AN ABSTRACT OF THE THESIS OF

Jeffrey Douglas Cavener for the degree of Master of Science in Computer Science presented on August 11, 1997. Title:

Visualization, Implementation, and Application of the Walking Tree Heuristics for Biological String Matching

Redacted for Privacy

Abstract approved: ___

Paul Cull

Biologists need tools to see the structural relationships encoded in biological sequences (strings). The Walking Tree heuristics calculate some of these relationships. I have designed and implemented graphic presentations which allow the biologist (user) to see these relations. This thesis contains background information on the biological sequences and some background on the Walking Tree heuristics. I demonstrate my methods by showing a visual matching of mitochondrial genomes. I also show matchings based on amino acids and on hydrophobicity. I also show how the parameters of the visualization can be varied to produce more useful pictures. I implemented a parallel version of the Walking Tree heuristic and used it to produce a phylogenetic tree for picornaviruses. I also implemented several user interfaces. These programs are available on my WWW page which allows a user to produce a picture of a matching by giving the sequences in GenBank format and by making a few mouse clicks.

Visualization, Implementation, and Application of the Walking Tree Heuristics for Biological String Matching

by

Jeffrey Douglas Cavener

A Thesis

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed August 11, 1997
Commencement June 1998

Master of Science thesis of Jeffrey Douglas Cavener presented on August 11, 1997

APPROVED:

Redacted for Privacy

Major Professor, representing Computer Science

Redacted for Privacy

Head of Department of Computer Science

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Jeffrey Douglas Cavener, Author

ACKNOWLEDGEMENT

I am indebted to Jim Holloway and Paul Cull, for the many conversations enabling me to extend their work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DEDICATION

To Mom, without whom this would have been impossible.

VISUALIZATION, IMPLEMENTATION, AND APPLICATION OF THE WALKING TREE HEURISTICS FOR BIOLOGICAL STRING MATCHING

1. INTRODUCTION

1.1. BIOLOGICAL PROBLEMS

"Know thyself" the philosopher says, and mankind from time immemorial has tried to obey this injunction.

Strangely enough computer science has contributed to this search for self-knowledge. Theory of computation has dealt with the meaning of knowledge and computation. Artificial intelligence has dealt with the modeling and simulation of thought. While these specialties have pursued mind, "the ghost in the machine", advances in biology have provided a glimpse of the genetic code, "the program within the machine". These advances have in turn produced a new series of problems for computer science. The major problem is, of course, understanding how this genetic programming language works and how the biological machine interprets and executes the instructions in this language. While this major problem is beyond our present abilities, there are several smaller problems whose solutions may aid in solving the major problem.

First and most obvious is the simple problem of storing masses of data. Megabytes, gigabytes and terabytes stream out of labs, and are safely stored in computer databases.

The next problem is how to access data within these databases. Annotation of the sequence provides a partial solution. It is often possible to use the annotations to find sequences from a particular species or from a particular chromosome or with a known specified gene. Of course, very few sequences are fully annotated, so finding all sequences within a particular gene may be much harder. Computer science has produced various string matching (actually string finding) algorithms that make finding a particular string within a large string a relatively easy operation. These algorithms allow for a certain degree of inexactness. If local changes like insertions, deletions, and substitutions are the only permitted changes then these algorithms can do a good job if finding approximate matches. On the other hand if nonlocal changes like translocations are allowed, these algorithms will be stymied. These algorithms can handle inversions by looking for matches with both the inverted string and the noninverted string. But again inversions within inversions will cause these algorithms to fail to find matches.

In the previous problems a "gene" was a contiguous substring, but in higher organisms a "gene" often consists of several exons which are contiguous substrings that are distributed within a large sequence. So local matching methods will have difficulty finding such genes.

Another problem is that when sequences are found their function may be completely unknown. If one could find strongly matching subregions between two or more strings, one could infer that these matching regions probably are biologically important. The location and characterization of these important substrings could have a large commercial impact.

Even if one does not know the function of a sequence, the similarity among sequences could be used to infer the relatedness of the organisms from which the sequences were derived. Recent work in this area has produced the hypothesis of a mitochondrial Eve, and the hypothesis that the Neanderthals were not our ancestors. If relatedness were a metric then one could use it to infer phylogenetic trees.

1.2. HEURISTICS

Many of the problems of the previous section are computationally intractable and/or ill-specified. For example, no one knows exactly what causes changes in genetic sequences, and so the calculation of relatedness between sequences is ill-specified. Of course, one can create models that exactly specify relatedness, but one does not know if these models are correct. Much of the effort in computer science has focused on the edit distance model because computing matchings in this model only takes time proportional to the product of the string lengths. But it is well known among biologists that this simple model does not capture all of the biological possibilities. Unfortunately when one considers more biologically reasonable models, the matching problems become computationally intractable. For example, even finding the minimum number of flips needed to sort a sequence is an NP-hard problem. For these reasons, one is forced to use heuristics to study biological string matching. A heuristic is a computational procedure which usually computes something close to the desired result.

In this thesis, I concentrated on the walking tree heuristics developed by Jim Holloway and Paul Cull [1–7]. This heuristic, or really family of heuristics, attempts to find a good approximate matching between two strings called the pattern and the text. The basic idea of the heuristic is to form a tree data structure based on the pattern and walk this data structure across the text. The leaves of the tree correspond to the characters of the pattern, and at each step each leaf simply reports on how similar its pattern character is to the text character it is looking at. A higher level node in the tree corresponds to the pattern substring represented by the leaf nodes which are descendents of the higher level node. A higher level node computes a score and a position for its substring based on the currently remembered score and the scores and positions compared by the node's two children. The functions computed by the nodes can be chosen for the desired application.

Modifications of this heuristic allow it to compute alignments between strings, and to deal with inversions and even inversions within inversions. Pleasantly heuristic and its various modifications all run in time approximately proportional to the product of the lengths of the two strings.

Because of the simple tree structure of this heuristic, it can be parallelized relatively easily. Again because of the simple structure and the low degree of communication needed between processors, one would expect almost full speedup, so that running a parallel version on P processors would take about 1/P of the time to run the job using only a single processor.

1.3. VISUALIZATION

Humans are largely visual animals. To understand a mass of data, humans find one picture preferable to thousands of numbers stored in a table. How can string matching be conveyed by a picture?

The simplest idea is to present an alignment as a graph in which there is an edge (or visually a line) from a character in one string to the aligned character in the other string. A little thought indicates that while this might be reasonable for very short strings, it would present a confusing mess for longer strings. Since typical biological strings are several thousand characters in length, one needs a method that suppresses single character matches in favor of longer matches. On the other hand, if one had a block from say character i to character j in one string that aligned with a character l to character m in the other string, one would not want the fact that character i+17 did not align with character l+17 to mess up the block match.

These considerations led me to consider two important parameters, contig length and percent identity.

Contig length is the length of a contiguous pattern substring which is matched to a contiguous text substring. Percent identity of a pattern substring is the percentage of the characters in that substring which are mapped to the identical characters in the text. By varying these two parameters I can decide to show only those substrings that are of at least the contig length and match by at least the specified percentage.

To make the visualization better, I decided to use different colors, green for direct matches and red for inverted matches. These distinct colors will make the two kinds of matches easier to see. The use of color also has another advantage. I could use a light color to indicate a match which had just met the percentage threshold, and use a deeper color to indicate a higher percentage match.

Finally, I realized that outlining matching bands in black made them much easier to see.

1.4. MY CONTRIBUTIONS

The main effort reported in this thesis is the design of visual interfaces for the walking tree heuristic. Of course, these interfaces could have been used on top of other heuristics, but I did not investigate this possibility.

The first interface I designed was based on Tcl/Tk. The second interface was designed using HTML and is available on the World Wide Web at http://www.cs.orst.edu/ cavenej/chasmview.html .

I used these interfaces to present some studies of the efficacy of the walking tree heuristic. I did this by taking a sequence, permuting and inverting this sequence, and then applying the walking tree heuristic to the pair of changed and unchanged sequences.

I also experimented with some real biological data to see which values of the parameters gave a reasonable presentation of the matching between sequences.

As mentioned above, the walking tree heuristics should parallelize easily. Jim Holloway had demonstrated this by creating a parallel form for the Sequent Balance which is a shared memory parallel machine [7]. Holloway demonstrated his program by using it to find the relatedness scores for a family of viral genomes. He then converted these scores to distances and used these distances to construct a phylogenetic tree for these viruses. Holloway showed that his program displayed an almost linear speedup in the number of processors used. Unfortunately, our Sequent has now been decommissioned, so his programs can no longer be run. I designed a parallel program using PVM which would run on a network of workstations. I used this program to compute distances between viral genomes and then fed these distances to an available program which constructed the phylogenetic tree. The major advance here is that my program could calculate the distances for a family of 38 genomes in less time than Holloway's program took to compute the distances for a family of 20 genomes. This speedup is mainly the result of the speed advantage of the workstations over the Sequent Balance.

1.5. OVERVIEW

In Chapter 2 I have given background on basic concepts of molecular biology. I introduce the applicability of the Walking Tree [1–7] to gene discovery, annotation of genetic structures, genome alignment and phylogenetic tree calculation in Chapter 3. I also present an alignment of the mitochondrial genomes of the human and the earthworm which I produced with my software for visualization of genomic alignments, as well as my phylogenetic tree of thirty-eight picornaviruses. In Chapter 4, I discuss bases of sequence interpretation and matching for this and future work, and relate my design considerations for current and future software implementations. In Chapter 5, I present my World Wide Web interface for genome alignment and visualization, which provides access to my software via a compiled CGI program. I also show a sample run of my software using the graphical user interface which I prototyped in Tcl/Tk, and a brief introduction to some of the underlying programs that I have written (which are detailed in the command-line reference in Appendix A). In Chapter 6, I provide further detail regarding my production of a genomic alignment visualization for mitochondrial genomes of the human and the earthworm, and of my fast construction of a phylogenetic tree for Picornaviridae using the Parallel Virtual Machine. I conclude in Chapter 7 with suggestions for further work. Appendix A is the command-line interface reference for my software. Appendix B is a supplement describing the Walking Tree heuristic algorithm, my contribution therein being only the reorganization of material drawn from [1–7] for a poster session presentation.

2. BIOLOGICAL BACKGROUND

2.1. MOTIVATION

Biologists are overwhelmed by the results of advances in DNA and protein sequencing technologies, which have caused an exponential growth in the amount of sequence data available for analysis. The extent to which this information can be navigated and classified drives advances in medicine and many sciences. New software tools are constantly being developed to leverage the data, many of which rely on the matching of one sequence of DNA, RNA, or protein to another, and those which are able to cope with the alignment difficulties of transpositions and inversions as well as indels, may prove to be the most useful. Herein, after mentioning a few of the high points (see [27] or a similar work for a more general introduction), I report on my experience extending one such tool which provides scored alignments for sequences, and suggest future directions for research.

2.1.1. Gene Finding and Annotation of Genetic Structures

Much of the available genetic sequence data has yet to be classified. Because similar sequences usually have similar biological functions, the matching of characterized sequences to new ones is of great importance in the annotation of new data. Additionally, matches between uncharacterized sequences may be useful in the detection of previously unknown genes.

2.1.2. Genome Alignment

Complete genome sequences are now known for some organisms, and the human genome should be available in a few years. Alignments of entire genomic sequences reveal similarities between them which may allow the discovery of shared mechanisms of molecular genetics, as well as exceptions to common mechanisms, and provide for cross-annotation of previously known genes and other features.

2.1.3. Phylogenetic Analysis

The phylogenetic study of molecular sequences is of help in understanding relationships between organisms. Local and global sequence comparison techniques enable the automatic generation of phylogenetic trees for the organisms involved.

2.2. BACKGROUND

The field of computational molecular biology is the application of computational methods to the study of relationships between biological molecules of interest. Common application areas include the understanding of the genetic mechanisms of organisms, viruses, or cancer and other genetic diseases. Much of the work is concerned with comparisons of deoxyribonucleic acid (DNA), ribonucleic acid (RNA), or proteins. DNA is a linear chain molecule, containing a sequence of the bases adenine (A), guanine (G), cytosine (C), and thymine (T), linked by a backbone. RNA is similar to DNA, but uses uracil (U) instead of thymine. Protein as produced is a linear chain molecule composed of a combination of the twenty amino acids in Table 2.1. We would like to exploit our knowledge of the relationship between DNA, RNA, and protein products, but there are complications.

2.2.1. The central dogma

The central dogma of molecular biology is illustrated in Figure 2.1. DNA serves as a template for the construction of an RNA copy in a process called tran-

Codes		Amino acid
A	Ala	Alanine
C	Cys	Cysteine
D	Asp	Aspartic Acid
E	Glu	Glutamic Acid
F	Phe	Phenylalanine
G	Gly	Glycine
H	His	Histidine
I	Ile	Isoleucine
K	Lys	Lysine
L	Leu	Leucine
M	Met	Methionine
N	Asn	Asparagine
P	Pro	Proline
Q	Gln	Glutamine
R	Arg	Arginine
S	Ser	Serine
T	Thr	Threonine
V	Val	Valine
W	Trp	Tryptophan
Y	Tyr	Tyrosine

Table 2.1. The standard amino acids.

scription. The transcribed RNA (messenger RNA, abbreviated as mRNA) is then used as a coded template to produce protein in the process known as translation. Triplets of RNA bases, called codons, specify each amino acid that is to be included in the protein, as shown in Table 2.2.

$$\begin{tikzcd} \text{DNA} \arrow[r, "\text{transcription}"'] & \text{mRNA} \arrow[r, "\text{translation}"'] & \text{Protein} \end{tikzcd}$$

Figure 2.1. The central dogma of molecular biology.

2.2.2. Complications

2.2.2.1. Coding and non-coding strands

The DNA molecule has directionality, and is usually present as double-stranded DNA (dsDNA), with the two strands running in opposite directions, as depicted in Figure 2.2. In dsDNA, each adenine base in one strand is paired with a thymine base in the other strand, and each cytosine base in one strand is paired with a guanine base in the other strand. Some bases code for genes, and some do not. In dsDNA, we often speak of a coding strand and a non-coding strand, wherein one strand codes for genes, and the other, complementary strand, running in the reverse direction, does not. In reality, either strand may contain genetic information at different locations in its sequence, and in some cases both strands actually code for genes along the same stretch of dsDNA.

	Codon (RNA)	Amino Acid
1	AAA	Lys
2	AAC	Asn
3	AAG	Lys
4	AAU	Asn
5	ACA	Thr
6	ACC	Thr
7	ACG	Thr
8	ACU	Thr
9	AGA	Arg
10	AGC	Ser
11	AGG	Arg
12	AGU	Ser
13	AUA	Ile
14	AUC	Ile
15	AUG	Met
16	AUU	Ile
17	CAA	Gln
18	CAC	His
19	CAG	Gln
20	CAU	His
21	CCA	Pro
22	CCC	Pro
23	CCG	Pro
24	CCU	Pro
25	CGA	Arg
26	CGC	Arg
27	CGG	Arg
28	CGU	Arg
29	CUA	Leu
30	CUC	Leu
31	CUG	Leu
32	CUU	Leu
33	GAA	Glu
34	GAC	Asp
35	GAG	Glu
36	GAU	Asp
37	GCA	Ala
38	GCC	Ala
39	GCG	Ala
40	GCU	Ala
41	GGA	Gly
42	GGC	Gly
43	GGG	Gly
44	GGU	Gly
45	GUA	Val
46	GUC	Val
47	GUG	Val
48	GUU	Val
49	UAA	END
50	UAC	Tyr
51	UAG	END
52	UAU	Tyr
53	UCA	Ser
54	UCC	Ser
55	UCG	Ser
56	UCU	Ser
57	UGA	END
58	UGC	Cys
59	UGG	Trp
60	UGU	Cys
61	UUA	Leu
62	UUC	Phe
63	UUG	Leu
64	UUU	Phe

Table 2.2. Codon translation table (standard).

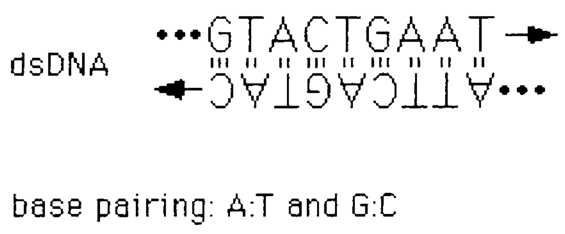


Figure 2.2. Double-stranded DNA.

2.2.2.2. Errors and Indels

Differences between reported DNA sequences and the actual DNA, and between reported DNA from different species, may arise naturally or as the result of errors in the lab. One case of this is the substitution of one DNA base for another, whether actual or as a detection error. Another common problem area is that of indels (insertions and deletions of one or more DNA bases) in one DNA sequence relative to another. If the error is one of detection in the lab, identifying the function of the gene encoded becomes that much more difficult. If the difference is naturally occurring, we may be faced with a sequence that looks like a gene but is not actually expressed correctly, or with a sequence that has markedly changed relative to another while retaining the same function. Some of these problems involve a shift in reading frame. Six possible reading frames of a sequence, and their resultant protein products (only one of which is usually correct), are shown in Figure 2.3. In most cases, only one of the six reading frames of a sequence is ever used in a healthy organism.

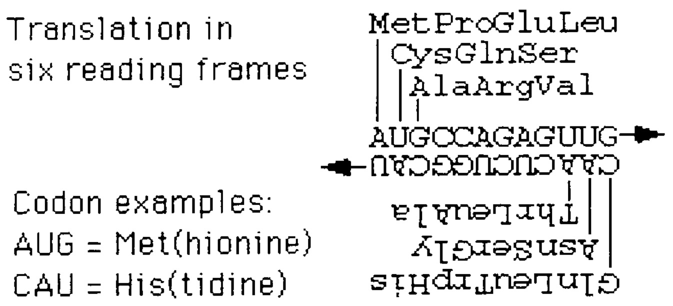


Figure 2.3. The six reading frames of translation.

2.2.2.3. Introns and Exons

In eukaryotes (organisms with nuclei), the DNA for a gene is often coded for by several parts called exons, which are separated by sections called introns. These introns must be removed from the RNA transcript, and the exons must be spliced together, before translation into protein takes place. This process is illustrated in Figure 2.4.

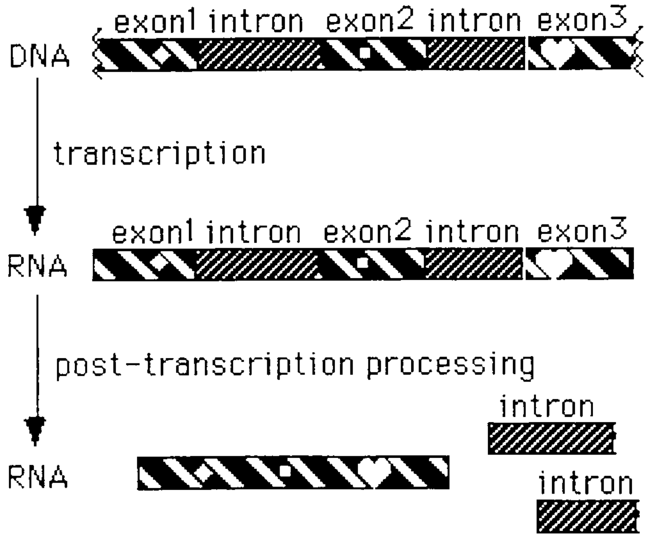

DNA
exon1 intron exon2 intron exon3
transcription
RNA
exon1 intron exon2 intron exon3
post-transcription processing
RNA
intron
intron

Figure 2.4. Introns and exons.

2.2.2.4. Transpositions

Two DNA sequences may differ by a transposition, as shown in Figure 2.5, where two sequences (the middle two in this case) are swapped in a copy of the DNA. This can occur via a number of mechanisms, including multiple inversions.

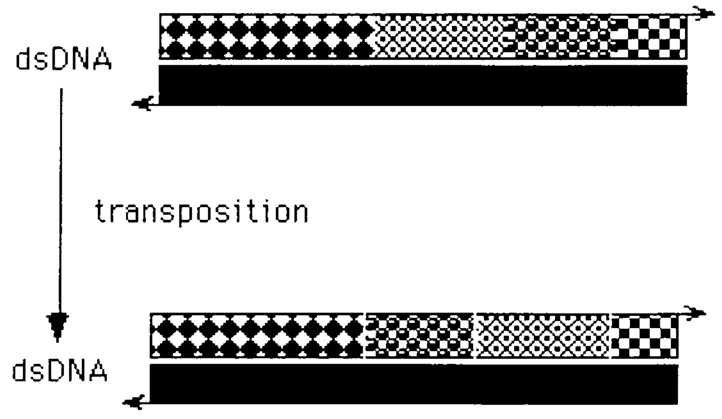


Figure 2.5. Transposition within dsDNA.

2.2.2.5. Inversions

Occasionally, a section of dsDNA gets inverted, as shown in Figure 2.6. Depending on where this occurs and how long a sequence is involved, genes may be changed, broken, regulated [20], or moved a great distance.

2.2.3. Summary

The central dogma provides a model which we may use to study relationships between DNA, RNA, and proteins, but comparison of these molecular sequences is complicated by many details, including errors, substitutions, insertions, deletions, gene segmentation into exons, reading frame shifts, transpositions, and inversions.

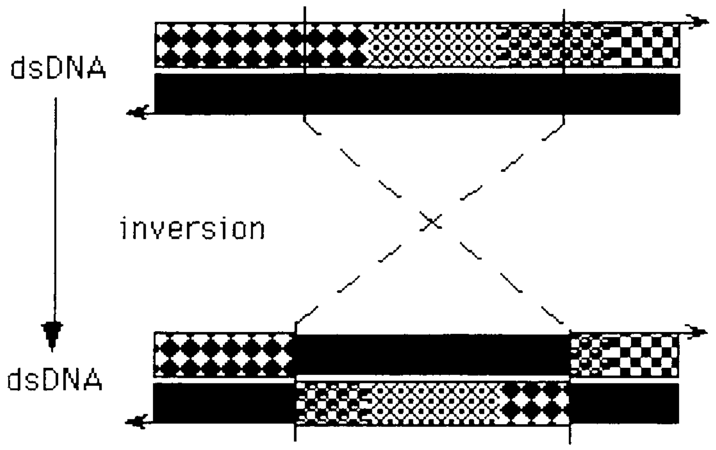


Figure 2.6. Inversion within dsDNA.

3. METHODS

The Walking Tree, a heuristic algorithm for approximate string matching, has been described in [1–7]. Here I recap some of its potential application areas in bioinformatics as they related to my investigations.

3.1. SEQUENCE MATCHING FOR GENE DISCOVERY

Similar sequences often have similar regulatory functions or code for related proteins. Thus, finding a close match between regions of two sequences may uncover related genes. There are many methods for sequence matching, including [8–10], and these vary in the degrees to which they handle the complications discussed in Section 2.2.2. The Walking Tree produces a similarity score by a method which seems to circumvent these difficulties. The Walking Tree can also produce an *inner alignment* between two sequences (an alignment which is no larger than the larger of the two sequences), providing information on the locations of mutually conserved regions which can provide clues to the nature of the relationship between the sequences. With Figures 3.1 through 3.3 I briefly illustrate the alignment capabilities of the Walking Tree with respect to inversions and translocations (transpositions). Figure 3.1 shows the true relationship between an artificial DNA sequence (on the left) and the same sequence after repeated inversions and translocations (right). Regions which are relatively inverted are shown in red, and those which are not are shown in green. Figure 3.2 shows a visualization of raw results from a Walking Tree alignment for these two sequences. Inverse complementary matches are shown in red, and direct matches are shown in green. Most of the features were detected. Figure 3.3 shows another visualization of the same Walking Tree alignment, with much of the noise filtered out by my software.

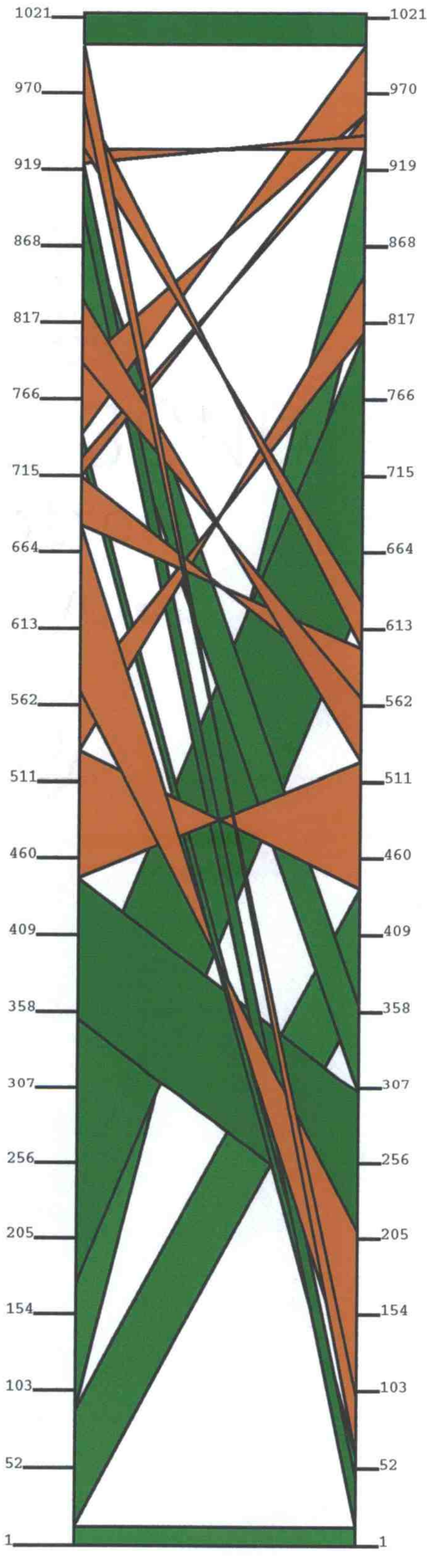


Figure 3.1. Artificial DNA and related sequence

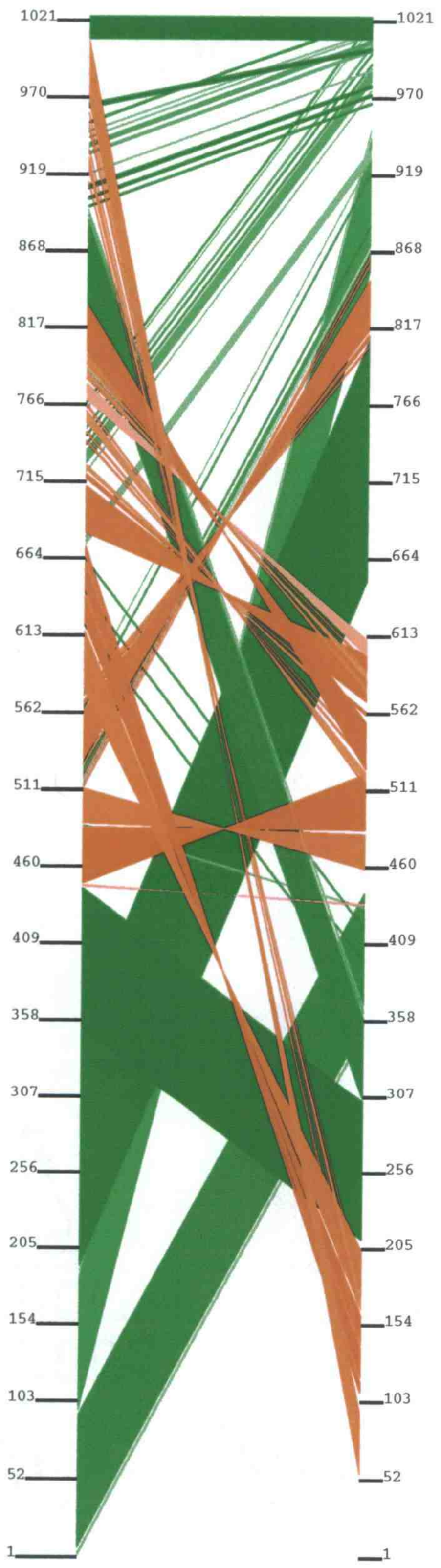


Figure 3.2. A Walking Tree alignment (raw)

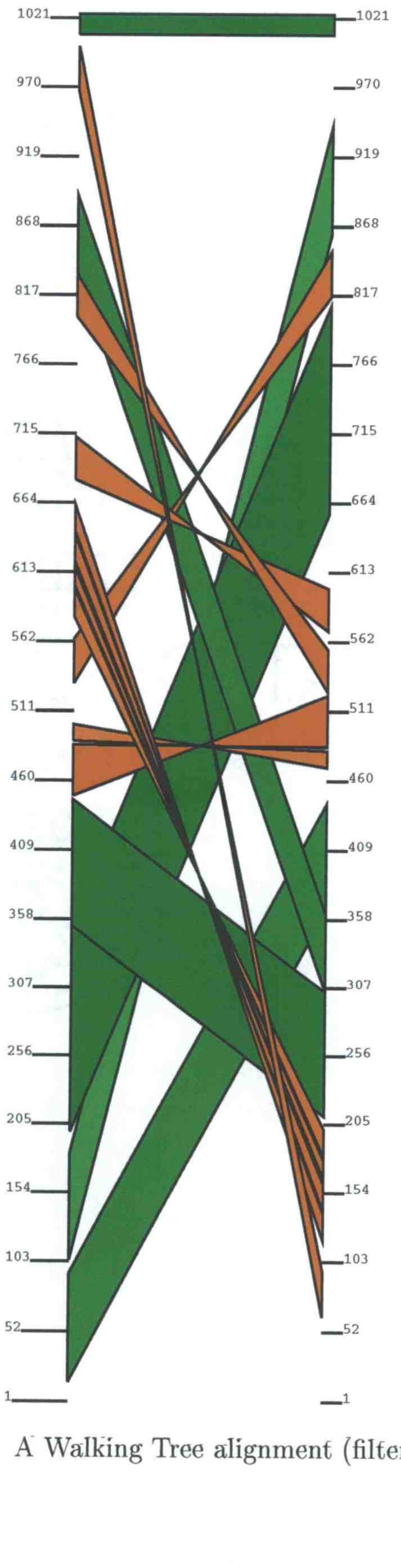


Figure 3.3. A Walking Tree alignment (filtered)

3.2. ANNOTATION OF GENETIC STRUCTURES AND GENOME ALIGNMENT

Many programs are available for classifying regions within DNA sequences, including BLAST [11] and GeneMark [12]. If large sequences which have many similarities are to be compared, the Walking Tree can be used to try to find the matching subsequences in the face of the complications mentioned in Section 2.2.2, and the two sequences may thusly be cross-annotated based upon previously classified regions. An example of this is shown in Figure 3.4, (discussed in more detail in Section 6.1.1) a visualization of sequence relationships between two mitochondrial genomes.

3.3. PHYLOGENETIC TREE CALCULATION

Phylogenetic trees serve to organize the relationships within a group of genetic sequences and the organisms from which they come. To produce a phylogenetic tree for related organisms, methods [19] are available which take as input a distance matrix containing pair-wise distances (based on arbitrary distance metrics) between the sequences. Similarity scores from the Walking Tree can be normalized to obtain approximate edit distances which can be used to construct the required matrix for a phylogenetic tree. Figure 3.5 shows one such tree which I produced for thirty-eight related viral genomes.

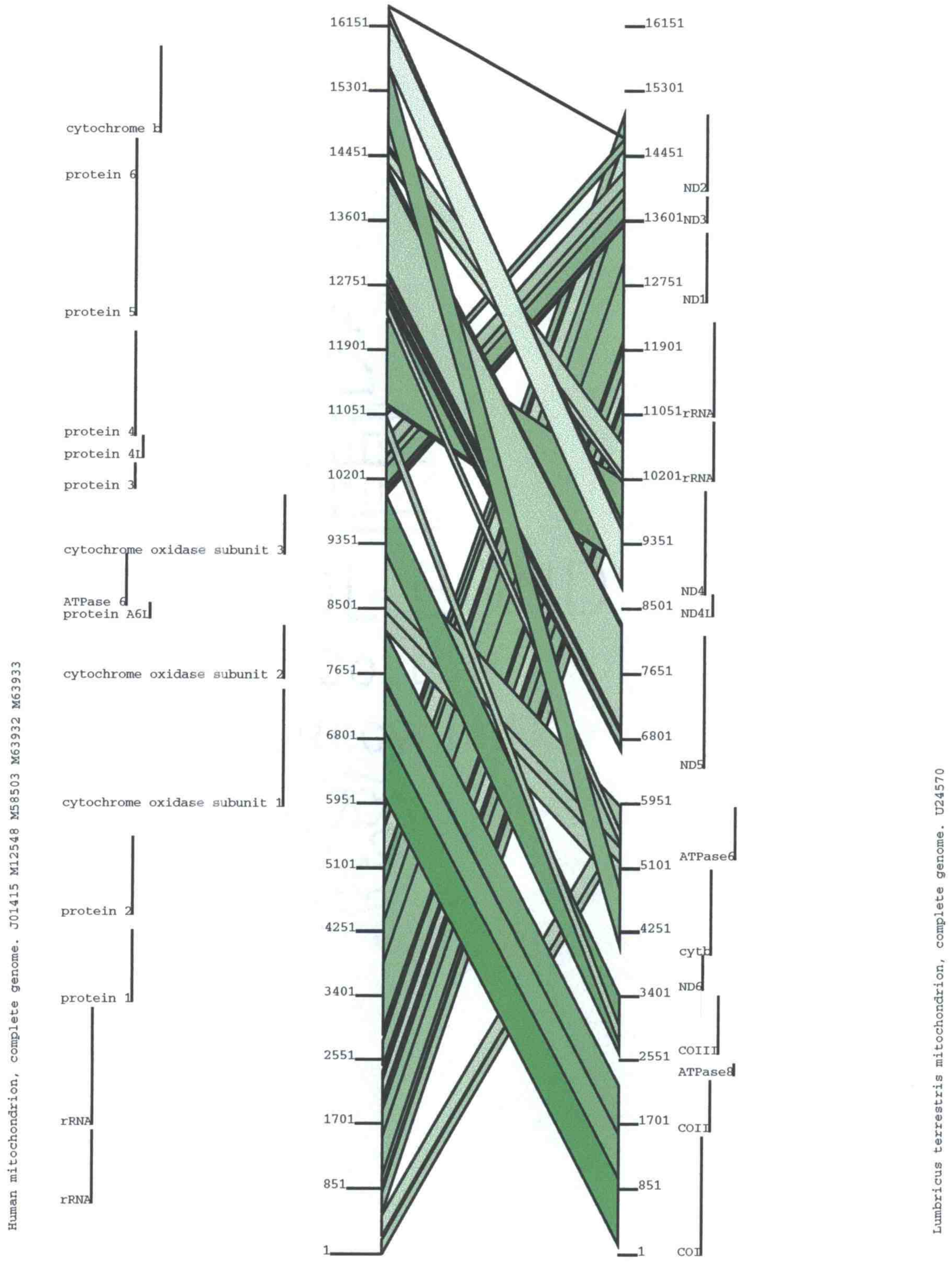


Figure 3.4. Visualization of a walking tree alignment of two mitochondrial genomes.

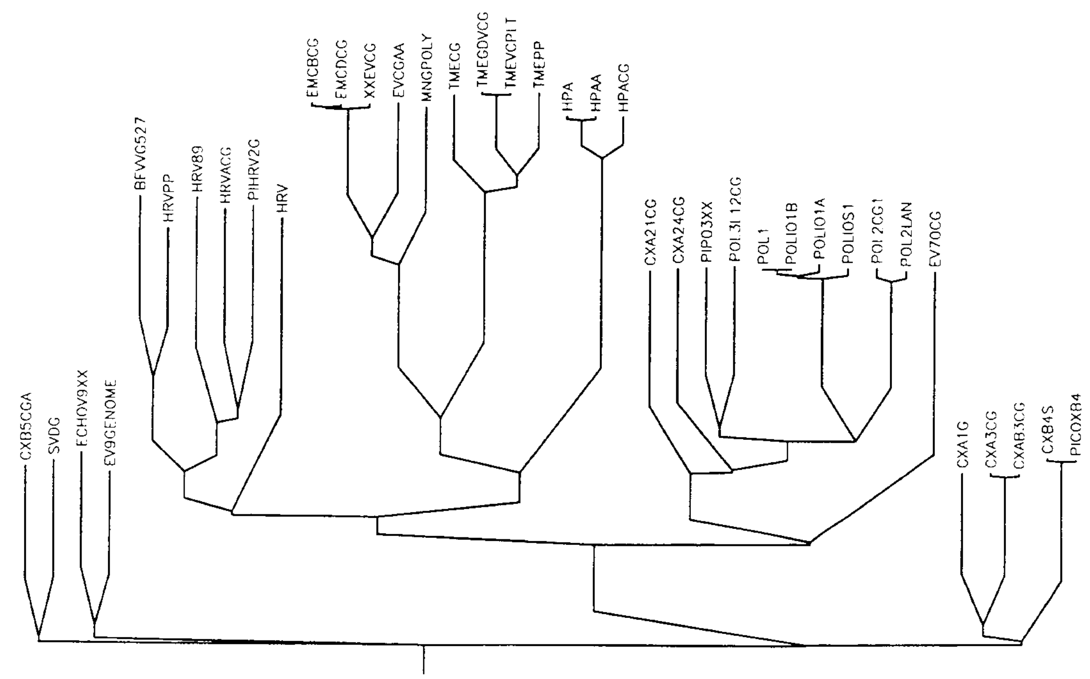


Figure 3.5. Phylogenetic tree of viruses based on pair-wise alignment scores.

4. APPROACHES AND CONSIDERATIONS

4.1. MATCHING AND SEQUENCE INTERPRETATION

4.1.1. Nucleic Acid Sequences

The visualization examples in Chapter 2 are based on identity comparisons between bases in DNA sequences. Figure 4.1 shows another example, in this case, a matching of the DNA sequences for genes for the protein Cytochrome C in two species. Such comparisons reveal genetic relationships. It should be noted that other comparisons are possible, particularly those of amino acid sequence comparison, which provide more information on protein function..

4.1.2. Amino Acid Sequences

Figure 4.2 is a comparison of the amino acid sequences for the protein Cytochrome C in two species, based on identity matching of the amino acids. A number of other amino-acid-based approaches to sequence comparison are possible, and the method used may be dictated by the form of the data available. The nature of these sequence interpretations is such that the use of different visualization methods for alignments produced by the Walking Tree may be beneficial, taking advantage of different measures of amino acid similarity.

4.1.2.1. Protein Back-translation

If a nucleic acid comparison is desired, but one or both sequences are proteins, amino acid sequences may be *back-translated* into inferred mRNA precursors. For example, given the tetramer MetProGluLeu in Figure 2.3, Table 2.2 can be used to infer the mRNA sequence one codon at a time. Starting with Met, only one codon, AUG, is found in the table, so the first three inferred bases

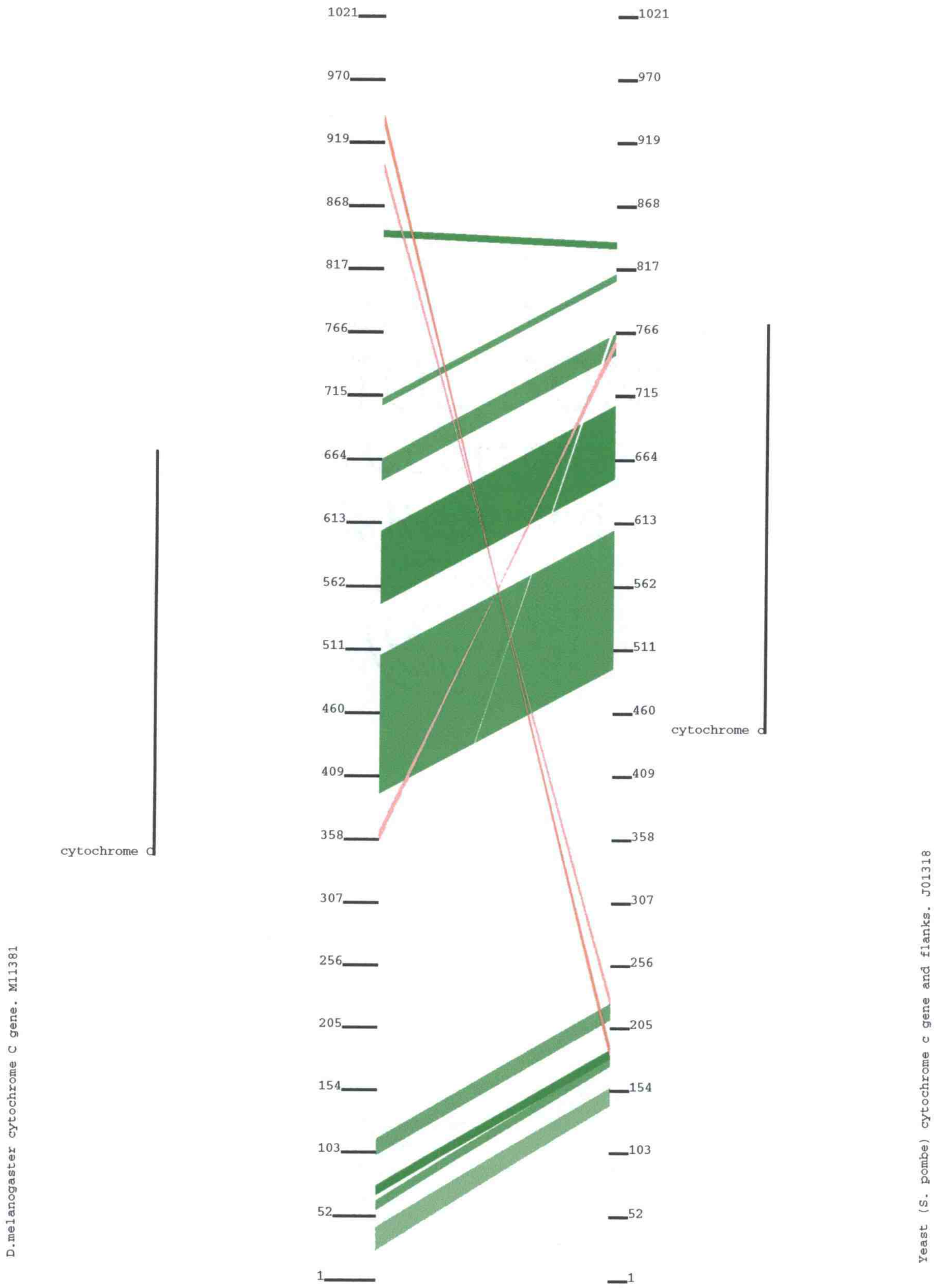


Figure 4.1. Cytochrome C - identity matching of DNA

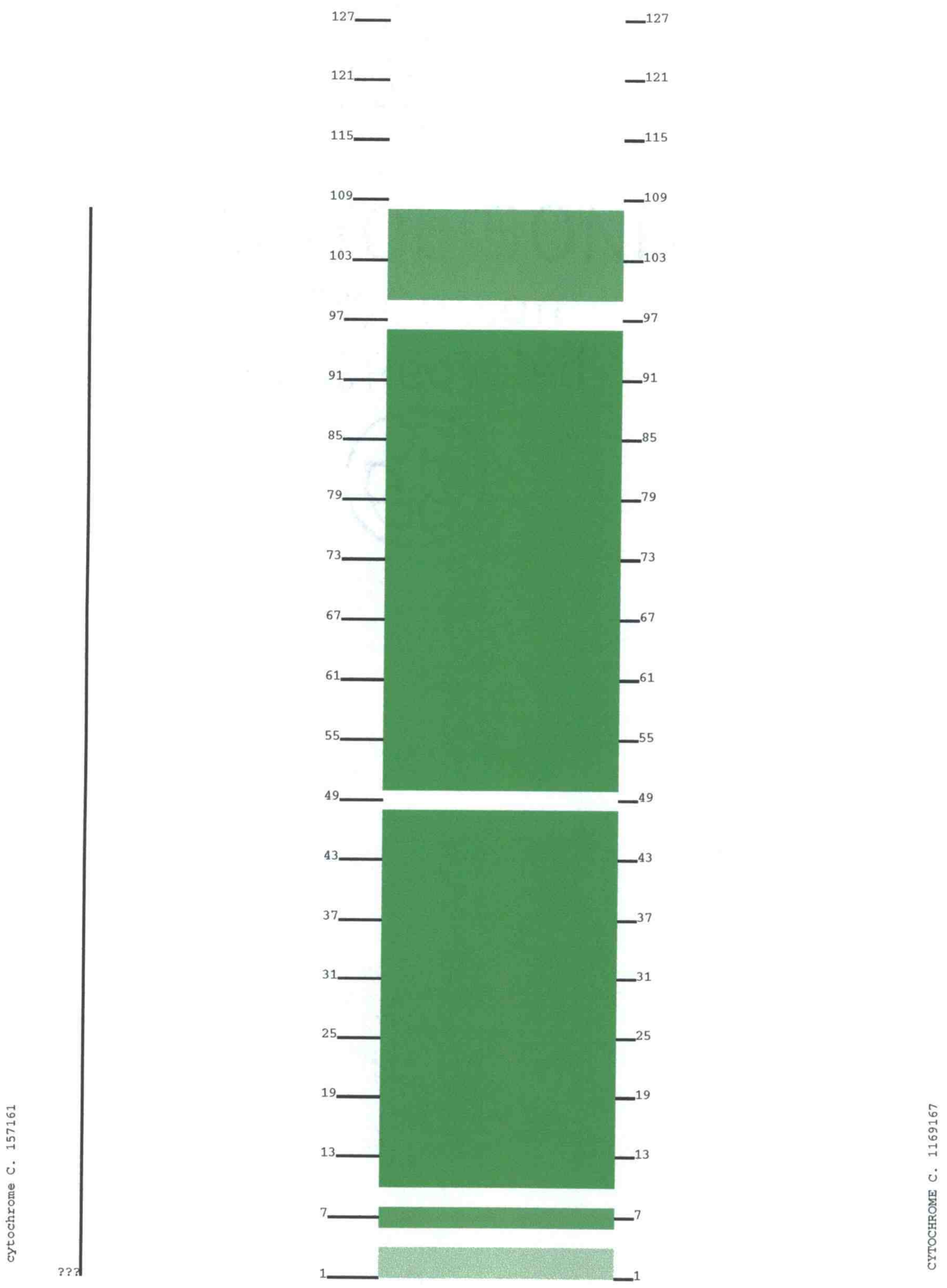


Figure 4.2. Cytochrome C - identity matching of amino acids

are AUG. For Pro, there are four possible codons, all four of which begin with CC. So far there are now four possible inferred sequences, AUGCCA, AUGCCC, AUGCCG, and AUGCCU. For the sake of convenience, these sequences may be written together as a single regular expression AUGCC[A,C,G,U]. For Glu, there are two codons from which to choose, GAA and GAG, extending our inference to eight possible sequences, AUGCC[A,C,G,U]GA[A,G]. For Leu, there are six different codons, UUA, UUG, CUA, CUC, CUG, and CUU. The ones beginning with uracil may be written as UU[A,G], and the four codons beginning with cytosine written as CU[A,C,G,U], so for Leu, our six codons are [UU[A,G],CU[A,C,G,U]]. So, for the amino acid sequence MetProGluLeu, the forty-eight mRNA sequences AUGCC[A,C,G,U]GA[A,G][UU[A,G],CU[A,C,G,U]] may be inferred and used for sequence matching. Alternatives to producing all possible back-translations include choosing codons based on their likelihoods, or using ambiguous nucleotide codes.

Codon likelihood Although the amino acids in our example sequence MetProGluLeu have several possible codons, some codons are more likely than others to be found in a particular type of protein or in a particular organism. Thus, a codon may be chosen over others based upon relative likelihood. For example, if our protein is known to have come from a human, a codon frequency table such as Table 4.3 [21] for *Homo sapiens* may be used to construct the most likely mRNA sequence. For our example, there is one choice of AUG for Met, and most likely choices of CCC for Pro, GAG for Glu, and CUG for Leu, for an inferred sequence of AUGCCCGAGCUG.

Ambiguous nucleotide codes Where the identities of nucleic acids in a sequence are ambiguous, extensions to nucleic acid representation such as that shown

Ala	GCA	0.13	end	UAG	0.21	Leu	UUA	0.02	Ser	UCA	0.05
Ala	GCG	0.17	end	UAA	0.23	Leu	CUA	0.03	Ser	UCG	0.09
Ala	GCU	0.17	end	UGA	0.55	Leu	CUU	0.05	Ser	AGU	0.10
Ala	GCC	0.53				Leu	UUG	0.06	Ser	UCU	0.13
			Gln	CAA	0.12	Leu	CUC	0.26	Ser	UCC	0.28
Arg	CGA	0.06	Gln	CAG	0.88	Leu	CUG	0.58	Ser	AGC	0.34
Arg	CGU	0.07									
Arg	AGA	0.10	Glu	GAA	0.25	Lys	AAA	0.18	Thr	ACA	0.14
Arg	AGG	0.18	Glu	GAG	0.75	Lys	AAG	0.82	Thr	ACU	0.14
Arg	CGG	0.21							Thr	ACG	0.15
Arg	CGC	0.37	Gly	GGU	0.12	Met	AUG	1.00	Thr	ACC	0.57
			Gly	GGA	0.14						
Asn	AAU	0.22	Gly	GGG	0.24	Phe	UUU	0.20	Trp	UGG	1.00
Asn	AAC	0.78	Gly	GGC	0.50	Phe	UUC	0.80			
									Tyr	UAU	0.26
Asp	GAU	0.25	His	CAU	0.21	Pro	CCA	0.16	Tyr	UAC	0.74
Asp	GAC	0.75	His	CAC	0.79	Pro	CCG	0.17			
						Pro	CCU	0.19	Val	GUA	0.05
Cys	UGU	0.32	Ile	AUA	0.05	Pro	CCC	0.48	Val	GUU	0.07
Cys	UGC	0.68	Ile	AUU	0.18				Val	GUC	0.25
			Ile	AUC	0.77				Val	GUG	0.64

Table 4.3. Codon frequency for highly expressed human genes

in Table 4.4 [22] are often used. Using Table 4.3 and Table 4.4, our example protein MetProGluLeu may be back-translated to the ambiguous sequence AUGCCNGARYUN, with AUG for Met, CCN for Pro, GAR for Glu, and YUN for Leu.

4.1.2.2. DNA Translation

DNA sequences may be translated (using an appropriate translation table, such as Table 2.2) in all six reading frames (see Figure 2.3) and the resulting amino acid sequences used in matching. This allows nucleic acid sequences to be compared to other sequences based upon the similarities of the amino acids for which they may code.

Amb. Code	Possible Nucleotides			
N	A	C	G	U
V	A	C	G	
H	A	C	U	
D	A	G	U	
B	C	G	U	
M	A	C		
R	A	G		
W	A	U		
K	G	U		
S	C	G		
Y	C	U		
A	A			
C	C			
G	G			
U	U			

Table 4.4. Common ambiguous nucleotide codes.

4.1.2.3. Amino Acid Similarity

Amino acids may be compared based on a variety of properties, including electric charge, hydrophobicity, size, codon mutation tolerance and distance, and roles typically played in secondary, tertiary and quaternary structures, all of which may be important to protein structure and function. The most widely successful comparison methods based on these properties are the intricate PAM and BLOSUM matrix families, including PAM250 [23] and BLOSUM62 [24], but for many applications, a simple binary measure of hydrophobicity is the method of choice.

4.1.2.4. Hydrophobicity

Hydrophobicity here is a measure of whether an amino acid residue's side-chain tends not to associate with water molecules. This serves as a predictor of whether or not the amino acid will be exposed to water in a protein's folded structure. Because related or similar proteins typically have hydrophobic regions in common, this quality is useful in matching sequences.

Table 4.5 [25] shows binary hydrophobicities for the standard amino acids. The primary advantage of reducing hydrophobicity to a binary variable is a computational one. A single amino acid, the product of a three-nucleotide codon, may be represented by one binary bit, sequences of which may be efficiently stored, manipulated, and compared.

Residue	Hydrophobic
Ala	1
Arg	0
Asn	0
Asp	0
Cys	1
Gln	0
Glu	0
Gly	1
His	0
Ile	1
Leu	1
Lys	0
Met	1
Phe	1
Pro	0
Ser	0
Thr	0
Trp	1
Tyr	1
Val	1

Table 4.5. Binary hydrophobicity of common amino acid residues.

Figure 4.3 shows a comparison of the amino acid sequences for the protein Cytochrome C in two species, based on the binary hydrophobicity of the amino acids. Gaps in the alignment are primarily due to false inversions [30] which have been filtered out. This suggests that in some cases the Walking Tree is more useful without the inversion capability.

4.1.2.5. Matching and Visualization

Although the matching and visualization studies herein are based on base identity between DNA sequences, other routes to comparison exist via translation and back-translation of sequence data. With protein sequence data, the application and rendering of knowledge of amino acid residue properties such as charge, size, typical roles, codon mutation distance, and hydrophobicity are possible.

4.2. DESIGN CONSIDERATIONS

It was desirable to produce a system amenable to the use of additional comparison methods such as those above.

To this end, I wrote programs allowing for the use of additional data input formats, alignment programs, visualization bases, and output formats.

I have initially configured the system for the comparison of DNA sequences based on identity and complementarity, with the GenBank format for data input, an existing Walking Tree implementation [26], and the PostScript format for graphical output.

4.2.1. Pre-alignment

I chose the GenBank format for input because of its widespread usage, simplicity, online availability, and incorporation of DNA, RNA, protein, and annotation data together in one file.

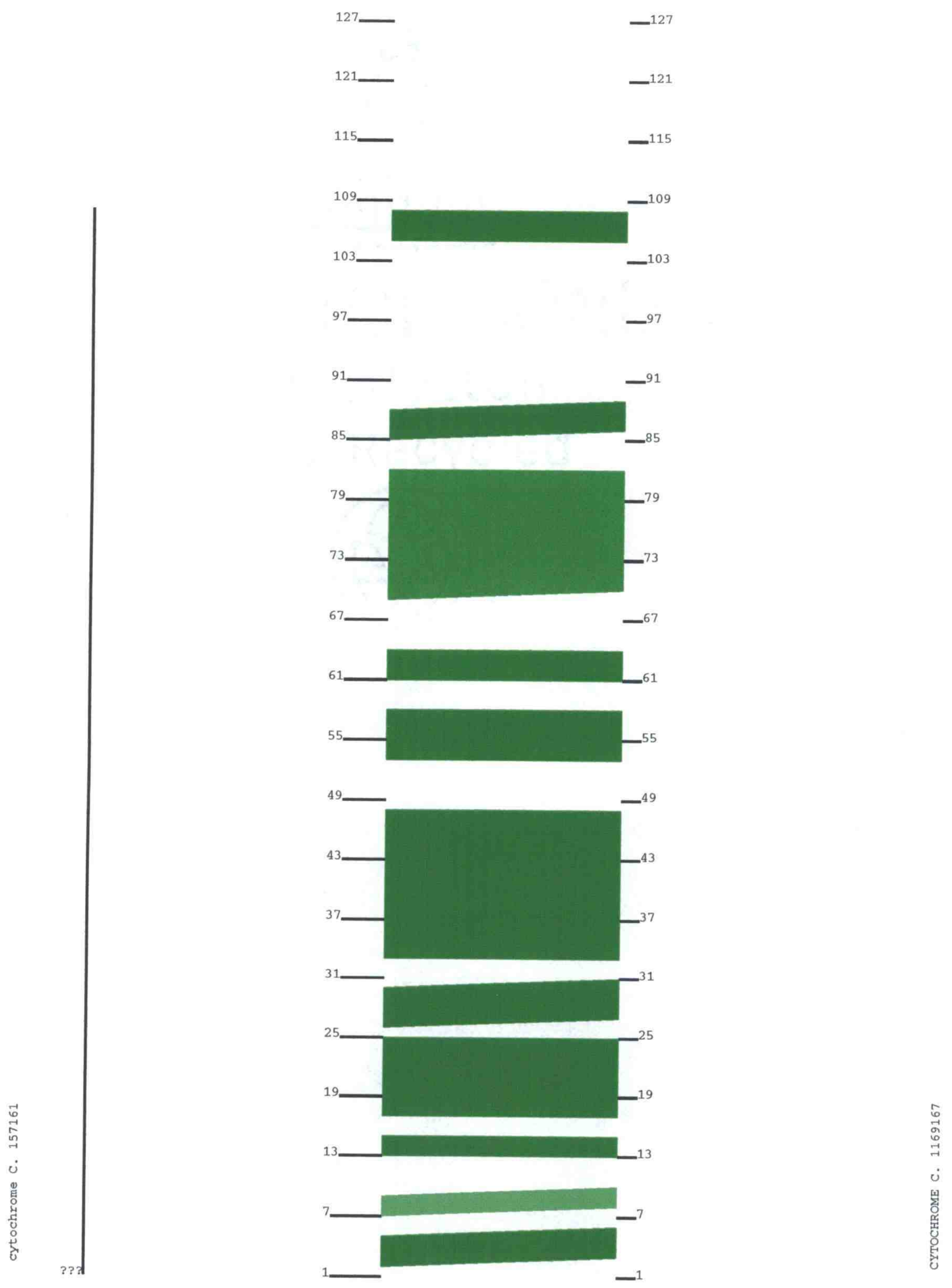


Figure 4.3. Cytochrome C - hydrophobicity matching of amino acids

4.2.2. Alignment

I chose the Walking Tree implementation of [26] because it had been well tested by prior usage, is highly portable, and was immediately available.

It should be noted that [26] is a single-processor implementation that takes time proportional to the product of the lengths of the two sequences being compared, which is prohibitive for very large sequences, and so parallel implementations are needed for large genomes. I have implemented a parallel version of the Walking Tree (direct tree only) on the Sequent Balance and noted a near-linear speedup, duplicating the parallelization results of [7], and I am also working on a checkpointing Walking Tree implementation in C++ for the PVM [18].

4.2.3. Visualization rendering

I chose the PostScript format for rendering visualizations because of its support of subroutines and vector graphics (which reduce file size), the widespread installed base of free viewing software on all major platforms, and the availability of PostScript printers for standard and oversized color printing.

5. IMPLEMENTATION

5.1. WWW INTERFACE

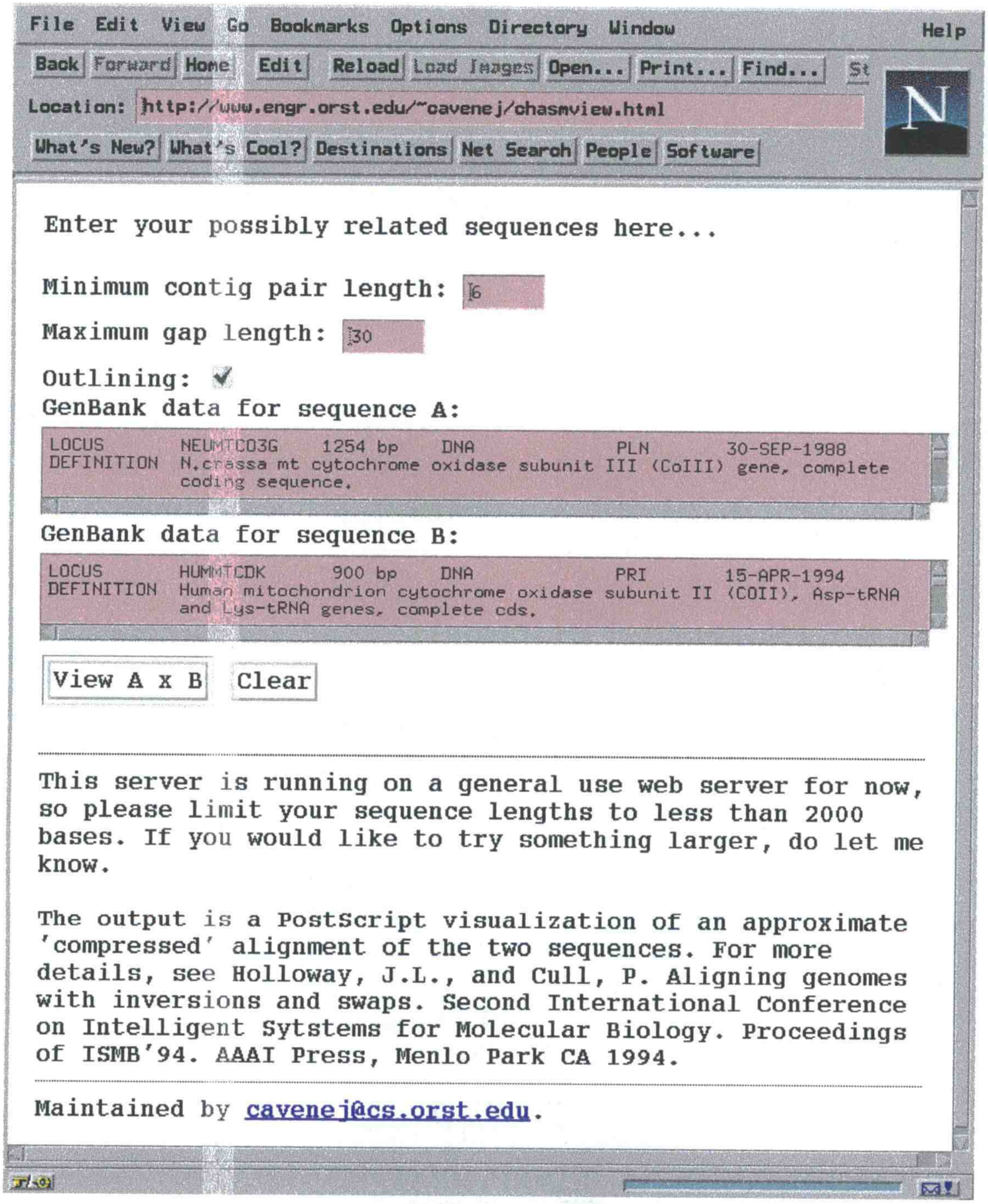
I developed a web page, shown in Figure 5.1, to allow others convenient use of the alignment visualization software. My compiled CGI program processes submitted data and provides the user with a PostScript format visualization of the alignment. This format is suitable for viewing or high quality printing. Refer to Chapter 5 for details on the visualization parameters available on the web page. Figure 5.1

5.2. GRAPHICAL INTERFACE

Before I created a WWW interface, I prototyped a graphical user interface for alignment visualization in Tcl/Tk. I chose Tcl/Tk because the GUI would be easy to modify, and because the Canvas widget would allow easy generation of PostScript output.

A screenshot of the original main window is shown in Figure 5.2. The help dialogs are shown in Figures 5.3, 5.4, and 5.5. The dialogs for alignment generation, viewing, and printing are shown in Figures 5.6, 5.7, 5.8, and 5.9.

I found the Tcl/Tk PostScript generation capabilities unsatisfactory, and changed to a completely customized PostScript model of representation for visualization and printing. It was easy to modify the GUI to call my new command-line interface code for alignment and visualization.


File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Edit Reload Load Images Open... Print... Find... St
What's New? What's Cool? Destinations Net Search People Software

Location: http://www.engr.orst.edu/~cavenej/chasmview.html

Enter your possibly related sequences here...

Minimum contig pair length: 6

Maximum gap length: 30

Outlining: ✓

GenBank data for sequence A:

```
LOCUS       NEUMTCO3G     1254 bp    DNA             PLN       30-SEP-1988
DEFINITION  N.crassa mt cytochrome oxidase subunit III (CoIII) gene, complete
            coding sequence.
```

GenBank data for sequence B:

```
LOCUS       HUMMTCDK       900 bp    DNA             PRI       15-APR-1994
DEFINITION  Human mitochondrion cytochrome oxidase subunit II (COII), Asp-tRNA
            and Lys-tRNA genes, complete cds.
```

View A x B   Clear

This server is running on a general use web server for now, so please limit your sequence lengths to less than 2000 bases. If you would like to try something larger, do let me know.

The output is a PostScript visualization of an approximate 'compressed' alignment of the two sequences. For more details, see Holloway, J.L., and Cull, P. Aligning genomes with inversions and swaps. Second International Conference on Intelligent Sytstems for Molecular Biology. Proceedings of ISMB'94. AAAI Press, Menlo Park CA 1994.

Maintained by cavenej@cs.orst.edu.

Figure 5.1. WWW access to a walking tree implementation.

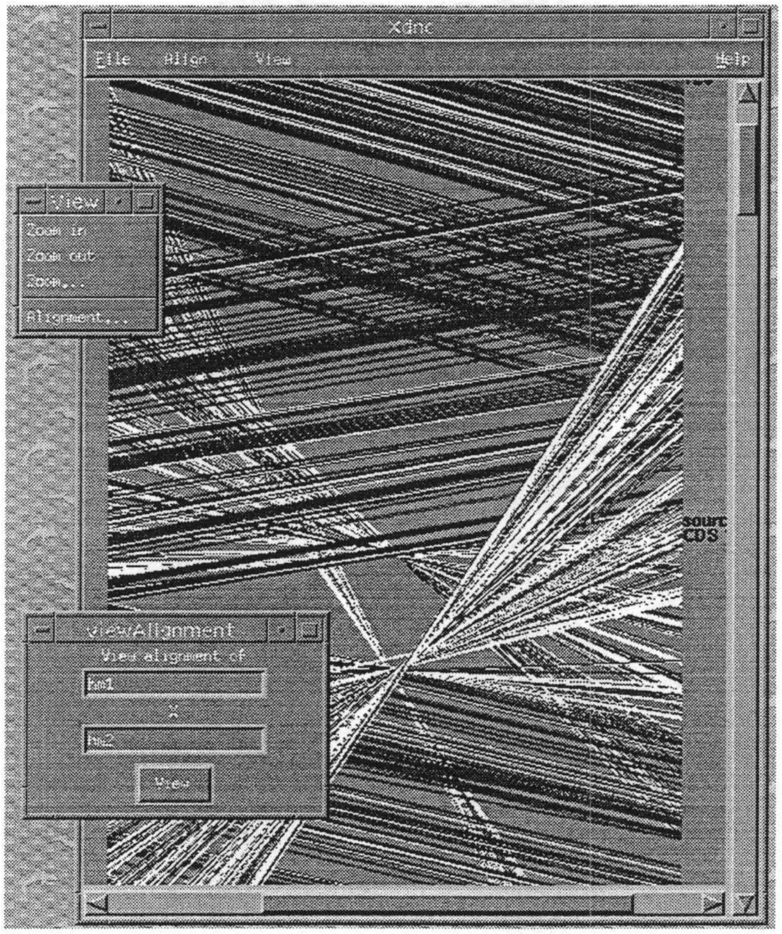


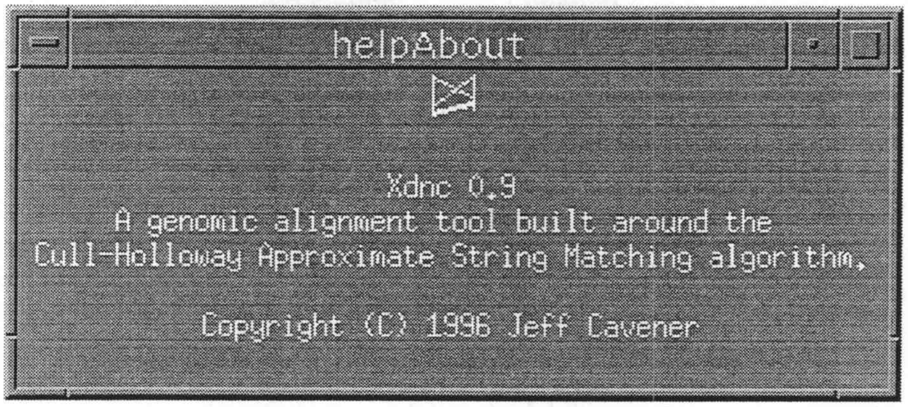
Figure 5.2. A first GUI.

5.2.1. A Sample Run

Figures 5.10 through 5.15 show the steps necessary to produce an alignment visualization via the graphical interface for two arbitrary GenBank data files, cyt2 and cyt3.

5.2.2. Invocation

The graphical interface is started by the Tcl/Tk script xdnc (Figure 5.10).


helpAbout

Xdnc 0.9
A genomic alignment tool built around the
Cull-Holloway Approximate String Matching algorithm.

Copyright (C) 1996 Jeff Cavener

Figure 5.3. The About dialog window.

5.2.3. Bringing up the Align dialog

The Align dialog is brought up via the Align menu (Figure 5.11).

5.2.4. Performing an alignment

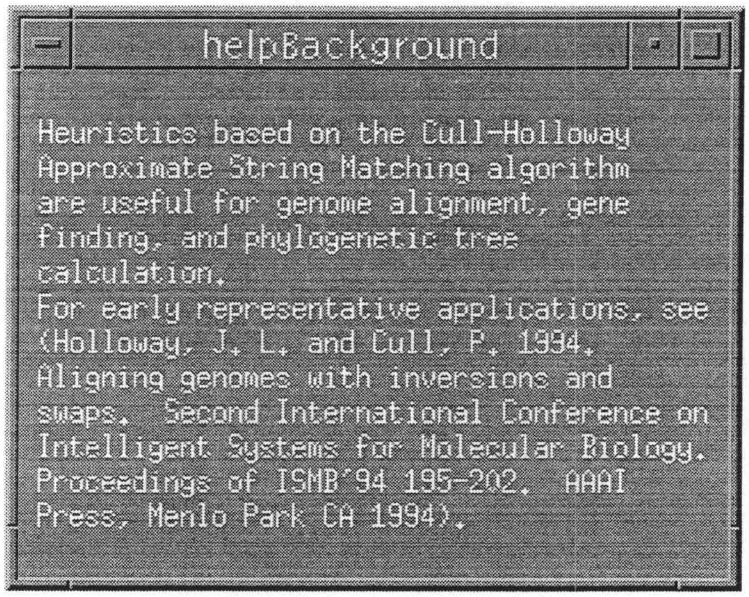
Figure 5.12 shows the Align dialog, in which the names of two local GenBank data files are entered. After the Align button is pushed, alignment is performed, and the resultant alignment visualization appears (Figure 5.13).

5.2.5. Printing the alignment visualization

The Print dialog is brought up via the Print item in the File menu (Figure 5.14), and desired printing options are selected in the Print dialog (Figure 5.15).

5.3. COMMAND-LINE INTERFACE AND UTILITIES

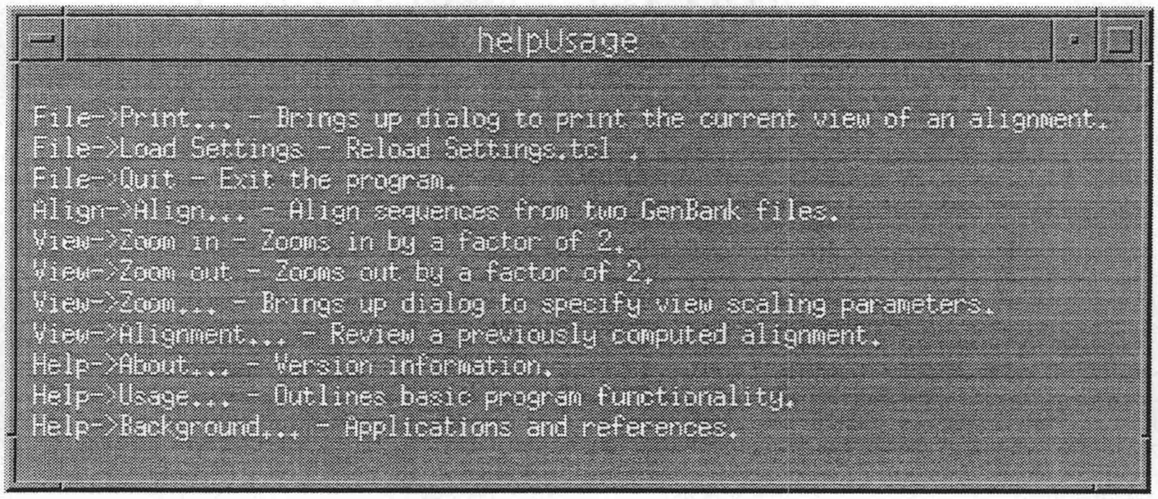
A variety of scripts are provided to aid the user in the initiation of alignments and the production of visualizations. Of particular interest to the new user will



Heuristics based on the Cull-Holloway
Approximate String Matching algorithm
are useful for genome alignment, gene
finding, and phylogenetic tree
calculation.
For early representative applications, see
(Holloway, J. L. and Cull, P. 1994.
Aligning genomes with inversions and
swaps. Second International Conference on
Intelligent Systems for Molecular Biology.
Proceedings of ISMB'94 195-202. AAAI
Press, Menlo Park CA 1994).

Figure 5.4. The Background dialog window.

be the programs Align and View, and the PostScript post-processing utilities outlinePSContigs and adjustPSMargins. See the command reference section (pages 63 through 93 of Appendix A) for details.



```
File->Print... - Brings up dialog to print the current view of an alignment.
File->Load Settings - Reload Settings.tcl .
File->Quit - Exit the program.
Align->Align... - Align sequences from two GenBank files.
View->Zoom in - Zooms in by a factor of 2.
View->Zoom out - Zooms out by a factor of 2.
View->Zoom... - Brings up dialog to specify view scaling parameters.
View->Alignment... - Review a previously computed alignment.
Help->About... - Version information.
Help->Usage... - Outlines basic program functionality.
Help->Background... - Applications and references.
```

Figure 5.5. The Usage dialog window.

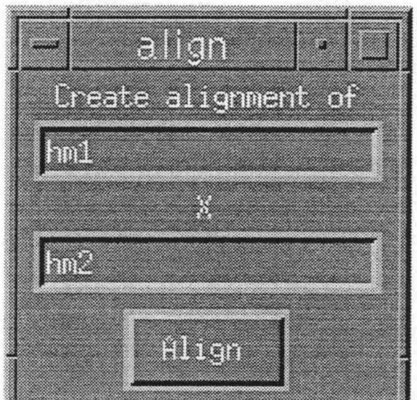


Figure 5.6. The Align dialog window.

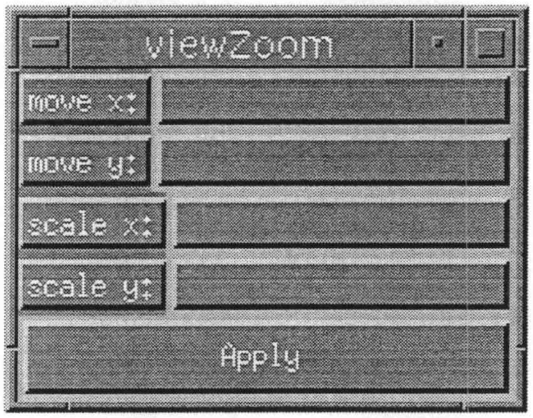
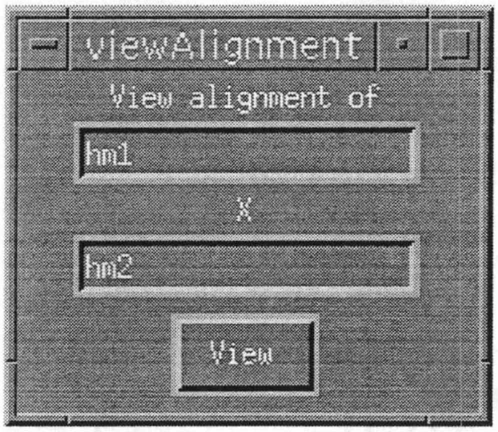


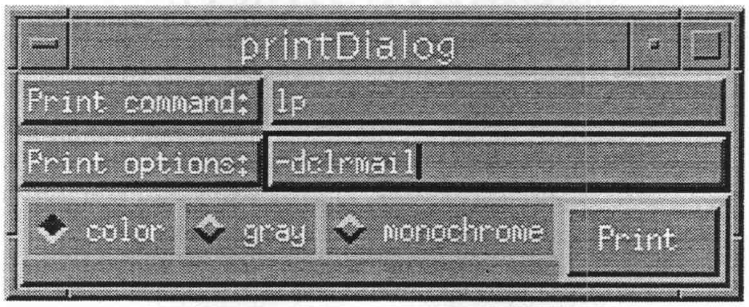
Figure 5.7. The Zoom dialog window.



hm1

hm2

Figure 5.8. The View Alignment dialog window.

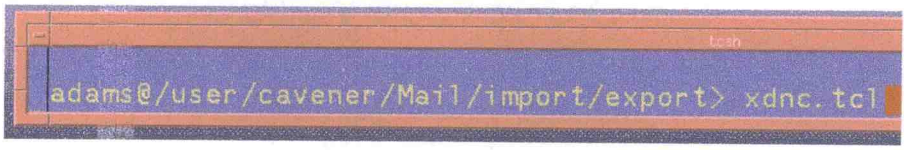


Print command: lp

Print options: -dclrmail

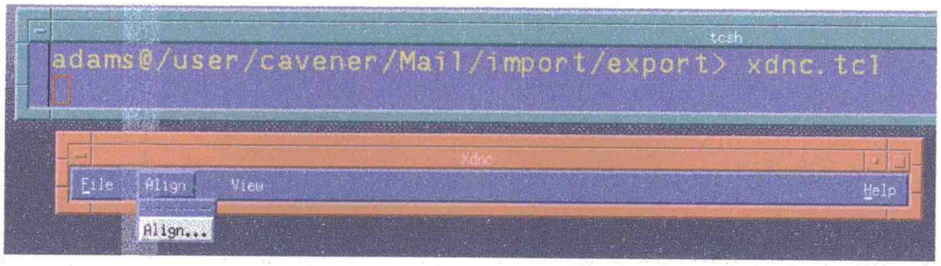
[✓ color] gray monochrome

Figure 5.9. The Print dialog window.



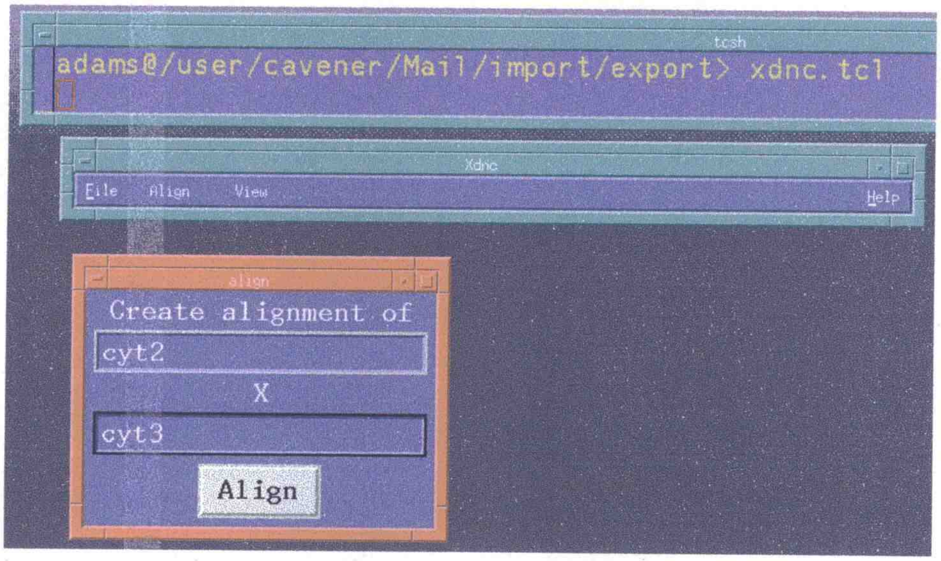
```
adams@/user/cavener/Mail/import/export> xdnc.tcl
```

Figure 5.10. GUI Invocation



```
adams@/user/cavener/Mail/import/export> xdnc.tcl
```

Figure 5.11. Bringing up the Align dialog



```
adams@/user/cavener/Mail/import/export> xdnc.tcl
```

Figure 5.12. Starting an alignment

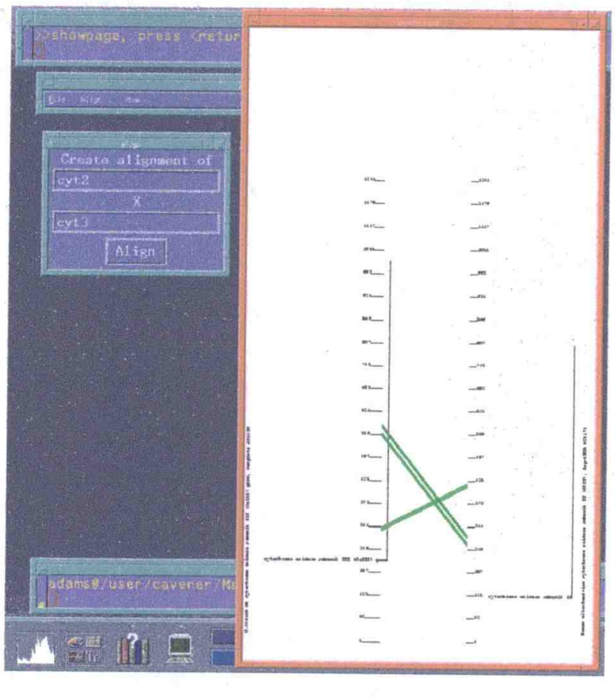
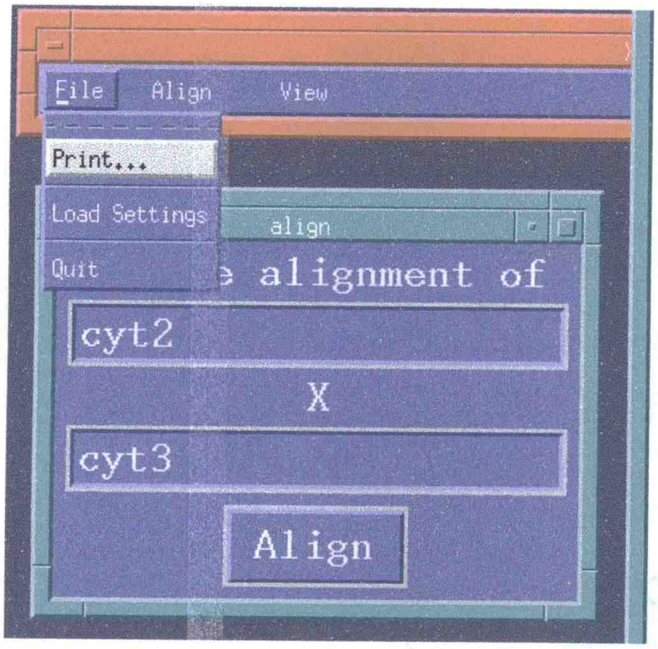


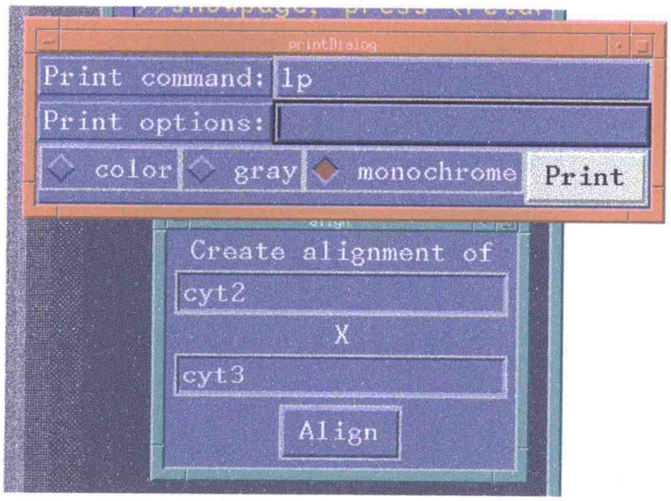
Figure 5.13. The resultant alignment visualization



cyt2

cyt3

Figure 5.14. Bringing up the print dialog



printDialog

Print command: lp

Print options:

color gray [✓ monochrome]

Figure 5.15. Printing options

6. APPLICATION

6.1. GENE ANNOTATION AND GENOME ALIGNMENT

6.1.1. An Alignment of Two Mitochondrial Genomes

Figure 3.4 on page 22 shows a visualization of an automated comparison of the complete mitochondrial genomes of the human [13] and the common earthworm [14]. I produced the visualization with my programs Align.pl [15], View.pl, and outlinePSContigs.pl from GenBank [16] entries for the two sequences. It is easily seen that the method produced reasonable matches between the known related regions of the sequences in most cases, e.g. the cytochrome oxidase subunits 1 and 2 of the human mitochondrion (left) match the corresponding genes COII and COIII of the worm mithchondrion (right), and protein 4 of the human matches the corresponding protein ND4 of the worm. The content of the visualizations produced from the alignment data may however be varied by adjusting the maximum gap length and the minimum contig length.

6.1.2. Maximum Gap Length

If there is a poorly conserved region within a gene, matching that same gene from two different species may result in two separate partial matches for the gene, with a gap in between them due to the poorly conserved region. By specifying a gap length which may be bridged to join separately matching regions, matches may be combined to aid in the identification of the gene as a single feature. Figures 6.1 through 6.3 show the effect of increasing the maximum gap length. In Figure 6.1, with a maximum gap length of one, there are hundreds of tightly packed matches, many more than the number of genes (and ribosomal subunits). Increasing the maximum to sixteen (Figure 6.2) results here in a slightly noticeable improvement. With the maximum increased to 64, pronounced matches for protein 4/ND4, cytochrome oxidase subunit 3/COIII, ATPase6, and cytochrome oxidase subunits 1,2/COI,II appear. Note that I used a constant minimum contig length of six, and that different results will be obtained by varying the minimum contig length parameter.

6.1.3. Minimum Contig Length

The inner alignment data provided by the walking tree contains subsequence matches of length increasing with the degree of conservation within a region of the genome. Here I show the effect of filtering out the lesser of these contiguous (gapless) regions by varying the minimum contig length. Note that here, I used a constant maximum gap length of one, and that different results will be obtained with lower and higher parameters. In Figure 6.4, contiguously matched regions of thirty-six bases or more are shown. Many appear in a band for cytochrome oxidase subunit 1, and matches are also shown (from the top left) within cytochrome b, NADH subunit 4, NADH subunit 2, and 16s ribosomal RNA. The lightness of the match within the 16s ribosomal RNA represents a low percentage base identity within the match. In Figure 6.5, done with a minimum contig length of twelve, matches appear for all of the shown features, excepting NADH subunits 6, 4L, and 3, and ATPase subunit 8 on the right. With a minimum contig length of six, (Figure 6.6), unlikely matches appear for these four subunits, as well as other noise, and with a minimum contig length of one, (Figure 6.7) all matches are shown as assigned in the raw walking tree alignment data, including questionable low significance inverted matches (in red) and noise.

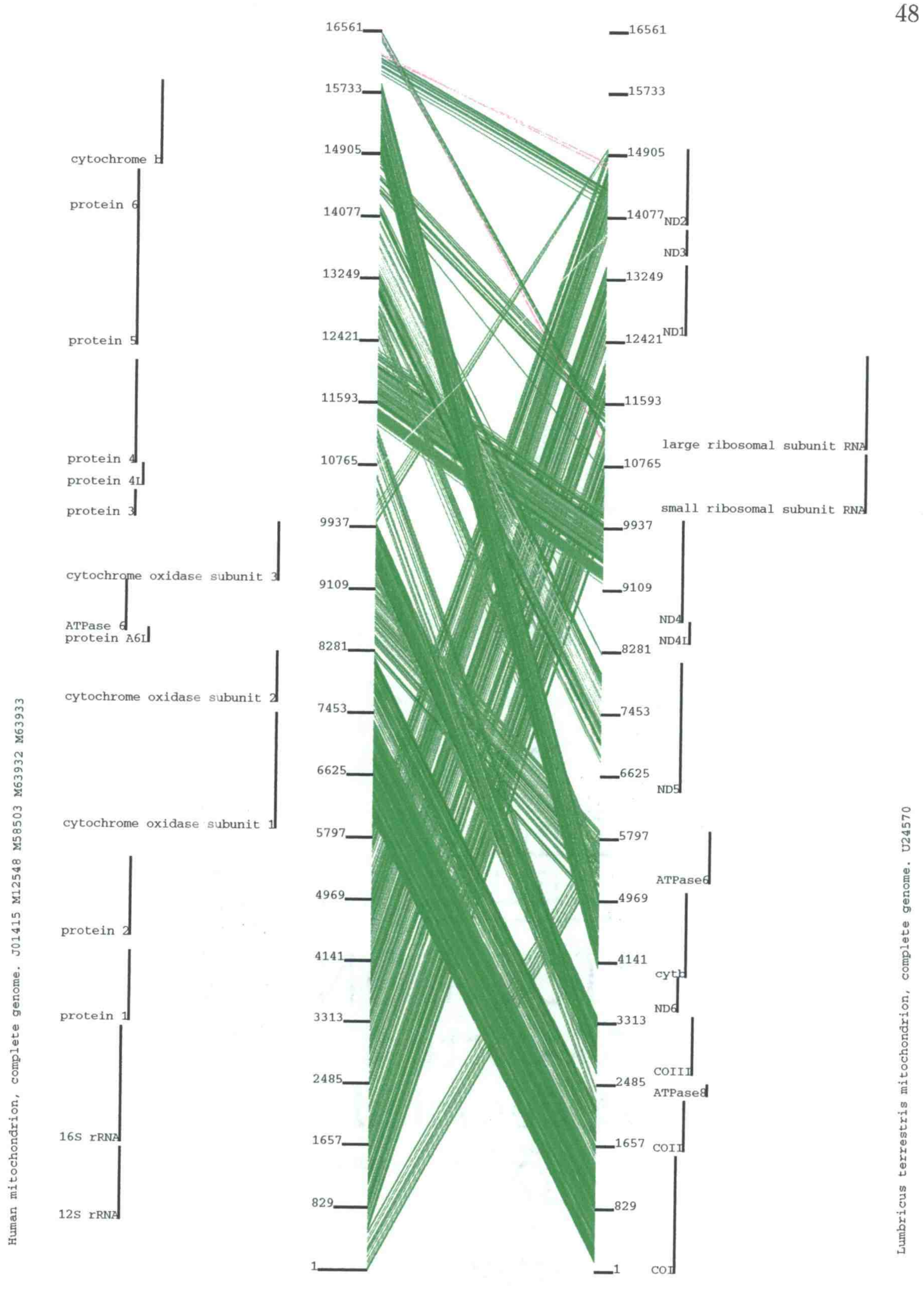


Figure 6.1. Maximum Gap Length = 1

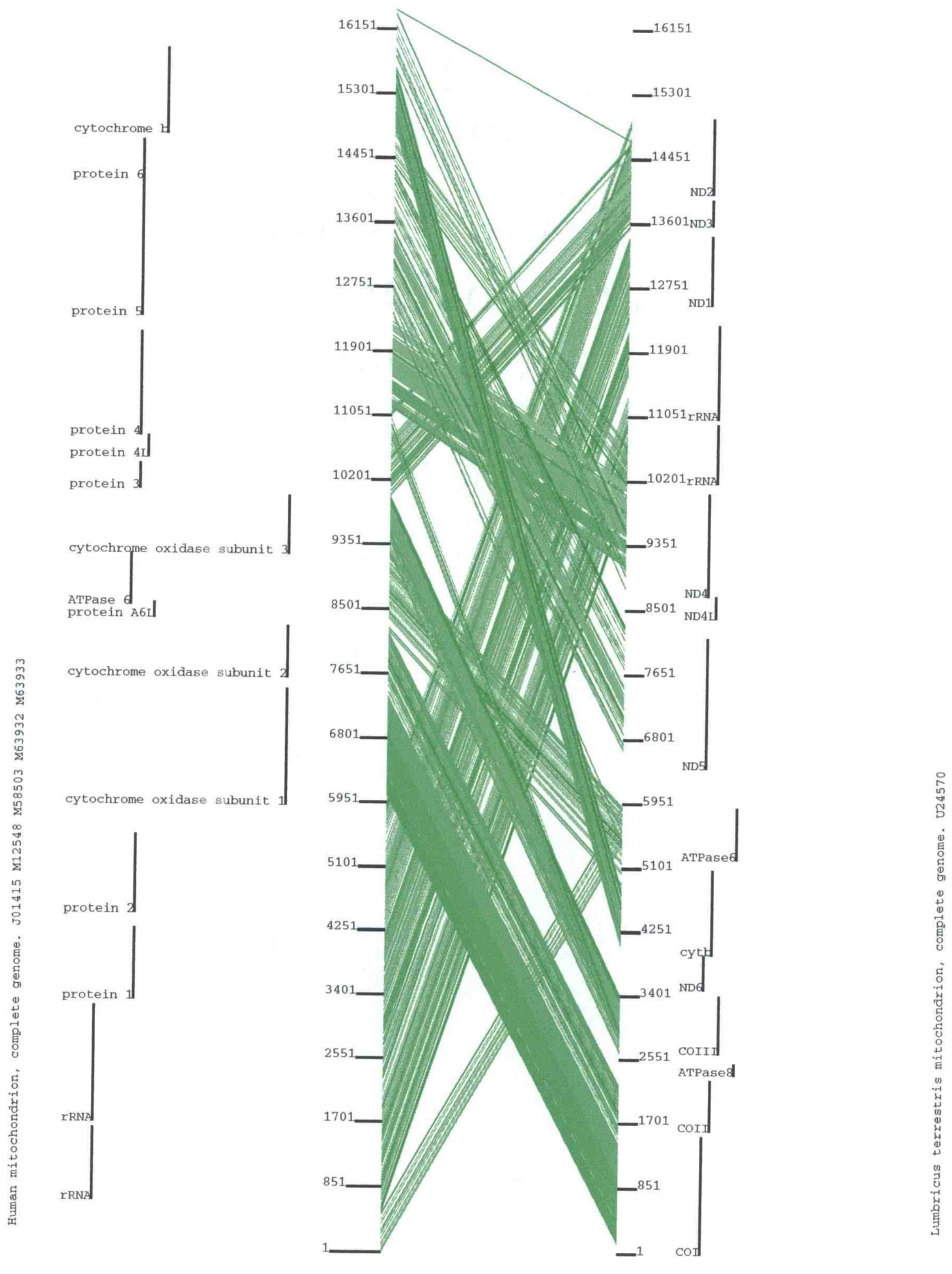


Figure 6.2. Maximum Gap Length = 16

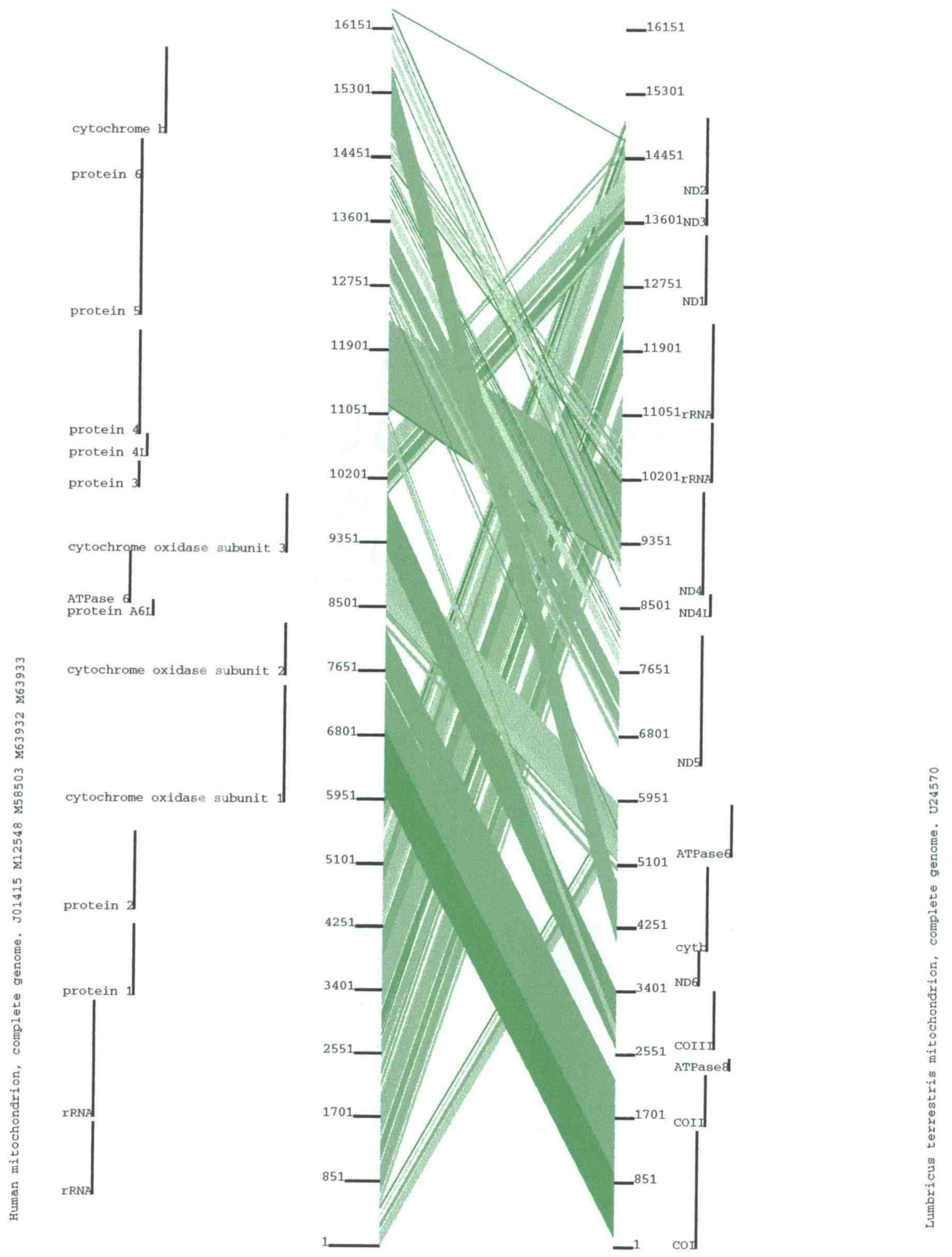
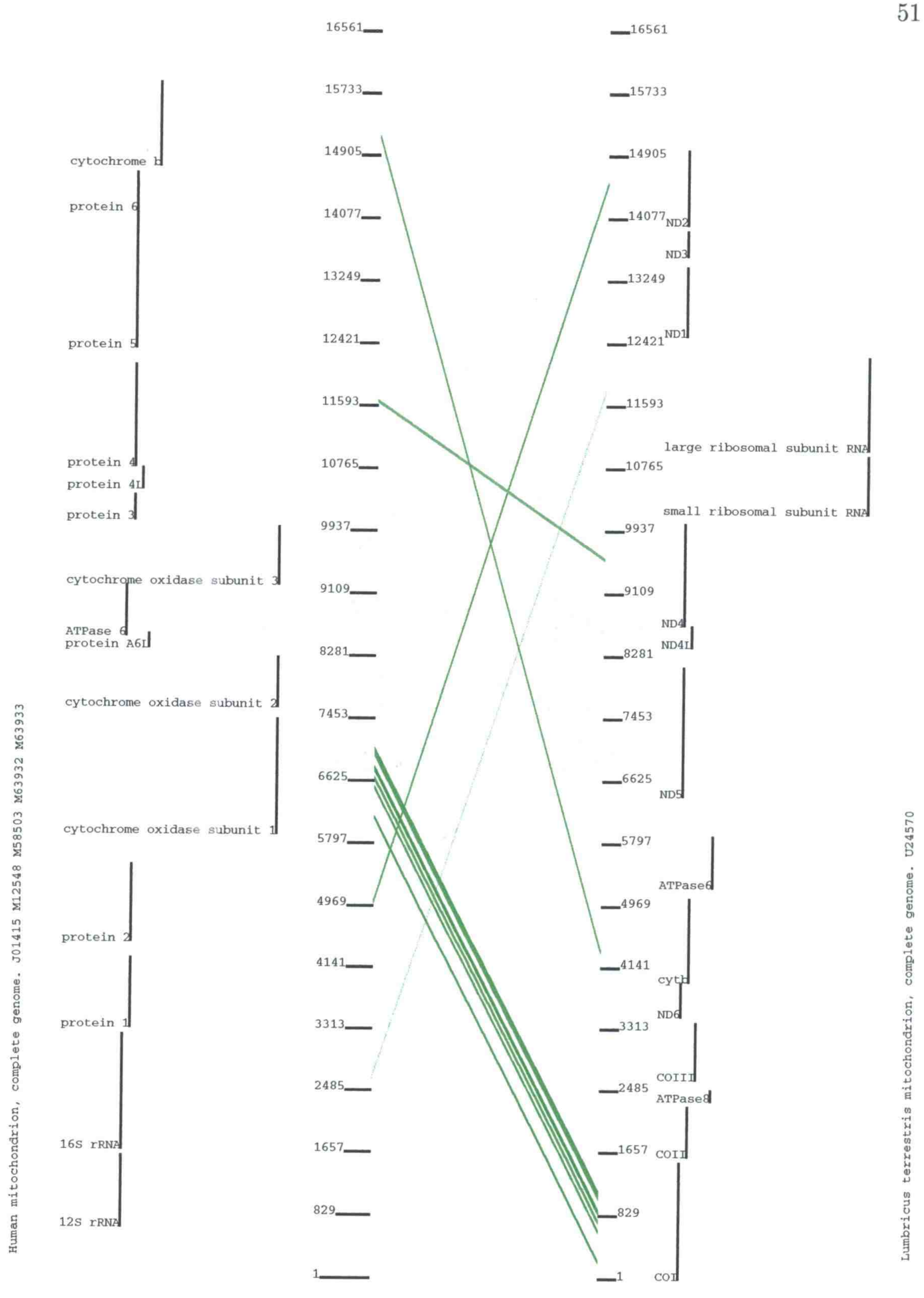


Figure 6.3. Maximum Gap Length = 64



Human mitochondrion, complete genome. J01415 M12548 M58503 M63932 M63933
cytochrome b
protein 6
protein 5
protein 4
protein 4L
protein 3
cytochrome oxidase subunit 3
ATPase 6
protein A6L
cytochrome oxidase subunit 2
cytochrome oxidase subunit 1
protein 2
protein 1
16S rRNA
12S rRNA
16561
15733
14905
14077
13249
12421
11593
10765
9937
9109
8281
7453
6625
5797
4969
4141
3313
2485
1657
829
1
ND2
ND3
ND1
large ribosomal subunit RNA
small ribosomal subunit RNA
ND4
ND4L
ND5
ATPase6
cytb
ND6
COIII
ATPase8
COII
COI
Lumbricus terrestris mitochondrion, complete genome. U24570

Figure 6.4. Minimum Contig Length = 36

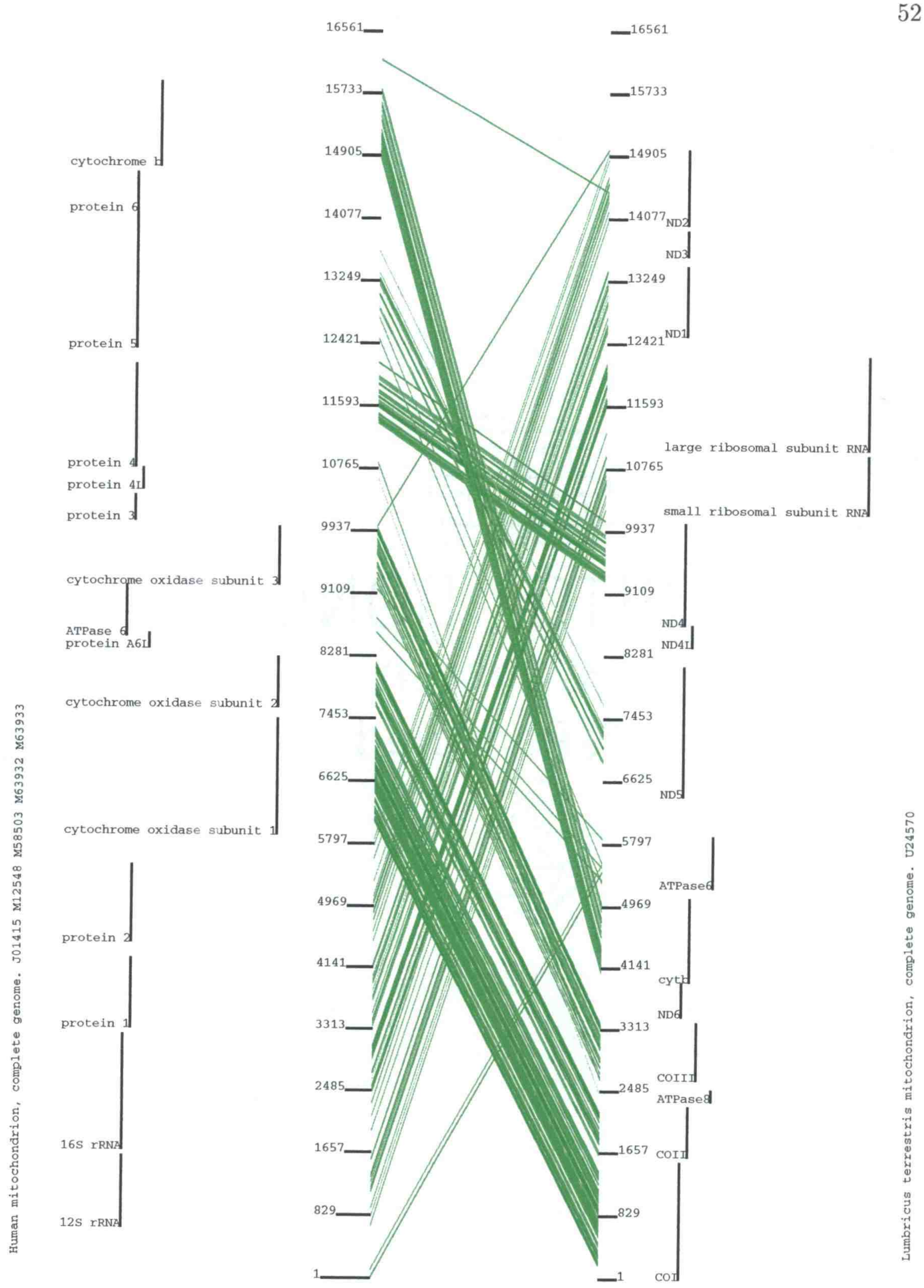


Figure 6.5. Minimum Contig Length = 12

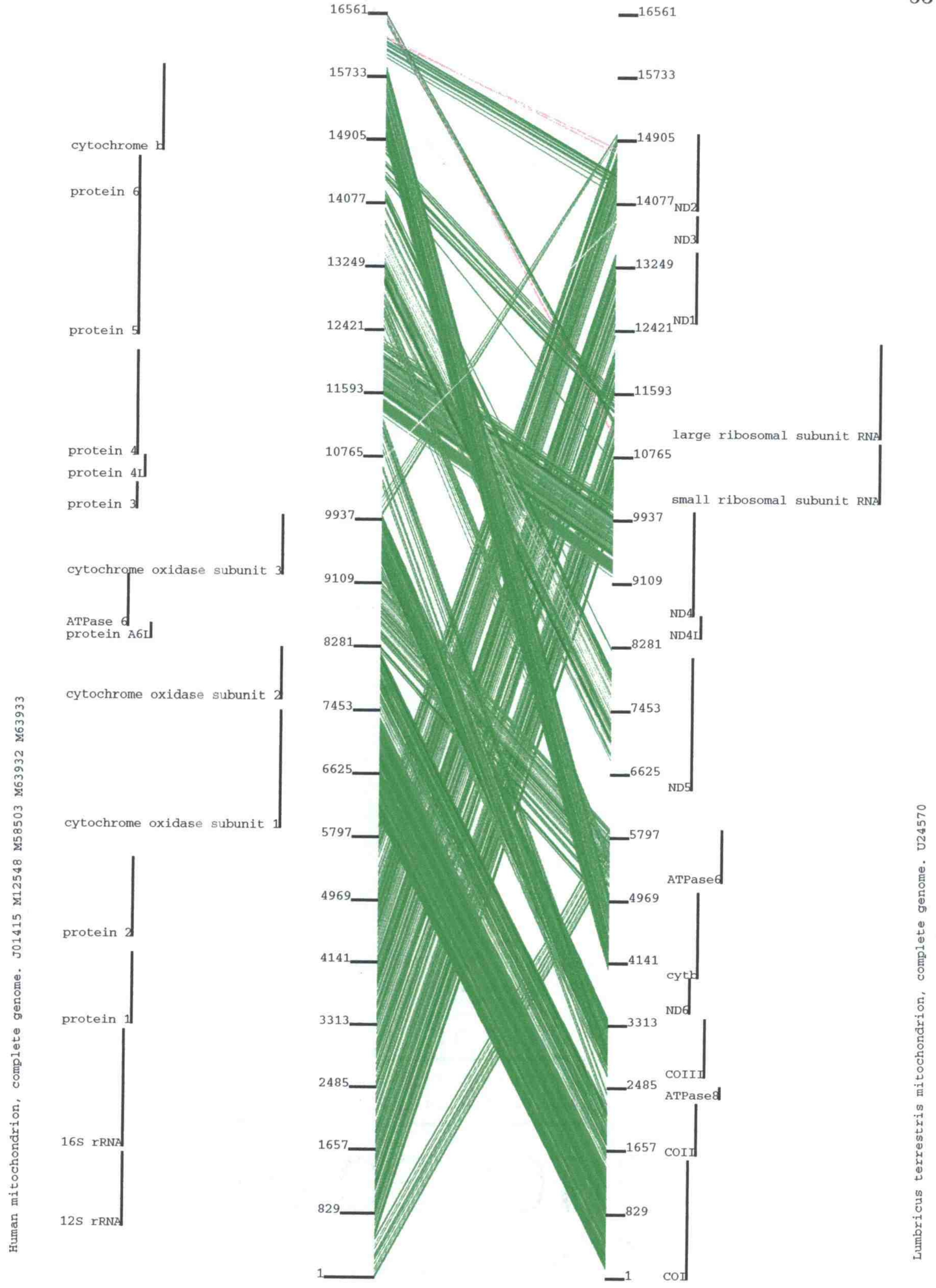


Figure 6.6. Minimum Contig Length = 6

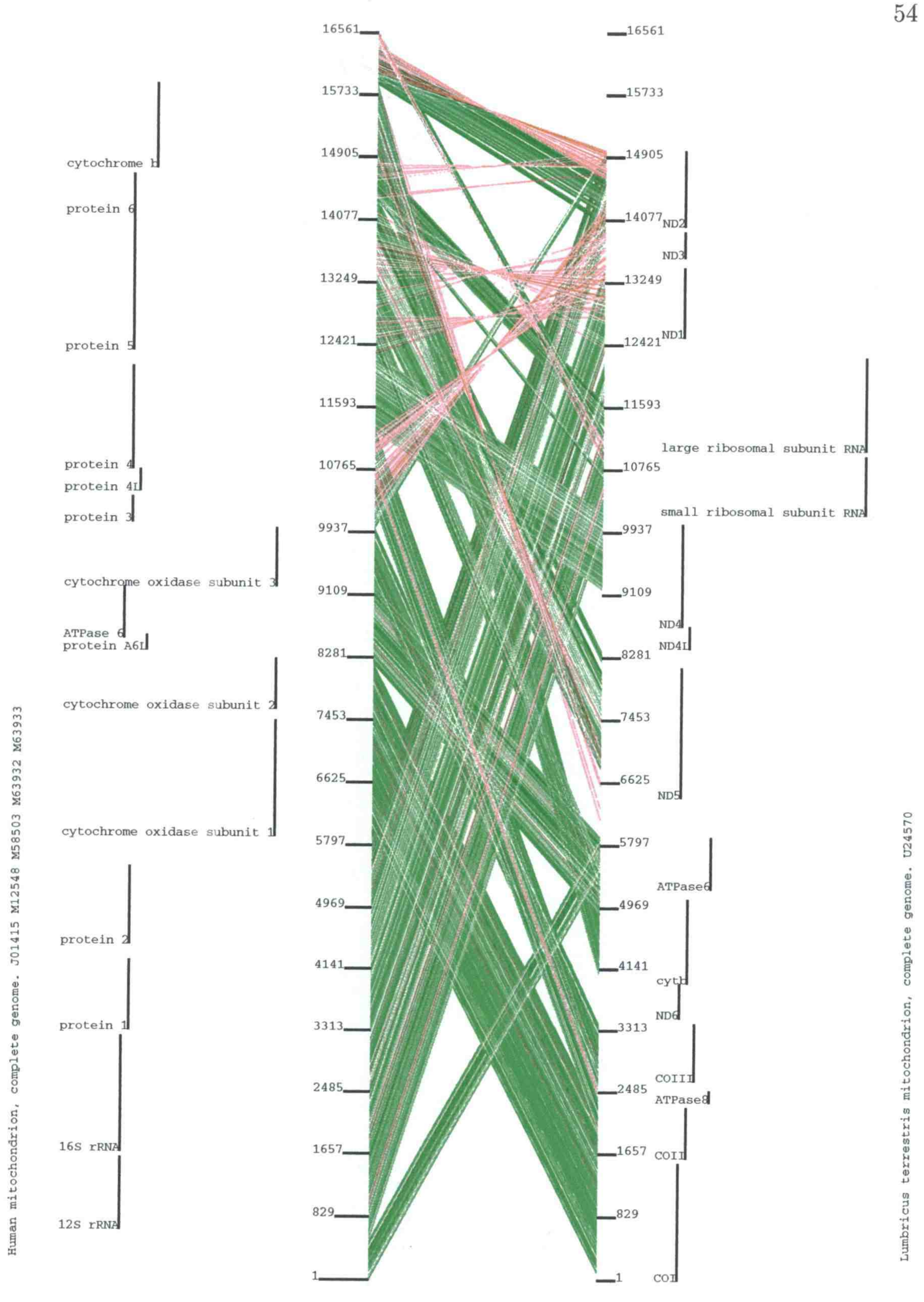


Figure 6.7. Minimum Contig Length = 1

6.2. PHYLOGENETIC TREE CALCULATION

Figure 3.5 on page 23 shows the resulting tree of my phylogenetic study on thirty-eight picornavirus genomes [17]. I performed a total of 1444 alignments via my program pairwiseAlignments.pl running under PVM [18] configured with fifty workstations. I then constructed a distance matrix based on the alignment files with my program alignments2distanceMatrix.pl (see Appendix A for more information). I used the programs neighbor and drawgram in the phylogenetic analysis package PHYLIP to construct the tree from the distance matrix (see [?, 19]or more information on these programs). Note that in my vertically oriented tree, it is the vertical component of the branches which corresponds to the distance from putative ancestors at the branching points.

My tree contains the same clustering patterns as those of an earlier tree of twenty picornavirus (Figure 1 of Appendix B, also in [7]), constructed by Jim Holloway using the Walking Tree, which is nearly identical to that of [28].

The Theiler murine encephalomyelitis cardioviruses (TMECG, TMEGDVCG, TMEPP, TMEVCPLT), the other cardioviruses (EMCBCG, EMCDCG, EVCGAA, MNGPOLY, XXEVCG), and the Hepatitis A viruses (HPA, HPAA, HPACG) display the same clustering patterns. The relationship [29] between the swine vesicular disease virus SVDG and Coxsackie B5 virus CXB5CGA is also retained.

Additionally, the polio viruses (PIP03XX, POL1, POL2CG1, POL2LAN, POL3L12CG, POLIO1A, POLIO1B, POLIOS1) cluster well, as do the human rhinoviruses (HRV, HRV89, HRVACG, HRVPP, PIHRV2G). Interestingly, the bovine enterovirus BEVVG527 would appear to be more closely related than is HRV to the bulk of the human rhinoviruses. The Coxsackie A viruses (CXA21CG, CXA24CG) and EV70CG (which incidentally causes acute hemorrhagic conjunctivitis, as does CXA24CG) cluster loosely with the polio viruses. The Coxsackie B viruses (CXA1G (B1); CXA3CG, CXAB3CG (B3); CXB4S, PICOXB4 (B4)) cluster near the echo viruses (ECHOV9XX, EV9GENOME), SVDG and CXB5CGA.

It appears that the method clusters the sequences well according to their relatedness, and the Walking Tree may prove useful for automating other such comparisons given raw sequence data.

7. CONCLUSION

7.1. SUMMARY

Biologists are overwhelmed by the task of utilizing the ever-increasing amount of biological sequence data available. The problem is complicated by errors in data collection and by biological transformations including translocation and inversion of genetic elements. I have outlined complications of the application of the central dogma of molecular biology to sequence matching, and suggested the Walking Tree heuristic algorithm [1–7] as a suitable method for matching biological sequences in light of these complications. I have presented several new user interfaces for the Walking Tree, including a World Wide Web interface, a GUI in Tcl/Tk, and a command-line interface, suitable for use with Walking Tree implementations such as [26]. With these tools, I have produced a genomic alignment visualization for mitochondrial genomes of the human and the earthworm, and a phylogenetic tree of picornaviruses, further demonstrating applicability of the Walking Tree to gene discovery, annotation of genetic structures, genome alignment and phylogenetic tree calculation.

7.2. FUTURE WORK

Additional investigation and implementation is needed to support two important application areas. The first, the use of the Walking Tree in protein database searches, requires a Walking Tree implementation which incorporates BLOSUM [24] or PAM [23] matrices for amino acid similarity scoring. The second, the use of the Walking Tree to align complete genomes of bacteria and higher organisms in reasonable time, requires parallel implementations of the Walking Tree for available machines. Both of these application areas would be addressed by a PVM [18] implementation of the Walking Tree with arbitrarily assignable subtree scoring functions.

BIBLIOGRAPHY

[1] J. L. Holloway. Algorithms for String Matching with Applications in Molecular Biology. *Doctoral dissertation, Oregon State University, Dept. of Computer Science.* 1992

[2] P. Cull and J. L. Holloway. A divide and conquer approach to approximate string matching. *Technical Report TR-91-50-1, Oregon State University, Dept. of Computer Science.* 1991

[3] P. Cull and J. L. Holloway. Algorithms for constructing a consensus sequence. *Technical Report TR-91-20-1, Oregon State University, Dept. of Computer Science.* 1991

[4] P. Cull and J. L. Holloway. Reconstructing sequences from shotgun data. *Sequences: Combinatorics, Compression, Security, and Transmission.* Springer-Verlag, 1991

[5] P. Cull and J. L. Holloway. Optimistically building a consensus sequence using inexact matches. *Proceedings of the Hawaii International Conference on System Sciences.* Volume 1, pages 643-652, 1992

[6] P. Cull and J. L. Holloway. Approximate String Matching for Biological Sequences with Swaps and Inversions. *Cybernetic Systems '94.* Pages 879-886. R. Trappl (editor). World Scientific, Singapore.

[7] J. L. Holloway and P. Cull. Aligning Genomes with Inversions and Swaps. *Second International Conference on Intelligent Systems for Molecular Biology, Proceedings of ISMB'94.* Pages 195-202. AAAI Press, Menlo Park CA. 1994

[8] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology.* 147:195-197,1981

[9] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences of the U.S.A.* 85:2444-2448,1988

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

[11] M. Borodovsky and J. McIninch. (GeneMark). *Computational Chemistry.* 17:123,1993

[12] Y. Xu, J. R. Einstein, R. J. Mural, M. B. Shah and E. C. Uberbacher. An Improved System for Exon Recognition and Gene Modeling in Human DNA

Sequences. *Second International Conference on Intelligent Systems for Molecular Biology, Proceedings of ISMB'94*. AAAI Press, Menlo Park CA. 1994

[13] Anderson,S., Bankier,A.T., Barrell,B.G., de Bruijn,M.H.L., Coulson,A.R., Drouin,J., Eperon,I.C., Nierlich,D.P., Roe,B.A., Sanger,F., Schreier,P.H., Smith,A.J.H., Staden,R. and Young,I.G. Sequence and organization of the human mitochondrial genome. *Nature.* 290 (5806), 457-465 (1981)

[14] Flook,P.K., Rowell,C.H. and Gellissen,G. The sequence, organization, and evolution of the Locusta migratoria mitochondrial genome. *J. Mol. Evol.* 41 (6), 928-941 (1995)

[15] The usage of this and my other programs may be found in the reference section in the Appendix. Align.pl incorporates a Walking Tree implementation, kindly provided by Jim Holloway, upon which this work is based.

[16] The GenBank database is accessible via http://www.ncbi.nlm.nih.gov.

[17] The GenBank entries for the picorna virus genomes used are accession numbers BEVVG527, CXA1G, CXA21CG, CXA24CG, CXA3CG, CXAB3CG, CXB4S, CXB5CGA, ECHOV9XX, EMCBCG, EMCDCG, EV70CG, EV9GENOME, EVCGAA, HPA, HPAA, HPACG, HRV, HRV89, HRVACG, HRVPP, MNGPOLY, PICOXB4, PIHRV2G, PIP03XX, POL1, POL2CG1, POL2LAN, POL3L12CG, POLIO1A, POLIO1B, POLIOS1, SVDG, TMECG, TMEGDVCG, TMEPP, TMEVCPLT, and XXEVCG.

[18] Parallel Virtual Machine. The latest user's guide is available via ftp from netlib2.cs.utk.edu.

[19] PHYLIP is available via http://evolution.genetics.washington.edu/phylip.html

[20] Dybvig,K. and Yu,H. Regulation of a restriction and modification system via DNA inversion in Mycoplasma pulmonis. *Molecular Microbiology* 12 (4), 547-560 (1994)

[21] Adapted from http://www.gcg.com/techsupport/data/human_high.cod which is also available in gcg (Wisconsin Package Version 9.0, Genetics Computer Group (GCG), Madison, Wisc.)

[22] Adapted from an internal table in dnaparse.c from the PHYLIP package.

[23] Schwartz, R. M. and Dayhoff, M. O. Atlas of Protein Sequence and Structure. *Dayhoff, M. O. Ed, National Biomedical Research Foundation, Washington D.C.* [1979] pp. 353-358.

[24] Henikoff, S. and Henikoff, J. G. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA.* 89: 10915-10919 (1992).

[25] J. Janin. Surface and Inside Volumes in Globular Proteins. *Nature.* 277(1979)491-492.

[26] One of J. L. Holloway's Walking Tree implementations, formerly available via ftp at ftp.cs.orst.edu/holloway/dnc.tar.Z

[27] Setubal, J. and Carlos, J. Introduction to computational molecular biology. *PWS Publishing Company.* 1997.

[28] Stanway,G. Structure, function, and evolution of picornaviruses. *Journal of General Virology.* 87:2483-2501 (1990).

[29] Zhang,G., Wilsden,G., Knowles,N.J. and McCauley,J.W. Complete nucleotide sequence of a coxsackie B5 virus and its relationship to swine vesicular disease virus. *J. Gen. Virol.* 74 (Pt 5), 845-853 (1993).

[30] Most of the gaps in this visualization are due to the filtering out of false inversions after matching. The false inversions occured frequently because the Walking Tree was used to match binary truth values. Turning off the inversion functionality of the Walking Tree would result in fewer gaps in this particular visualization.

APPENDICES

APPENDIX A: COMMAND LINE INTERFACE REFERENCE

Align

Usage

Align.pl pattern-root-name patternGBfile text-root-name textGBfile [maxBaseIndex]

Description

This wrapper script takes two GenBank files and produces an alignment of the first into the second.

Arguments

In order, the arguments are:
pattern-root-name - The desired root name to use to represent the pattern sequence.
patternGBfile - The GenBank file containing the pattern sequence.
text-root-name - The desired root name to use to represent the text sequence.
textGBfile - The GenBank file containing the text sequence.
maxBaseIndex - Optionally specified maximum sequence length to assume for purposes of visualization construction.

Files

Required files involved:
patternGBfile - The GenBank file for the pattern sequence.
textGBfile - The GenBank file for the text sequence.

Notes

The default parameters defaultMinContigLength and defaultMaxGap are set rather conservatively. Several intermediate files are produced.

View

Usage

View.pl pattern-root-name patternGBfile text-root-name textGBfile min-contig-length max-gap [maxBaseIndex]

Description

This wrapper script takes two GenBank files for which an alignment has been previously computed, and produces a visualization of the alignment. The filename for the resulting PostScript file is constructed as pattern-root-name_X_text-root-name.min-contig-length.max-gap.ps .

Arguments

In order, the arguments are:
pattern-root-name - The desired root name to use to represent the pattern sequence.
patternGBfile - The GenBank file containing the pattern sequence.
text-root-name - The desired root name to use to represent the text sequence.
textGBfile - The GenBank file containing the text sequence.
min-contig-length - Aligned subsequences must be at least this long to be included in the visualization.
max-gap - Aligned subsequence pairs separated by gaps no longer than max-gap on

both sequences will be joined.

maxBaseIndex - Optionally specified maximum sequence length to assume for purposes of visualization construction.

Files

Required files involved:

patternGBfile - The GenBank file for the pattern sequence.

textGBfile - The GenBank file for the text sequence.

pattern-root-name_X_text-root-name.alignment - The previously generated alignment of the pattern sequence into the text sequence.

Notes

Several intermediate files are produced.

a2c

Usage

a2c.pl minimum-contig-length max-gap pattern-root-name text-root-name

Description

This wrapper script assembles aligned contig pairs from a raw alignment.

Arguments

In order, the arguments are:
minimum-contig-length max-gap pattern-root-name text-root-name minimum-contig-length - Aligned subsequences must be at least this long to be included in the visualization.
max-gap - Aligned subsequence pairs separated by gaps no longer than max-gap on both sequences will be joined.
pattern-root-name - The root name to use to represent the pattern sequence.
text-root-name - The root name to use to represent the text sequence.

Files

Required files involved:
pattern_X_text.alignment - The raw alignment file.

Notes

None.

adjustPSMargins

Usage

adjustPSMargins.pl left_in_points bottom_in_points [yourfile.ps]

Description

This utility script changes the left and bottom margins of a PostScript alignment to those specified as arguments.

Arguments

In order, the arguments are:
left_in_points - The new left margin in printer's points.
bottom_in_points - The new bottom margin in printer's points.
yourfile.ps - Optionally specified alignment visualization file.

Files

Required files involved:

Notes

72 printer's points = 1 inch.

alignment2contigsList

Usage

```
alignment2contigsList.pl   minLengthForContig   [alignmentFileName]
```

Description

This script extracts aligned contig pairs from an alignment.

Arguments

In order, the arguments are:

minLengthForContig - Contigs must be at least this long to be included in the list.

Files

Required files involved:

Notes

None.

alignments2reflexive

Usage

alignments2reflexive firstAlignmentFile secondAlignmentFile

Description

This script finds the reflexive alignment of two alignments.

Arguments

In order, the arguments are:

firstAlignmentFile - a_X_b.alignment

secondAlignmentFile - b_X_a.alignment

Files

Required files involved:
Two alignment files.

Notes

For various reasons, an alignment of a sequence A into a sequence B may differ from that of B into A. Given two such alignments, the reflexive alignment consists of those paired (matched) bases in the first alignment which were also a matched pair in the second alignment.

annotateContigsListWithPatternAnnotations

Usage

annotateContigsListWithPatternAnnotations.pl contigsList patternAnnotationFeatures

Description

Where two individually contiguous subsequences are aligned to one another, it is insightful to share annotations between the two. This script annotates a contigsList with annotations of the original pattern sequence.

Arguments

In order, the arguments are:
contigsList - File containing aligned contig pairs.

patternAnnotationFeatures - Features file of the pattern sequence to annotate the contigsList with.

Files

Required files involved:
p_X_t.contigsList* and p.features

Notes on containment and coverage

It is useful to know the degree to which a contig covers another, and also the degree to which a contig is contained within another, to assess the relevance of such shared annotations. Herein, the degrees of containment and of coverage are defined as follows. Given two sequence ranges X and Y, the coverage by X of Y, COV(X,Y), is defined to be (min(Xend,Yend)-max(Xstart,Ystart))/(Yend-Ystart) where this value is non-negative. (see Figure). The containment of X by Y, CON(X,Y), is defined to be (min(Xend,Yend)-max(Xstart,Ystart))/(Xend-Xstart) where this value is non-negative. (see Figure). Note that COV(X,Y)=CON(Y,X).

annotateContigsListWithTextAnnotations

Usage

annotateContigsListWithTextAnnotations.pl contigsList
textAnnotationFeatures

Description

This script annotates a contigsList with annotations of the original text sequence. See annotateContigsListWithPatternAnnotations for discussion.

Arguments

In order, the arguments are:
contigsList - File containing aligned contig pairs.
textAnnotationFeatures - Features file of the text sequence to annotate the contigsList with.

Files

Required files involved:
p_X_t.contigsList* and t.features

Notes

None.

c2ps

Usage

c2ps.pl a-rootWord b-rootWord min-length-contig

Description

Wrapper script for PostScript rendering of aligned contig pairs.

Arguments

In order, the arguments are:

a-rootWord - The root name to use to represent the pattern sequence.

b-rootWord - The root name to use to represent the text sequence.

min-length-contig - Used here to choose a contigsList file.

Files

Required files involved:

See contigsList2ps.pl and filterPSFeatures.pl.

Notes

Contains default parameters for layout of rendering.

changePSFeatures

Usage

changePSFeatures.pl feature-text new-feature-text [contigsList.ps-file]

Description

Modifies feature annotation text in existing PostScript visualizations.

Arguments

In order, the arguments are:

feature-text - The text that is to be found and modified.

new-feature-text - The desired new text.

contigsList.ps-file - Optionally specified alignment visualization file.

Files

Required files involved:

None.

Notes

None.

contigsList2ps

Usage

contigsList2ps.pl patTitlexOffset patTitleyOffset psPatternTitleFile textTitlexOffset textTitleyOffset psTextTitleFile patLabelOffset patRangeBarOffset patNumberOffset patternFeaturesPSfile textLabelOffset textRangeBarOffset textNumberOffset textFeaturesPSfile startBaseIndex maxBaseIndex pattern-x-Offset text-x-Offset contigsListFile

Description

Assembles and generates PostScript fragments for rendering aligned contig pairs.

Arguments

In order, the arguments are:

patTitlexOffset - Horizontal offset for the pattern title.

patTitleyOffset - Vertical offset for the pattern title.

psPatternTitleFile - PostScript file holding the pattern title.

textTitlexOffset - Horizontal offset for the text title.

textTitleyOffset - Vertical offset for the text title.

psTextTitleFile - PostScript file holding the text title.

patLabelOffset - Horizontal offset for pattern feature annotation labels.

patRangeBarOffset - Horizontal offset for pattern feature range bars.

patNumberOffset - Horizontal offset for pattern base index numbers.

patternFeaturesPSfile - PostScript file holding the pattern's feature annotations.

textLabelOffset - Horizontal offset for text feature annotation labels.

textRangeBarOffset - Horizontal offset for text feature range bars.

textNumberOffset - Horizontal offset for text base index numbers.

textFeaturesPSfile - PostScript file holding the pattern's feature annotations.

startBaseIndex - Index of the first base location rendered.

maxBaseIndex - Maximum base index for which to allow.

pattern-x-Offset - Horizontal offset for the pattern-side of aligned contig pairs.

text-x-Offset - Horizontal offset for the text-side of aligned contig pairs.

contigsListFile - File containing the list of aligned contig pairs.

Files

Required files involved:

pattern-root-name.title.pat.ps - psPatternTitleFile text-root-name.title.text.ps - psTextTitleFile

pattern-root-name.features.pat.ps - patternFeaturesPSfile

text-root-name.features.text.ps - textFeaturesPSfile

contigsListFile - File containing aligned contig pairs.

Notes

Offsets are in printer's points.

f2ps

Usage

f2ps.pl pattern-root-name pat-GB-file text-root-name text-GB-file

Description

This wrapper script extracts features from GenBank files and generates PostScript fragment files for rendering their labels at their locations along the pattern and text sequences.

Arguments

In order, the arguments are:

pattern-root-name - The root name used to represent the pattern sequence.

pat-GB-file - The GenBank file containing the pattern sequence.

text-root-name - The root name used to represent the text sequence.

text-GB-file - The GenBank file containing the text sequence.

Files

Required files involved:

pat-GB-file - The GenBank file for the pattern sequence.

text-GB-file - The GenBank file for the text sequence.

Notes

None.

filterPSFeatures

Usage

filterPSFeatures.pl [ps-file]

Description

This script takes a PostScript alignment visualization as input, and outputs the visualization with certain feature annotation labels removed. By default, mRNA is stripped.

Arguments

In order, the arguments are:

ps-file - Optionally specified PostScript alignment visualization.

Files

Required files involved:

None.

Notes

Additional undesirable feature annotations may be specified by completing the template inside of this script.

g2i

Usage

g2i.pl pattern-root-name pattern-GB-file text-root-name text-GB-file

Description

This wrapper script prepares input files for the sequence alignment executable. Sequences are extracted from the specified GenBank files and assembled in an input file, which is then specified in the generated simscript file.

Arguments

In order, the arguments are:

pattern-root-name - The root name used to represent the pattern sequence.

pattern-GB-file - The GenBank file containing the pattern sequence.

text-root-name - The root name used to represent the text sequence.

text-GB-file - The GenBank file containing the text sequence.

Files

Required files involved:

pattern-GB-file - The GenBank file containing the pattern sequence.

text-GB-file - The GenBank file containing the text sequence.

Notes

None.

gb2features

Usage

```
gb2features.pl   [gb-file]
```

Description

Extracts and labels annotations from the features section of a GenBank file.

Arguments

In order, the arguments are:

gb-file - Optionally specified GenBank file containing the sequence of interest.

Files

Required files involved:

None.

Notes

Preprocess input with gbFeatureFix.pl to handle long annotations.

gb2psPatternTitle

Usage

gb2psPatternTitle.pl [GenBankFile]

Description

This script extracts the definition and accession fields from a GenBank file, and uses them to produce a PostScript fragment for the pattern title.

Arguments

In order, the arguments are:
GenBankFile - Optionally specified GenBank file containing the sequence of interest.

Files

Required files involved:
None.

Notes

None.

gb2psTextTitle

Usage

gb2psTextTitle.pl [GenBankFile]

Description

This script extracts the definition and accession fields from a GenBank file, and uses them to produce a PostScript fragment for the text title.

Arguments

In order, the arguments are:
GenBankFile - Optionally specified GenBank file containing the sequence of interest.

Files

Required files involved:
None.

Notes

None.

gb2seq

Usage

gb2seq.pl [gb-file]

Description

This script extracts the sequence data from a GenBank file, and prints it on a single line without the digits and whitespace.

Arguments

In order, the arguments are:

gb-file - Optionally specified GenBank file containing the sequence of interest.

Files

Required files involved:

None.

Notes

A newline character is appended to the output sequence.

gbFeatureFix

Usage

```
gbFeatureFix.pl   [gb-file]
```

Description

This filter prepares GenBank input for feature extraction.

Arguments

In order, the arguments are:

gb-file - Optionally specified GenBank file containing the sequence of interest.

Files

Required files involved:

None.

Notes

This filter serves to merge each multi-line join of subsequences onto a single line.

grid

Usage

```
grid.pl   pgb   tgb
```

Description

This script serves only as an example of running View.pl with a spread of parameters.

Arguments

In order, the arguments are:

pgb - The root name used to represent the pattern sequence, and here also the GenBank file.

tgb - The root name used to represent the text sequence, and here also the GenBank file.

Files

Required files involved:
Two GenBank files, specified by the arguments pgb and tgb.

Notes

This is only an exemplary script, and the name of each GenBank file doubles as the root name for that sequence's files.

itoa

Usage

itoa.pl pattern-root-name text-root-name

Description

This wrapper script calls the sequence alignment executable, giving it a simscript file as input, and capturing output in an alignment file..

Arguments

In order, the arguments are:
pattern-root-name - The root name used to represent the pattern sequence.

text-root-name - The root name used to represent the text sequence.

Files

Required files involved:

pattern-root-name_X_text-root-name.input - Input file holding the text and pattern sequences.

pattern-root-name_X_text-root-name.simscript - Commands to be executed by the sequence alignment executable.

Notes

None.

joinContigs

Usage

joinContigs.pl   max-gap   [contigsListFile]

Description

This script joins (merges) adjacent pairs of aligned contigs which are separated by no more than max-gap base pairs, creating larger aligned contig pairs from smaller ones.

Arguments

In order, the arguments are:

max-gap - Aligned subsequence pairs separated by gaps no longer than max-gap on both sequences will be joined.

contigsListFile - Optionally specified contigsList file.

Files

Required files involved:

None.

Notes

Resultant contigs within an aligned pair may be of different lengths if the corresponding gaps are of different lengths.

makeSimScript

Usage

makeSimScript.pl input-file-name

Description

This script generates commands for the sequence alignment executable.

Arguments

In order, the arguments are:

input-file-name - The file containing the text and pattern sequences.

Files

Required files involved:
The argument input-file-name is intended to be of the form p_X_t.input representing an existing file.

Notes

None.

outlinePSContigs

Usage

outlinePSContigs.pl [PostScript-alignment-visualization-file]

Description

This utility script outlines the bars of each aligned contig pair in the specified PostScript file.

Arguments

In order, the arguments are:
PostScript-alignment-visualization-file - Optionally specified.

Files

Required files involved:
None.

Notes

The specified file is modified after a backup copy is created.

patternFeatures2ps

Usage

patternFeatures2ps.pl [pattern-features-file]

Description

This script produces PostScript fragments for pattern feature annotations.

Arguments

In order, the arguments are:
pattern-features-file - Optionally specified file containing feature annotations for the text sequence.

Files

Required files involved:
None.

Notes

The script heuristically extracts (or excludes) a meaningful annotation from the wealth of available feature information. The selection criteria are easily modified in the script.

rawAlignment2alignment

Usage

rawAlignment2alignment.pl [raw-alignment-file]

Description

This script extracts the alignment information from the raw output of the sequence alignment executable.

Arguments

In order, the arguments are:
raw-alignment-file - Optionally specified.

Files

Required files involved:
None.

Notes

None.

seq2input

Usage

seq2input.pl text.seq pattern.seq

Description

This script simply takes two sequence file names, and prints the sequence data in a form consumable by the sequence alignment executable.

Arguments

In order, the arguments are:
text.seq - File containing the text sequence, ended with a newline character. pattern.seq - File containing the text sequence, ended with a newline character.

Files

Required files involved:
Two sequence files specified by text.seq and pattern.seq.

Notes

Yes, the text file is specified here before the pattern file.

setMaxBaseIndex

Usage

setMaxBaseIndex.pl pattern-root-name patternGBfile text-root-name textGBfile [maxBaseIndex]

Description

Creates a file containing either the length of the larger of the sequences, or a user-specified substitute, to use in the layout of rendered visualizations.

Arguments

In order, the arguments are:
pattern-root-name - The root name used to represent the pattern sequence.
patternGBfile - The GenBank file containing the pattern sequence.
text-root-name - The root name used to represent the text sequence.
textGBfile - The GenBank file containing the text sequence.
maxBaseIndex - Optionally specified maximum base index to use in alignemnt visualizations.

Files

Required files involved:
patternGBfile and textGBfile.

Notes

A new maxBaseIndex file is created unless the file already exists and no argument is given for maxBaseIndex on the command line.

stripPSFeatures

Usage

```
stripPSFeatures.pl   feature-text   [ps-file]
```

Description

This script strips (comments out) features matching the feature-text argument. Either a PostScript alignment visualization or a PostScript contigsList fragment are acceptable as input.

Arguments

In order, the arguments are:

feature-text - Features matching this argument will be stripped from the PostScript view.

ps-file - Optionally specified PostScript file.

Files

Required files involved:

None.

Notes

Using non-specific feature specifiers may result in unintended stripping, e.g., using the feature-text argument "tRNA" would also strip mtRNA.

t2ps

Usage

t2ps.pl pattern-root-name pattern-GB-file text-root-name text-GB-file

Description

This wrapper script creates PostScript fragment files for the pattern title and the text title.

Arguments

In order, the arguments are:
pattern-root-name - The root name used to represent the pattern sequence.
pattern-GB-file - The GenBank file containing the pattern sequence.
text-root-name - The root name used to represent the text sequence.
text-GB-file - The GenBank file containing the text sequence.

Files

Required files involved:
pattern-GB-file and text-GB-file

Notes

None.

textFeatures2ps

Usage

```
textFeatures2ps.pl   [text-features-file]
```

Description

This script produces PostScript fragments for text feature annotations.

Arguments

In order, the arguments are:

text-features-file - Optionally specified file containing feature annotations for the text sequence.

Files

Required files involved:

None.

Notes

The script heuristically extracts (or excludes) a meaningful annotation from the wealth of available feature information. The selection criteria are easily modified in the script.

APPENDIX B: SUPPLEMENTAL PAPER ON THE WALKING TREE

Walking Tree Heuristics for String Matching and Gene Location

J. L. Holloway
Crop and Soil Science
Oregon State University
holloway@bcc.orst.edu

P. Cull
Computer Science
Oregon State University
pc@cs.orst.edu

J. D. Cavener*
Computer Science
Oregon State University
cavenej@cs.orst.edu

Abstract

Our walking tree heuristics, for approximate string matching, align large biological sequences which may include nested inversions and translocations.

Here we show their use in phylogenetic tree calculation, genome alignment, and gene finding.

The basic heuristic takes time proportional to the product of the lengths of the two sequences, uses workspace proportional to the length of the shorter sequence, and parallelizes well.

Introduction

There is evidence that evolution proceeds by transposing and inverting segments of a genome in addition to changing, inserting, and deleting individual bases in the genome. Many methods that are currently in use to align genetic sequences fail to consider the transpose and invert operations. We introduce treatments given in (Holloway & Cull 1994) of a heuristic to facilitate the comparison of sequences using both types of operations; the change, insert, and delete operations on individual bases, and the transpose and invert operations on segments of the sequences.

Examples

Phylogenetic Analysis

Picornaviridae is a family of single stranded RNA viruses that are 7.2 to 8.4 kb in length. It is composed of the five genera Aphthovirus, Cardiovirus, Enterovirus, Hepatovirus, and Rhinovirus. The RNA typically codes for four major polypeptides and several proteases.

We selected the sequences from GenBank release 81.0 (February 1994) with the keywords "Picornaviridae", "complete", "genome", and "sequence". From these GenBank entries we selected the twenty entries that contained a complete Picornavirus genome sequence. Using the heuristic, we computed an alignment score between each pair of sequences. The alignment score, s, is converted to a normalized "distance", d, using

$$d = 1 - \frac{s}{\max_s}$$

where $\max_s$ is the maximum possible alignment score for the pattern sequence. We then used the Fitch–Margoliash distance matrix method as implemented by Joe Felsenstein in the Phylip 3.53c package to construct the phylogenetic tree in Figure 1.

The phylogeny presented in Figure 1 is based on the complete viral genomes and is nearly identical to the phylogeny presented by (Stanway 1990) based on the P1 (capsid–encoding) regions of each genome. The Hepatovirus genera (HPAACG, HPACG, HPA, HPAA) cluster tightly as expected since they are nearly identical sequences. The Cardioviruses (EVCGAA, EMCDCG, EMCBCG, MNGPOLY, TMECG, TMEPP, TMEVCPLT, and TMEGDVCG) form three groups, the encephalomyocarditis viruses, a mengovirus, and the Theiler murine encephalomyelitis viruses. The Enteroviruses (SVDG, CXB5CGA, CXB4S, CXA21CG, POLIOS1, CXA24CG, and BEVVG527) cluster loosely. The Rhinoviruses (HRV) are not near any of the other Picornaviruses.

The importance of studying methods that are capable of aligning genetic sequences that include inversions and translocations is evidenced by the frequent mention of the rearrangement of the order and orientation of genes between organisms in the literature, for example (Perry, Thomsen and Roeder 1985; Prombona and Subramanian 1989; Devos *et al.* 1993). Further the known order and orientation of genes on 16 mitochondrial genomes has recently been used to construct a phylogenetic tree (Sankoff *et al.* 1992). With walking tree heuristics we can construct such phylogenetic trees easily from either the DNA sequences or from the gene positions and orientations.

Alignment and Gene Finding

To demonstrate the utility of our heuristic, we use it to align two pairs of sequences. The result of the heuristics discussed in this paper is an alignment for each character of the pattern into the text. We are developing tools to filter out the uninteresting regions of the alignment and leave only the interesting regions. Currently we use two simple filters. The first filter selects regions that are aligned with no gaps. A minimum

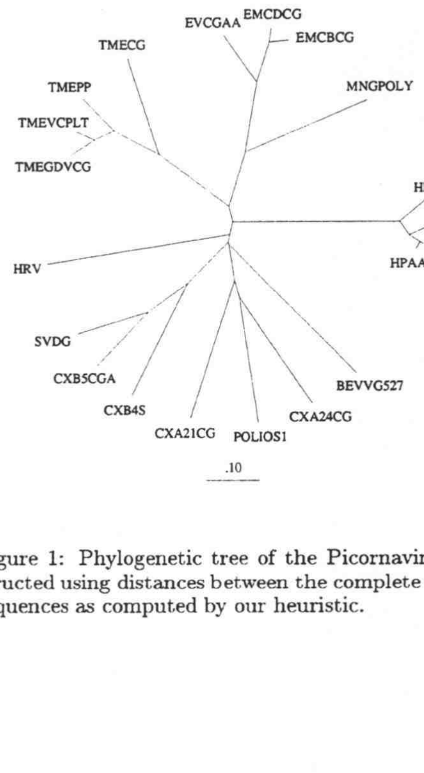


Figure 1: Phylogenetic tree of the Picornavirus constructed using distances between the complete genome sequences as computed by our heuristic.

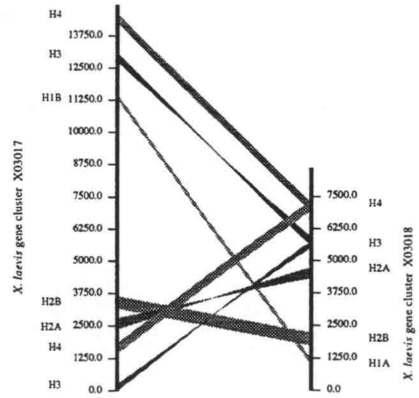


Figure 2: An alignment of two histone gene clusters from *Xenopus laevis*, GenBank loci XELHISH2 and XELHISH3A.

length is specified and only regions of the alignment that are at least the minimum length with no gaps are shown. The second filter selects only regions that have a percent identity greater than a specified minimum percent identity. In Figures 2 and 3, the regions that pass through both filters are shown with red bars connecting regions that align directly and blue bars connecting regions that are the inverse complement of one another. The intensity of the color increases as the percent identity increases.

Histone gene cluster of *X. laevis* We use two histone gene clusters from *Xenopus laevis* to demonstrate aligning sequences with inversions. The histone gene cluster from *X. laevis* with GenBank accession number X03017 is 14942 base pairs in length and the histone gene cluster from *X. laevis* with GenBank accession number X03018 is 8592 base pairs long (Perry, Thomsen and Roeder 1985). The orientation of exons H2A and H3 in X03017 is inverted to the orientation of exons H2A and H3 in X03018. Our heuristic constructed the alignment shown in Figure 2. The position of the histone genes H2A, H2B, H3, and H4 are marked on both sequences. Notice that the genes H3 and H4 appear twice in the left sequence, X03017, in the figure and both copies are aligned with the single copy of the gene in the right sequence, X03018. The alignment shows that the orientation of H2A and H3 regions are reversed in the two sequences. The regions of the two

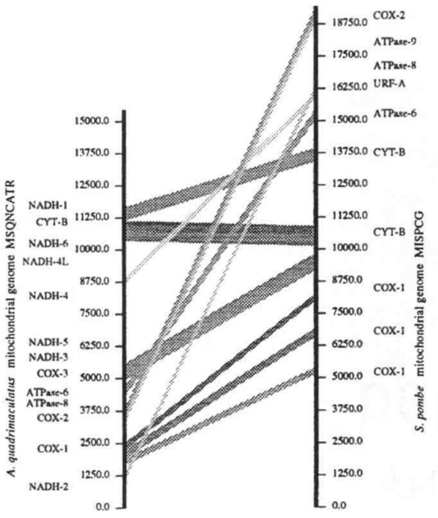


Figure 3: An alignment of the mitochondrial genomes of *Anopheles quadrimaculatus*, GenBank locus MSQN-CATR, and *Schizosaccharomyces pombe*, GenBank locus MISPCG. This alignment was calculated using about five minutes of computer time.

sequences that show the highest similarity are the corresponding genes. The walking tree heuristic properly aligns the histone gene clusters from *X. laevis* because it is capable of inverting and transposing subsequences in the alignment that it creates.

Mitochondrial DNA genomes Our heuristic was used to align the mitochondrial genomes of *Anopheles quadrimaculatus*, GenBank locus MSQNCATR, composed of 15455 bases and *Schizosaccharomyces pombe*, GenBank locus MISPCG composed of 19431 bases. The resulting alignment is given in Figure 3. The two genomes are labeled with the CDS features as given in the GenBank entries. The two entries have the cytochrome c oxidase 1 (COX–1), cytochrome b (CYT–B), cytochrome c oxidase 2 (COX–2), ATPase–6, and ATPase–8 CDS features in common. There are two striking features of this alignment. The first is that the introns that appear in the *S. pombe* sequences for cytochrome c oxidase 1 and cytochrome b, but do not appear in the *A. quadrimaculatus*, are correctly represented in the alignment. The second striking feature of Figure 3 is the strong alignment of the cytochrome c oxidase 3 (COX–3) region on the mitochondrial genome of *A. quadrimaculatus* with an unlabeled region on the mitochondrial genome of *S. pombe*. We later found the cytochrome c oxidase subunit 3 from the mitochondria of *S. pombe* in GenBank with accession number X16868 (Trinkl and Wolf 1989) and it exactly matches the region from bases 8959 to 9768 of the complete mitochondrial genome of *S. pombe*. This demonstrates the capability of our heuristic to identify a previously unrecognized region of DNA by aligning similar regions in other sequences. The heuristic correctly aligns each of these pairs of products with the exception of the ATPase 8 product. ATPase 8 is a short region, 162 bases in the *A. quadrimaculatus* genome, and 147 bases in the *S. pombe* genome. The simple filters that we currently use with the heutistic fail to identify this region because it is short and not highly conserved relative to the rest of the genome. When relaxing the stringency of the filters to the level that alignments appear for ATPase–8, the rest of the alignment becomes difficult to see with simple filters due to the large number of short, less significant regions that align.

Walking Tree Performance

The basic heuristic computes the score of an alignment with no inversions. We modify this heuristic in three ways: 1) to compute a better placement of gaps, 2) to construct the alignment, and 3) to use inversions in the alignment. The extensions to the basic heuristic may be used individually or in combination.

We have shown the following resource usage results for the heuristic computing only the score of an alignment with inversions in (Holloway 1992).

- The heuristic will execute in time proportional to the product of the length of the text and the length of the pattern.
- The work space used by the heuristic is proportional to the length of the pattern. The work space used is independent of the length of the text.
- The heuristic underestimates the actual alignment score of a pattern at a given position in the text by, at most, the sum of the gap penalties in the alignment at that position.

We have shown the following resource usage results for the heuristic for constructing an alignment with inversions in (Holloway 1992).

- In worst case, the heuristic to construct the alignment will run in $O(|T||P|\log|P|)$ time given a text string T and a pattern P. In practice (see Figure 4), alignment construction takes $O(|T||P|)$ time, as the $\log|P|$ factor for constructing the alignment does not appear since a "better" alignment, requiring that the best alignment be updated, is only rarely found as the walking tree moves along the text.
- Work space of $O(|P|\log|P|)$ is required by the heuristic to construct an alignment given a text string T and a pattern P.
- The heuristic never needs to go backwards in the text.

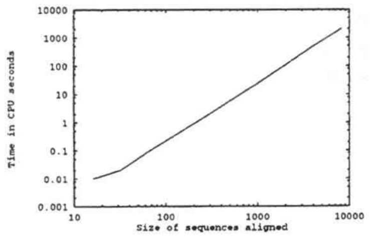


Figure 4: CPU time used by our heuristic to align pairs of equal length sequences on a Sun SPARC–10.

Basic Heuristic

Our metaphor is to consider the data structure for the basic heuristic as a walking tree (see Figure 5) with $|P|$ leaves, one for each character in the pattern. When the heuristic is considering position $l+1$ of the text, the leaves of the tree are positioned over the $|P|$ contiguous characters of the text up to and including character $l+1$. The leaves also remember some of the information for the best alignment within the first l characters of the text. On the basis of this remembered information and the comparisons of the leaves with the text characters under them, the leaves update their information and pass this information to their parents. The data will percolate up to the root where a new best score is calculated. The tree can then walk to the next position by moving each of its leaves one character to the right. The whole text has been processed when the leftmost leaf of the walking tree has processed the rightmost character of the text.

To define a scoring system that captures some biological intuitions, we currently use a function that gives a positive contribution based on the similarity between aligned characters, and a negative contribution that is related to the number and length of gaps, translocations, and inversions. A gap in an alignment occurs when adjacent characters in the pattern are aligned with non–adjacent characters in the text. The length of the gap is the number of characters between the non–adjacent characters in the text. An inversion occurs when a substring, $P_1 = p_i p_{i+1} \cdots p_j$, is matched with text that has the form $\overline{p_j}\overline{p_{j-1}} \cdots \overline{p_i}$. We use $\overline{p_i}$ to indicate the complement of p_i. A translocation occurs when a substring P_1 is matched with a text substring T_1, and a substring P_2 is matched with a text substring T_2, but P_1 occurs before P_2 in the pattern string, while T_1 occurs after T_2 in the text string.

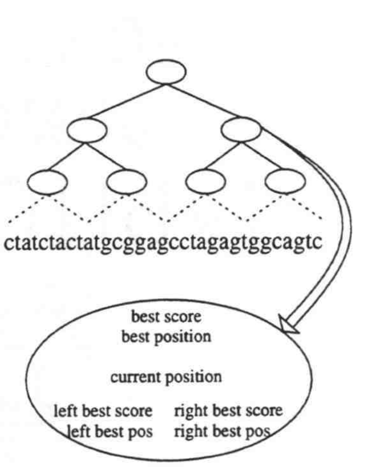


Figure 5: Data structure used to align the pattern within the text. In this picture, each leaf node represents 8 characters of the pattern, each of the internal nodes represents 16 characters of the pattern, and the root node represents the entire pattern. Each of the nodes contains the fields shown in the expanded node.

Adjusting Gaps

The basic heuristic places gaps close to their proper positions. If we use the heuristic to align the string "ABCDEF" in the string "ABCXXDEF" the gap may be placed between 'B' and 'C', rather than between 'C' and 'D'. This is a result of the halving behavior of the basic heuristic. By searching in the vicinity of the position that the basic heuristic places a gap we can find any increase in score that can be obtained by sliding the gap to the left or right. The cost of finding better placements of the gaps is a factor of $\log |P|$ increase in running time since at each node we have to search a region of the text of length proportional to the size of the substring represented by the node.

Including Inversions

The basic heuristic can be modified to find alignments when substrings of the pattern need to be inverted to match substrings of the text. The idea is to invert the pattern and move, along the text, a walking tree of the inverted pattern in parallel with the walking tree of the original pattern. Each pair of nodes in the forward and inverse walking trees that represent the same region of the pattern are referred to as sister nodes. When the match score of a region of the inverted pattern is sufficiently higher than the match score of the corresponding region of the pattern, the region of the inverted pattern is used to compute the alignment score. The introduction of an inversion can be penalized using a function similar to the gap penalty function.

Note that inversions are incorporated into both the walking tree and the inverted walking tree so that it is possible to have inversions nested within a larger inverted substring.

Constructing the alignment

The basic heuristic will report the score and position of the alignment, but does not give enough information to construct the alignment.

Alignment construction introduces a factor of log $|P|$ into theoretical time and space usage which does not occur in practice. An additional field to hold the best alignment of the pattern substring represented by a node is added to each node of the walking tree.

Note that we must save the entire alignment and not just pointers to the alignments of a node's children because the maximum scoring alignments of the children may change without the maximum scoring alignment of the node changing. When the modified heuristic completes, the alignment that produced the best score will be stored in the root node of the walking tree.

Parallelization

We have implemented and optimized our heuristic on a single node of a Meiko CS-2 computer. A node consists of one Texas Instruments SuperSPARC scalar processor and two Fujitsu μVP vector processors.

In practice, we let each leaf of the walking tree represent more than a single character, typically between 30 and 100 characters. This does not decrease the number of character comparisons that the heuristic performs since the walking tree is still moving one character at a time across the text, but it does decrease the size of the walking tree. The smaller walking tree decreases the time required to align two sequences.

Using the vector-optimized heuristic on one node of the Meiko CS-2, two 8192-base sequences can be aligned in less than 1.5 CPU minutes, and aligning a pair of sequences of length 32768 requires less than 25 minutes of CPU time. This agrees with the predicted run time which says that increasing both the pattern and the text by a factor of 4 should increase the run time by a factor of 16.

Extrapolating, an alignment of sequences one million bases long may take two weeks, but fortunately our parallel implementations have shown near-linear speedup. The table below shows speedup results on a 28 processor Sequent Balance 21000.

Processors	1	2	4	8	12	16	20
Speedup	1.0	1.97	3.83	7.26	10.5	13.6	16.7

It may also be possible to construct a "smart" disk controller based on our walking tree heuristic. The heuristic uses only a few simple operations and never needs to back up in the text. Each processor would need a small, constant sized memory, and would need to communicate with at most four other processors. Such a disk controller would allow database search heuristics to start with only the sequences that are similar to the query sequence.

Summary

Complete viral and organellar genomes have been sequenced and are in the biological sequence databases. Very soon, complete sequences of bacterial genomes and some eukaryotic genomes will be available. In the past, while aligning short biological sequences, it was reasonable to model the evolution of biological sequences with the operations: substitute one base for another, insert a base, and delete a base. Today, with the arrival of sequences of complete genomes, the operations inversion and transposition need to be added in order to appropriately model the evolution of these large biological sequences. In this paper we described a family of heuristics designed to align biological sequences that may include inversions and transpositions. Our heuristics use time approximately proportional to the product of the lengths of the sequences being aligned and use work space approximately proportional to the length of the shorter of the sequences. As an example of the utility of our heuristic, it was used successfully to align the complete mitochondrial genomes of *Schizosaccharomyces pombe* and *Anopheles quadrimaculatus*. This alignment shows the previously unreported location of the cytochrome c oxidase subunit 3 on the mitochondrial genome of *S. pombe* and was completed using about five minutes of computer time.

Acknowledgments

This work was supported in part by NSF grant CDA-9216172.

References

Devos, K. M.; Atkinson, M. D.; Chinoy, C. N.; Francis, H. A.; Harcourt, R. L.; Koebner, R. M. D.; Liu, C. J.; Masojc, P.; Xie, D. X.; and Gale, M. D. 1993. Chromosomal rearrangements in the rye genome relative to that of wheat. *Theoretical and Applied Genetics* 85:673-680.

Holloway, J. L., and Cull, P. 1994. Aligning genomes with inversions and swaps. *Second International Conference on Intelligent Sytstems for Molecular Biology. Proceedings of ISMB'94* 195-202. AAAI Press, Menlo Park CA 1994

Holloway, J. L. 1992. *String Matching Algorithms with Applications in Molecular Biology.* Ph.D. Dissertation, Oregon State Universtiy, Corvallis, OR.

Perry, M., Thomsen, G. H. and Roeder, R. G. (1985). Genomic organization and nucleotide sequence of two distinct histone gene clusters from *xenopus laevis*. *Journal of Molecular Biology* **185**: 479-499.

Prombona, A. and Subramanian, A. R. (1989). A new rearrangement of angiosperm chloroplast DNA in rye (*secale cereale*) involving translocation and duplication of the ribosomal rpS15 gene. *The Journal of Biological Chemistry* **264**(32): 19060-19065.

Sankoff, D.; Leduc, G.; Antoine, N.; Paquin, B.; Lang, B. F.; and Cedergren, R. 1992. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proceedings of the National Academy of Science* 89:6875–6579.

Stanway, G. 1990. Structure, function, and evolution of picornaviruses. *Journal of General Virology* 87:2483–2501.

Trinkl, H. and Wolf, K. (1989). Nucleotide sequence of the gene encoding subunit 3 of cytochrome *c* oxidase (*cox3*) in the mitochondrial genome of *schizosaccharomyces pombe* strain ef1. *Nucleic Acids Research* **17**(23): 10104.