variable both in sequence and structure. Mitochondrial tRNAs are also occasionally missing some paired arms or are otherwise varying in structure (see, for example, Wolstenholme *et al.* 1987 [114]).

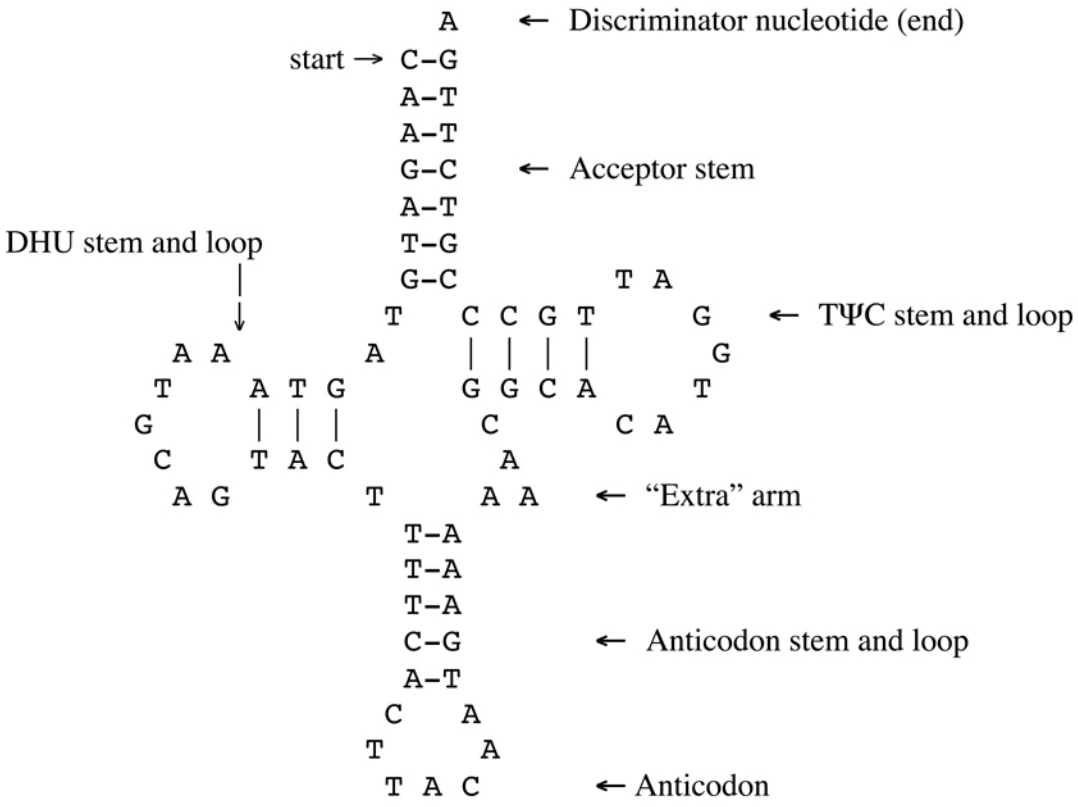


Figure 5.1: Schematic representation of a typical tRNA encoded by an animal mitochondrial genome. Nucleotides paired by hydrogen bonds are indicated by dashes. The tRNA is folded from a single string of ribonucleotides starting with "CAAGATG", reading counterclockwise around the structure ("TAGTAAATGCAGTACTTTTC ACTTACAATGAAAAAACGGCACATGGATTGCC") and ending with "CGTCTTGA".

5.3 Methods

Methods for identifying tRNAs typically fall into three basic categories: covariation analysis based on a generalization of hidden markov models (HMMs) [34, 96], motif-finding algorithms which include structural as well as nucleotide sequence motifs [35, 37, 40, 60], and minimum-energy based algorithms [119]. We tested four of these programs for identifying tRNAs in animal mt genomes. The four programs are COVE, tRNAscan-SE, RNAMotif, and our own method and are discussed briefly below.

5.3.1 COVE

Identifying tRNAs using HMMs was first proposed in 1994 simultaneously by Eddy and Durbin [34] and Sakakibarra *et al.* [96] and was implemented by Eddy and Durbin in the COVE software package. A covariation model (CM) is created in COVE by training the model with a set of sequences and adjusting the parameters and structure of the model so that high probabilities are assigned to the training sequences. The set of sequences may be previously aligned or not. This is well-suited to the animal mt tRNA problem since we are not required (and, in fact, would not be able to) align the training sequences based on primary structure. Once the model has been created, a candidate RNA sequence is aligned to the CM using a three-dimensional dynamic programming algorithm, and the score is then calculated based on the probability of the alignment.

Although this method is one which initially showed promise and was very accurate, it was prohibitively time-consuming to run on large nuclear genomes. Lowe [68] estimated that searching the human genome with a tRNA covariation model would take about nine and a half CPU years. However, animal mt genomes, which are typically about 15,000 bp long, are a very reasonable size and indeed, COVE typically took about two minutes to identify all the tRNAs in an entire

animal mt genome.

5.3.2 tRNAscan-SE

tRNAscan-SE is primarily targeted towards non-organellar tRNAs. It uses two methods as a prepass for identifying candidate sequences based on sequence content and then passes the candidates to COVE as a subroutine. For organellar tRNAs, it bypasses the prepass step (because the sequences that they search for don't exist in organellar genomes) and gives the sequences directly to the covariation model. This process, without the preprocessing, is equivalent to running the COVE program on the tRNA CM included in the software package.

5.3.3 RNAMotif

RNAMotif is a descendent of past motif-based programs [8, 40, 60], but has a more powerful descriptor language than its predecessor, RNAMOT [40], and it has a global scoring scheme. The user creates a descriptor for a molecule based on stem and loop motifs, including any specific information regarding nucleotide values or numbers of mismatches in the stem. The descriptor also includes a scoring section in which the user can query assignments made to the motifs by the search and from this, can compute a score. RNAMotif first creates a tree representation from the descriptor file. It then does a depth first search through the tree, trying to match the input sequence. The successful candidates are then passed to the scoring routine, where they are evaluated and optionally accepted or rejected based on rules in the score section.

5.3.4 Our Method

Our method was implemented as part of `DOGMA` [117] a whole genome annotation package for organellar genomes. It is a pattern-matching algorithm which combines

structural motif searching with an integrated scoring system designed specifically for animal mt tRNAs. The scoring system is based on an empirical method for identifying tRNAs in animal mt sequence data. The method's search relies almost exclusively on secondary structure for identification. The pseudo-code for our method is shown in Figure 5.2 It searches one-by-one for each tRNA's anticodons so that the user is given the best-scoring candidates for *each* tRNA. This way, none of the tRNAs are missed entirely and the best-scoring possibilities are returned to the user. The program first identifies the anticodon arm (see Figure 5.1) and then searches to either side for a basepairing arm of length seven which is the acceptor stem, awarding points for each positive pattern that is matched. The remaining sequence between the anticodon and acceptor stems is then folded to maximize the score. The best-scoring candidates are then saved and reported to the user.

```
For each strand (forward and reverse)
    For each amino acid A
        For each occurrence of A in the genome
            If there exists a acceptor stem that is
              -within 29 nt to left and 32 nt to the right
               of anticodon arm stem
              -of length 7
              -with 5 or more bases paired
            And the anticodon arm stem is has 3 or more bases paired
                    Fold and score D arm
                    Fold and score T arm
                    Fold and score extra arm
Report top-scoring A for each tRNA
```

Figure 5.2: Pseudo-code for our tRNA algorithm. Scoring rules are shown in Table 5.3.4

Description	Points
Pairing bases in stem	+2
All bases in stem are pairing	+2
All bases in anticodon arm stem are pairing	+12
All bases in acceptor stem are pairing	+16
First bases in stem are not pairing	-1
Single nt closest to acceptor stem is C or T	+1
Single nt second closest to acceptor stem is A or G	+1
Single nt between D arm and anticodon arm stem is A or G	+1
D arm stem length greater than 7	-3
T arm stem length greater than 8	-3
D arm stem length greater than 10	-6
D arm stem length greater than 11	-6
D and T arm stem length between 2 and 6	+2
D arm loop length less than 15	+2
T arm loop length less than 15	+2
T arm loop length less than 10	+2
Extra arm is length 4	+4
Extra arm is length 5	+2
Extra arm is length 3	+2
Extra arm is length 2	-1
Extra arm is length 1	-1
Extra arm is length 0	-1
Extra arm is length 6	-2
Extra arm is length 7	-2
Extra arm is length 8	-2
Extra arm is length 9	-2
Consecutive non-pairing bases in stem	reject
Stem is length 4 with 2 or more non-pairing bases	reject
Stem length is greater than 4 with 3 or more non-pairing bases	reject

Table 5.1: Animal mitochondrial tRNA scoring rules.

5.4 Experimental Setup

Our dataset consists of 300 complete animal mitochondrial genomes from GenBank with their accompanying annotation of 6,600 tRNAs. This is the complete list of all animal mitochondrial genomes in GenBank at the time of this writing.

In testing the COVE method, a covariation model specifically trained to identify animal mitochondrial tRNAs was created. The model was trained with 1,432 tRNAs from 65 complete animal mt genomes taken from the set of 300 genomes. COVE was then run on the datasets with and without the training sequences. The results reported here are on the whole dataset (including the genomes used to train the model) so that they can be more easily compared to the other methods which are tested on the whole dataset. For tRNAscan-SE, we used the tRNA CM that the program came with. This was trained on a dataset of 1415 aligned tRNAs from the 1993 Sprinzl database [104]. Although this model was not trained on organellar tRNAs, when a user queries the tRNAscan-SE web site with a mitochondrial sequence, this is the model which is used. For RNAMotif, we created a descriptor file for animal mt tRNAs and went through several iterations of testing and modifying the descriptor until we were convinced it was performing as well as it could. We then ran the output through the pruning routine of RNAMotif which is meant to prune out subsets of solutions.

After determining how well COVE performed with our CM trained on animal mt tRNAs, we decided to explore different techniques for training the CM to see if it could be further improved upon. The methods we tried and the results are discussed in Section 5.7.

5.5 Evaluation

Each program was tested on the 300 genome dataset. Each genome has 22 tRNAs (one for each amino acid and two for both serine (tRNA-Ser) and leucine (tRNA-Leu)) to be identified. The methods were evaluated based on false negatives (FN) and false positives (FP). A false negative occurs when a program fails to identify an actual tRNA, and a false positive occurs when a program identifies a sequence as a tRNA when it, in fact, is not. The false negatives for each genome were counted and plotted in Figure 5.3. It shows, for each of the four methods, for each number of false negatives, how many genomes missed that many tRNAs. The false negatives for each of the 22 tRNAs were counted individually and are presented in Figure 5.4 with the FN for the two tRNA-Ser and tRNA-Leu combined. This figure shows for each tRNA, how many genomes missed it for each method.

5.6 Results and Discussion

As can be seen in Figure 5.3, the best-performing method was, by far, COVE. In the figure, the COVE results are for all 300 genomes, including the training set. (Even without the subset of genomes which the model was trained on, it performed very well) COVE found all 22 of the tRNAs for 215 out of 300 genomes. This is compared to just 11 for tRNAscan-SE, 44 for our method and *none* for RNAMotif. Even as the best performer, however, the CM trained for animal mitochondrial tRNAs still missed some tRNAs, in the worst case, missing 8 of the tRNAs. Figure 5.4 shows that for COVE, unlike the other methods, one can't say that it performed especially poorly on particular tRNAs or for particular genomes except that there is a spike on the FN for COVE on serine. The FN for both serines were combined, but the FN is still high. However, it does appear that COVE is much less sensitive to degenerate cases of animal mt tRNA secondary structure than the other methods.

tRNAscan-SE is probably *the* most popular method for identifying tRNAs. Although very successful for non-organellar genomes [68], the tRNAscan-SE CM, which had been trained on non-organellar genomes, does not perform nearly as well on animal mitochondrial tRNAs. As is illustrated in the results in Figure 5.3, a CM needs to be appropriately trained for its target molecule. tRNAscan-SE only found all 22 of the tRNAs in 11 genomes (compared to 215 for COVE) and even missed *every* tRNA for one of the genomes. This can also be seen in Figure 5.4 where tRNAscan-SE missed tRNA-Ser (which is often missing the D arm) for most of the genomes. This illustrates how important it is to use care when using probabilistic methods which require training. They can be extremely rigorous and accurate when properly trained (as in COVE, above).

The least successful of the methods was RNAMotif, with over half of the genomes missing more than 25% of the tRNAs and not finding all of the tRNAs in any of the genomes. One of the problems with assessing the performance of RNAMotif is the volume of output. This is meant to be a feature – that RNAMotif will not overlook any potential RNA molecule which fits the descriptor, but it turns out to not be appropriate for animal mt tRNAs. The descriptor for mitochondrial tRNAs must be very general because individual nucleotides are not constrained, but this also allows for a huge number of matches. Even when the output is run through the pruning routine that comes with RNAMotif, the deluge of output is more than even the most diligent user could wade through. For the graph in Figure 5.3, if a candidate with the correct coordinates was found in the top 20 answers for each tRNA, it was counted. This is a very generous way of counting (one could not have much confidence in a method where the correct tRNA is ranked eighteenth in a list) and yet it still did not perform very well. One of the drawbacks to this method is its "all or nothing" approach. The descriptor language does have some nice regular expression types of features, but it does not allow for boolean expressions. As an

example, one would want to allow for a missing D arm in a tRNA by matching a stem and loop strongly or not at all. We suspect this is also the reason that RNAMotif is the method most sensitive to missing particular tRNAs. In Figure 5.4, it appears that there are a group of tRNAs for which RNAMotif performs exceptionally poorly.

Our own method found all 22 of the tRNAs for 44 of the genomes, and usually only missed 1 or 2 tRNAs. When our method identified the correct tRNA, it was usually the top scoring tRNA or within the top 3 for each tRNA. A feature of our method is that it selects the top-scoring candidates for each tRNA and presents them to the user. The results from our method also illustrate how difficult it can be to develop a system for recognizing a set of tRNAs with such diverse secondary structure and so many exceptions to the canonical tRNA structure. However, our scoring scheme is very good at constructing putative secondary structures and can be used as a postprocessing routine for candidate regions to fold and score potential tRNAs.

With respect to false positives, the COVE program had very few and while this is indeed a feature, it also doesn't present the user with "second choices" if the folding is not to their satisfaction. The number of false positives for our method is not reported because the user can choose how many of the best-scoring candidates for each tRNA should be returned. The number of false positives reported by RNAMotif is so large as to make the method impractical, sometimes giving hundreds of false positives per tRNA.

5.7 Further Training of Covariation Models

Encouraged by the strong performance of COVE, we attempted to improve upon the method by trying different approaches to training the model. Despite coming up with several strategies for training, we found that none of the new CMs that we trained performed better than the original CM we trained with 1432 animal mt

tRNAs.

The first strategy was to increase the number of training sequences in the training set. For the second CM, we trained the model with 3300 sequences (half of the tRNAs in the dataset) and found that it did not outperform our first CM. The 3300 CM found all of the tRNAs in 221 genomes versus 215 in the 1432 CM but this was not uniform over different FN values. For individual tRNAs, the 3300 CM only missed 8 tRNA-Pro versus 15 in the 1432 CM but it missed 20 tRNA-Cys when the 1432 CM only missed 14.

We then tried to construct a CM which would do a better job identifying tRNA-Ser (usually one of the two tRNA-Ser is missing an arm). Using just the tRNA-Ser sequences, we tried using all of the sequences for training as well as for the test set, as well as trying a third as the test set (200 sequences), and finally tried using the maximum likelihood model-making strategy option (which was *very* slow to train) did not improve the results either. The original model which we trained had a total of 41 false negatives, but the CM trained with all the tRNA-Ser sequences had 52 false negatives. The model trained on 200 tRNA-Ser sequences missed 49 and the maximum likelihood trained model missed 64. We then tried targeting each tRNA family individually. We created a CM for each individual tRNA family using one third of the tRNAs (100) as the training set and then testing on the genomes which weren't part of the training set. Just as with the CMs targeted for tRNA-Ser, there was not a discernible improvement in performance.

Our next idea was to explore the taxonomic sampling of the dataset to see if there might be some kind of bias towards particular genomes. If most of the animal mitochondrial genomes in GenBank were from the same taxonomic group, then the model trained with the sequences would be biased toward identifying tRNAs in these genomes. In examining the genomes which had the most false positives, we found that many were actually within the same class. Three were in the class

Actinopterygii (ray-finned fishes) which had 114 complete genomes—more than any other group in the whole set of genomes. This indicated that a bias towards certain genomes in training was not cause of the false negatives. Rather than having a CM which was more evenly sampled, we decided to construct one that was trained specifically towards the genomes which had high numbers of false positives. We trained a CM with 2508 tRNAs from the class Actinopterygii and then tested it on all 114 genomes in that class, but to our disappointment, there was no improvement in the number of false negatives.

5.8 Conclusions and Future Directions

Here we have presented an analysis of existing tRNA identification methods and evaluated their performance with respect to animal mt genomes. We have shown that COVE is the most effective and promising method and why other methods are not successful at identifying animal mt tRNAs. COVE is also the most robust method with respect to identifying non-canonical foldings. Future work will include trying a combination of our method with COVE and modification of COVE (since different training techniques did not prove fruitful).

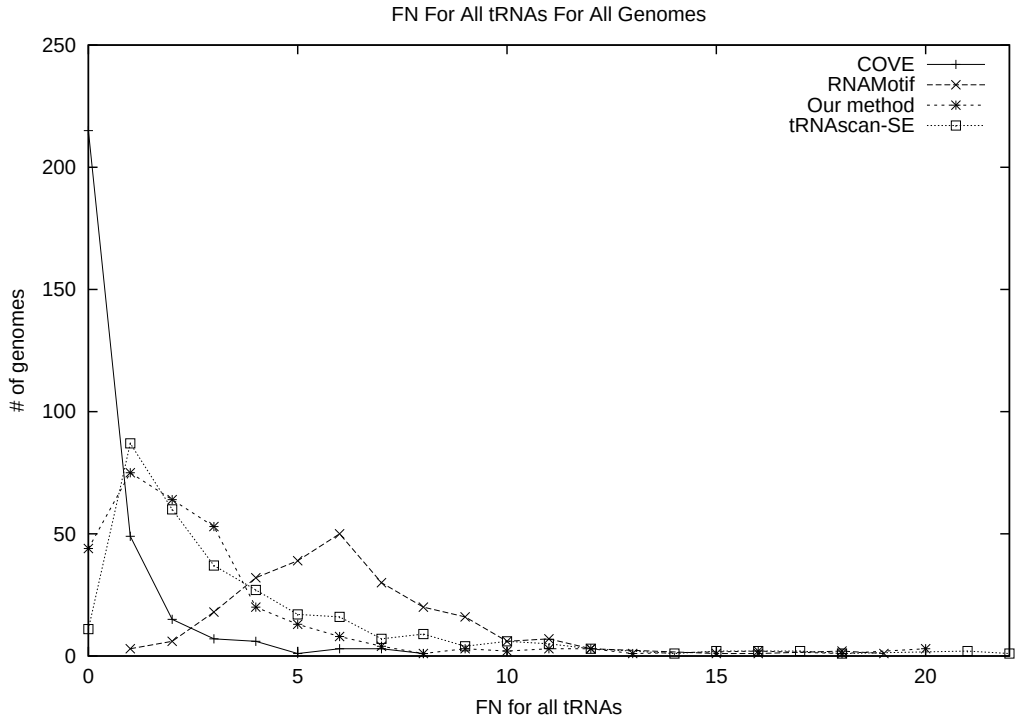


Figure 5.3: The number of false negatives for the four methods. For each number of missed tRNAs (x-axis), the number of genomes with that number of false negatives is plotted.

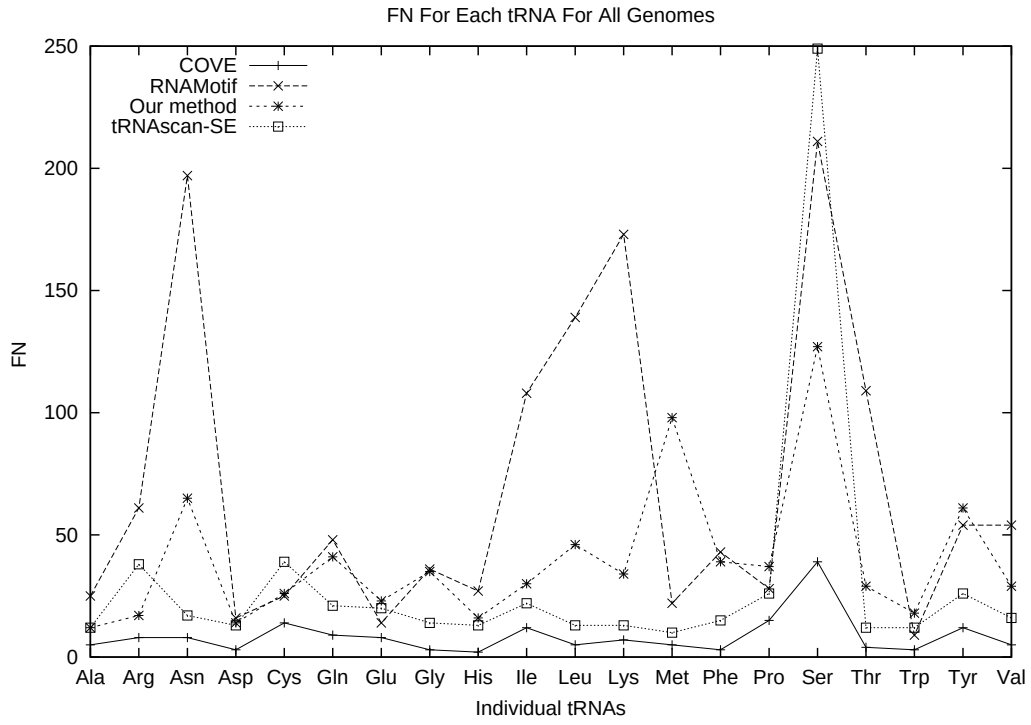


Figure 5.4: The number of false negatives for each tRNA for the four methods. For each tRNA, the number of genomes that missed that tRNA is plotted.

Chapter 6

Organellar Genome Annotation

6.1 Introduction

The Dual Organellar GenoMe Annotator (DOGMA) automates the annotation of organellar (plant chloroplast and animal mitochondrial) genomes. It is a web-based package that allows the use of BLAST [2] searches against a custom database, and conservation of basepairing in the secondary structure of animal mitochondrial tRNAs to identify and annotate genes. DOGMA provides a graphical user interface for viewing and editing annotations. Annotations are stored on our password-protected server to enable repeated sessions of working on the same genome. Finished annotations can be extracted for direct submission to GenBank.

The comparison of complete mt genome sequences is becoming increasingly important for reconstructing the evolutionary relationships of organisms [17, 21, 71, 75], for studying population structure and history [92], including those of humans [52], for identifying forensic materials [86], and for understanding the inheritance of certain human diseases [109]. In the past, annotating organellar genomes has been a time-consuming and error-fraught process and, with the input of high-throughput genome sequencing centers, has been the rate-limiting step in the pro-

duction of complete chloroplast and mitochondrial genome sequences. DOGMA is the first tool which automates this process.

DOGMA is a web-based annotation package. It takes as input a file containing the complete nucleotide sequence of an animal mitochondrial or plant chloroplast genome in FASTA format. For protein coding genes, the genome is translated in all six reading frames and queried against our custom amino acid sequence databases using BLAST. BLAST (or Basic Local Alignment Search Tool) is program for searching with a query sequence against a database of sequences. (BLASTN is a nucleotide query search against a nucleotide sequence database, and BLASTX is a nucleotide query search against an amino acid sequence database.) Ribosomal RNAs (rRNAs) and transfer RNAs (tRNAs) are queried against a nucleotide sequence database. The databases were constructed from a set of annotated animal mitochondrial and green plant chloroplast genomes. DOGMA constructs a list of genes from the BLAST output, and graphically displays the list of genes to the user for annotation. When a gene is selected, a detailed view of the gene's nucleotide and amino acid sequence and BLAST hits is displayed. Because the putative genes are located using sequence similarity with genes in other genomes, and because BLAST is not explicitly looking for a start and stop codon for the protein coding genes, the start and stop codons must be chosen by the user. The start and end positions for each tRNA and rRNA also must be verified. Annotations are stored on our password-protected server so they can be retrieved and edited. When complete, the annotation may be retrieved in a format suitable for direct submission to GenBank. DOGMA also allows the user to extract sub-sequences of the genome (including intergenic regions, introns, amino acid sequences of protein coding genes, etc.) for further analysis.

6.2 DOGMA Databases

Chloroplasts and mitochondria typically have circular, double-stranded chromosomes that vary little in gene content. This similarity in gene content allows us to use a comparative approach for annotations. Genes which have been previously identified in closely-related taxa can be used to identify those genes in newly sequenced genomes.

6.2.1 Animal Mitochondrial Genomes

Animal mitochondrial genomes typically are about 15,000 basepairs (bp) in length and contain 37 genes: 13 protein coding genes, 22 tRNA and 2 rRNA genes [15]. Gene content is mostly fixed, though the gene order can be highly rearranged. Duplications or deletions of genes are rare, most genes do not overlap (though there are some well-identified exceptions), and genes do not contain introns (except for some cnidarians). The animal mitochondrial genome databases were compiled from the 367 annotated genomes in GenBank. Each database contains the amino acid sequence for a specific gene from each of the genomes in which it appears. There is a database for each of the 13 protein coding genes, plus one for each of the two rRNA genes.

6.2.2 Chloroplast Genomes

Chloroplast genomes, on the other hand, are usually about 150,000 bp (but can be as long as 220,000 bp) and contain 110-130 genes. There are 4 ribosomal RNA genes, about 30 transfer RNAs and about 80 protein coding genes. Introns are infrequent in chloroplast genomes, occurring in 20 genes in *Nicotiana*, which makes identifying genes much more straightforward. Chloroplast genomes contain 4 distinct regions. Two of the regions (IRA and IRB) are identical inverted repeats. The other two regions are the large and small single-copy regions. In general, gene

content and order are highly conserved [84], although in some groups numerous structural rearrangements have been identified [30]. There are currently 26 complete plant chloroplast genomes in GenBank and 18 of these are green plants. For chloroplast genomes, we created databases for genes from 16 complete genomes of green plants. They include *Adiantum*, *Arabidopsis*, *Epifagus*, *Lotus*, *Marchantia*, *Mesostigma*, *Nephroselmis*, *Nicotiana*, *Oenothera*, *Oryza*, *Pinus*, *Psilotum*, *Spinacia*, *Triticum*, and *Zea*. Database files were created for 98 chloroplast protein coding genes (with two entries for the each of the trans-spliced pieces of rps12). We did not include open reading frames (ORFs) in the database, however, hypothetical chloroplast reading frames (ycfs) were included. There are 4 rRNA nucleotide sequence databases and 35 tRNA nucleotide sequence databases.

6.2.3 Gene Nomenclature

GenBank is fraught with errors in annotation. These errors include typos, incorrect sequences and gene names, and inconsistencies is naming conventions. We have endeavored to clarify this issue by correcting and standardizing all gene names for both plant chloroplast and animal mitochondrial genomes. The naming conventions follow those set forth by Martin *et al.* [71] for plastid genes and used by Boore [16] for mitochondrial genes. In each case, these are generally the same as the gene names established for their bacterial homologs.

6.3 Identifying Genes

6.3.1 Protein Coding Genes

Protein coding genes are identified in the input genome based upon conservation of sequence similarity in genes in other genomes in the database. The input nucleotide sequence is queried in all 6 reading frames against the amino acid sequence database

for each gene using BLASTX. Various BLAST parameters (such as e-value and number of hits returned) may be set by the user. Once DOGMA has identified the putative protein coding genes, the user then selects start and stop codons for each gene. The program displays to the user the nucleotide sequence for the gene from the input genome with the translation to amino acids, along with the amino acid sequences from the BLAST hits. For genes containing introns, DOGMA will identify exon boundaries based upon the BLAST hit boundaries which must then be verified by the user. When a gene contains an intron, the exons are the pieces of coding sequence which make up the gene surrounding the intron. This has proven to work quite well; however, genes with very small exons (two or three amino acids) will be missed by BLAST and they must be located by hand. These exons occur in three well-documented genes in chloroplast genomes (petB, petD, and rpl16), and the user will know to look for them when there is no start or stop codon for one of these three genes.

6.3.2 Identifying tRNAs

In chloroplast genomes, the nucleotide sequences for tRNAs are highly conserved, and we have found that sequence similarity is sufficient for their detection. Databases of nucleotide sequences for chloroplast tRNAs are used for searching with BLASTN. DOGMA identifies the anticodon for each tRNA based on the database entry.

Transfer RNAs diverge rapidly in sequence in animal mitochondrial genomes and therefore, sequence similarity is not a sufficient criterion to locate the genes. They must be identified based on conservation of basepairing in the cloverleaf-shaped secondary structure. This is a difficult task, and we have found that methods based on hidden Markov models can do well [116] and DOGMA uses the COVE [34] program. COVE identifies candidate sequences of the tRNA genes based on secondary structure, but does not give a putative folding, and so DOGMA then uses a custom

program to infer the stem and loop folding of the secondary structure.

6.3.3 Identifying rRNAs

Ribosomal RNAs can be detected through BLAST searches for sequence similarity for both mitochondrial and chloroplast genomes. For mitochondrial genomes, the BLAST parameters (such as gap penalty or percent identity) must be optimized since some portions of the rRNA genes can be highly diverged.

6.4 Web-based Display and Editing Tool

DOGMA is a web-based display and editing tool. On first use, a researcher creates a userid and password which keeps their (perhaps unpublished) data private from other users of the software. Users may also save and retrieve existing annotations.

The tool consists of three panels (Figure 6.1). The main (middle) panel displays all the putative genes, and genes are color coded by strand and gene type and labeled with the gene name. When a gene in the middle panel is selected, details for annotating that gene appear in the top panel and a new window appears for recording the annotation information for that gene for input to Sequin (NCBI's software for submitting annotations to GenBank). DOGMA displays the nucleotide sequence for both strands, with the translation to amino acids lined up above the nucleotides, and the amino acid sequences for the BLAST hits in the other taxa above that. All of the potential in-frame start and stop codons for a gene appear as links and the user simply clicks on the codon to select it as the start or stop codon of the gene. For annotation of rRNAs and chloroplast tRNAs, DOGMA functions similarly to the protein coding genes, except that nucleotide sequences are displayed rather than amino acid sequences, and the user chooses the start and end of the gene. The user can also view the putative secondary structure of the tRNAs.

Animal mitochondrial tRNAs are notoriously difficult to annotate. DOGMA

uses Eddy and Durbin's COVE software to identify a list of putative tRNA sequences and then tries to infer the secondary structure. When a tRNA is selected in the middle panel, a list of the possible tRNAs for that amino acid, with schematic drawings of its secondary structure, are shown in the top panel. The user can choose the tRNA based on the quality of the secondary structure folding and its COVE score.

6.5 Future Work

In the future, the chloroplast database will be expanded to include more taxa. The mitochondrial genomes of plants, fungi, and protists will be added to `DOGMA`, as well as private custom databases for individuals. Researchers only interested in a subset of the database will be able to identify the genomes they are interested in for comparison. This will also allow people interested in phylogenetic reconstruction to identify evolutionary similarity in anticipation of alignment of the whole genomes. It will also allow users to use their own unpublished data for annotation. We plan to construct a searchable database of folded tRNA structures for all organellar genomes as well as adding the capability to `DOGMA` of searching for tRNAs using a variety of types of methods. Future versions of `DOGMA` will additionally address the difficult issue of RNA editing of start and stop codons. There are also plans for including an ORF finder as a subroutine so putative new genes can be identified.

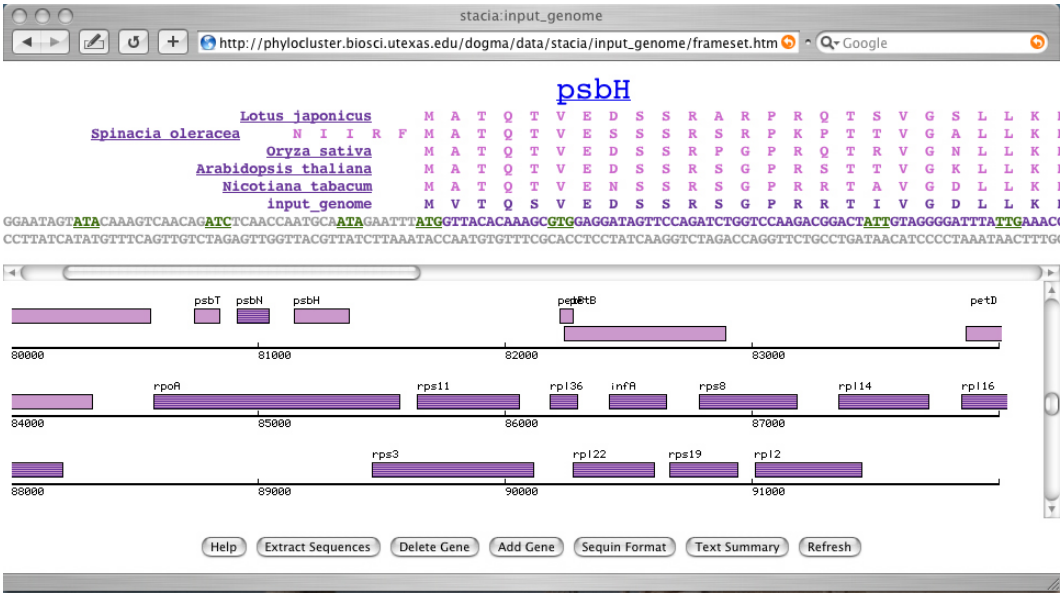


Figure 6.1: The DOGMA annotation window showing details of the *psbH* gene in a chloroplast genome. The bottom panel shows the option buttons, while the middle panel shows the genes in the genome laid out on a number line. The top panel is where the start and stop codons are annotated for each gene.

Chapter 7

Phylogenetic Footprinting: A Comparative Genomics Method for Identifying Regulatory Elements

7.1 Introduction

Understanding gene expression is one of the most pressing unsolved problems in molecular biology today. Gene expression controls the metabolic and developmental processes in all organisms. Gene expression is controlled by regulatory elements (or motifs) which are short sequences (about 6 to 20 nucleotides) to which enzymes attach in order to control the expression of a gene. They are responsible for turning on and off enzymes required for the function of metabolic processes. Regulatory elements (e.g. promoters, transcription factor binding sites) are typically found in the non-coding regions upstream of the genes that they regulate (other motifs may appear downstream of genes or within coding regions or introns). Identifying regula-

tory elements is an extremely arduous laboratory process, and while computational methods can provide information about *potential* regulatory elements, laboratory experiments will be needed to demonstrate the function of these motifs. Successful computational methods for identifying putative regulatory elements which are verified in the laboratory would change our understanding of the most widely studied processes in cell and molecular biology.

7.2 Prior Work

Many approaches have been taken in addressing this problem. Existing methods for finding regulatory elements include combinatorial methods [46, 90, 105], expectation maximization (EM) methods [7, 23, 63], and Gibbs sampling methods [62, 66, 67, 72, 82, 115]. These methods have achieved some success, however, identifying regulatory elements remains an unsolved problem in computational biology. Unfortunately, most computational methods produce an avalanche of false positives [101, 113], but it has been shown that a phylogenetic footprinting approach, which incorporates phylogenetic information into the analysis, can significantly reduce the number of false positives [64]. There has been much empirical evidence for the advantages of using sequences from several related genomes [14, 18, 42, 54, 72, 113], which was first suggested by Tagle *et al.* [108].

Phylogenetic footprinting is based on the observation that regulatory elements, under selective pressure, evolve more slowly than the the surrounding non-functional sequence, and therefore, the regulatory elements will be conserved among related species (or *taxa*) with respect to the surrounding sequence [80]. Developing algorithms for phylogenetic footprinting is a difficult problem because not only do regulatory elements for a particular gene vary from genome to genome, non-coding regions (where regulatory elements are found) can vary greatly in length (from less than 100 to 1000 or more nucleotides) and regulatory elements do not appear in

a fixed location with respect to the gene they regulate. Because most of the nucleotides in non-coding regions are not functional, they evolve at a much higher rate than functional regions of the genome (i.e. protein coding genes). In comparing sequences for organisms for which the phylogenetic relationships are known, we would like to align the sequences in order to identify the conserved elements among the sequences, however, because the regulatory elements are not in a fixed position relative to the gene they regulate, and because non-coding regions evolve rapidly, those regions often cannot be aligned and the motifs prove to be elusive.

Blanchette *et al.* [13] introduced the first algorithm which explicitly incorporates not just phylogenetic relationships, but an actual phylogenetic tree into motif discovery. Their algorithm, implemented in the program `FootPrinter`, makes a significant contribution because it is the first to explicitly incorporate a phylogenetic tree, but has the drawback of using parsimony as an optimization criterion. We developed a new approach to phylogenetic footprinting, one which builds upon `FootPrinter` by using it as a first pass over the data in order to identify candidate regulatory elements. In the second pass, the candidate motifs are evaluated under a maximum likelihood framework. The advantages of maximum likelihood methods over parsimony in phylogenetic analysis are well-documented [48, 59, 106], and incorporating models of evolution into the second pass addresses many of the drawbacks inherent in the parsimony approach.

7.3 The Substring Parsimony Problem

A collection of sequences are *homologous* if they are descended from a common ancestor. Phylogenetic footprinting is the problem of identifying regulatory elements in a collection of homologous sequences related by a known phylogenetic tree. It allows us to identify regulatory elements in a collection of sequences which are evolving from species to species. The formal problem which is addressed by Blanchette *et*

al.'s approach to phylogenetic footprinting is the Substring Parsimony Problem [12]. Given a tree T, leaf-labeled by a set of sequences S of length l, the *parsimony score* of a set of sequences is the minimum total number of substitutions over the tree T needed to explain the observed sequences. This is the minimum Hamming distance over all possible labelings of the internal nodes of T by sequences of length l. The Substring Parsimony Problem can then be stated as: given a set of homologous sequences $S_1, ..., S_n$ from n different species, the phylogenetic tree T indicating the relationship among these species, leaf-labeled with the sequences $S_1, ..., S_n$, the length k of the motifs to search for, and an integer d, find all sets of substrings $s_1, ..., s_n$ of $S_1, ..., S_n$, each of length k, such that the parsimony score of $s_1, ..., s_n$ on T is at most d.

7.4 Blanchette *et al.*'s Approach

The Substring Parsimony Problem is NP-hard [1], but Blanchette *et al.* have developed a dynamic programming algorithm which solves it exactly. Their algorithm arbitrarily roots the unrooted input tree T at an internal node r and proceeds from the leaves to the root. At each node u of the tree, they compute a table W_u each with 4^k entries, one for each of the possible subsequences of length k. For a string s of length k, $W_u[s]$ is the best parsimony score for the subtree rooted at node u if u is labeled with s. $C(u)$ is the set of children of u, and $d(s,t)$ is the Hamming distance between sequences s and t, and $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. The tables can be computed by the following dynamic programming algorithm:

$$W_u[s] = \begin{cases} 0 & \text{if } u \text{ is a leaf, and } s \text{ a substring of } S_u \\ \infty & \text{if } u \text{ is a leaf, and } s \text{ not a substring of } S_u \\ \sum_{v \in C(u)} min_{t \in \Sigma^k}(W_v[t] + d(s,t)) & \text{if } u \text{ is not a leaf.} \end{cases} \tag{7.1}$$

The score of the solution to the Substring Parsimony Problem is then given by $min_{s \in \Sigma^k}(W_r[s])$. This algorithm runs in $O(nk(4^{2k} + l))$ time where l is the average length of the input sequences, and for each of the $O(n)$ edges, for each of the 4^k strings of length k at each endpoint of the edges, the distance $d(s, t)$ is computed in k time. The ancestral sequences can be recovered by tracing back the recurrence in time linear in the size of the tables. The k in the exponent makes the algorithm infeasible to run as is, but they describe some bounding techniques which take advantage of the input parameter d, the maximum allowable number of mutations over the tree. These improvements reduce the running time to $O(nk(min(l(3k)^{\frac{d}{2}}, 4^k+l)))$.

7.5 Drawbacks to FootPrinter: Improvements with Maximum Likelihood

Although recent papers have made it clear that explicitly incorporating phylogenetic information is becoming the approach of choice for researchers [14, 18], the fundamental problem with `FootPrinter` is that it seeks motifs with the minimum number of changes (the most parsimonious solution) to explain the sequences. The most parsimonious solution is not always the most biologically accurate [106] and does not incorporate models of evolutionary change into the solution. Parsimony inherently makes assumptions about the data, which can lead to incorrect solutions. Incorporating models of evolution allows us to account for different rates of evolution on the tree and between sites in a sequence, as well as to distinguish between the kinds of mutations which occur (i.e. `G` may be more likely to mutate to `C` than to `A`). Parsimony may not be computing a biologically accurate distance, and therefore can miss motifs which are biologically significant (but have more than the d mutations `FootPrinter` allows) in more distantly related taxa.

The advantages of a maximum likelihood approach are multifold. Given a

well-accepted tree computed under ML, we have at our hands not only estimates of branch lengths of the tree, but also the model parameters which best fit the data under which the tree was estimated. We can make use of this information in assessing how likely it is for a motif to have evolved along an edge of the tree. A significant improvement over Blanchette *et al.*'s approach is that we are actually using the phylogenetic tree to explain the data and identify motifs that are conserved in a way that is consistent with the input tree, its branch lengths, and its model parameters. This makes much better use of the phylogenetic information at hand and is more biologically accurate.

7.6 Maximum Likelihood Approach

7.6.1 Definitions

A substitution model Q is a table of rates at which each nucleotide is replaced by each alternative nucleotide. This is represented by a 4x4 matrix where each entry i, j represents the rate of change from base i to base j for some infinitesimal amount of time. The model incorporates frequency parameters which represent the frequencies of the bases `A, C, G, T`. The model also incorporates relative rate parameters representing the probability of each base i to mutate to base j for all four bases. When the matrix is symmetric around the diagonal, it is a time-reversible model where the probability of a mutation from i to j is equal to the probability of a mutation from j to i. The matrix Q specifies the rates of change from one nucleotide to another per instant of time. The probabilities of change from one state to another along a branch of length t is the likelihood. Given the matrix Q and a tree T labeled with branch lengths, leaf-labeled with nucleotide sequences, we can calculate the likelihood of the tree T with respect to the model Q. To incorporate rate heterogeneity among sites, we also include a relative rate component into the

model, drawn from a gamma distribution, and represented by the shape parameter α. And finally, the percent of expected invariable sites in the data is also specified in the model.

7.6.2 ML FootPrinter

Our approach, in which we incorporate models of evolution into the analysis to aid in the identification of regulatory elements, is ML FootPrinter. It has two phases. Given an accepted phylogeny T for the set S of n input sequences, and an integer d, the bound on the parsimony score, `FootPrinter` is run on the dataset in Phase I. The input parameter d is set to a relatively high value in order to find motifs which may have evolved over longer branches of the tree. This gives us a large set of candidate motifs for the input sequences. `FootPrinter` often returns many motif sequences *per taxon* when the motifs are short and there are low-complexity regions in the sequences. If there are many motifs reported in an individual taxon of a particular motif, it is unlikely that it is truly a regulatory element, as this would mean that enzymes could bind to many different sites in the intergenic spacers. For this reason, the next step in Phase I is to filter out the motifs which have returned multiple motif sequences for *e*ach taxon. Phase II evaluates the candidate motifs. In Phase II, the candidate sequences are evaluated under both functional and non-functional models of evolution to see if the sequences are more likely given the functional or non-functional model. Although we refer to models constructed from intergenic regions as *non-functional* to distinguish them from the genic, functional regions, obviously these regions contain functional elements, which are exactly the motifs we are looking for.

Define Q_F and Q_{NF} to be the substitution models for the functional and non-functional regions of the input genomes, respectively. Given the tree T with branch lengths, a set s of n candidate motifs and the two models of evolution Q_F

and Q_{NF}, we evaluate the maximum likelihood of these motif sequences on the given tree with respect to each model using the program `PAUP*` [107]. Once these scores have been computed, the sequences which have a higher likelihood with respect to the functional model are the sequences we would suggest to be true motifs or regulatory elements. Once the likelihood scores are computed, we can use Akaike's Information Criteria (AIC) [19] to evaluate how well one model fits the data versus the other. AIC values are calculated from likelihood scores for each model. We can then calculate Akaike weights from the AIC values where the Akaike weight w_i is the weight of evidence that model i is the best approximating model, given the the data and set of candidate models. The true test of whether a motif actually regulates a gene must be done in the lab; our method is meant to reduce the number of false positives a researcher must consider, and hopefully improve the quality of the candidate motifs.

7.7 Real Data Analysis: The *atpB-rbcL* Region of a Set of Chloroplast Genomes

Datasets suitable for phylogenetic footprinting must be chosen carefully; if there is not enough evolution on the tree, there will not be enough change in the non-coding regions to identify true motifs. If there is too large an amount of evolution in the dataset, common motifs may not even exist among taxa. The intergenic region between the *rbcL* and *atpB* genes of chloroplast genomes was chosen as a real dataset for evaluation because the region has been well-studied [24, 43, 70, 120] with two known promoter sequences for *rbcL* in that region. The *rbcL* gene encodes the large subunit of the enzyme ribulose-1,5-bisphosphate carboxylase (Rubisco) which is needed for fixation of carbon dioxide in photosynthesis. This enzyme is arguably the most abundant and important enzyme because it controls the earth's carbon

cycle [25]. Additionally, the whole genomes (and therefore the intergenic spacers) have been sequenced for an appropriately divergent set of genomes and there exists an accepted phylogeny for the genomes. Taxa included in the dataset are are shown in Table 7.1. The accepted phylogeny for these taxa is shown in Figure 7.1 [24].

We used `MultiPipMaker` [102] to compute a Percent Identity Plot (PIP) for a selection of the genomes in our analysis (Figure 7.2). `MultiPipMaker` compares long DNA sequences to identify conserved segments and produces a percent identity plot, which shows both the position in one sequence and the degree of similarity for each aligning segment between all the sequences and the reference sequence. Positions along the horizontal axis are labeled with genes. Horizontal lines represent a match with the reference sequence, and the horizontal box for each genome represents degree of similarity in percent from 50 (at the bottom of the box) to 100 (at the top of the box). It can be seen from the figure that the genic regions of *rbcL* and *atpB* are quite well-conserved, while the intergenic region has a large amount of variability. This illustrates that the rate of evolution in the intergenic region is much higher than in the functional genic region, making it a suitable test case for our algorithm because conserved sequences would not be expected to occur spuriously.

7.7.1 Methods

To create the two models of evolution (Q_F for the genic region and Q_{NF} for the intergenic region) for the chloroplast data, two separate datasets were compiled: a functional dataset consisting of the DNA sequences of the *rbcL* gene for each of the 11 genomes (detailed in Table 7.1), and a non-functional dataset consisting of the DNA sequences of the intergenic region between *rbcL* and *atpB* for all the genomes. For each of the two datasets, sequences were extracted from GenBank and then aligned using ClustalW [53]. Using the accepted phylogeny (Figure 7.1) for the aligned sequences [24], `PAUP*` was then used to score the tree under maximum likelihood

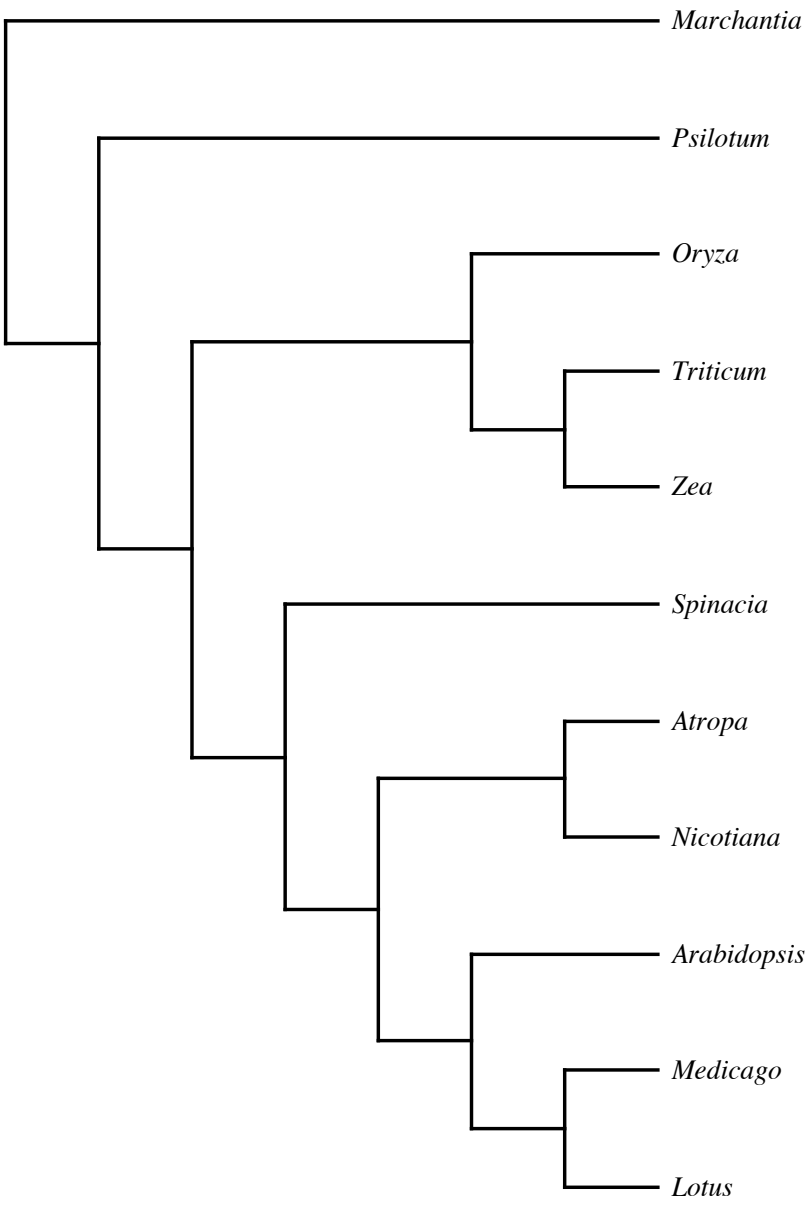


Figure 7.1: The accepted phylogeny of the *rbcL* gene for the 11 chloroplast genomes in our analysis.

Taxon	*rbcL* Coordinates	Strand	Intergenic Sequence	Length
Atropa	57248-58681	+	56433-57247	814
Arabidopsis	54958-56397	+	54156-54957	801
Lotus	5158-6585	−	6586-7367	781
Marchantia	56355-57782	+	55847-56354	507
Medicago	117623-119047	+	116867-117622	755
Nicotiana	57595-59028	+	56777-57594	817
Psilotum	55824-57251	+	55232-55823	591
Oryza	54082-55536	+	53314-54081	767
Spinacia	53825-55252	+	53038-53824	786
Triticum	54910-56343	+	54128-54909	781
Zea	56874-58304	+	56115-56873	758

Table 7.1: The 11 taxa used to create the chloroplast models and the coordinates of the *rbcL* gene and the *rbcL-atpB* intergenic sequence. When *rbcL* was on the reverse strand, the reverse complement of the downstream sequence was used.

for the genic and for the intergenic sequences to compute the model parameters for each of the two datasets. For each model, parameters include the substitution rates, the nucleotide frequencies, the shape parameter of the gamma distribution (α) which describes the among-site variation in rates of substitution, and the percent of invariable sites, all of which are estimated from the data by `PAUP*`. The result is Q_F, the model from the *rbcL* gene, and Q_{NF}, the model estimated from the intergenic sequences. The parameter values for these models are shown in Table 7.2. The branch lengths for the functional sequences were saved with the tree topology as these represent the amount of change we would expect to see on a branch for functional sequences (such as regulatory elements).

7.7.2 Experiments

We ran the 11-taxon chloroplast dataset consisting of sequences from the intergenic regions through `FootPrinter` with various parameter settings. Because the dataset

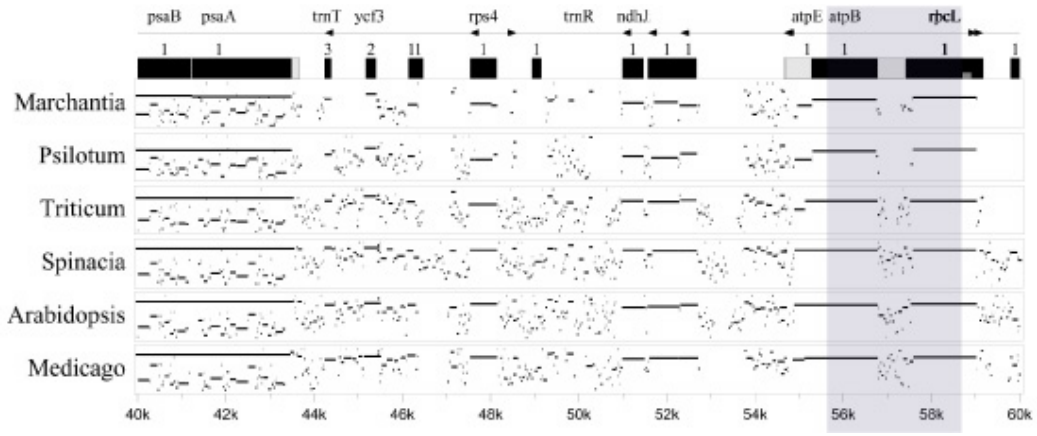


Figure 7.2: `MultiPipMaker` output with *Nicotiana* used as the reference genome. The region containing *atpB* and *rbcL* is highlighted.

Model	Rate Matrix			Base Freqs. A	C	G	T	α	% Inv. Sites
Q_F	1.20	3.39	0.77						
		0.99	5.38	0.29	0.18	0.22	0.31	0.828	0.43
			1.00						
Q_{NF}	1.16	2.13	0.57						
		1.78	2.45	0.38	0.13	0.13	0.36	5.514	0.04
			1.00						

Table 7.2: The parameter values for the Q_F and Q_{NF} models estimated from the chloroplast dataset using `PAUP*`.

is large and because `FootPrinter` uses as large amount of memory, we were restricted to searching for motifs of size 6, 8, 10, and under limited conditions, 12. In searching for motifs of size 6, we searched with a maximum allowed parsimony score of $d = 4$, a parsimony score of $d = 5$ for motifs of size 8, and a parsimony score of $d = 6$ for motifs of size 10 and 12. We filtered out the motifs which had multiple motif sequences for individual taxa. This is a particularly important step for the *rbcL* dataset because intergenic regions in chloroplast genomes often evolve in ways that create many small repeats (such as slipped-strand mispairing and insertions linked with secondary structure formations) [57]. There are occasions when regulatory elements do exist as multiple copies, but this is typically limited to just pairs of regulatory elements [57] rather than many copies, as `FootPrinter` returns for low complexity regions (i.e. a long string of `A`s). The filtered motifs from the first pass were then scored using maximum likelihood using `PAUP*` under both the functional and non-functional models of evolution. The results were compared with the two known regulatory elements for *rbcL* in that region. The motifs were also evaluated in terms of their potential as novel regulatory elements.

The questions we wanted to answer in the real data analysis were as follows. Can ML FootPrinter identify the known regulatory elements for this dataset? If found, can they be evaluated based on their likelihood scores with respect to the two models? Can it distinguish between the known promoter and spurious motifs (false positives) using likelihood scores? A false positive occurs when a motif is identified as a regulatory element although it is not one. Of course, besides the two known promoters we are looking for, we don't actually know when a sequence in the real dataset is a false positive, that must be determined in the lab, but our hope is that we could rule out a large number. While filtering eliminates a good number of unlikely candidate motifs, the results from phase one still produce more motifs than could reasonably be investigated in a lab.

7.7.3 Results for the Chloroplast Dataset

After being filtered, we were left with 24 motifs of length 6, 13 of length 8, 31 of length 10, 7 of length 12, and 5 of length 14 to be evaluated. The two known *rbcL* promoters for this dataset are `cpt1` with the nucleotide sequence `TTGCGC` and `cpt2` with the nucleotide sequence `TACAAT` [120]. `FootPrinter` identified the `cpt1` promoter while searching for motifs of length 8 and 10, but interestingly did not identify it while searching for motifs of length 6. This is because there were mutations in the regulatory element for the more distant taxa in the tree and the parsimony score d may not have been high enough when search for motifs of length 6 (thought it was set as high as possible). The promoter sequence appears exactly in 9 of the 11 sequences. The second known promoter was identified when searching for motifs of length 10, 12 and 14. It appears in 8 of the 11 sequences without mutation.

When the promoter sequence `cpt1` was found in the results for the motif length 8 search, it had a log likelihood score of -23.440 with respect to Q_F and -28.370 with respect to Q_{NF} which gives us AIC scores of 54.88 for Q_F and 64.74 for Q_{NF} and AIC weights $w_F = 0.992$ and $w_{NF} = 0.007$ making the data 138 times more likely to fit Q_F. When `cpt1` was found in the results for the motif length 10 search, it had a log likelihood score of -39.979 with respect to Q_F and -43.383 with respect to Q_{NF} which gives us AIC scores of 87.96 for Q_F and 94.77 for Q_{NF} and AIC weights $w_F = 0.968$ and $w_{NF} = 0.032$ making the data 30 times more likely to fit Q_F. These numbers suggest that these motif sequences are more likely with respect to the functional model and therefore it is a good candidate motif, as would be hoped.

When the promoter sequence `cpt2` was found in the results for the motif length 10 search, it had a log likelihood score of -39.421 with respect to Q_F and -34.447 with respect to Q_{NF} which gives us AIC scores of 86.84 for Q_F and 76.89 for Q_{NF} and AIC weights $w_F = 0.007$ and $w_{NF} = 0.993$ making the data 144 times

more likely to fit Q_{NF}. When `cpt2` was found in the results for the motif length 12 search, it had a log likelihood score of -36.439 with respect to Q_F and -35.100 with respect to Q_{NF} which gives us AIC scores of 80.88 for Q_F and 78.20 for Q_{NF} and AIC weights $w_F = 0.207$ and $w_{NF} = 0.792$ making the data 4 times more likely to fit Q_{NF}. When `cpt2` was found in the results for the motif length 14 search, it had a log likelihood score of -52.900 with respect to Q_F and -49.093 with respect to Q_{NF} which gives us AIC scores of 113.8 for Q_F and 106.19 for Q_{NF} and AIC weights $w_F = 0.021$ and $w_{NF} = 0.978$ making the data 44 times more likely to fit Q_{NF}. Obviously, the results for the `cpt2` promoter are not what what would be hoped, however, the results are probably due to to low complexity of the motif (i.e. it contains 3 `A`s). It suggests that perhaps having one model for all motifs may not be enough to identify all the regulatory elements.

7.8 Simulated Data: An Experimental Investigation

In order to thoroughly explore the limitations and boundaries of the ML FootPrinter method, we conducted a simulation study with trees and sequences of varying sizes and randomly planted motifs of varying lengths into the sequences. With the simulation study, we sought to answer the following questions. How often does it locate all the planted motifs? How often did it miss just one motif? How often does it miss all the planted motifs? What is the likelihood of the planted motifs with respect to the other motifs found and with respect to both the functional and the non-functional models? Can ML FootPrinter sufficiently distinguish between potentially functional motifs and spurious ones to significantly reduce the number of false positives? What are the limits with respect to tree size, motif length and sequence length?

7.8.1 Methods

Trees: In order to create the simulated datasets, we first constructed a set of tree topologies with varying numbers of taxa and branch lengths. This was done using the `r8s` package [97] which generates random binary trees under a birth-death model of evolution. The birth-death model, in adding and removing new lineages with some probability, has the result of creating trees that are not complete binary trees. For each of the numbers of leaves (5, 7, 10, 12, 15, and 20), four different tree topologies with different branch lengths were generated, giving us 24 trees topologies total.

Sequences and planted motifs: We then used the program `SeqGen` [91] to generate sequences using the tree topologies generated by `r8s`. `SeqGen` allows the user to specify the parameters of the evolutionary model to use in generating the sequences. We generated sequences under two models. Because we wanted the two models to be representative of real models, we used the Q_F and Q_{NF} models from the chloroplast dataset (see Table 7.2) representing the intergenic (non-functional) region with a high rate of evolution, and the other from the functional *rbcL* region of the dataset representing a region with a low rate of evolution. For each model tree, we generated 4 simulated datasets each with sequences of length 250, 500, 750, 1000 and 2000 under the Q_{NF} model to represent the intergenic sequences, and sequences of length 6, 8, 10, 12, 14 and 16 under the Q_F model to represent the functional motifs. The short motif sequences generated from the Q_F model were then randomly planted into the sets of sequences generated from the Q_{NF} model in order to represent regulatory elements in intergenic regions. The result was a total of 2,880 datasets for the simulation study.

7.8.2 Experiments and Results

All of the 2,880 simulated datasets with the planted motifs were then run through `FootPrinter` with a parsimony score which varied according to the length of the motif sought and the size of the tree. Typical values for d were half of the motif size and half of the tree size. `FootPrinter` failed to produce motifs either because it was not able to run to completion, or it simply did not find any motifs for the dataset for the specified values of d. We found that , when the value of d was low, no motifs were found, however too large values of d (for large trees with long sequences) caused the program to run out of memory. All three parameters (tree size, motif size, and sequence length) contributed to `FootPrinter` running out of memory or failing to produce motifs. Sequences longer than 1,000 nucleotides, or trees with more than 12 taxa, or motifs larger than 12 often caused `FootPrinter` to fail.

`FootPrinter` produced a set of motifs for $1,494$ of the $2,880$ datasets. This discrepancy was not surprising since we were trying to test the boundary conditions for the possible numbers of taxa and size of motifs for the datasets. Once we filtered out the motifs which had multiple sequences for individual taxa, we were left with putative motifs for 1,369 datasets. We found that of the datasets which produced motifs, `FootPrinter` successfully found all of the planted motifs in 62 of the 1,369 datasets and it missed just one of the planted motifs for 27 of the datasets. For 1,226 of the 1,369 datasets, `FootPrinter` failed to find any of the planted motifs.

With respect to the planted motifs, they had AIC weights that overwhelmingly favored the functional model for 2,860 of the datasets. This, of course, is not surprising since the data was generated using the the functional model. When the likelihoods of the planted motifs were compared with the other motifs found by `FootPrinter` they did not suggest that any particular motif found was any better than any of the other motifs found.

7.9 Discussion and Future Work

These are preliminary results for ML FootPrinter and, although it did not identify all the regulatory elements, we still learned what improvements could be made and what to explore next. A better implementation of `FootPrinter`, particularly incorporating a model-based approach, is needed. Modifications were attempted to improve `FootPrinter`, but it was clear that either a serious overhaul or a reimplementation would be required to improve it. More real data analysis would be helpful in evaluating the method; this would demonstrate whether perhaps the chloroplast dataset was not a good one to try and analyze, and it would demonstrate if there are better ones to select with more closely or less closely related taxa. Because these are the initial models, it may mean that they are not the correct models to use and that new ones need to be tested. Perhaps deriving the models from the protein coding genes of the genome does not generate the correct model. It may be that creating models specific to collections of regulatory elements (as more regulatory elements are identified), rather than functional regions of the genome, would perform better. After more consideration of how protein coding genes evolve, and the specific selective pressures they evolve under, these seems even more probable. For example, protein coding genes often evolve with codon position bias. Each codon (triplet of nucleotides) has three positions, and the first position is under the most selective pressure in some genes, while the third position of the codon is under almost no selective pressure. Mutations in the third position of a codon may not even change the amino acid that the codon codes for in the protein or the gene function. No such selective pressure exists in regulatory elements since they do not have codons. Therefore, as we learn more about exactly how regulatory elements control the function of genes, then we will be able to construct more accurate models specific to regulatory elements incorporating the specific kinds of selective pressure they are under. It would be interesting to see how using the incorrect topology (if the phy-

logeny is not known for a dataset) affects the identification of regulatory elements. There is more evidence accumulating daily that regulatory elements are certainly conserved functional elements in intergenic regions, and methods which exploit this fact will be the most successful. Much work is still required in this area, and as more is known about gene function, automated methods will improve.

Bibliography

[1] T. Akutsu. Hardness results on gapless local multiple sequence alignment. *Technical Report 98-MPS-24-2, Information Processing Society of Japan*, 1998.

[2] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[3] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. CONCORDE: Combinatorial Optimization and Networked Combinatorial Optimization Research and Development Environment. `www.tsp.gatech.edu/concorde.html`.

[4] D.A. Bader, A.K. Illendula, and B.M.E. Moret. Using PRAM algorithms on a uniform-memory-access shared-memory architecture. Report 2001-03, Deptartment of Computer Science, Univeristy of New Mexico, 2001.

[5] D.A. Bader and B.M.E. Moret. GRAPPA runs in record time. *HPC Wire*, 9(47), 2000.

[6] D.A. Bader, B.M.E. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *J. Comput. Biol.*, 8(5):483–491, 2001.

[7] T.L. Bailey and C. Elkan. Unsupervised Learning of Multiple Motifs in

Biopolymers Using Expectation Maximization. *Machine Learning*, 21:51–83, 1995.

[8] B. Billoud, M. Kontic, and A. Viari. Palingol: a declarative programming language to describe nucleic acids' secondary sructures and to scan sequence databases. *Nucleic Acids Res.*, 24:1395–1403, 1996.

[9] M. Blanchette. `http://www.cs.washington.edu/homes/blanchem/software.html`.

[10] M. Blanchette, G. Bourque, and D. Sankoff. Breakpoint Phylogenies. In S. Miyano and T. Takagi, editors, *Genome Informatics*, pages 25–34. University Academy Press, Tokyo, Japan, 1997.

[11] M. Blanchette, M. Kunisawa, and D. Sankoff. Gene order breakpoint evidence in animal mitochondrial phylogeny. *J. Mol. Evol.*, 49:193–203, 1999.

[12] M. Blanchette, B. Schwikowski, and M. Tompa. Algorithms for phylogenetic footprinting. *Journal of Computational Biology*, 9(2):211–223, 2002.

[13] M. Blanchette and M. Tompa. Discovery of Regulatory Elements by a Computational Method for Phylogenetic Footprinting. *Genome Research*, 12:739–748, 2002.

[14] D. Boffelli, J. McAuliffe, D. Ovcharenko, K.D. Lewis, I. Ovcharenko, L. Pachter, and E.R. Rubin. Phylogenetic Shadowing of Primate Sequences to Find Functional Regions of the Human Genome. *Science*, 299:1391–1394, 2003.

[15] J.L. Boore. Animal Mitochondrial Genomes. *Nucleic Acids Res.*, 27:1767–1780, 1999.

[16] J.L. Boore. The duplication/random loss model for gene rearrangement exemplified by mitochondrial genomes of deuterostome animals. In D. Sankoff

and J. Nadeau, editors, *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families*, pages 133–147, Dordrecht, Netherlands, 2000. Kluwer Academic Publishers.

[17] J.L. Boore and W.M. Brown. Big trees from little genomes: mitochondrial gene order as a phylogenetic tool. *Curr. Opin. Genet. Dev.*, 8(6):668–674, 1998.

[18] M.L. Bulyk. Computational prediction of transcription-factor binding site locations. *Genome Biology* (Online journal), 5(201), 2003.

[19] K.P. Burnham and D.R. Anderson. *Model selection and inference: a practical information-theoretic approach*. Springer-Verlag, New York, NY, 1998.

[20] R.M. Bush, C.A. Bender, K. Subbarao, N.J. Cox, and W.M. Fitch. Predicting the evolution of human influenza A. *Science*, 286:1921–1925, 1999.

[21] Y. Cao, M. Fujiwara, M. Nikaido, N. Okada, and M. Hasegawa. Interordinal relationships and timesecale of eutherian evolution as inferred from mitochondrial genome data. *Gene*, 259:149–158, 2000.

[22] A. Caprara. Formulations and hardness of multiple sorting by reversals. In *Proc. 3rd Int'l Conf. on Comput. Mol. Bio. (RECOMB99)*, pages 84–93. ACM Press, NY, 1999.

[23] L.R. Cardon and G.D. Stormo. Expectation maximization algorithm for identifying protein-binding sites with variable lengths from unaligned DNA fragments. *Journal of Molecular Biology*, 223:159–170, 1992.

[24] M.W. Chase, D.E. Soltis, R.G. Olmstead, D. Morgan, D.H. Les, B.D. Mishler, M.R. Duvall, R.A. Price, H.G. Hills, Y.L. Qiu, K.A. Kron, J.H. Rettig, E. Conti, J.D. Palmer, J.R. Manhart, K.J. Sytsma, H.J. Michaels, W.J.

Kress, K.G. Karol, W.D. Clark, M. Hedren, B.S. Gaut, R.K. Jansen, K.J. Kim, C.F. Wimpee, J.F. Smith, G.R. Furnier, S.H. Strauss, Q.Y. Xiang, G.M. Plunkett, P.S. Soltis, S.M. Swensen, S.E. Williams, P.A. Gadek, C.J. Quinn, L.E. Eguiarte, E. Golenberg, Jr. G.H. Learn, S.W. Graham, S.C.H. Barrett, S. Dayanandan, and V.A. Albert. Phylogenetics of Seed Plants: An Analysis of Nucleotide Sequences from the Plastid Gene rbcL. *Annals of the Missouri Botanical Garden*, 80(3):528–548,550–580, 1993.

[25] M.T. Clegg. Chloroplast gene sequences and the study of plant evolution. *Proc. Natl. Acad. Sci.*, 90:363–367, 1993.

[26] W.J. Conover. *Practical Nonparametric Statistics, 3rd ed.* John Wiley & Sons, 1999.

[27] M. E. Cosner. *Phylogenetic and molecular evolutionary studies of chloroplast DNA variations in the Campanulaceae.* PhD thesis, Ohio State University, Columbus, OH, Department of Botany, 1993.

[28] M.E. Cosner, R.K. Jansen, B.M.E. Moret, L.A. Raubeson, L. Wang, T. Warnow, and S.K. Wyman. A New Fast Heuristic for Computing the Breakpoint Phylogeny and Experimental Phylogenetic Analyses of Real and Synthetic data. In *Proc. 8th Int'l Conf. on Intelligent Systems for Mol. Biol. ISMB-2000*, pages 104–115, 2000.

[29] M.E. Cosner, R.K. Jansen, B.M.E. Moret, L.A. Raubeson, L. Wang, T. Warnow, and S.K. Wyman. An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae. In D. Sankoff and J.H. Nadeau, editors, *Comparative Genomics*, pages 99–122. Kluwer Acad. Pubs., 2000.

[30] M.E. Cosner, R.K. Jansen, J.D. Palmer, and S.R. Downie. The highly rear-

ranged chloroplast genome of *trachelium caeruleum* (Campanulaceae): Multiple inversions, inverted repeat expansion and contraction, transposition, insertions/deletions, and several repeat families. *Curr. Genet.*, 31:419–429, 1997.

[31] DCAF: Gene Order Dynamics, Comparative Maps and Multigene Families (workshop). Montreal, Canada, August 2000.

[32] DIMACS Workshop on Whole Genome Comparison. Piscataway, New Jersey, USA, February 2001.

[33] S.R. Downie and J.D. Palmer. Use of chloroplast DNA rearrangements in reconstructing plant phylogeny. In P. Soltis, D. Soltis, and J.J. Doyle, editors, *Plant Molecular Systematics*, pages 14–35. Chapman and Hall, 1992.

[34] S.R. Eddy and R. Durbin. RNA sequence analysis using covariance models. *Nucleic Acids Research*, 22:2079–2088, 1994.

[35] N. El-Mabrouk and F. Lisacek. Very fast identification of RNA motifs in genomic DNA. Application to tRNA search in the yeast genome. *Journal of Molecular Biology*, 264:46–55, 1996.

[36] J. Felsenstein. PHYLIP. `evolution.genetics.washington.edu/phylip.html`.

[37] G.A. Fichant and C. Burks. Identifying potential tRNA genes in genomic DNA sequences. *Journal of Molecular Biology*, 220:659–671, 1991.

[38] W. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 1955:279–284, 1967.

[39] L.R. Foulds and R.L. Graham. The Steiner problem in phylogeny is NP-Complete. *Advances in Applied Mathematics*, 3:43–49, 1982.

[40] D. Gautheret, F. Major, and R. Cedergren. Pattern searching/alignment with RNA primary and secondary structures: An effective descriptor for tRNA. *Comput. Applic. Biosci.*, 6:325–331, 1990.

[41] GenBank: `http://ncbi.nlm.nih.gov/Genbank`.

[42] M.S. Gelfand, E.V. Koonin, and A.A. Mironov. Prediction of transcription regulatory sites in Archaea by a comparative genomic approach. *Nucleic Acids Research*, 28:695–705, 2000.

[43] L. Hanley-Bowdoin and N.H. Chua. Transcriptional interaction between the promoters of the maize chloroplast genes which encode the β subunit of ATP synthase and the large subunit of ribulose 1,5-bisphosphate carboxylase. *Mol. Gen. Genet.*, 215:217–224, 1989.

[44] S. Hannenhalli and P.A. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *Proc. 27th Ann. Symp. on Theory of Computing (STOC95)*, pages 178–189, Las Vegas, NV, 1995. ACM.

[45] M. Held and R.M. Karp. The travelling salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

[46] G.Z. Hertz and G.D. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, 15:563–577, 1999.

[47] D.M. Hillis. Origins of HIV. *Science*, 288:1757–1759, 2000.

[48] D.M. Hillis, J.P. Huelsenbeck, and D.L. Swofford. Hobgoblin of phylogenetics? *Nature*, 369:363–364, 1994.

[49] S.B. Hoot and J.D. Palmer. Structural rearrangements, including parallel inversions, within the chloroplast genome of Anemone and related genera. *J. Molecular Evolution*, 38:274–281, 1994.

[50] D. Huson, S. Nettles, K. Rice, T. Warnow, and S. Yooseph. Hybrid tree reconstruction methods. *ACM Journal of Experimental Algorithmics*, 4(5), 1999.

[51] D. Huson, K.A. Smith, and T. Warnow. Correcting large distances for phylogenetic reconstruction. In *Proc. 3rd Workshop on Algorithm Engineering WAE99*, pages 273–286. Springer Verlag, 1999.

[52] M. Ingman, H. Kaessmann, S. Pääbo, and U. Gyllensten. Mitochondrial genome variation and the origin of modern humans. *Nature*, 408:708–713, 2001.

[53] T.J. Gibson J.D. Thompon, D.G. Higgins. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 2004.

[54] K. Jiao, J. Nau, M. Cool, W. Gray, J. Fassler, and R. Malone. Phylogenetic footprinting reveals multiple regulatory elements involved in control of the meiotic recombination gene, rec102. *Yeast*, 19:99–114, 2002.

[55] D.S. Johnson and L.A. McGeoch. The traveling salesman problem: a case study. In E. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley, New York, 1997.

[56] H. Kaplan, R. Shamir, and R.E. Tarjan. A faster and simpler algorithm for sorting signed permutations by reversals. *SIAM J. Comput.*, 29(3):880–892, 1999.

[57] S.A. Kelchner. The evolution of non-coding chloroplast DNA and its application in plant systematics. *Annals of the Missouri Botanical Garden*, 87:482–498, 2000.

[58] E.B. Knox, S.R. Downie, and J.D. Palmer. Chloroplast genome rearrangements and the evolution of giant lobelias from herbaceous ancestors . *Molecular Biology and Evolution*, 10:414–430, 1993.

[59] M.K. Kuhner and J. Felsenstein. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Molecular Biology and Evolution*, 11:459–468, 1994.

[60] A. Laferriere, D. Gautheret, and R. Cedergren. An RNA pattern matching program with enhanced performance and portability. *Comput. Applic. Biosci.*, 10:211–212, 1994.

[61] A. LaMarca and R.E. Ladner. The Influence of Caches on the Performance of Heaps. *ACM J. Experimental Algorithmics*, 1(4), 1996.

[62] C.E. Lawrence, S.F. Altschul, M.S. Boguski, J.S. Lui, A.F. Neuwald, and J.C. Wootton. Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment. *Science*, 262:208–214, 1993.

[63] C.E. Lawrence and A.A. Reilly. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins*, 7:41–51, 1990.

[64] B. Lenhard, A. Sandelin, L. Mendoza, P. Engstrom, and W.W. Wasserman N. Jareborg. Identification of conserved regulatory elements by comparative genome analysis. *Journal of Biology*, 13(2), 2003.

[65] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[66] J.S. Liu and C.E. Lawrence. Bayesian inference on biopolymer models. *Bioinformatics*, 15:38–52, 1999.

[67] J.S. Liu, A.F. Neuwald, and C.E. Lawrence. Bayesian models for multiple local sequence alignment and Gibbs sampling strategies. *J. Am. Stat. Assoc.*, 90:1156–1170, 1995.

[68] T.M. Lowe and S.R. Eddy. tRNAscan-SE: A program for improved detection of transfer RNA genes in genomic sequence. *Nucleic Acids Res.*, 25:955–964, 1997.

[69] T.J. Macke, D.J. Ecker, R.R. Gutell, D. Gautheret, D.A. Case, and R. Sampath. RNAMotif, an RNA secondary structure definition and search algorithm. *Nucleic Acids Res.*, 29:4724–4735, 2001.

[70] J.F. Manen, A. Natali, and F. Ehrendorfer. Phylogeny of *rubiaceae-rubieae* inferred from the sequence of a cpDNA intergene region. *Plant Systematics and Evolution*, 190:195–211, 1994.

[71] M. Martin, T. Rujan, T. Richly, A. Hansen, S. Cornelsen, T. Lins, D. Leister, B. Stoebe, M. Hasegawa, and D. Penny. Evolutionary analysis of Arabidopsis, cyanobacterial, and chloroplast genomes reveals plastid phylogeny and thousands of cyanobacterial genes in the nucleus. *Proc. Natl. Acad. Sci.*, 99:12246–12251, 2002.

[72] L. McCue, W. Thompson, C. Carmack, M. Ryan, J. Lui, V. Debyshire, and C. Lawrence. Phylogenetic footprinting of transcription factor binding sites in proteobacterial genomes. *Nucleic Acids Research*, 29:774–782, 2001.

[73] C.C. McGeoch. Toward an experimental method for algorithm simulation. *INFORMS J. Comput.*, 8:1–15, 1996.

[74] J. Meidanis, M.E.M.T. Walter, and Z. Dias. A lower bound on the reversal and transposition diameter. *Journal of Computational Biology*, 9(5):743–745, 2002.

[75] M. Miya, A. Kawaguchi, and M. Nishida. Mitogenomic exploration of higher teleostean phylogenies: A case study for moderate-scale evolutionary genomics with 38 newly determined complete mitochondrial DNA sequences. *Mol. Biol. Evol.*, 18:1993–2009, 2001.

[76] B.M.E. Moret. Towards a discipline of experimental algorithmics. In *DIMACS Monographs in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 2002.

[77] B.M.E. Moret, D.A. Bader, and T. Warnow. High-performance algorithm engineering for computational phylogenetics. In *Proc. 2001 Int'l Conf. Computational Sci. ICCS 2001*. Springer Verlag, 2001.

[78] B.M.E. Moret and H.D. Shapiro. *Algorithms from P to NP, Vol. I: Design and Efficiency*. Benjamin-Cummings, Menlo Park, CA, 1991.

[79] B.M.E. Moret, S.K. Wyman, D.A. Bader, T. Warnow, and M. Yan. A new implementation and detailed study of breakpoint analysis. In *Proc. 6th Pacific Symp. Biocomputing PSB 2001*, pages 583–594. World Scientific Pub., 2001.

[80] A.M. Moses, D.Y. Chiang, M. Kellis, E.S. Lander, and M.B. Eisen. Position specific variation in the rate of evolution in transcription factor binding sites. *BMC Evolutionary Biology*, 3(19), 2003.

[81] J.H. Nadeau and B.A. Taylor. Lengths of chromosome segments conserved since divergence of man and mouse. *Proc. Nat'l Acad. Sci. USA*, 81:814–818, 1984.

[82] A.F. Neuwald, J.S. Liu, and C.E. Lawrence. Gibbs motif sampling: detection of bacterial outer membrane protein repeats. *Protein Science*, 4:1618–1632, 1995.

[83] R.G. Olmstead and J.D. Palmer. Chloroplast DNA systematics: a review of methods and data analysis. *Amer. J. Bot.*, 81:1205–1224, 1994.

[84] J.D. Palmer. Plastid chromosomes: structure and evolution. *Cell Culture and Somatic Cell Genetics of Plants*, 7A:5–53, 1991.

[85] J.D. Palmer. Chloroplast and mitochondrial genome evolution in land plants. In R. Herrmann, editor, *Cell Organelles*, pages 99–133. Wein, 1992.

[86] T.J. Parsons and M.D. Coble. Increasing forensic discrimination of mitochondrial DNA testing through the analysis of the entire mitochondrial DNA genome. *Croatian Med. J.*, 42:304–309, 2001.

[87] A. Pavesi, F. Conterlo, A. Bolchi, G. Dieci, and S. Ottonello. Identification of new eukaryotic tRNA genes in genomic DNA databases by a multistep weight matrix analysis of transriptional control regions. *Nucleic Acids Res.*, 22:1247–1256, 1994.

[88] W.R. Pearson. Rapid and sensitive sequence comparison with FASTP and FASTA. *Methods in Enzymology*, 183:63–9, 1990.

[89] I. Pe'er and R. Shamir. The median problems for breakpoints are NP-complete. *Elec. Colloq. on Comput. Complexity*, 71, 1998.

[90] P.A. Pevzner and S.H. Sze. *Combinatorial Approaches to Finding Subtle Signals in DNA Sequences*. AAAI Press, 2000.

[91] A. Rambaut and N.C. Grassly. Seq-Gen: an application for the Monte Carlo

simulation of DNA sequence evolution along phylogenetic trees. *Comput. Appl. Biosci.*, 13:235–238, 1997.

[92] D.M. Rand. The Units of Selection on Mitochondrial DNA. *Ann. Rev. Ecol. Syst.*, 32:415–448, 2001.

[93] L.A. Raubeson and R.K. Jansen. Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants. *Science*, 255:1697–1699, 1992.

[94] K. Rice and T. Warnow. Parsimony is Hard to Beat! In *Proc. 3rd Ann. Int'l Conf. Comput. Comb. (COCOON97)*, pages 124–133, 1997.

[95] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstruction of phylogenetic trees. *Molec. Biol. Evol.*, 4:406–425, 1987.

[96] Y. Sakakibarra, M. Brown, R. Hughey, I.S. Mian, K. Sjölander, R.C. Underwood, and D. Haussler. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Res.*, 22:5112–5120, 1994.

[97] M.J. Sanderson. r8s: inferring absolute rates of evolution and divergence times in the absence of a molecular clock. *Bioinformatics*, 19:301–302, 2003.

[98] D. Sankoff. Edit Distances for Genome Comparisons Based on Non-Local Operations. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *3rd Ann. Symp. on Combinatorial Pattern Matching*, volume 644 of *Lecture Notes in Computer Science*, pages 121–135, Tucson, AZ, April/May 1992. Springer-Verlag.

[99] D. Sankoff. Personal communication, February 2000.

[100] D. Sankoff and M. Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *J. Comp. Biol.*, 5:555–570, 1998.

[101] C.D. Schmid, V. Praz, M. Delorenzi, R. Perier, and P. Bucher. The Eukaryotic Promoter Database EPD: the impact of *in silico* primer extension. *Nucleic Acids Research*, 32:D82–D85, 2004.

[102] S. Schwartz, Z. Zhang, K.A. Frazer, A. Smit, C. Riemer, J. Bouck, R. Gibbs, R. Hardison, and W. Miller. PipMaker: A Web Server for Aligning Two Genomic DNA Sequences. *Ann. Rev. Plant Physiol.*, 38:391–418, 1987.

[103] D. Söll and U. RajBhandary, editors. *tRNA: Structure, Biosynthesis, and Function*. American Society for Microbiology, Washington, DC, 1995.

[104] S. Steinberg, A. Misch, and M. Sprinzl. Compilation of tRNA sequences and sequences of tRNA genes. *Nucleic Acids Res.*, 21:3011–3015, 1993.

[105] G.D. Stormo and G.W. Hartzell. Identifying protein-binding sites from unaligned DNA sequences. *Proceedings of the National Academy of Sciences*, 86:1183–1187, 1989.

[106] D. Swofford, G. Olson, P. Waddell, and D. Hillis. Phylogenetic inference. In D. Hillis, C. Moritz, and B. Mable, editors, *Molecular Systematics*, chapter 11, pages 407–514. Sinauer Associates Inc, 2 edition, 1996.

[107] D.L. Swofford. PAUP*: Phylogenetic analysis under parsimony and other methods. *Sinauer Assoc.*, 2000.

[108] D. Tagle, B. Koop, M. Goodman, J. Slightom, D. Hess, and R. Jones. Embryonic ϵ and γ globin genes of a prosimian primate (Galago crassicaudatus) nucleotide and amino acid sequences, developmental regulation and phylogenetic footprints. *Journal of Molecular Biology*, 203:439–455, 1988.

[109] D.C. Wallace. Mitochondrial diseases in man and mouse. *Science*, 283:482–488, 1999.

[110] L.-S. Wang. Exact-iebp: a new technique for estimating evolutionary distances between whole genomes. In *Lecture Notes in Computer Science: Proc. 1st Workshop for Alg. & Bio. Inform. WABI 2001*, pages 175–188. Springer Verlag, 2001.

[111] L.-S. Wang and T. Warnow. Estimating true evolutionary distances between genomes. In *Proc. 33th Ann. Symp. on Theory of Comp. (STOC 2001)*, pages 637–646. ACM, 2001.

[112] T. Warnow. Some combinatorial problems in phylogenetics. In *Proc. Int'l Colloquium on Combinatorics and Graph Theory*, Balatonlelle, Hungary, 1996.

[113] W.W. Wasserman and A. Sandelin. Applied bioinformatics for the identification of regulatory elements. *Nature*, 5:276–287, 2004.

[114] D.R. Wolstenholme, J.L. MacFalane, R. Okimoto, D.O. Clary, and J.A. Wahleithner. Bizarre tRNAs inferred from DNA sequences of mitochondrial genomes of nematode worms. *Proc. Natl. Acad. Sci.*, 84:1324–1328, 1987.

[115] C.T. Workman and G.D. Stormo. Ann-Spec: a method for discovering transcription factor binding sites with improved specificity. *Cancer Research*, 61:2492–2499, 2000.

[116] S.K. Wyman and J.L. Boore. Annotating animal mitochondrial tRNAs: An experimental evaluation of four methods. In *Proc. of European Conf. on Computational Biology (ECCB03)*, pages 44–46, 2003.

[117] S.K. Wyman, R.K. Jansen, and J.L. Boore. Automatic annotation of organellar genomes with DOGMA. *Bioinformatics*, 2004. To appear.

[118] L. Xiao, X. Zhang, and S.A. Kubricht. Improving memory performance of sorting algorithms. *ACM J. Experimental Algorithmics*, 5(3), 2000.

[119] M. Zuker and P. Stiegler. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nucleic Acids Research*, 9:133–148, 1981.

[120] G. Zuraawski and M.T. Clegg. Evolution of higher-plant chloroplast DNA-encoded genes: implications for structure-fuction and phylogenetic studies. *Ann. Rev. Plant Physiol.*, 38:391–418, 1987.

Vita

Stacia Kathleen Wyman was born in Morristown, New Jersey on September 15, 1966, the daughter of Kathleen Mongan Wyman and Michael Thayer Wyman. After graduating from The Williston Northampton School in Easthampton, Massachusetts, in 1985, she enjoyed a one-year hiatus from academia while on Grateful Dead tour with friends. Refreshed from a year of travel and music, she went on to receive her Bachelor of Arts degree from Smith College in Northampton, Massachusetts, in 1990. After a mere year and a half, she next completed her Master of Science degree in Computer Science at The University of Wisconsin, Madison in 1991. She went on to attend the Automotive Program of The College of Alameda in Alameda, California, for two years and then was employed as an auto mechanic for one year in Corvallis, Oregon. Continuing her migration northward, she then lived and worked in Seattle, Washington, before entering the University of Texas at Austin in September of 1999.

Permanent Address: 2304 Longview Street

Austin, TX 78705

This dissertation was typeset with LaTeX 2_ε[1] by the author.

[1] LaTeX 2_ε is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay and James A. Bednar.