Improvements in the Accuracy of Pairwise Genomic Alignment

by

Alexander Karl Hudek

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2010

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Pairwise sequence alignment is a fundamental problem in bioinformatics with wide applicability. This thesis presents three new algorithms for this well-studied problem. First, we present a new algorithm, RDA, which aligns sequences in small segments, rather than by individual bases. Then, we present two algorithms for aligning long genomic sequences: CAPE, a pairwise global aligner, and FEAST, a pairwise local aligner.

RDA produces interesting alignments that can be substantially different in structure than traditional alignments. It is also better than traditional alignment at the task of homology detection. However, its main negative is a very slow run time. Further, although it produces alignments with different structure, it is not clear if the differences have a practical value in genomic research.

Our main success comes from our local aligner, FEAST. We describe two main improvements: a new more descriptive model of evolution, and a new local extension algorithm that considers all possible evolutionary histories rather than only the most likely. Our new model of evolution provides for improved alignment accuracy, and substantially improved parameter training. In particular, we produce a new parameter set for aligning human and mouse sequences that properly describes regions of weak similarity and regions of strong similarity. The second result is our new extension algorithm. Depending on heuristic settings, our new algorithm can provide for more sensitivity than existing extension algorithms, more specificity, or a combination of the two.

By comparing to CAPE, our global aligner, we find that the sensitivity increase provided by our local extension algorithm is so substantial that it outperforms CAPE on sequence with 0.9 or more expected substitutions per site. CAPE itself gives improved sensitivity for sequence with 0.7 or more expected substitutions per site, but at a great run time cost. FEAST and our local extension algorithm improves on this too, the run time is only slightly slower than existing local alignment algorithms and asymptotically the same.

Acknowledgements

I would like to thank my supervisor, Daniel G. Brown, for his support and guidance in producing this thesis. Thanks also, to my committee members for their helpful comments, suggestions, and aid in proofing this document. I am grateful to my family, for listening to my thoughts and ideas. Finally, I would like to thank NSERC for funding this work.

Dedication

To my sister.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In 1859 Charles Darwin published *On the Origin of Species* [29] in which he proposed the idea of *evolution*. His idea, that organisms change over generations in response to environmental pressures, has had a profound impact on both culture and science. Under the theory of evolution, species may differentiate into one or more new species over time. These new species are related to one another by their common ancestor and share many traits. Although Darwin observed these shared traits, he did not know the precise mechanism by which species change over time.

Today, we know that species are described by DNA, a complex molecule comprised of many smaller molecules called nucleotides. Evolutionary change occurs through changes to this molecule, where beneficial mutations remain in populations and damaging mutations quickly disappear. Recently, advances in biotechnology have allowed us to rapidly read this molecule into computer systems as strings over the four-symbol alphabet, A, T, C, G, where each symbol represents a single nucleotide. The data describing a single species, commonly called a genome, can be millions or billions of symbols long. Online databases [3] now contain the genomes for thousands of species, and the number is quickly growing.

The size of genomes necessitates the use of computer algorithms to analyze and understand them. One of the most basic computational tasks that we perform on genomic data is identifying the evolutionary relationships between DNA from two or more species. On a small scale, we wish to identify which individual nucleotides are unique to a species, and which nucleotides share ancestry. On a larger scale, we look to find entire subsequences that share a common ancestry.

In this thesis we focus on improving these basic sequence analysis tasks by using a probabilistic model of evolution. In Chapter 3, we present a new algorithm for discover-

ing evolutionary relationships that lie somewhere between the large-scale and small-scale versions of the problem. Instead of focusing on the relationship between individual bases, we look for related segments. The problem is substantially different from the large-scale problem of finding related subsequences: in the large-scale problem, we make no requirements on the types of mutations that may occur between the subsequences. As a result, the large-scale related subsequences are much larger than a typical segment in our new algorithm.

In Chapter 4, we present three new algorithms. First, we improve the small-scale problem by extending our model of evolution to be more descriptive. Our new model can describe a fixed number of distinct evolutionary regimes, each of which mutates at a different rate. Many parts of the genome change at a slower rate than average due to being less tolerant of mutation. Typically, these sections describe crucial cell mechanics that organisms require to survive. Changes that break these mechanisms quickly disappear from the genome since the host organism dies quickly.

Second, we present two techniques for working with very long DNA sequences. The long length of genome sequences introduces a related problem: our algorithms become impractical due to computation time and memory constraints. Thus, in practice we must use heuristics to constrain the computation. We present improved techniques for two classic problems in this domain. The first, *global alignment*, assumes that two input sequences are completely related to one another and that no rearrangements occur. The second, called *local alignment*, relaxes these assumptions by instead finding subsequences that exhibit an evolutionary relationship. Our improvements to both methods come from evaluating not just the single most likely evolutionary relationship between sequences, but all possible relationships under a model.

We now describe some notation, give a more detailed introduction to sequence alignment, and describe the mathematical framework on which we base most of our new techniques.

1.1 Sequence alignment: definitions and interpretation

First we define some basic notation. We use x and y to denote sequences. For sequence x, we refer to position i as x_i and let the first position be x_1. We denote the subsequence of x from i to j as $x_{i...j}$. We use capital letters for matrices, vectors, and sets. The size of vector V is $|V|$ and the size of set S is $|S|$. For vector V, let $V(i)$ be element i where

$0 \leq i < |V|$. Similarly, we denote elements of matrix M as $M(i, j)$, where i is the row and j is the column. We now describe the sequence alignment problem and related notation.

Sequence alignment is the primary tool for finding evolutionary relationships between DNA sequences. A DNA sequence is a string over four symbols: A, T, C, and G. These symbols represent the nucleotides adenine, thymine, cytosine, and guanine, respectively. As time passes, DNA sequences incur mutations from a variety of physical processes. Thus, DNA sequences from individuals of a species contain many differences. Over longer periods of time, these mutations combined with environmental conditions lead to speciation. When comparing DNA sequences from different species, large scale changes, such as long insertions and deletions, duplications, reversals and translocations, are common. The goal of sequence alignment is to infer which physical mutations occurred using only the mutated sequences and a mathematical model that abstracts the physical mutation processes.

Given sequences $x = x_1 x_2 \dots x_n$ and $y = y_1 y_2 \dots y_m$, we can produce a sequence alignment by inserting special gap characters, usually indicated with a dash, into each sequence such that the *aligned* sequences have equal length. An alignment, like the one in Figure 1.1, corresponds to an evolutionary history described in terms of three types of physical mutations: point mutations, insertions, and deletions. To interpret an alignment, we stack the aligned sequences on top of one another and look at the content of the columns.

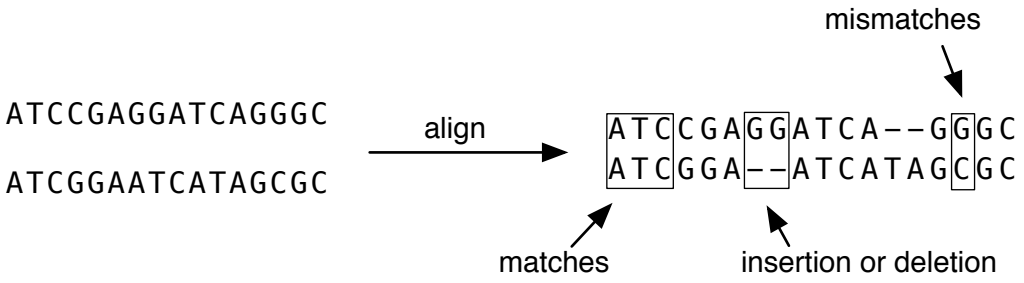

Figure 1.1: An alignment of two input sequences.

When two nucleotide symbols are in the same column, they either *match* or they *mismatch*. These columns describe two evolutionary predictions. First, the nucleotides corresponding to each symbol are predicted to have a common ancestor. Second, a match/mismatch column makes a prediction about point mutations having occurred in the time since the original sequences diverged. For example, if the symbols mismatch, then one or both of the nucleotides corresponding to the mismatching symbols incurred a point mutation. We let $x_i \diamond y_j$ mean that symbols x_i and y_j in sequences x and y are in the same alignment column.

When a nucleotide symbol shares a column with a gap symbol, we refer to this as either an insertion or deletion (*indel*). Typically, we consider consecutive gap columns as a single unit, corresponding to a single mutation event. Columns containing only gap symbols have no meaning and are thus disallowed. For a pair of sequences, we make no distinction between insertion and deletion events as it is impossible to know the true sequence of the common ancestor. It could be that the subsequence in question was deleted from the common ancestor in one sequence, or that it was never in the common ancestor at all and was instead inserted into the other sequence.

There are many possible alignments for two input sequences. There are two approaches to producing alignments: score-based and probabilistically. These approaches differ in the way they modelWe give a brief introduction to both frameworks, although our work uses probabilistic alignment.

1.1.1 Score-based alignment

Score-based alignment measures sequence similarity with edit distance. Here, *matches*, *mismatches*, and *indels* are edit operations transforming one sequence into the other. We assign each edit operation a cost, and the sum of the costs allows us to assign a similarity measure to a particular alignment. The goal of sequence alignment in this framework is to find an alignment with maximum score.

This framework was introduced in 1970 by Needleman and Wunsch [62] and is the foundation for many modern alignment algorithms. The original Needleman-Wunsch alignment algorithm uses dynamic programming to find the optimal alignment in $O(m^2n^2)$ time and space for two sequences of length m and n. In 1982 Gotoh [37] presented an improved version that takes $O(mn)$ time and space. We describe the improved version here.

Let $s(x_i, y_j)$ be the score of pairing x_i with y_j. Most simple scoring schemes assign all matching pairs one score, and all mismatching pairs another. Originally, gaps are scored linearly: each gap symbol incurs a penalty d. We can compute the score of the optimal alignment with a simple recurrence:

$$D(i,j) = \max \begin{cases} D(i-1, j-1) + s(x_i, x_j) \\ D(i-1, j) + d \\ D(i, j-1) + d \end{cases} \tag{1.1}$$

To compute the actual alignment, we remember our choices for each $D(i, j)$ and then trace these backwards from $D(n, m)$. The value of $D(n, m)$ is the score of the optimal alignment.

A major disadvantage of the above algorithm is that linear gaps do not reflect important properties of gaps in real biological sequences. With linear gaps, the frequency of gaps and their length are tied. In his 1982 paper introducing the more efficient Needleman-Wunsch algorithm presented above, Gotoh also introduced *affine* gap costs. Affine gap costs incur a penalty to start a gap, and a separate penalty to extend it. The alignment algorithm for affine gaps follows the same basic recurrence as above, but requires two extra matrices to model being in a gap:

$$D(i,j) = \max[D(i-1,j-1) + s(x_i, x_j), P(i,j), Q(i,j)] \tag{1.2}$$

where

$$P(i,j) = \max[D(i-1,j) + g, P(i-1,j) + d] \tag{1.3}$$

and

$$Q(i,j) = \max[D(i,j-1) + g, Q(i,j-1) + d]. \tag{1.4}$$

As before, the value of $D(n,m)$ is the score of the optimal alignment.

In 2009, Cartwright [23] introduced an alignment algorithm that scores gaps using a power law. This scoring scheme closely matches observed gap distributions in real sequences, but the alignment algorithm by Cartwright requires $O(n^3)$ time for two sequences of length n. Thus, it is practical only for very short input sequences.

1.1.2 Hidden Markov models and alignment

An alternate way of interpreting an alignment is to consider it to be a description of a specific evolutionary history of two sequences. *Matches* and *mismatches* represent symbols that come from a symbol in a common ancestor. *Indels* represent either an insertion or deletion in one of the sequences since the common ancestor. We note that an alignment in this framework does not completely pin down the evolutionary history. For example, a match may represent either no mutations since the common ancestor, or a series of mutations ending in the same symbol as in the ancestor. However, an alignment is still a very specific overall description of evolutionary history.

The score-based Needleman-Wunsch algorithm is equivalent to probabilistic alignment under a simple model of evolution. To describe how this is so, we first describe hidden Markov models, the framework on which we base probabilistic alignment. A hidden Markov model (HMM) is a description of a probabilistic process for generating sequences over a finite alphabet. HMMs have two distinct uses: generating random sequences from a model,

and, given a model and output sequences, inferring information about how those sequences were produced; this second use is often called HMM decoding [33].

An HMM consists of set S of states, a matrix T of transitions, a matrix E of emissions, and an alphabet Σ. The transition parameter $t_{i,j}$ is the probability that a process in state i jumps to state j in the next step. The emission parameter $e_{i,a}$ is the probability that we emit symbol a from state i. For fixed i we must have $\sum_j t_{i,j} = 1$, where $t_{i,j} \geq 0$ and $\sum_{a \in \Sigma} e_{i,a} = 1$, where $e_{i,a} \geq 0$. States may also be *silent*, emitting no symbol, which we denote as ϵ. We must always define a *start state* where the process begins. HMMs may generate infinite or finite sequences. In the latter case, we must also designate a *stop state*. See Figure 1.2 for an example of a simple HMM that generates infinite sequences over the alphabet $\Sigma = \{a, b\}$.

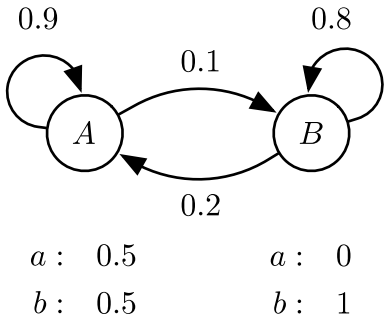

Figure 1.2: A hidden Markov model that generates strings over $\Sigma = \{a, b\}$. State A emits a and b equally as often and state B always emits b. Once in state B, we stay there with probability 0.8 and transition with probability 0.2.

Generating a sequence given an HMM is straightforward. Given an HMM in state i, we jump to state j with probability $t_{i,j}$ and emit symbol a with probability $e_{j,a}$. We repeat this step, jumping from state to state and emitting symbols in a left to right fashion, either indefinitely or until we reach a stop state. We call the sequence of states we use a *path*. Each path π has a probability value that is the product of all the transitions in the path. The joint probability of a path and a particular output string is the product of both the transitions in the path and the emissions from the path states. This later probability corresponds to the probability of a particular alignment.

1.1.3 The Viterbi algorithm

While generating random strings is useful in itself, the real power of hidden Markov models comes from using them to infer how supplied output strings were generated. There are several important HMM inference or decoding algorithms; for now, we focus on one technique which, when applied to a particular model, is equivalent to score-based alignment.

Given an HMM and a sequence x, there are many possible state sequences, or *paths*, that produce x. A path of highest probability, π^*, is often useful. For HMMs that correspond to physical processes, such a path corresponds to a sequence of events that has the highest likelihood of having taken place.

$$\Pr(x|\pi^*) = \operatorname*{argmax}_{\pi} \Pr(x|\pi) \tag{1.5}$$

For an HMM with $|S|$ states that produces a single string of length n, we can find π^* in $O(|S|^2 n)$ time using the Viterbi algorithm. A path that satisfies equation 1.5 is often called a Viterbi path. The Viterbi algorithm is a simple dynamic programming method:

$$\begin{array}{lrcl} \text{Initialization:} & V(1,s) & = & 1 \\ \text{Recursion:} & V(i,k) & = & \max_\ell V(i-1,\ell) t_{\ell,k} e_{x_i,k} \end{array} \tag{1.6}$$

Here, s is the starting state and e is the ending state. The joint path and emission probability is

$$\Pr(x|\pi^*) = V(n,e). \tag{1.7}$$

Variations on this algorithm allow multiple starting or ending states [33]. To recover π^*, we must also remember the choice we made in assigning each $V(i,k)$. We build π^* backwards by tracing our choices of $V(i,k)$ from $V(n,e)$ to $V(1,s)$.

1.1.4 Pair hidden Markov models

So far we have described hidden Markov models that generate a single sequence. However, HMMs may also emit two or more sequences. We call an HMM that generates two sequences a pair hidden Markov model (pair-HMM). States now emit, two, one, or no symbols. Let $e_{a,b,k}$ be the probability that state k emits a in the first sequence and b in the second sequence. As before, let ϵ represent the case where we emit no symbol in a particular sequence.

As with HMMs that generate one sequence, pair-HMMs can both generate random sequences and perform inferences given two supplied sequences. The Viterbi algorithm for a pair-HMM takes time and space $\Theta(|S|^2 mn)$ for sequences of length n and m. We modify the dynamic programming recurrence in equation 1.6 as shown in equation 1.8 below.

$$V(i,j,k) = \max_{\ell} \begin{cases} V(i-1,j-1,\ell)t_{\ell,k}e_{x_i,y_j,k} \\ V(i-1,j,\ell)t_{\ell,k}e_{x_i,\epsilon,k} \\ V(i,j-1,\ell)t_{\ell,k}e_{\epsilon,y_j,k} \end{cases} \tag{1.8}$$

There are pair-HMM versions of many standard HMM inference algorithms. We discuss these in Chapter 3.

1.1.5 The alignment pair-HMM

The standard HMM for alignment, shown in Figure 1.3 is a simple three state pair-HMM that produces two infinite sequences of nucleotides over the DNA alphabet $\Sigma = \{A, T, C, G\}$. State M emits pairs of symbols that share a common ancestor. These symbols may mismatch, representing mutation, or they may match representing either no mutation or compensatory mutations. States I_x and I_y emit single nucleotides into sequence x and sequence y respectively. These states model insertions (or deletions) in one sequence relative to the common ancestor. Parameter α is the probability that we start an insertion and parameter β is the probability that we continue an insertion having already started it.

Given two biological sequences x and y, we use the Viterbi algorithm to find a state sequence π^*. The state sequence π^* corresponds to an optimal alignment of x and y according to the model. State M corresponds to alignment columns with two nucleotides and states I_x and I_y correspond to columns with a nucleotide over a gap character, or vice versa. To complete the mapping to score-based sequence alignment we convert probabilities to scores with an odds ratio. An odds ratio compares the probability that our sequences come from the pair-HMM in Figure 1.3 to the probability that our sequences come from a model of random DNA. We convert this ratio to a score by taking the logarithm, usually in base-2. The result is a score in *bits*, where positive values indicate that the data comes from our pair-HMM and negative values indicate it comes from the model of random DNA. Our model of random DNA is very simple, we draw symbols one at a time from a distribution over Σ. This distribution may be uniform over the DNA symbols, or some symbols may occur more often than others. For example, some areas of genomes have a high frequency of A's and T's, while other regions have a high frequency of G's and C's.

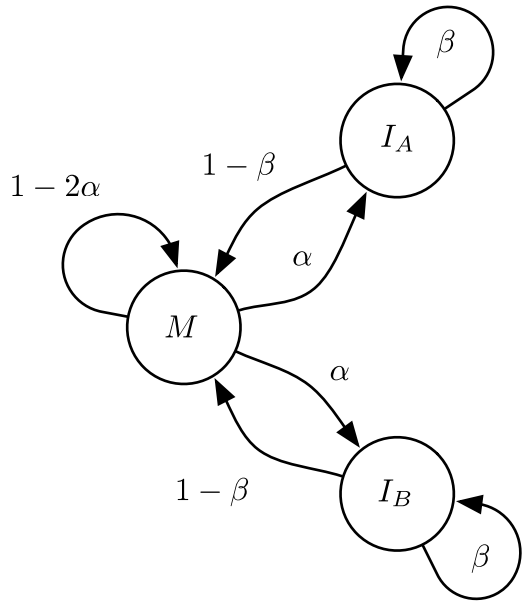

Figure 1.3: The traditional pair-HMM for alignment. State M describes bases that share a common ancestor. States I_x and I_y describe insertions or deletions into sequences x and y respectively. Parameters α and β model affine gaps where α corresponds to the gap open cost and β to the gap extension cost. To translate probabilities into scores we take the logarithm of an odds ratio comparing the model probabilities to those from a null hypothesis. The null hypothesis is that the input symbols are drawn randomly from a background distribution.

In this framework, Viterbi decoding is identical to the Needleman-Wunsch algorithm with affine gaps [33]. We can model linear gaps by setting α and β to be equal. Both Viterbi and Needleman-Wunsch take time and space $\Theta(mn)$.

1.2 Summary

In this thesis we focus on improving probabilistic sequence alignment. While basic sequence alignment has enjoyed much success, it fails to capture critical properties of real biological data. We address this by extending the basic pair-hidden Markov model for alignment to allow for more descriptive alignments. Additionally, we show how to use more sensitive inference algorithms for pair-HMMs to improve homology detection. These techniques have no score-based counterpart, nor do they return an alignment. However, they still fall within the context of probabilistic alignment.

In the next chapter we describe related work that includes both probabilistic techniques as well as score-based techniques. We also discuss several heuristics for aligning very long sequences. In the remaining chapters, we describe our contributions to the area of probabilistic alignment.

Chapter 2

Related Work

In this chapter we discuss work that is both directly related to our improvements, and supporting work which we use to build fully functional, and practical alignment programs. We describe related work on six basic topics. First, we describe methods for using hidden Markov models to perform various types of inferences. Second, we describe a heuristic called *seeding* that finds very short highly similar subsequences in linear time in the length of the input. Third, we describe a heuristic for constraining the computation time and space of global alignment called *anchoring.* Fourth, we describe existing alignment algorithms and models of evolution. Fifth, we describe heuristics and techniques for performing local alignment. And finally, we describe work related to inferring alignment parameters from real DNA sequences.

2.1 HMM decoding

Despite the importance of sequence alignment, there are relatively few alternatives to Viterbi decoding for sequence alignment. The most common alternative decoding technique is posterior decoding. Here, rather than optimizing over the paths through a pair-HMM, we optimize the locations of individual symbol pairs over all possible paths. This technique maximizes the accuracy of matches and mismatches by considering all possible evolutionary histories, and all of the input data, when choosing to pair two symbols. Its main downside is inaccurate insertions, as the algorithm does not consider gap structure when choosing to pair a symbol or leave it unpaired.

A variant of posterior decoding, called posterior-Viterbi, attempts to get the best of

both Viterbi and posterior decoding by considering a particular path in addition to posterior probabilities in the final alignment step. In the next two sections we describe both these techniques in detail and describe existing work that uses them.

2.1.1 State posterior decoding

Posterior decoding [33] is one of the most useful alternate decoding techniques to Viterbi. Its power comes from the use of posterior probabilities. Given a sequence x and an HMM, the state posterior probability $\Pr(\pi_i = k|x)$ describes the likelihood that symbol x_i is emitted from state k. In contrast to Viterbi, which concerns itself with single paths, the state posterior probability considers *all* paths that produce the sequence x.

The algorithm for posterior decoding has three components: the forward algorithm, the backward algorithm, and a final step combining the forward and backward probabilities. Each step can be implemented with dynamic programming taking $O(|S|^2 n)$ time and space for an input sequence of size n and HMM with $|S|$ states.

Forward algorithm

The forward algorithm computes the overall probability that a hidden Markov model produces a sequence x. Conceptually, it produces the sum of the probabilities of all paths that generate x. The forward algorithm is a dynamic programming algorithm that takes $O(|S|^2 n)$ time and space for a sequence of length n and HMM with $|S|$ states, start state s, and end state e.

$$\begin{array}{lrcl} \text{Initialization:} & F(1,s) & = & 1 \\ \text{Recursion:} & F(i,k) & = & \sum_\ell F(i-1,\ell) t_{\ell,k} e_{x_i,k} \end{array} \tag{2.1}$$

In the recurrence in equation 2.1, each $F(i,k)$ computes the probability that we produce $x_{1\ldots i}$ given that we emit x_i from state k. The probability that we produce x is

$$\Pr(x) = \sum_\pi \Pr(x|\pi) = F(n,e). \tag{2.2}$$

Backward algorithm

The backward algorithm is the counterpart to the forward algorithm. It computes the same value, that is, the probability that an HMM produces a sequence x, but does so with a recurrence that processes x from the end rather than the beginning.

$$\begin{array}{lrcl} \text{Initialization:} & B(n,e) & = & 1 \\ \text{Recursion:} & B(i,k) & = & \sum_\ell t_{k,\ell} e_{x_i,\ell} B(i+1,\ell) \end{array} \tag{2.3}$$

The value $B(i,k)$ in equation 2.3 is the probability that we produce the subsequence $x_{i+1\ldots n}$ starting from state k. The probability that we produce x is

$$\Pr(x) = \sum_\pi \Pr(x|\pi) = B(1,s). \tag{2.4}$$

State posterior sequence

Combining the forward probabilities and backward probabilities, we obtain a formula for the state posterior probability.

$$Pr(\pi_i = k|x) = F(i,k)B(i,k)/Pr(x) \tag{2.5}$$

As a final step, we produce a sequence $\hat{\pi}$ where each $\hat{\pi}_i$ maximizes equation 2.5 for a position i. The path $\hat{\pi}$ may not represent a valid path through the HMM, nor does it preserve length distributions on states.

2.1.2 State posterior decoding for pair-HMMs

The forward and backward algorithms extend naturally to pair hidden Markov models, although the time and space complexity increase substantially. For sequences x and y with lengths n and m, and a pair-HMM with k states, both the forward and backward algorithms take time and space $\Theta(kmn)$. The forward recurrence becomes

$$F(i,j,k) = \sum_\ell \begin{cases} F(i-1,j-1,\ell) t_{\ell,k} e_{x_i,y_j,k} \\ F(i-1,j,\ell) t_{\ell,k} e_{x_i,\epsilon,k} \\ F(i,j-1,\ell) t_{\ell,k} e_{\epsilon,y_j,k} \end{cases} \tag{2.6}$$

where $F(i,j,k)$ is the probability that we produce $x_{1\ldots i}$ and $y_{1\ldots j}$ ending in state k. The backward recurrence is defined similarly with $B(i,j,k)$ being the probability that we produce $x_{i+1\ldots n}$ and $y_{j+1\ldots m}$ starting from state k. We define $\Pr(x_i \diamond y_j|x,y,k)$ to be the probability that we emit x_i and y_j from state k. In terms of forward and backward values we have

$$\Pr(x_i \diamond y_j|x,y,k) = F(i,j,k)B(i,j,k)/\Pr(x,y). \tag{2.7}$$

There is no straight-forward analogue to the final step of the posterior decoding algorithm for HMMs that output one sequence. However, for pair-HMM for sequence alignment Miyazawa [58] describes a *probability alignment* which captures a similar intention. In a probability alignment we assign to each x_i the y_j that maximizes

$$\Pr(x_i \diamond y_j | x, y) = \operatorname*{argmax}_k \Pr(x_i \diamond y_j | x, y, k). \tag{2.8}$$

As with posterior decoding on HMMs that output a single sequence, this alignment may not represent a valid path through the pair-HMM. In that sense it may not even be a valid alignment. However, Miyazawa shows that if we only assign pairs where $\Pr(x_i \diamond y_j | x, y) \geq 0.5$, and assign the remaining symbols to gaps, the alignment will always be valid. In addition to ignoring the gap length distributions, the biggest problem with this approach is that very few symbols are aligned.

2.1.3 State posterior-Viterbi decoding

Fariselli, Martelli, and Casadio [35] introduce posterior-Viterbi decoding in which they limit the state path to valid paths through the HMM. They replace the final step of the posterior decoding algorithm with a Viterbi-like recurrence. Let

$$\delta(s, k) = \begin{cases} 1 & \text{if } t_{s,k} > 0 \\ 0 & \text{if } t_{s,k} = 0 \end{cases} \tag{2.9}$$

and the posterior-Viterbi recurrence be

$$PV(i, k) = \max_\ell PV(i-1, \ell)\delta(\ell, k) \Pr(\pi_i = k | x). \tag{2.10}$$

The optimal state path is recovered by remembering each choice of $PV(i, k)$ and tracing backwards from the end state.

2.1.4 Optimal accuracy alignment

A similar algorithm was proposed earlier for the alignment pair-HMM. In 1998, Holmes and Durbin [43] introduced the *optimal accuracy alignment*. This alignment maximizes the state posterior probabilities for match states. The recurrence is simple

$$OA(i, j) = \max_\ell \begin{cases} OA(i-1, j-1) \Pr(x_i \diamond y_j | x, y, M) \\ OA(i-1, j) \\ OA(i, j-1) \end{cases} . \tag{2.11}$$

Here we let M be the match state and our recurrence is over the the lengths of input sequences x and y.

2.1.5 AMAP alignments and marginalized posterior decoding

Introduced in 2007 by Schwartz and Pachter [71, 72], and later used in FSA [11], the AMAP alignment algorithm is similar to the optimal accuracy alignment, but also defines posterior probabilities for insertion states. Specifically, they define $\Pr(x_i \diamond -|x, y) = 1 - \sum_{j=1}^{m} \Pr(x_i \diamond y_j|x, y)$, the probability of no homology for x_i, and give a similar definition for $\Pr(- \diamond y_j|x, y)$, the probability of no homology for y_j. Schwartz also adds a heuristic parameter γ for weighting the impact of insertion posterior probabilities. An optimal AMAP alignment h^γ is defined as

$$\begin{aligned} h^\gamma &= \text{argmax}_h \, 2 \sum_{(i,j)\in h_P} \Pr(x_i \diamond y_j|x, y) + \\ &\quad \gamma \left(\sum_{i\in h_{I_x}} \Pr(x_i \diamond -|x, y) + \sum_{j\in h_{I_y}} \Pr(- \diamond y_j|x, y) \right), \end{aligned} \tag{2.12}$$

where h_P is the set of positions that are paired in h, h_{I_x} is the set of positions in x that are unpaired, and h_{I_y} is the set of positions in y that are unpaired. The parameter γ must be between 0 and 1 and controls how much impact the gap posterior probabilities have on the final alignment.

In 2008 Lunter *et al.* [54] introduce marginalized posterior decoding (MPD) for both the standard alignment pair-HMM and an extended alignment pair-HMM that models short and long insertions. MPD decoding is equivalent to AMAP with $\gamma = 1$.

2.1.6 Summary

We have described four decoding techniques for hidden Markov models. Where appropriate, we have also described related algorithms designed specifically for the alignment pair-HMM. See Table 2.1 for a summary. The techniques in this thesis focus mainly on the forward algorithm for pair-HMMs. Although typically the forward and backwards algorithms are used as part of a more complicated decoding technique, they may also be used on their own.

Optimality Goal	HMM	Alignment pair-HMM
single path	Viterbi	Viterbi
state posterior sequence	posterior decoding	probability alignment
valid state posterior sequence	posterior-Viterbi decoding	optimal accuracy alignment AMAP alignment marginalized posterior decoding
transition posterior sequence	transition posterior decoding	none
probability related	forward or backward	forward or backward

Table 2.1: Summary of HMM decoding techniques relevant to sequence alignment. For each goal, we describe the relevant algorithm to achieve it on a single HMM as well as the alignment pair-HMM.

2.2 Seeding

Seeding is the first heuristic step in many anchoring techniques. It is also crucial to other heuristic forms of alignment such as *local alignment*, one of the topics of Chapter 4. The goal of seeding is to find a first set of candidate points for further evaluation.

The basic idea is to look for short subsequences that satisfy some simple matching requirements. The simplest seed, called a *consecutive seed*, looks for k consecutive matches between two sequences. Most early alignment programs such as BLASTN [5, 6] and FASTA [52] use this type of seed. We call a pair of subsequences that satisfy a seed a *seed hit*. See Figure 2.1 for an example. Finding all consecutive seed hits for sequences of length m and n takes $O(m+n)$ time with an implementation that uses hashing.

```
   111111
ATTGGATACGGATAG
ATCCGATACGGATCA
```

Figure 2.1: Example of a consecutive seed hit between two sequences. We denote the seed requiring $k = 6$ matches as a sequence of '1's. Each '1' indicates that the position must match. We adopt this notation for consistency with more complicated seeding descriptions. Symbols joined with a dot indicate sequence positions satisfying the seed.

2.2.1 Suffix trees and maximal matching

Extending the concept of consecutive matches are maximal unique matches (MUMs) and maximal exact matches (MEMs). A MEM is a pair of subsequences where extending the sequences to the left or right introduces a mismatch. A MUM adds the constraint that the pair of subsequences must be unique among all such pairs between two sequences. We can find both MUMs and MEMs in $O(n+m)$ time and space using a data structure called a suffix tree [39]. MUMs are used for anchoring in the MUMMER and MUMMER2 [30, 31, 53] genomic alignment programs. MEMs are used in MAVID and AVID [12, 13] and GAME [26].

2.2.2 Spaced seeds

Although FLASH [22] used a collection of spaced seeds in 1993, and WABA [49] used a specific spaced seed in 2000, the general idea of spaced seeds was made popular by PatternHunter [55] in 2002. Here, we may denote positions that can either match or mismatch in our matching criteria. The most common notation for a spaced seed is a string of 1's and 0's, where 1 positions denote that a match is required, and a 0 position denotes a position we may ignore. For example, the seed 1101 requires two matches followed by either a match or mismatch, and ending with a match. Under this scheme, consecutive seeds become strings of k 1's. See Figure 2.2 for an illustrated example.

```
    1101101001
ATTGGCTACGGTTAG
 : :  : :  :
ATCGGATACGGATCA
```

Figure 2.2: Example of a spaced seed hit between two sequences. The seed requires six matches, denoted by '1's, and has four *do not care* positions, denoted by '0's. Symbols joined with a dot indicate sequence positions satisfying the '1' positions of the seed.

The staggering of the required matches has two purposes. First, it reduces the total number of possible substrings by extending the length of the seed. And secondly, by reducing the likelihood of adjacent hits. If a consecutive seed hits a location containing many consecutive matches, it will also hit all adjacent locations until a mismatch occurs. In contrast, a space seed hit does not guarantee that the hit location contains many consecutive matches, thus adjacent positions may not hit the seed. For long regions with

long consecutive matches, the longer length of spaced seeds also contributes to fewer adjacent matches. Sensitivity improvements depend on the choice of seed pattern. Many authors [14, 21, 47, 27] have designed algorithms for finding optimal seeds given a statistical profile for homologous sequences.

PatternHunter II [46] achieves further improvement in sensitivity by using more than one spaced seed. This introduces the problem of choosing an optimal *set* of spaced seeds, rather than finding an optimal single seed. There are many approaches to choosing an optimal set of spaced seeds [46, 77, 80]. As with a single seed, a set of seeds can be optimized to detect particular types of homology.

2.2.3 Other seeding strategies

While the above seeding strategies are the most popular, there are still many more strategies. Chen and Sung describe half gapped seeds [24], a variation on spaced seeds. They allow positions to be labelled as half matches, which require either a match in that position, or that the symbol in the first subsequence at the positions matches one of the two neighbouring positions in the second subsequence. See Figure 2.3 for an example of the ways a half position can be satisfied. The YASS [64] local aligner uses transition seeds that extend spaced seeds with positions that match transition mutations (A $\leftrightarrow$ T or C $\leftrightarrow$ G). Mismatch seeds, used by CHAOS [19] and BLAT [48], limit the number of mismatches that occur in a fixed window of size k. A hit is considered to be any pair of subsequences of length k, with fewer than n mismatches.

Figure 2.3: Example of the ways a half gap seed can be satisfied. Here we specify a half match position as '1/2'. Symbols joined with a dot indicate sequence positions satisfying the '1' positions of the seed and symbols joined with a line indicate sequence positions satisfying the '1/2' position.

Vector seeds [15, 17] generalize and extend both spaced seeds and mismatch seeds. A vector seed is represented as a sequence of numbers and a threshold. Given a pair of subsequences and a vector seed, we compute a score by summing the numbers at each match position. We call the pair a seed hit if the score is greater than or equal to the threshold. For example the seed $((1, 0, 1, 1), 3)$, is a vector seed that is equivalent to the spaced seed

"1011." and the seed $((1,1,1,1,1,1),5)$, is a mismatch seed of length 6 allowing one mismatch. Vector seeds can also express more complex match criteria that combine both spaced seed features, mismatch seed features, as well as the ability to assign different weights to each match location. For example, the seed $((1,1,0.5,1,1,0.5),4)$ scores the third match positions by half each and only requires a total score of 4. This seed is not equivalent to any spaced seed or mismatch seed.

Finally, Mak, Gelfand, and Benson [56] describe indel seeds that allow small insertions and deletions within hit constraints. An indel seed pattern may contain an 'X' symbol, which indicates that the position is either a match, or is *missing* from one of the two subsequences under consideration. For example the seed "11XX11" allows either two missing symbols at the 'X' positions, one missing symbol, or no missing symbols.

2.3 Anchoring alignments of long sequences

The techniques of Section 2.1 work well on sequences with lengths of up to several thousand symbols. However, the quadratic run time of those algorithms make their use on sequences of millions of symbols impractical. We solve this problem using *anchoring*, a heuristic for constraining the dynamic programming space of the algorithms in Section 2.1.

Anchoring algorithms by necessity must be fast. The goal is to find pairs of short subsequences from both input sequences that are highly likely to be part of the final alignment or alignments. Anchoring typically uses several steps: finding a set of candidate position pairs, evaluating each pair, and finally choosing a subset of pairs to use as anchors. We use these three phases in the following sections that describe existing anchoring techniques for global genomic alignment. Although our focus in on pairwise alignment, we also include global multiple aligners when appropriate. Pairwise alignment is a sub-problem of multiple alignment.

2.3.1 Notation and definitions

Given sequences x and y, a *fragment* consists of a pair of subsequences $x_{i...j}$ and $y_{k...\ell}$. An *anchor* is a fragment that we use to constrain a dynamic programming algorithm operating on sequences x and y. We say an anchor or fragment is *correct* if at least one base from the subsequence of x and one base from the subsequence of y truly share a common ancestor, and *incorrect* otherwise. The *area* of subsequences $x_{i...j}$ and $y_{k...\ell}$ is $(j-i+1)(\ell-k+1)$.

We say a set of fragments *covers* x and y if every base of x and y is contained in exactly one fragment.

Since score-based alignment may be more familiar, we present the probabilities of opening and extending gaps as scores in the remainder of this document. To convert a probability to a score in bits, we take the negative base-2 logarithm of the probability value.

2.3.2 Fragment chaining heuristic

Fragment chaining is a widely used heuristic for choosing a set of anchors from a set of potential anchors, or *fragments*. Chaining has two goals: choosing a set of anchors consistent with a global alignment, and choosing a set of anchors that is most likely to be correct among the set of consistent fragment sets.

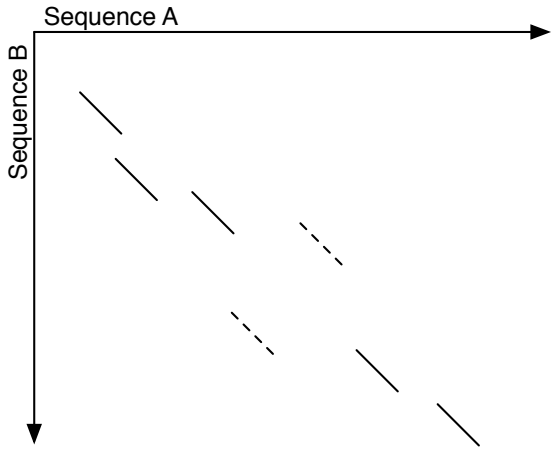

Figure 2.4: Example of fragments in a chain. The diagonal lines represent paired subsequences from sequences A and B. The solid fragments can be part of a valid chain. The dashed fragments are said to *conflict* because they cannot both be included in the chain of solid fragments.

A *consistent* set of fragments is a set that we are able to build a global alignment around. A set of fragments under consideration for anchoring typically includes many false fragments. These false fragments represent positions that satisfy the requirements of the previous steps despite not being truly homologous. As a result, the set of potential anchors will contain many conflicting fragments. A pair of fragments *conflict* if they cannot both be part of a single global alignment. For example, in Figure 2.4 there is no global alignment

that contains both of the designated fragments. Each fragment may also have a score that represents its likelihood to be correct. Chaining can take such scores into consideration when resolving conflicts.

Optimal fragment chaining uses dynamic programming to choose a set of fragments that do not conflict and have the highest combined score. For n fragments the naive algorithm takes time $O(n^2)$, although more sophisticated chaining algorithms bring this down [34].

2.3.3 MUMMER and MUMMER2

MUMmer [30] and MUMmer 2 [31], introduced in 1999 and 2002 respectively, use suffix trees to find a set of fragments. In MUMmer, the set of fragments is the set of maximal matches that are unique in all sequences (MUMs). MUMmer 2 relaxes the uniqueness requirement in the query sequence, finding all maximum exact matches (MEM) instead. The final set of anchors is taken to be the longest non-conflicting chain of fragments. This is equivalent to fragment chaining with each fragment having a score of 1.

2.3.4 GLASS

In 2000, GLASS [8] introduced *recursive anchoring*. In this scheme, anchors are chosen in passes, where the anchoring requirements are relaxed in each pass. The intuitive idea is that obviously correct anchors should be chosen first, followed by weaker fragments in regions that are still too large to align with dynamic programming.

Each phase consists of four steps. First, GLASS uses consecutive seeding to find a set of initial fragments. Second, the score of each fragment is set to the sum of the alignment scores for the left and right flanking twelve positions. In the third step, GLASS uses fragment chaining to reduce the set of fragments to the highest scoring set of consistent fragments. Finally, any fragments within this set that have a score below a threshold are discarded.

GLASS recursively applies this procedure between anchors, reducing the size of the initial consecutive seed each time it recurses.

2.3.5 OWEN

OWEN [69, 65], in 2002, introduced an anchoring procedure that combined automated heuristics with manual adjustments from a user. OWEN allows users to manually issue

commands that perform specific heuristics. For example, users can ask the system to find similarities between two subsequences, either exhaustively, or using consecutive seeding. A similarity in OWEN is a set of *islands* in close proximity to each other. Each island, similar to our fragments, represents a pair of subsequences with at least m matches in each frame of length n, and at least k overall frames. This is similar to an ungapped local alignment.

Other actions include filtering similarities by p-value, resolving conflicts by chaining similarities or by using a greedy conflict resolution algorithm, and performing final refined alignment with Needleman-Wunsch.

2.3.6 LAGAN and CHAOS

LAGAN [20], introduced in 2003, uses CHAOS [19] alignments as fragments in its anchoring procedure. CHAOS forms local alignments by using mismatch seeds to find an initial set of fragments, then forming *local* chains of seeds using a cost function that considers both the diagonal distance between hits as well as the off diagonal distance, which are forced indels. Chains terminate when the distance to the next fragment is too large. The score of each CHAOS alignment is the total number of matches in the seed hits that it contains.

LAGAN chooses a set of CHAOS alignments as anchors using the standard chaining heuristic. Unlike previous aligners that require the final alignment to pass through each anchor, LAGAN uses anchors as a loose constraint, requiring only that the final alignment pass within several bases of each anchoring CHAOS alignment. The CHAOS based anchoring procedure from LAGAN is also used in conjunction with DIALIGN [19].

2.3.7 AVID

AVID [12] is a global aligner that uses suffix trees to seed its anchoring procedure. Unlike MUMmer, AVID uses both maximal exact matches (MEMs) and maximal unique matches (MUMs) from both sequences. AVID also uses a recursive anchoring strategy similar to GLASS.

The anchoring procedure starts by finding all MEMs, which includes the MUMs, then discards all that are less than half the length of the longest MEM. Each MEM is assigned a score that is based on the length of the MEM and the score of the alignments for the 10 bp to the left and right of the MEM. The optimal chaining heuristic is then used to select anchors first from all the MUMs, then recursively from the MEMs. The algorithm recurses from the beginning on all unanchored subsequence pairs that are too long to align.

2.3.8 ACANA

ACANA [44] uses a greedy approach to anchoring genomic sequences of up to several thousand bases. Although ACANA requires $O(mn)$ time for sequences of length m and n, and is thus too slow to anchor long genomic sequences, we include it here for its ability to align distantly related sequence, a major focus of our work.

ACANA uses local alignments, the topic of section 2.5, to anchor global alignments of genomic sequences in a greedy, GLASS-like recursive anchoring procedure. In each recursive step, ACANA first finds all local alignments and assigned each a modified score of $v \log u$, where v is the score of the alignment and u is the length. Alignments with score below 1 or length below 5 are assigned a modified score of 0. Alignments with modified score less than 90% of the top scoring alignment are filtered out. The remaining alignments are assigned a new score $G_a = u_a + \sum_b u_b$, where u_a is the previous modified score for alignment a, and b iterates over all remaining non-conflicting alignments. The alignment a with highest G_a is taken as an anchor. The algorithm then recurses in each flanking region.

2.3.9 Work of Brown and Hudek

In 2004, we [18] introduced a prototype global multiple aligner that uses a mathematical model of global alignment to select a seed weight and a score threshold to minimize the number of incorrect anchors. Given a weight w and threshold T, they use a prefix or repeat of a single spaced seed with weight w to find a set of fragments. The score of a fragment for two sequences is the sum of the matches and mismatches within it. Fragments with score below T are discarded and anchors are selected from the remaining fragments with the standard chaining heuristic. I extended this work in my masters thesis [45] to include GLASS-like recursive anchoring, both for fragment finding and in the model of anchored global alignment.

2.3.10 Other anchoring techniques

PECAN [67] uses exonerate [75] local alignments as anchor candidates. FSA [11] uses either MUMs or exonerate local alignments as anchor candidates. On pairs of sequences, FSA and PECAN use anchor-choosing procedures equivalent to the standard fragment chaining heuristic.

2.4 Models of pairwise alignment

In this section we describe both score-based models of alignment as well as models based on pair hidden Markov models (pair-HMMs). Many of these algorithms are part of either local or multiple aligners, but we include them as the techniques are relevant to our work.

WABA [49] uses a complex five state alignment pair-HMM. Their model emits symbols on transitions, rather than on entering a new state as presented in Chapter 1. The WABA pair-HMM models three new features: it models short gaps and very long gaps separately, highly conserved and weakly conserved non-coding DNA, and coding DNA.

MCALIGN2 [79] extends the standard pair-HMM for alignment to explicitly model gaps of lengths 1 bp, 2 bp, and 3 bp or more. This change is made due to the observation that gaps of length 1 and 2 bp are over-expressed in real alignments of two *Drosophila* species, if one assumes a geometric length distribution.

Michael *et al.* [57] introduced a score-based alignment algorithm that allows for the use of different scoring schemes for different sections of the alignment. Their *jumping alignment* model charges a *jump cost* to switch scoring schemes. We use this work as the basis for our own probabilistic model of evolution, which we describe in section 4.2 of Chapter 4.

Schultz *et al.* [70] present a jumping profile hidden Markov model (profile-HMM) for aligning a sequence to a multiple alignment A of sequence families. In their model the alignment of the sequence may jump between family subtypes of the multiple alignment. A family subtype can be thought of as a multiple alignment consisting only of sequences from A that belong to the same sequence subtype. There is a probabilistic jump cost associated with switching subtypes during alignment.

Pecan [67] is a multiple aligner based on Probcons [32] that uses posterior alignment constrained to the set of valid alignments. The Pecan alignment pair-HMM is extended to allow transitions between gap states, in addition to transitions between gap and match states. Paten et. al. also explore the use of a mixture model for gaps, extending the pair-HMM to model short and long affine gaps.

Arribas-Gil, Metzler, and Plouhinec [7] propose a pair-HMM for alignment that contains sub-models for strong and weak conservation. The sub-model for strong conservation has no gaps, and the sub-model for weak conservation has affine gaps. The alignment can transition between these models with a given probability.

Lunter *et al.* [54] use a pair-HMM for local alignment to study how model parameters and decoding techniques affect alignment quality. Their pair-HMM includes an optional model of long and short gaps. Using parameters obtained from blastz [73] alignments of

human and mouse sequence, they simulate sequences with various base compositions and mutation rates. For each set of training sequences, they test various properties including: using correct indel rates, correct substitution rates, a mixture gap model, the correct base composition, and two posterior decoding methods in addition to Viterbi. They find that the mixture gap model and posterior decoding had the largest impact on accuracy, with the other features providing little improvement. We note, however, that the fraction of homologous positions correct in all their data was fairly high, and they did not test substitution rates greater than 0.4. Our work is mostly concerned with alignment regions with accuracy rates much below 80% of positions being correct. Chapter 5 describes experiments on sequences containing much more mutation.

FSA [11] extends the standard pair-HMM for alignment by replacing the insertion states with two insertion states, one for short gaps and one for long gaps. For pairs of sequences, its alignment algorithm find the optimal accuracy alignment described in section 2.1.4.

DIALIGN [60, 61] constructs alignments by assembling a set of non-conflicting, fixed length, segment pairs, or *diagonals*, based on a weighting property for each diagonal. They define a *diagonal* as a pair of subsequences, both of length ℓ, from sequences x and y. While DIALIGN is capable of aligning more than two sequences, we focus only on the pairwise case. Although DIALIGN is more closely related to a local alignment procedure, we include it here for its significant departure from hidden Markov model based alignment techniques.

A diagonal of length ℓ containing m matches is given a weight that represents the likelihood of seeing m matches given p, the probability of seeing a match in neutral DNA. Two diagonals are said to conflict if they share any sequence positions. DIALIGN builds alignments by using dynamic programming to find the highest weight set of non-conflicting diagonals. To reduce noise, only diagonals with a weight above threshold T are used as input to the dynamic programming.

DIALIGN 2 [59] is improved by allowing different match probabilities for different pairs of bases, as well as modifying the weight of a diagonal to account for the length. The modified weighting function prevents bias towards aligning many shorter diagonals instead of one large diagonal. DIALIGN-T [76] introduces two additional heuristics: an extra filter to prevent bias towards local alignments rather than global alignments, and a diagonal weight adjustment to take into account matches between multiple sequences when aligning more than two sequences.

Sigma [74] is an alignment program that follows the general framework of DIALIGN. It uses a weight function similar to that of DIALIGN 2, but includes a measure of nucleotide pairs, known as *dinucleotides*. This allows Sigma to weight commonly occurring simple

repeats lower than DIALIGN further reducing false alignments.

Cartwright [23] describes a pair-HMM that models indels according to a power law by using a zeta distribution for gap lengths instead of the usual geometric distribution. The downside to this approach is increased run time. The training algorithm presented by Cartwright takes time $O(n^3)$ for two sequences of length n. Despite this, the zeta distribution models gaps with high accuracy compared to traditional insertion and deletion models.

2.5 Local pairwise alignment

In this section we focus on heuristics and extension algorithms for local alignment. Instead of aligning all of the input sequences, local alignment involves finding alignments of subsequences of the input sequences. Thus, local alignment indirectly solves the large scale problem of finding subsequences related by evolution, and also solves the small scale problem of finding which individual nucleotides share a common ancestor.

2.5.1 BLAST, BLASTZ, and LASTZ

The majority of local alignment algorithms are based on the techniques introduced in BLAST [5, 6]. In its current form, the BLAST algorithm consists of three distinct steps: find positions likely to be part of a local alignment, extend these resulting in a set of high scoring segment pairs (HSPs), and extend HSPs into full gapped alignments.

The original BLAST [5] used simple consecutive seeds to find an initial set of fragments to extend into HSPs. This was later extended to two-hit consecutive seeds [6], where two seed hits in close proximity are required for a region to be extended into an HSP. More recently, BLAST has been updated to use spaced seeds [1].

As a quick initial filter, BLAST uses the number of non-overlapping hits in a window of size A, ignoring regions with too few hits. For the remaining seed pairs, BLAST extends each to an HSP by performing an ungapped local extension from both the left and right ends of the second hit in a pair. To determine when to stop an extension, BLAST introduces the *x-drop* heuristic. Given a threshold X, we stop extending when the current local alignment score drops X below the best score seen so far. In BLAST, only the second seed hit in a two-hit seed pair is extended.

BLAST filters the HSPs according to the ungapped extension score and another threshold S_g. All HSPs with score above S_g are extended with full gapped extensions. Although a

version of BLAST exists to align pairs of sequences, the full BLAST algorithm is designed to align a query sequence against a database of target sequences. In this context, BLAST computes an expectation value (e-value) in addition to an alignment score for each local alignment. The e-value gives the expectation of seeing an alignment with a given score by chance, given a particular query length and database size. The e-value is used as a final filter in a BLAST search.

BLASTZ and LASTZ

BLASTZ [73] introduces a few additions to the BLAST framework. Instead of two-hit consecutive seeds, BLASTZ uses spaced seeds that additionally allow a single transition in one of the match positions. When scoring ungapped alignments, BLASTZ uses a measure of the local sequence composition rather than assuming a global background composition.

BLASTZ also introduces recursive searches, a technique that resembles recursive anchoring in global alignment. Unaligned regions between two alignments that are no more than 50 kb apart are aligned again, but with a shorter spaced seed. If the flanking alignments are within 10 kb, the filtering threshold for ungapped alignments is reduced.

Finally, BLASTZ allows local alignments to be constrained using the global alignment heuristic. After seed hits are expanded to HSPs, BLASTZ uses the fragment chaining heuristic to select only those HSPs that are part of the maximum scoring chain. BLASTZ allows local alignments in the reverse orientation to be included in the chain. Chaining a reverse orientation local alignment incurs a separate anti-diagonal chaining penalty.

LASTZ [40] is a reimplementation of BLASTZ that expands the options for seeding to include two-hit seeding as in BLAST, user defined seeds, and spaced seeds with more than one transition.

2.5.2 WABA

WABA [49] is a local aligner that follows the overall strategy of BLAST, but differs significantly in several ways. Because WABA uses a much more complicated, and thus slower, model of alignment (see Section 2.4), it aligns two sequences by splitting up the query into smaller segments, aligning each segment, then recombining these segments into longer local alignments.

Specifically, WABA splits up the input sequence into 2000 bp segments that overlap by 1000 bp. WABA then uses seeded ungapped alignments to find a region of 5000 bp in the

target that is mostly likely to represent a true homology to the 2000 bp query sequence. This region is then aligned to the query with the detailed alignment model. When each segment is aligned with the target, WABA attempts to merge short alignments by looking for locations that have at least 15 identical alignment positions and joined them at that point.

To find a target region for a given query sequence, WABA uses a technique similar to BLAST. First, WABA uses a spaced seed that ignores every third position. This is often known as the *wobble base* due to the way in which DNA codes for amino acids. Three DNA bases encode a single amino acid. However, there is redundancy in this encoding in that the encoding for particular amino acids often allows more than one base in the third position. Because of this, every third base in sequence that codes for proteins can often incur a mutation without affecting the protein. Thus, this base is often said to *wobble*.

Similar to the BLAST window filter for seed hits, WABA then greedily clumps hits that are within 48 bp of each other and on the same diagonal. The score of each clump is the square of the number of hits within it. All clumps with score that is less than 25% of the maximum-scoring clump are filtered out. Ungapped extensions are then performed on the remaining clumps. As before, the extensions with score below 25% of the best-scoring extension are discarded. The centre of the highest-scoring extension is taken to be the target region for the detailed alignment phase.

2.5.3 GAME

GAME [26] follows the three stage method of BLAST, but adds a global alignment-inspired anchor-chaining procedure that globally aligns regions between fragments. The GAME alignment procedure starts by finding all maximal exact matches (MEMs) using a suffix tree. It filters out all MEMs with length below a threshold T_m, then performs ungapped extensions on the remaining MEMs, expanding them into HSPs. Rather than using the x-drop heuristic to decide when to stop extending, GAME stops when the fraction of matches in the HSP drops below a threshold T_π. Finally, HSPs with length below T_e are discarded.

The remaining HSPs are then used to build local alignment using a greedy approach that combines global alignment with x-drop based gapped extensions. On each greedy step, GAME finds a pair of anchors that are within a distance T_d, of one another. The region between them is globally aligned using Viterbi and the classic pair-HMM for evolution. If the alignment score is above a threshold, the alignment is kept, including the pair of anchors. When no more anchor pairs are left, the ends of the remaining local alignments are extended using the x-drop heuristic.

In addition to using maximal exact matches, GAME can optionally use translated maximal exact matches (tMEMs). Given two DNA input strings, GAME finds all translated matches by first converting the input sequence into a protein sequence. Since there are three possible translation frames, three protein sequences are produced for each input sequences, one for each frame. A suffix tree finds all the MEMs among these translated sequences.

2.5.4 YASS

YASS [63, 64] uses the BLAST alignment procedure, but instead of using two-hit seeds, it uses transition spaced seeds described in section 2.2.3. In addition to positions that must match and positions that are ignored, YASS allows match or transition positions. As an initial filter before performing ungapped extensions, YASS groups seed hits based on both the diagonal distance and the off diagonal distance.

2.5.5 Exonerate

Exonerate [75] introduces a variety of heuristics for producing local alignments from a set of HSPs. When extending an HSP, exonerate extends from the point on the HSP in which the score of the regions on either side is equal. The intuition is that this is the point of highest quality, and thus the most likely to be a correct starting point.

Exonerate's alignment heuristics automatically modify a set of pair hidden Markov models depending on the situation to which they are being applied. For example, when trying to align an area between two close HSPs, exonerate may modify the traditional model of alignment to produce a global alignment. For HSPs on the edges of alignments, it may produce a pair-HMM for one sided local alignment, connecting it to the model used for the adjacent alignment region.

The specific models it uses depends on the task at hand. For example, exonerate comes with models for aligning arbitrary genomic sequence as well as models that align sequence that codes for amino acids, to a genomic sequence.

2.6 Parameters and training

Here, we describe techniques for performing parameter training on real biological sequences. We include parametric alignment, which is a technique that avoids the issue of parameter

training by finding alignments under all possible parameters under specific constraints.

2.6.1 Pair-HMM parameter training

Parameters for pair-HMMs are typically trained with an expectation maximization procedure that uses one of two algorithms: Viterbi training [33], or Baum-Welch training [9]. For both algorithms, training is an iterative procedure where we refine the parameters on each iterative step, stopping when the change in probability or score is below some threshold.

For Viterbi training, each iterative step consists of finding the optimal Viterbi path with the current parameters, and computing new parameters from this path. In this case, the stopping criteria depends on the the change in the probability of the Viterbi path. Viterbi training is popular in sequence alignment due to both run time and ease of implementation: one simply finds an alignment and computes statistics from that alignment. For example, see the study by Chiaromonte, Yap, and Miller [42] described in the next section.

Baum-Welch training uses the forward and backward algorithms to compute the expected number of times each transition and emission is used. In this case, the stopping procedure is concerned with $\Pr(x|T, E)$, the probability we produce sequence x given transition matrix T and emission matrix E. Baum-Welch is slightly slower in run time due to having to compute both the forward values as well as the backwards values, but benefits from the sensitivity of prior probabilities and is guaranteed to converge. Both Viterbi and Baum-Welch training take $O(mn)$ time and space for input sequences of length n and m. Baum-Welch training is a special case of the expectation maximization algorithm [33].

2.6.2 Work of Chiaromonte, Yap, and Miller

Many popular aligners for genomic DNA use alignment parameters trained in 2002 by Chiaromonte, Yap, and Miller [25]. They use an approach similar to that used to form the BLOSUM [42] scoring schemes for protein alignment. They point out that highly similar sequences score well even when model parameters describe alignments with much weaker similarities.

Their approach is to filter out alignment regions that have similarity above 70% identity, and use Viterbi training only on the remaining low similarity regions. They train on human and mouse alignments at three different genome locations: the HOXD region, the CFTR region, and the hum16pter region. They evaluate each scoring scheme using simple alignment metrics on ungapped alignments of several genome locations and conclude that

the HOXD scoring scheme is the most sensitive. They do not train gap costs. Harris [40] improves on the procedure by using Viterbi training to train gap parameters.

2.6.3 Work of Arribas-Gil, Metzler, and Plouhinec

Arribas-Gil, Metzler, and Plouhinec [7] present an expectation maximization procedure that samples a path from a pair-HMM on each iteration. A second procedure also samples parameters at each iteration, given a prior distribution over possible parameters. This procedure either accepts the new parameters or keeps the old parameters.

2.6.4 FSA and query specific training

FSA [11] uses Baum-Welch training to train parameters for their alignment pair-HMM. Rather than training on a specific set of training data, and subsequently using those parameters in all alignments for sequences from a given pair of species, FSA trains parameters for each individual pair of input sequences. If the sequences are too short for reliable training, FSA uses a default set of parameters.

2.6.5 Parametric alignment

Pachter and Sturmfels [66] take a different approach to choosing parameters for sequence alignment. They note that the Viterbi algorithm, applied to the standard pair-HMM for alignment, has a small,finite number of possible alignments, despite the large space of possible model parameters. They derive formulas for computing each possible alignment and propose to simply study all possible alignments. The main problem with this approach is that it does not indicate which of the resulting alignments are likely to be correct.

Chapter 3

RDA: alignment by segments

In the first chapter, we described how Viterbi decoding with a particular pair-HMM is equivalent to score-based alignment. In terms of the underlying model of evolution, the Viterbi path represents a single evolutionary history maximum probability. However, for some input sequences there may be many possible histories with high probability. In these cases, it is more useful to identify pairs of subsequences that are highly likely to be related by the model in *any* way, rather than identifying only one of the possible alignments.

In this chapter we describe a new method for decoding the traditional pair-HMM for alignment that addresses the above scenario. Rather than producing a standard alignment, our method produces an alignment of segments where each segment captures a set of alternate evolutionary histories with similar likelihood.

3.1 Right-Down alignments

We now describe our new technique for producing alignments with the standard alignment pair-HMM. The underlying problem we address is that a single alignment gives only one possible explanation (in the case of the optimal alignment, the explanation of highest probability) for the evolution of two sequences. In practice, many distinct explanations may have essentially the same semantic meaning, and when considered together, these may give rise to a better mathematical explanation of the homology between two sequences.

For example, one may feel that the optimal alignment in Figure 3.1 contains a surprising number of mismatches. Even more surprising, we may add an equal number of mismatched A's to the top sequence and C's to the bottom sequence and the optimal alignment for

typical parameters *never* contains a gap. The biological interpretation of this is that each of the mismatched A's and C's represents point mutations. How is it that the optimal alignment represents such an improbable history?

```
AAAAAACCC
AAACCCCCC
```

Figure 3.1: Optimal alignment of two sequences according to global alignment, using typical parameters.

For this example, the problem comes from the existence of many sub-optimal alignments representing highly similar alternate histories. For example, Figure 3.2 gives three alternate alignments of the two sequences from Figure 3.1. While each alignment has a lower score than the alignment with no gaps, there are many more alignments involving gaps than the single alignment with no gaps.

```
AAAAAACCC---        AAAAAACCC---        AAAAAA---CCC
AAA---CCCCCC        ---AAACCCCCC        ---AAACCCCCC
```

Figure 3.2: Three possible alignments of two sequences. Each alignment represents an equally likely evolutionary history, according to the standard model of alignment.

Now consider a different question: do the *group* of A's in the first sequence correspond to the group of A's in the second sequence, or do some A's and C's share a common ancestor? To answer this, we need a new way to decode the alignment pair-HMM. Traditional Viterbi decoding finds only the single most likely history, ignoring all other possibilities. Posterior methods consider all histories, but ignore significant structural properties of alignments such as gap lengths.

We suggest the use of segmentation of sequences, rather than alignment of sequences, as a way of joining together alternative histories of sequences, when the story that they tell about the sequences is similar in its meaning. This approach gives rise to a new algorithm for exploring sequence homology, which we call the right-down aligner, or RDA. Our approach is compatible with a dynamic-programming algorithm which has $O(n^2m^2)$ runtime on sequences of lengths n and m. We also present a heuristic $O(k^2nm)$ algorithm for a parameter k, which has a quite practical runtime. Our heuristic algorithm produces similar segmentations to the exact algorithm.

RDA is particularly appropriate for repeat-rich DNA sequences, as there it becomes unreasonable to assume that a single history is useful in the presence of numerous explanations for the evolution of repeated bases or short sequences.

3.1.1 Segmentation

To encapsulate our idea of segmentation, we wish to join several paths through the HMM, by saying that they represent the same semantic meaning: they give us a blocking of the sequence into segments inside which we are less concerned about the details of the homology.

A *segment* consists of a sequence u of length n and sequence v of length m where either u or v is distinguished. We call a segment with v distinguished a *right segment* and require that $m < n$. A right segment represent all the ways to align u to v by inserting gap characters only in v. Similarly, we call a segment with u distinguished a *down segment* and require $n < m$. In this case, we may insert characters only into u. We say a segment is *correct* if one of these alignments represents the true history. A *segmentation* of sequences x and y is a sequence of segments with three properties.

1. In each segment, the distinguished sequence is a subsequence of x and the other a subsequence of y.
2. Every base of x and y is contained in exactly one segment.
3. Right and down segments always alternate.

This is, of course, not the only way to do such segmentation, but has the advantage that we can use dynamic programming for it, once we make some small modifications to the standard alignment pair-HMM.

To describe segments, we colour the states of the pair-HMM with black and white such that black sub-paths correspond to alignments in down segments and white sub-paths correspond to right segments. Since state M can only be of one colour, we modify the traditional HMM as shown in Figure 3.3. Our new HMM contains two identical match states, one for each segment type. Additionally, we add silent states, states that emit no symbols, to allow paths to transition from one segment type to another in a precise way. When adding these states we take care to avoid creating cycles among silent states as this prevents accurate decoding. We also add an additional silent starting state, distinguished by a double circle. We transition from the start state by first choosing a segment type with equal probability. Then, we pick a coloured emission state within the chosen segment using the stationary distribution conditioned on having already chosen a segment type. Since the hidden states T_R and T_D transition immediately to the gap states, we add the probability of starting in T_R or T_D to the respective gap state. To mimic the traditional pair-HMM, we allow our new pair-HMM to stop only in the match and gap states. Now, each alignment

corresponds to a single sequence of colours, except that the first segment might be white or black. However, if we identify each base as white or black, these correspond to more than one alignment. In fact, they correspond to a segmentation. The optimal segmentation corresponds to the most probable labelling of the sequence pair [16].

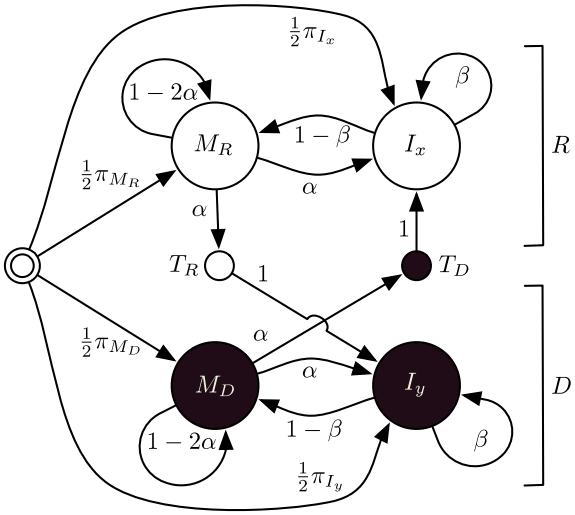

Figure 3.3: Pair-HMM for segmenting alignment. As before, α corresponds to the gap open cost and β to the gap extension probability. The small states are silent and the double circled state is the start state. White states represent right segments (R) which only allow gaps in sequence y. Black states represent down segments (D) which only allow gaps in sequence x. States T_R and T_D are silent transition states between the two segment types.

3.1.2 Computing an optimal segmentation

We can compute the optimal segmentation of the two sequences by dynamic programming: we note that, if the optimal segmentation of x and y ends with a "down" block, then the period up to and including the "right" block before that "down" block must be optimal. Here, we give the details using the modified HMM from the previous section, which give rise to an $O(n^2m^2)$ algorithm for computing optimal segmentation and an $O(k^2mn)$ heuristic algorithm (for a small parameter k). There is a connection to the maximum probability annotation algorithm presented by Brejova, Brown, and Vinar [16], which also looks to divide sequences into intervals corresponding to states of the same colours, subject to some constraints on the topology of the HMM.

$$
\begin{aligned}
M_R(i,j|k,\ell) &= e_{x_i,y_j,M_R}\left[t_{M_R,M_R}M_R(i-1,j-1|k,\ell) + t_{M_R,I_x}I_x(i-1,j-1|k,\ell)\right] \\
I_x(i,j|k,\ell) &= e_{x_i,I_x}\left[t_{I_x,M_R}M_R(i-1,j|k,\ell) + t_{I_x,I_x}I_x(i-1,j|k,\ell)\right] \\
T_R(i,j|k,\ell) &= M_R(i,j|k,\ell)
\end{aligned}
$$

$$
\begin{aligned}
M_D(i,j|k,\ell) &= e_{x_i,y_j,M_D}\left[t_{M_D,M_D}M_D(i-1,j-1|k,\ell) + t_{M_R,I_x}I_x(i-1,j-1|k,\ell)\right] \\
I_y(i,j|k,\ell) &= e_{y_j,I_y}\left[t_{I_y,M_D}M_D(i,j-1|k,\ell) + t_{I_y,I_y}I_y(i,j-1|k,\ell)\right] \\
T_D(i,j|k,\ell) &= M_D(i,j|k,\ell)
\end{aligned}
$$

Without look-back:

$$
\begin{aligned}
R(k,\ell) &= \max_{1\le i\le k, 1\le j\le \ell} T_R(i,j|k,\ell) + D(i,j) \\
D(k,\ell) &= \max_{1\le i\le k, 1\le j\le \ell} T_D(i,j|k,\ell) + R(i,j)
\end{aligned}
$$

With look-back:

$$
\begin{aligned}
R(k,\ell) &= \max_{\substack{k-k'\le i\le k, \ell-k'\le j\le \ell \\ k-k'-z\le i\le k-k', \ell-f(k-k'-i)\le j\le \ell-k'}} T_R(i,j|k,\ell) + D(i,j) \\
D(k,\ell) &= \max_{\substack{k-k'\le i\le k, \ell-k'\le j\le \ell \\ k-f(\ell-k'-j)\le i\le k-k', \ell-k'-z\le j\le \ell-k'}} T_D(i,j|k,\ell) + R(i,j) \\
f(x') &= 2 + (k'-2)(z-x')^{y'}/z^{y'} \\
z &= (y'+1)k'^2/(k'+2y')
\end{aligned}
$$

Figure 3.4: Dynamic programming to compute the most probable labelling. The look-back is defined by a k' by k' box with a length z tail defined by $f(x')$.

Figure 3.4 shows part of the recurrences we must solve to compute the most probable labelling. Here, $M_R(i, j|k, \ell)$ is the probability that we end by matching x_i and y_j having output $x_{k\ldots i-1}$ and $y_{\ell\ldots j-1}$ while staying in the white states, entering at x_k, y_ℓ. Similarly, $I_x(i, j|k, \ell)$ is the probability that we end by outputting only x_i having output $x_{k\ldots i-1}$ and $y_{\ell\ldots j}$. The probability of ending in state T_R after outputting $x_{k\ldots i}$ and $y_{\ell\ldots j}$ is $T_R(i, j|k, \ell)$.

The optimal segmentation ending in T_R for x from 1 to k and y from 1 to ℓ is $R(k, \ell)$. The probability of the optimal segmentation ending in state T_D is $D(k, \ell)$, defined using a similar recurrence for the black states. We compute the table $T_R(i, j|k, \ell)$ for $i \leq k$ and $j \leq \ell$ in a single pass giving a run time of $O(m^2n^2)$.

Since this is impractical for long sequences, we create a heuristic algorithm where, at each cell of the matrices R and D, we limit the look-back to a k by k box: this gives a maximum length to any segment. In practice, however, some segments are very long in one sequence and quite short in the other, and adding this possibility to the algorithm is not difficult, by adding to the box a length z tail allowing for long gaps. We define the tail with the function $f(x) = 2 + (k-2)(z-x)^y/z^y$ where $z = (y+1)k^2/(k+2y)$. Here, y controls the steepness of the curve, $f(0) = k$, $f(z) = 2$, and $\int_0^z f(x)dx = k^2$. In our implementation we set $y = 7$. Figure 3.5 shows the shape of the tail and look-back area for $k = 40$; again, any segment must fall within this shape, in terms of the number of positions in the segment for each of the two sequences. The overall size of the look-back area is $\Theta(k^2)$ giving a heuristic algorithm with run time $O(k^2nm)$.

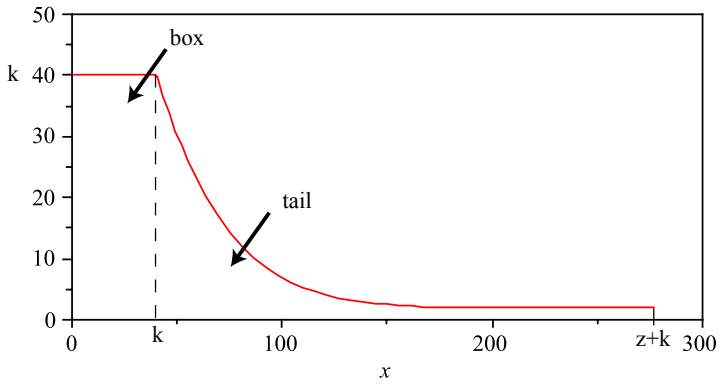

Figure 3.5: Shape of the look-back area with $k = 40$ and $y = 7$. The look-back area consists of a k by k box with a tail defined by the function $f(x) = 2 + (k-2)(z-x)^y/z^y$ where $z = (y+1)k^2/(k+2y)$. We shift the tail to the right by k.

3.2 Experiments with RDA

We demonstrate the success of our segmentation algorithm, RDA, the Right-Down Aligner, with experiments on synthetic data. We include comparisons to both the Needleman-Wunsch algorithm as well as the forward algorithm for pair-HMMs when applicable. First, we describe our hypotheses. Then, we describe experiments that validate our hypotheses.

3.2.1 Hypotheses

We have four hypotheses about RDA and the forward and Viterbi algorithms for the standard alignment pair-HMM. First, we expect that RDA is a better discriminator of related and unrelated sequences than Viterbi, and that the forward algorithm is better than RDA. This is because both the forward algorithm and RDA consider many alignments while Viterbi considers only the optimal alignment.

Second, we expect that the difference in discrimination accuracy between the forward algorithm, RDA, and Viterbi increases as the baseline nucleotide distribution becomes biased toward producing only two or fewer symbols. As the diversity of nucleotides decreases, the number of alignments with similar score increases and more repeats will arise by chance. The likelihood that the true alignment is the optimal alignment decreases as the number of high scoring alternate alignments increases.

Third, we hypothesize that the accuracy improvements of RDA and the forward algorithms increase with alignment length. As sequences get longer, the number of possible segmentations and alignments increases. Since the improvements of RDA and the forward algorithm come from exploring many alignments, we believe the accuracy should increase as the number of possible alignments increases.

Fourth, we expect that the segments produced by RDA represent a better overall alignment structure than that given by Viterbi. Our motivating example is one case where alignment of segments produces a better structure than Viterbi alignment. We expect this to generalize to more complicated cases.

Finally, we expect that on real data, RDA often gives substantially different results than Viterbi when given correct parameters. We expect this to be especially true in repeat rich regions due to the large number of different alignments with equivalent semantic meaning.

3.2.2 Experiments and results

To test our hypothesis we use two experiments. The first experiment uses alignment score to detect homologous pairs of sequences. The second experiment explores the accuracy of alignment structure. Both experiments use synthetic data.

Synthetic data

We generate our synthetic data using the pair Hidden Markov Model (pair-HMM) in Figure 1.3. In all tests, we vary the number of expected substitutions per site with a fixed gap open probability of 0.0812 and mean gap length of 3.6 bp. The gap parameters correspond to a gap open cost of -3.6 bits and gap extend cost of -0.5 bits. We choose these parameters to produce alignments that are qualitatively similar to alignments of intergenic sequence with evolutionary distance slightly more than that of human and mouse.

Detecting homology

To test the first four hypothesis about RDA and the forward algorithm, we use a sequence pair rank test. We start with 100 sequence pairs, $(A_1, B_1), (A_2, B_2), \ldots, (A_{100}, B_{100})$, taken from length n alignments generated by our pair-HMM. In this situation, each A_i and B_i were generated together, and thus are homologous. All other pairs are independently generated. As the amount of mutation increases, it becomes increasingly likely to see a high scoring alignment for an independently generated pair. Thus, the test increases in difficulty.

We analyze the homology between each A_i with B_1 through B_{100} and rank the results by score. If the homology between A_i and B_i ranks first, we count a success. Otherwise, we count a failure. Each test has a maximum possible score of 100. When testing RDA, we use our heuristic version with $k = 40$. Table 3.1 shows test results for various program and parameter combinations.

Our experiments verify all three of our hypotheses about RDA and the forward algorithm. In all cases the forward algorithm is a better discriminator than RDA and RDA is better than Viterbi. The difference between the test results increases with increasing GC content and sequence length.

Expected substitutions per site	50% GC $n = 200$			60% GC $n = 200$			60% GC $n = 400$		
	FA	RDA	VT	FA	RDA	VT	FA	RDA	VT
0.2	100	100	100	100	100	100	100	100	100
0.3	94	94	95	99	99	98	100	100	100
0.4	94	88	86	90	90	85	98	98	97
0.5	80	75	69	77	72	69	97	96	93
0.6	66	61	52	64	57	42	80	72	70
0.7	50	34	32	48	42	28	59	48	38
0.8	33	23	18	23	20	9	47	35	25
0.9	26	17	8	23	10	7	26	23	18
1.0	14	7	5	17	13	8	22	14	7
1.1	6	4	3	3	5	5	8	5	5
1.2	7	5	1	4	5	0	9	0	3

Table 3.1: Results of pair rank test for the right-down alignment (RDA), forward algorithm (FA) and Viterbi algorithm (VT). The approximate sequence length for each set of tests is denoted by n. We use the heuristic version of the RDA algorithm with $k = 40$. The entries in the table are the number of successes out of 100 trials.

Alignment structure

It is difficult to directly compare the segments produced by RDA with alignments by Viterbi. To get a lower bound on the accuracy of a segmentation produced by RDA, we compare the most probable alignment allowed by the RDA segmentation to the Viterbi alignment. Note that in this case a segment can still be correct even if the most probable alignment within the segmentation is wrong. Thus, this comparison only provides a lower bound on the possible structural improvements of RDA. We use the data generated in the first experiment but only align the homologous sequences A_i and B_i. To obtain the most probable alignment for a given segmentation, we run NW within each segment. We measure alignment accuracy as the percentage of completely correct columns found in the alignment.

d	50% GC $n = 200$					60% GC $n = 200$				
	RDA		VT			RDA		VT		
	μ	s	μ	s	p-value	μ	s	μ	s	p-value
0.2	67.8	8.7	67.6	9.0	7.4×10^{-1}	66.8	8.8	66.5	9.6	6.1×10^{-1}
0.3	59.8	9.1	59.7	11.8	9.6×10^{-1}	57.4	10.6	57.6	10.7	7.5×10^{-1}
0.4	48.5	11.1	49.1	12.4	4.0×10^{-1}	50.4	12.3	48.9	13.3	5.8×10^{-2}
0.5	44.8	11.3	43.5	12.0	2.4×10^{-1}	41.3	10.3	**43.1**	10.9	$\mathbf{3.8 \times 10^{-2}}$
0.6	38.9	10.9	37.0	12.5	5.7×10^{-2}	35.0	10.0	34.0	11.3	3.0×10^{-1}
0.7	29.3	11.9	28.8	13.7	6.4×10^{-1}	**29.9**	10.4	27.8	11.0	$\mathbf{7.2 \times 10^{-3}}$
0.8	25.5	10.2	24.3	10.7	1.9×10^{-1}	**25.2**	10.1	23.5	12.5	$\mathbf{3.6 \times 10^{-2}}$
0.9	21.1	9.5	19.5	10.4	1.1×10^{-1}	**20.7**	9.2	17.7	11.8	$\mathbf{1.8 \times 10^{-3}}$
1.0	18.2	7.8	17.5	10.0	5.2×10^{-1}	**19.7**	10.0	16.7	9.7	$\mathbf{2.5 \times 10^{-3}}$
1.1	**17.6**	8.6	15.6	9.8	$\mathbf{2.7 \times 10^{-2}}$	15.9	7.5	14.4	10.0	1.4×10^{-1}
1.2	**16.1**	8.0	13.9	8.3	$\mathbf{1.1 \times 10^{-2}}$	**15.7**	8.1	12.6	8.5	$\mathbf{1.7 \times 10^{-3}}$

Table 3.2: Percent of columns correct for various rates of mutation d, measured in expected substitutions per site. We compare the right-down alignment followed by Viterbi within the segments (RDA), versus straight Viterbi (VT). The approximate sequence length for each set of tests is denoted by n. Each data set contains 100 alignments and we give the mean μ, sample standard deviation s, and p-value for each. We calculate the p-value with a paired t-test using the false discovery rate correction. For RDA, we use the heuristic version with $k = 40$.

The experiment verifies our fourth hypothesis. Using a 5% confidence level, from Table 3.2 we see that for sequences with 50% GC content, there is no clear difference in

d	60% GC $n = 400$				
	RDA		VT		
	μ	s	μ	s	p-value
0.2	67.0	6.3	67.4	6.4	1.8×10^{-1}
0.3	57.6	7.8	58.4	7.4	1.6×10^{-1}
0.4	47.9	8.1	48.4	8.8	4.1×10^{-1}
0.5	41.9	7.5	40.6	8.9	6.6×10^{-2}
0.6	**31.3**	8.8	29.6	8.7	$\mathbf{1.6 \times 10^{-2}}$
0.7	**26.0**	8.4	24.2	10.2	$\mathbf{1.5 \times 10^{-2}}$
0.8	**23.6**	7.8	21.0	8.7	$\mathbf{1.4 \times 10^{-4}}$
0.9	**19.5**	7.1	16.2	8.0	$\mathbf{2.6 \times 10^{-5}}$
1.0	**16.0**	7.1	13.3	7.2	$\mathbf{2.7 \times 10^{-4}}$
1.1	**14.8**	5.7	12.1	7.1	$\mathbf{4.6 \times 10^{-4}}$
1.2	**13.1**	5.7	9.8	5.9	$\mathbf{1.7 \times 10^{-5}}$

Table 3.3: Percent of columns correct for various rates of mutation d, measured in expected substitutions per site. We compare the right-down alignment followed by Viterbi within the segments (RDA), versus straight Viterbi (VT). The approximate sequence length for each set of tests is denoted by n. Each data set contains 100 alignments and we give the mean μ, sample standard deviation s, and p-value for each. We calculate the p-value with a paired t-test using the false discovery rate correction. For RDA, we use the heuristic version with $k = 40$.

accuracy between RDA followed by Viterbi and straight Viterbi until we reach 1.1 expected substitutions per site. However, for sequences with 60% GC content and 0.7 or more expected substitutions per site, we observe that RDA with Viterbi within each segment produces more accurate alignments than Viterbi alone. Table 3.3 shows that the significance of our results increases with longer sequences. At best, we see an absolute improvement in accuracy of 3.3%. If we look at improvement relative to the Viterbi accuracy, the best RDA plus Viterbi score is 33.7% better than the Viterbi score. This occurs at 1.2 substitutions per site with 60% GC content and $n = 400$, which is highly distant sequence.

Aligning real data

To test our final hypothesis, we align short human and mouse sequences from the first ENCODE target [28]. We identify a homologous repeat rich region using a long global alignment and align it with both the exact RDA algorithm and Needleman-Wunsch. We use the same set of parameters, inferred from the entire first ENCODE target, for both the Needleman-Wunsch algorithm and the RDA.

A
```
CAAA------AGCAACTATTAATATTTTTAGCTCAGTG-GTCAAATATGCCTCTCTCATGTGTGCAC
AAAAATCTTGAGTAACTTATTATCTT----------TGAGGCAAACACACCTCC-------------
```

B
```
CAAA------AGCAACTATTAATATTTTTAGCTCAGTGGTCAAATATGCCTCTCTCATGTGTGCACA
AAAAATCTTGAGTAAC--TTATTATCTTTG-------AGGCAAACACACCTC-----------CACA
```

Figure 3.6: An alignment of human and mouse sequence from human chromosome 7 with (A) RDA and (B) Viterbi.

Figure 3.6 gives a subset of the resulting alignments. Our hypothesis is correct, RDA plus Needleman-Wunsch gives substantially different results on real data than Needleman-Wunsch alone. Unfortunately, we cannot identify a specific use case for these differences aside from statistical studies. Regardless, we feel RDA provides an interesting alternative to traditional alignment, specifically for repeat-rich sequences.

3.3 Summary

We present a new algorithm, RDA, that aligns sequences in segments where we consider all possible evolutionary histories in each segment. This algorithm gives improved homology detection over Viterbi alignments, but not quite as good as with the forward algorithm. It

also gives improved column level alignment accuracy due to an improved overall alignment structure. Finally, on real sequence, RDA with Needleman-Wunsch gives substantially different alignments than Needleman-Wunsch alone. We expect differences to be greater on repeat rich sequences.

On the other hand, it is unclear what practical benefits our alignment of segments give beyond improved statistical studies of evolution. The main negative aspect of RDA is a run time of $O(n^2m^2)$ for sequences of length n and m, compared to $O(nm)$ required for traditional alignment. Our heuristic version of RDA gives an improved runtime of $O(k^2nm)$ where larger k gives more accurate results. However, in practice even the heuristic version of the algorithm is significantly slower than traditional alignment. Thus, we conclude that RDA is an interesting algorithm, but it is not clear if it has practical applications.

Chapter 4

Sensitive genomic alignment: methods

In Chapter 3 we introduced a new technique for aligning sequences in segments. We now focus on three problems in genomic alignment: aligning genomic sequences with varying rates of mutation, anchoring long homologous genomic sequences, and discovering local segments of homology in pairs of genomic sequences. We address these with two new alignment programs: CAPE, a global aligner for very weakly homologous sequences, and FEAST, a local aligner with improved homology detection. Both these algorithms perform a final alignment with a more descriptive model of evolution that describes alignments comprised of distinct segments, each with its own statistical properties.

First, we describe the anchoring algorithm in CAPE, our new global alignment program. Then, we describe our model of evolution followed by our new local alignment algorithm implemented in FEAST. We test all these ideas with experiments on both synthetic and real DNA in the next chapter.

4.1 CAPE: Anchoring weakly homologous sequence pairs

The goal for CAPE is to answer two questions. First, what is the theoretical limit on the amount of mutation, beyond which we can no longer recover a good global alignment? And second, is it possible to perform accurate anchoring of long homologous sequences when we do not know the correct model parameters?

CAPE's anchoring strategy follows a recursive GLASS-like approach. Each recursive anchoring phase has three steps: finding fragments which may serve as anchors, scoring fragments, and choosing anchors. We find fragments with good random spaced seeds [50, 55, 47, 14] choosing the seed weight to fix the expected number of hits. We introduce a new algorithm to score fragments and a mixed global/greedy approach to choose anchors.

4.1.1 Fixing the number of seed hits

Given a region $(x_{i...j}, y_{k...l})$ we wish to anchor, we choose a seed weight w and length u to fix the number of expected random hits E_r. Let the background distribution over the alphabet $\Sigma = \{A, T, C, G\}$ be represented by vector q of length four. The probability of a match between randomly selected x_i and y_j is $\Pr[x_i = y_j] = q \cdot q$. The area of region $(x_{i...j}, y_{k...l})$ is $A = (j - i + 1)/(k - \ell + 1)$. To obtain E_r random hits, we choose

$$w = \log(E_r/A)/\log(\Pr[x_i = y_j]). \tag{4.1}$$

To ensure we choose a spaced seed with a good amount of *don't-care* positions, we set the length to $u = 1.4w$. We then choose a good random spaced seed with weight w and length u [51]. To prevent unsuitable values of w and u in small regions, we enforce $w \geq 3$ and $u \geq 5$.

4.1.2 Scoring fragments using local segmentation

Our goal in scoring fragments is to discriminate between *decoy fragments*, seed hits we find in unrelated sequence, and fragments in regions of true homology. We base our algorithm on the pair Hidden Markov Model (pair-HMM) for local alignment in Figure 4.1. The white states represent homologous sequences, and the black states represent random unrelated sequences. Given this model, sequence x with length n, and sequence y with length m, applying Viterbi decoding to the subsequences $x_{i...n}$ and $y_{j...m}$ represents a local alignment extension starting from symbols x_i and y_j. In terms of evolution, this path is the most likely evolutionary history for these two subsequences. The score is a direct measure of the probability of this history versus the probability that the two subsequences are taken from a random distribution. Thus the score of the optimal path is often used as a measure of homology.

However, as we have repeatedly noted, the optimal path only represents one possible evolutionary history among many. As a result, the optimal path score is limited in its

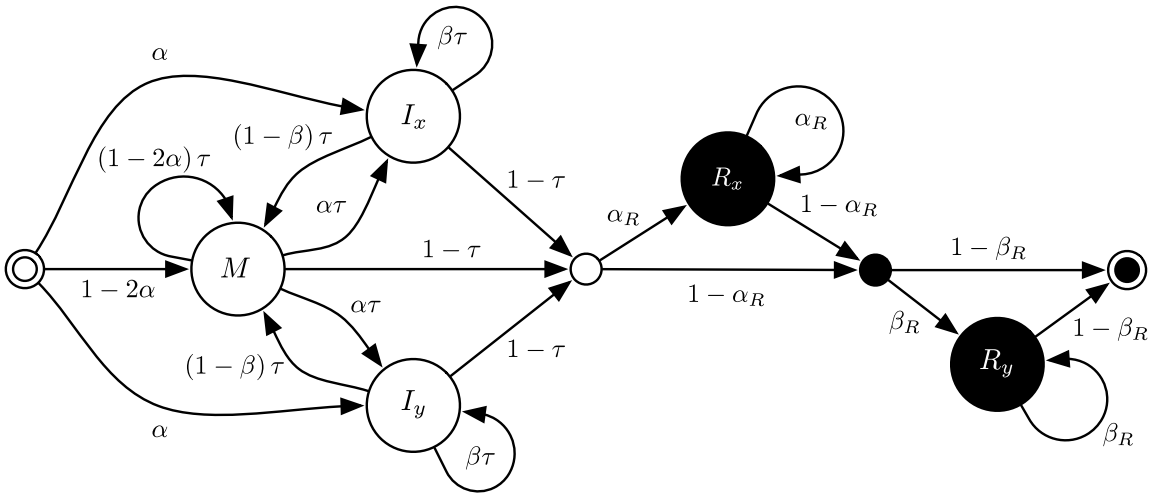

Figure 4.1: Pair-HMM for local segmentation. White states represent homologous sequence and the black states represent unrelated sequence. Unlabelled states are silent.

representation of the true homology between two sequences. To increase sensitivity, we wish to evaluate all possible histories. We do this by finding the most probable segmentation of the sequences into a white, homologous segment, and a black, unrelated segment.

The Markov model imposes one additional complication: a distribution on the lengths of the segments. We have no reason to prefer a segmentation of one length over another and thus set $\tau = \alpha_R = \beta_R = 1/2$. Because these components amount to a fixed adjustment to the final computed value, in practice, we simply omitting the contributions of τ, α_R, and β_R to our computation.

Due to the simplicity of the black states, we can compute the optimal segmentation in $O(mn)$ time, given input sequences of length m and n. We start by computing the forward probabilities for the white states of the pair-HMM using the standard dynamic programming implementation [33]. This returns three m by n matrices: $M(i,j)$ is the probability of all paths ending in state M such that $x_i \diamond y_i$, $I_x(i,j)$ contains the probability of all paths ending in state I_x emitting only x_i, and $I_y(i,j)$ containing the probability of all paths ending in state I_y emitting only y_j. The probability of the segmentation $L(i,j)$, where the break between the two segments is at x_i and y_j is

$$L(i,j) = S(i,j)R(i,j) \tag{4.2}$$

where

$$S(i,j) = M(i,j) + I_x(i,j) + I_y(i,j) \tag{4.3}$$

and

$$R(i,j) = R_x(i)R_y(j). \tag{4.4}$$

Here, $S(i,j)$ is the probability that we end in the white switch state after having emitted $x_{1\ldots i}$ and $y_{1\ldots j}$, and $R(i,j)$ is the probability that, starting from the white switch state, we emit $x_{i+1\ldots m}$ and $y_{j+1\ldots n}$ using only black states. We can compute $R(i,j)$ using the backward algorithm for pair-HMMs in time $O(mn)$. In practice, we express $R(i,j)$ in terms of $R_x(i)$ and $R_y(j)$, where $R_x(i)$ is the probability of emitting $x_{i+1\ldots m}$ using only state R_x and $R_y(j)$ is the probability of emitting $y_{j+1\ldots n}$ using only state R_y. We can compute these values in time $O(m+n)$. Finally, we iterate over all possible segmentations in time $O(mn)$, keeping the segmentation with highest probability.

For each fragment, we perform a single local segmentation of the input, starting from the midpoint of the fragment and extending 200 bp in the forward direction in both sequences. The size of the extension is user adjustable. A high-scoring fragment is likely to be used as an anchor in our global alignment.

4.1.3 Segmentation with multiple parameters

One of the primary goals of CAPE is to anchor sequences in which the local conservation level is unknown. For strongly or moderately-conserved homologies, it is easy enough to anchor long sequences correctly, even with incorrect parameters. However, given long sequences containing very weak homology, it is difficult to anchor without knowing the correct parameters beforehand. At the same time, without correct anchors, it is impossible to train correct parameters. Worse yet, if parameters of the weak homologies vary, even if we have a set of correct parameters describing each level of homology, we still do not know which particular parameter set to use for a given fragment. We solve this problem by performing segmentations with several sets of parameters for each anchor and choosing the highest scoring among all parameter sets to discriminate between real and random sequences. This procedure answers the following question: which model of homologous sequences fit the data best? If all the scores are negative, then the model of random data fits the data best.

We define a parameter set to be a triple (d, o, e) where d is the number of substitutions per site, o is the gap open probability in bits, and e is the gap extension probability in bits. To use a parameter set (d, o, e) on a particular fragment, we expand d into emission

probabilities for state M in Figure 4.1. Our implementation uses the HKY [41] model of substitutions to do this, although we assume equal numbers of transversions and transitions in our experiments. We use an estimate of the background nucleotide composition measured from the entire input.

The specific set of parameters that we use in the anchoring procedure is greedily chosen based on simulations of anchor discovery. We describe these simulations in detail in the next chapter. We also allow the user to provide a custom set of parameters for a given pair of input sequences.

4.1.4 Handling repeats in real sequence

Real biological sequences contain repeats that interfere with anchor finding. To address this problem we allow input sequences to be pre-masked and use this information during anchoring. We exclude any fragments containing a base annotated as a repeat. Similarly, our fragment scoring algorithm terminates if it encounters a repeat and returns the result up until that point.

4.1.5 Complete anchoring strategy

In describing our complete anchoring strategy, we work with two types of fragments: fragments defined by seed hits, and large fragments that describe subsequence pairs from input sequences x and y that we wish to anchor and align. We use the definition for the area of a fragment introduced in section 2.3.1 of Chapter 2.

Our anchoring strategy, which depends on our memory and time tolerance, requires two additional parameters. Let S be the area of the maximum size fragment we can align without anchors and let t be a score threshold. In our implementation we choose S to be 4×10^6, the area of two subsequences of size 2000 each, and t to be 70. We arrived at these values through trial and error and in part due to machine limitations. Additionally, we define P to be the number of parameter sets we explore and R to be a set of large fragments over sequences x and y; that is, each element of R is a pair consisting of an interval from x and an interval from y. We start with a single large fragment in R consisting of all of x and y and apply the following steps until finished.

1. Choose a fragment a in R with area greater than S. If there are none, return the set of fragments and finish; we are ready for the next phase.

2. Find k small fragments in a by seeding, score them, and choose one or more anchors.

   (a) Choose seed parameters to obtain k expected decoy fragments.

   (b) Score all discovered seed fragments with our local segmentation algorithm for each parameter set in P.

   (c) For each seed fragment, assign the highest local segmentation score from the set we obtain by performing segmentations with all parameter sets.

   (d) Filter out all seed fragments with score below t.

   (e) If any fragments remain, return the maximum weight collinear chain of remaining fragments as anchors. Otherwise, return the highest scoring seed fragment from all seed fragments as an anchor.

3. Replace fragment a in R with two fragments. The boundary of these two fragments is defined by the midpoint of the anchor fragment.

On termination of this procedure we have a set of fragments that cover sequences x and y. Each fragment has area less than or equal to S and the boundaries of each fragment represent anchor points. We retain the scores for each original anchor fragment used to form R.. The number of decoy fragments in the above procedure, k, defaults to 500, but is user adjustable.

We define two types of anchors based on Step 2(e) in the above procedure. We call anchors chosen by the maximum weight collinear chain *regular anchors*, and those chosen because they have the highest score, *greedy anchors.* We only choose greedy anchors when all seed fragment scores are below t. Thus, greedy anchors have a higher likelihood of being incorrect and we want to minimize their number in our final set of anchors.

To do this, we make use of another property of our procedure. During anchoring, we may often choose several greedy anchors before choosing one that sufficiently divides a large fragment into sub-fragments small enough to align. Some of these greedy fragments may not be needed. We use a final greedy strategy to remove any unnecessary anchors. The procedure iterates over the final anchor set in sequential order. We remove an anchor if the fragment defined by the flanking anchors has area less than or equal to S. In this case, we replace the two fragments in R that surround the removed anchor by a single fragment that is the union of the original two.

After we have anchored our input data, we perform banded alignment using an alignment based on a new pair-HMM model of evolution. We describe our new model in the next section.

4.2 A more descriptive model of evolution

Our model of evolution is a pair Hidden Markov Model (pair-HMM) producing sequences x and y. Figure 4.2 shows our entire model. White states describe sequences related by evolution, and black states describe unrelated sequence. The white states consist of k sub-models, S_1 through S_k, each of which generate two sequences with a consistent level of divergence. We allow switches from one sub-model to another through a silent switch state.

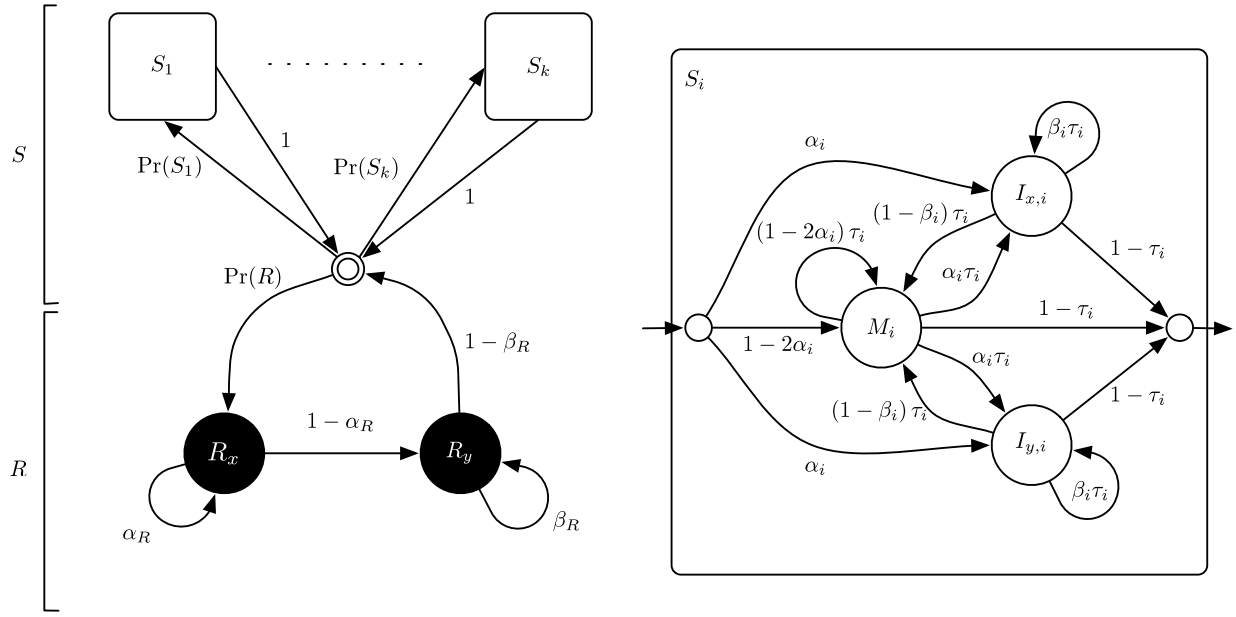

Figure 4.2: Pair-HMM for global alignment with multiple scoring schemes. The left gives an overview of the entire pair-HMM. The right box gives the structure of each sub-model, S_i. The white states (S) represent sequence related by a model of evolution that switches between sub-models S_1 through S_k. Black states (R) represent unrelated sequence. The double circled white state is the start and stop state. To prevent a cycle of silent states, CAPE uses a simplified model of random sequence that forces at least one symbol to be emitted in each sequence.

We base each sub-model S_i on the standard pair-HMM for alignment, with additional silent start and end states as well as a distribution on the sub-alignment length described by the parameter τ_i. States M_i emit pairs of symbols and correspond to match and mismatch scores in score-based alignment. Parameter α_i corresponds to the gap open penalty (GOP)

and β_i to the gap extension penalty (GEP). States $I_{x,i}$ and $I_{y,i}$ emit symbols in only sequence x or y respectively. These correspond to gaps in an alignment. In terms of score-based alignment, each sub-model corresponds to a unique alignment scoring scheme. The probabilities $\Pr(S_1)$ through $\Pr(S_k)$ correspond to a jump cost for changing scoring schemes [57].

In a seeded alignment framework, we assume the start position is in the alignment and perform an extension in two directions. We model this by assigning the white model switch state to be the start state, and including a model of unrelated sequences represented by black states. The transition with probability $\Pr(R)$ represents the end of an extension in one direction.

For global alignments, we use Viterbi decoding [33] to produce an alignment. The score of an alignment is the logarithm of the Viterbi path probability minus the logarithm of the probability according to a simple background model that uniformly samples from the symbol distribution as measured from the input. Unlike traditional alignments, our alignments also produce an annotation indicating which sub-model produced each column of the alignment. Viterbi decoding takes time $O(knm)$ where n and m are the lengths of the input sequences x and y, and k is the number of sub-models. While this is the same time complexity as existing local aligners for a constant k, each additional sub-model incurs a substantial constant-factor increase to the run time.

We use this model for the final alignment stage of CAPE. The number of sub-models and their parameters is set to be the same as the parameter sets we use to score fragments during anchoring. Optionally, we allow the alignment to move from the final random state back to the starting state. This feature is user configured and is intended to eliminate poorly scoring regions from the final alignment. We do not use this feature in our experiments.

In the next section, we introduce a new local aligner that uses our new pair-HMM throughout.

4.3 FEAST: local alignment and a new extension algorithm

Local alignment has two distinct goals: find subsequence pairs that represent homology according to a model, and to find alignments of those subsequence pairs. Traditionally, local aligners use a seed and extend approach to address both goals simultaneously. The seeding step uses a fast heuristic to find points likely to be in a local homology. The

extension step uses an algorithm based on Viterbi decoding to expand each seed hit into a local alignment. The score of each local alignment is used to determine if the seed hit was truly in a local homology, or if it was simply a false positive. Each local alignment also implies an answer to the first question: the subsequences represented in the alignment represent a homologous pair.

In our work we answer each question separately. First, we find local homologies according to a model, but do not find alignments of those homologies. In a second step we find alignments of each local homology. In both steps, we use a modified version of the new pair-HMM for alignment from the previous section.

4.3.1 An extension algorithm for homologies

Extensions that use the Viterbi algorithm find a single path of maximum probability. In terms of evolution, a path through our pair-HMM represents a set of constraints over the possible evolutionary histories. In this context, the Viterbi algorithm finds the single most likely evolutionary history of the two sequences. However, in the extension phase, we are interested in the probability that two sequences are related by homology in *any* way. That is, we wish to include all paths, and thus all possible evolutionary histories, in our computation. In our pair-HMM, this translates into finding the optimal *labelling* of the input. That is, we wish to colour the input with white and black, where black represents unrelated sequence and white represents related sequence. In a local alignment context, once we enter the model of random sequence we never leave. This property allows us to use a model of random sequence that includes the case where we emit no random symbols, without introducing a cycle of silent states. Figure 4.3 shows our pair-HMM for local alignment.

We let $L(i, j)$ be the probability of a labelling that divides the input sequences into related and unrelated segments. Specifically, $L(i, j)$ is the probability that we output $x_{1\ldots i}$ and $y_{1\ldots j}$ from the white states, and substrings $x_{i+1\ldots m}$ and $y_{j+1\ldots n}$ from the black states. We define $L(i, j)$ as

$$L(i, j) = S(i, j)R(i, j) \tag{4.5}$$

where $S(i, j)$ is the probability that we end in the white switch state after having emitted $x_{1\ldots i}$ and $y_{1\ldots j}$, and $R(i, j)$ is the probability that we emit $x_{i+1\ldots m}$ and $y_{j+1\ldots n}$ using only black states, starting from the white switch state. We compute all $S(i, j)$ in $O(kmn)$ given k sub-models and input sequences of length m and n by using the forward algorithm for pair-HMMs [33]. The probabilities $R(i, j)$ can be computed in $O(mn)$ using the backward

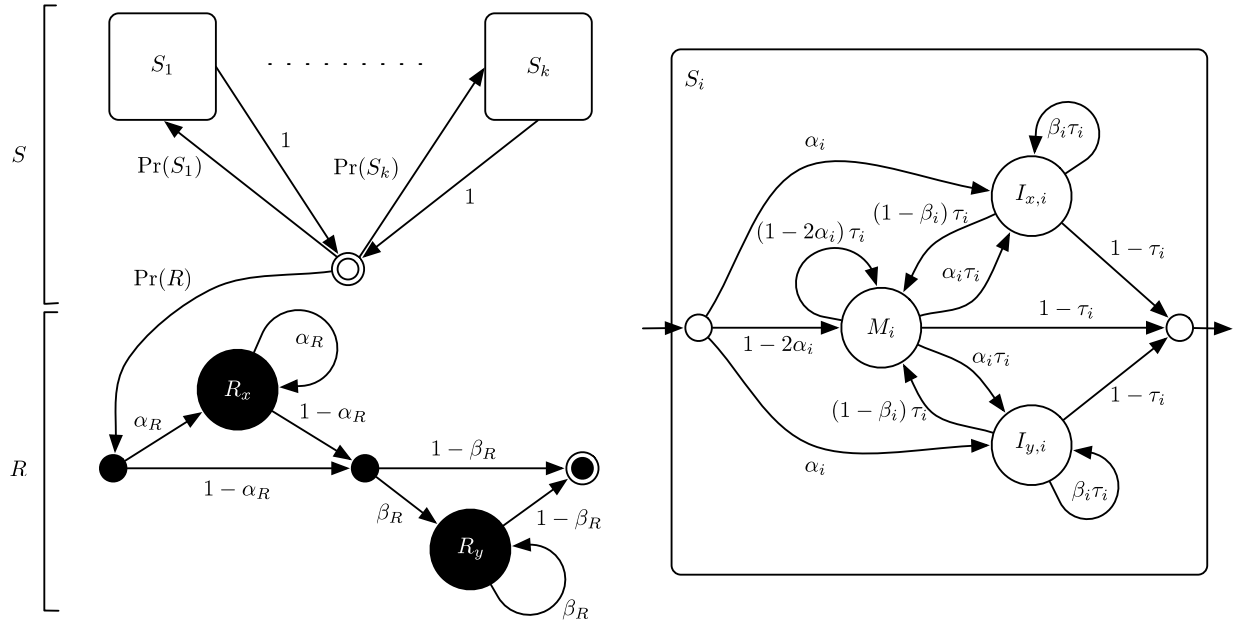

Figure 4.3: Pair-HMM for local alignment with multiple scoring schemes. The left gives an overview of the entire pair-HMM. The right box gives the structure of each sub-model, S_i. The white states (S) represent sequence related by a model of evolution that switches between sub-models S_1 through S_k. Black states (R) represent unrelated sequence. The double circled white state is the start state and the double circled black state is the stop state.

algorithm for pairHMMs. We find i and j that maximizes equation 4.5 using a final pass over all $L(i, j)$.

In practice, we perform many extensions in a single local alignment. Thus, a run time of $O(kmn)$ is too slow. We greatly improve the speed of our extension algorithm using several additional shortcuts and heuristics from BLAST [6, 5]. See section 2.5.1 in Chapter 2 for details.

First, we express an extension score as

$$s(i, j) = \lg[L(i, j)/\Pr[x_{1\ldots i}, y_{1\ldots j}|q]]$$

the base-2 logarithm of $L(i, j)$ divided by, $\Pr[x_{1\ldots i}, y_{1\ldots j}|q]$ the probability that we draw $x_{1\ldots i}$ and $y_{1\ldots j}$ from a background distribution q over the input alphabet. Since the black states are nearly identical to our background model, they cancel out in the odds score. Thus we ignore the black states completely and simply take the maximum forward probability, $S(i, j)$, as our segmentation probability.

Second, we adopt the x-drop heuristic to constrain the size of the forward algorithm dynamic programming matrix during extension. Assume we compute the forward algorithm recurrence for $S(i, j)$,

$$S(i, j, k) = \sum_{\ell} \begin{cases} S(i-1, j-1, \ell) t_{\ell,k} e_{x_i, y_j, k} \\ S(i-1, j, \ell) t_{\ell,k} e_{x_i, \epsilon, k} \\ S(i, j-1, \ell) t_{\ell,k} e_{\epsilon, y_j, k} \end{cases} \tag{4.6}$$

by filling S row by row. For each row i, we limit which j positions we include in the computation by comparing $s(i, j)$ to s_{max}, the maximum score we have seen to that point. If $s_{max} - s(i, j)$ falls below a threshold t, we stop computing columns of row i and move to the next row. We call t the x-drop threshold. The starting j position for $i + 1$ is the first j from row i such that $s_{max} - s(i, j) \leq t$.

In a BLAST-like seed-and-extend framework we perform two extensions from each seed hit: one in the forward direction and one in the reverse direction. The score for a complete two-way extension is the sum of the forward and reverse extension scores. In FEAST, we discard complete extensions whose score falls below a user defined threshold. Finally, during the extension phase, FEAST prevents local extensions from overlapping existing local extensions.

4.3.2 The alignment phase

After we find subsequence pairs that represent homologies according to our pair-HMM, we find global alignments for each pair. Since extensions can often be long, we need to

constrain each global alignment. One option is to perform anchoring as we do in CAPE. However, to save time, we use a different approach and find anchors during the extension phase.

The basic idea is to set anchors based on the forward probabilities that we compute during extension. We set an anchor every 100 bp. Specifically, every time a local extension moves at least 100 bases in both input sequences since we last set an anchor, or since the start of the extension, we set another anchor on the cell of maximum probability.

We perform global alignments of each sequence pair with Viterbi decoding using our new pair-HMM model alignment. We constrain the dynamic programming matrix to pass within 80 bases of each anchor set during the extension phase. Our Viterbi alignment algorithm also returns an annotation for the alignment indicating which sub-model produced each alignment column.

4.3.3 A data structure for detecting overlaps

An important part of our local alignment algorithm is preventing local extensions from covering the same subsequence pairs more than once. For input sequences x and y with lengths n and m, we view local extensions as constrained computations in an n by m matrix, where rows correspond to positions in x and columns to positions in y. Our goal is to prevent computing values for a particular position more than once. In this section we describe a data structure that allows us to quickly detect and prevent any overlapping extensions.

We have several requirements that we need to meet. First, we need to be able to quickly determine if a seed hit is within an existing extension. Second, for seed hits that are in an unexplored region of our matrix, we need to be able to quickly find the nearest column to the left and right that has already been explored. And finally, we need to be able to update these boundaries in near constant time as we step from one row to another during extension. We meet these requirements in two ways. First, we break up extensions into segments that meet certain properties. And second, we store these segments in a tree-based data structure. We now describe the details of this structure, and how and when we break up an extension into segments.

Our data structure stores a collection of segments. Every previously computed extension is represented by one or more linked segments. A *segment* is a sequence of pairs, (i, j), and a starting row r. Each pair, (i, j), represents the left and right boundaries of columns explored by previous extensions. Each pair corresponds to a matrix row according to its position in the sequence. For example, the first pair in the sequence corresponds to row

r, the second to row $r+1$, and so on. Our data structure ensures that for every segment, all overlapping segments start at the same row, r, and have the same number of column pairs in the sequence. To maintain this property, whenever we add a new segment to the data structure, we break both the newly added segment, and existing segments, into smaller segments linked by pointers. To insert a new extension, we represent it as a single large segment and let the data structure modify it to maintain our overlapping segment property.

The root of our data structure is a balanced tree where each tree node holds a row number, r, and pointer to a secondary balanced tree. We order the primary balanced tree by row number. Each secondary tree stores a collection of overlapping segments that share starting row r. We order the segments in each secondary tree by the right-most column in the segment: the j value from the last pair in the sequence. This basic structure allows us to find out if a point is within an existing extension in $O(\log n \log m)$ time. For points not within an existing extension, we can find any flanking extensions at the same time. Figure 4.4 gives an example of two extensions represented by three segments.

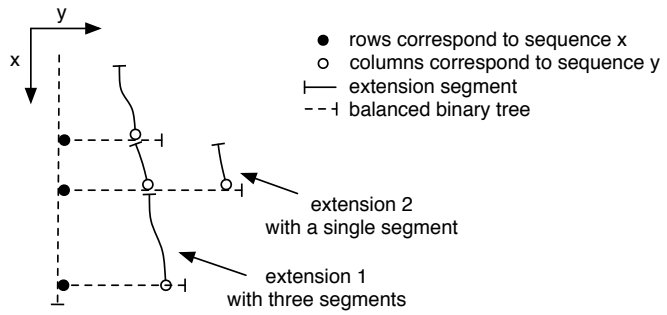

Figure 4.4: Layout of tree based data structure for quickly detecting extension overlaps. We represent extensions as a collection of linked segments. A primary balanced tree indexes collections of overlapping segments by their starting row. Inserting a new segment causes it, and existing segments, to break into smaller segments such that starting and ending row of all overlapping segments are the same. We store overlapping segments in their own balanced tree by the rightmost column of their lower extremity.

To enable near-constant time updates as we change rows during our forward extension algorithm, we use several more tricks. Assume that we are computing values on the row that corresponds to the symbol x_r. If no existing segment flanks this position on either side, then we simply store the number of rows until the next segment in the data structure. Thus we need only increment a counter until we reach the next segment. At that point, we spend $O(\log m)$ time to find the new flanking segments. Assume we do have a flanking

segment on the left. In this case, we can iterate through the sequence of pairs in the stored segment as we change rows. Each time we obtain new column boundaries in constant time. Once we reach the end of the segment, we must again perform an $O(\log m)$ lookup to find updated segment boundaries. We also perform an additional optimization in this case. If two segments are part of the same local extension, and we run off the end of the first segment, then we use the pointer to the second segment as the starting point for our binary search.

4.3.4 Seeding local extensions

To find a set of fragments to extend, we use a two step process. First, we use multiple spaced seeds to find starting positions. Specifically, we use a collection of six spaced seeds from Sun and Buhler [77], three of which target alignments of intergenic sequences and three which target alignments of exonic sequences. Second, we follow the technique BLASTZ and use a fast ungapped alignment algorithm to filter seed hits.

Our ungapped alignment algorithm follows our gapped extension algorithm, but disallows transitions to the insertion states of our pair-HMM. As before, we use the forward algorithm instead of the Viterbi algorithm, although the difference in this case is minimal since without gap states, the number of possible paths through is greatly reduced. None-the-less, using the forward algorithm allows ungapped extensions to switch scoring schemes. As in our gapped extensions, we use the x-drop heuristic to terminate extensions. The ungapped score of a seed is the sum of the extensions in two directions. We discard any seed hit that has an ungapped score below a user-defined threshold. For most cases, we find that a threshold of 20 is appropriate. See section 5.2 of the next chapter for details on how we arrive at this value.

4.4 Parameter training

We use FEAST, our local aligner, in a new approach to training alignment parameters. Our goal is to find a good systematic method for training alignment parameters for two species. When training general-purpose alignment parameters for two large genomes, several problems arise.

First, different regions of each genome may have different statistical properties. For example, the background distribution of nucleotides often varies, as does the level of homology

when compared to a second genome. Functional sequence is also often much more conserved than non-functional sequence, and different genome regions have different amounts of functional sequence.

The methodology of Chiaromonte, Yap, and Miller [25], which they apply to training parameters for human and mouse genomic sequence, address these issues in two ways. To address large-scale variability, they train parameters on three manually-selected regions of the human and mouse genomes. From the resulting three parameter sets, they select a best parameter set by evaluating ungapped alignment performance on nine manually selected genome regions. Their performance metric considers off-diagonal alignments, those not part of the optimal chain, to be false positives. To address variability between conserved and non-conserved genomic sequence regions, they note that highly similar sequence regions are easy to align regardless of model parameters. On the other hand, to correctly align weakly similar sequence, good parameters are needed. Based on this, Chiaromonte *et al.* train only on weakly similar sequence by filtering out strong alignments from their training data.

Although useful, their training method suffers from several shortcomings. First, three regions may not be enough to properly sample a large genome. Additionally, manual selection of genome regions may unintentionally introduce bias into the training data. The second problem is the use of only ungapped alignments to train match and mismatch probabilities. Ungapped alignments have great difficulty detecting very weakly similar sequence and thus biases training towards stronger similarities. Related to this, the methodology of Chiaromonte *et al.* does not address training of gap parameters, an important part of sequence alignment. Third, their performance metric considers alignments off the main diagonal to be false positives. Genomic rearrangements may often result in correct alignments being off the main diagonal. Thus, considering all off-diagonal alignments to be false may not be a good idea. Finally, their procedure requires the user to manually select a similarity threshold above which alignments are discarded. It is not clear what this threshold should be.

We address all these concerns in our own training methodology. Our procedure starts by randomly sampling homologous regions from the two target genomes. Initial homologies may be roughly identified by incorrect alignment parameters, or by parameters for a related species pair. In our own study, which we describe in detail in the next chapter, we use pre-identified homologies. After a set of training regions is identified, we train parameters for each region using a new expectation maximization (EM) training algorithm based on FEAST.

Our EM algorithm takes one target sequence and any number of query sequences, as well as a set of initial parameters for our new more descriptive alignment model. We expect

to see highly conserved functional sequence and more weakly conserved non-functional sequence. We model this by initializing two sub-models, one for strongly sequence with little mutation and one for sequence with a lot of mutation. Given a mutation rate in terms of substitutions per site, FEAST automatically creates a match-mismatch matrix using the background distribution of the input.

Given appropriate input and initialization of our model, we use the following iterative procedure:

1. Find all forward extensions with a score above a threshold.
2. We sum the extension scores and compare the resulting value with that of the previous iteration, if there is one. If the difference is less than 5 bits, we stop and return the current parameters.
3. Otherwise, we perform training on the set of extensions using either Baum-Welch or Viterbi training [33]. We constrain the dynamic programming space of these algorithms using the anchors set during extension.
4. Adjust the learned parameters according to our model constraints.
5. Go to step 1.

Repeats are of particular concern for our training procedure. Repetitive sequence causes an explosion of local extensions that can dominate the trained alignment parameters. We address this by masking known repeats and disallowing seed hits and extensions into masked regions.

We expect that many of our randomly selected genome regions will result in similar parameters. To save time in selecting an best final parameter set, we identify similar parameter sets and select only one parameter set from each group of similar sets. We do this by clustering the parameter sets using a reasonable clustering algorithm. See section 5.1 of the next chapter for details on the clustering method we use in applying this method to the human and mouse genomes. After clustering, we select a small number of parameter sets to test based on the results. These parameter sets should be significantly different from each other. In our study of the human and mouse genomes in the next chapter, we choose just three parameter sets.

To evaluate the performance of a given parameter set, we use a simple metric: the number of bases of the first species that are aligned to the second. This metric does not distinguish true alignments from false alignments, instead both types of alignments are

included. By choosing the parameter set that aligns the most sequence, we attempt to optimize for sensitivity in detecting homologies, rather than specificity. An additional positive property of this metric is that it requires no prior knowledge of the genomes being studied. This is important for new genomes. However, prior knowledge, if it exists, may also be incorporated into the evaluation step. For example, in our study of human and mouse sequence in the next chapter, we make use of known exons in addition to global alignment statistics.

Our procedure as a whole addresses all the problems we identified with the methodology of Chiaromonte *et al.* We eliminate bias in selecting training data by using random sampling. Our EM training algorithm uses our new pair-HMM for alignment to naturally find a boundary between weakly similar and strongly similar sequence regions. It also incorporates training of gaps for each sub-model, as well as parameters unique to our model such as sub-model length. By incorporating gaps, we also achieve higher sensitivity to weakly homologous sequence. Finally, our simple performance metric for choosing a final parameter set allows for rearrangements and makes no assumptions about which alignments are correct or incorrect.

4.5 Summary

In this chapter we presented two new pair-wise aligners: CAPE, a global aligner, and FEAST, a local aligner. Both algorithms share two common improvements, the use of the forward algorithm for homology detection rather than the Viterbi algorithm, and a new more descriptive model of alignment.

CAPE is a global aligner that scores fragments with a local homology detection algorithm that uses the forward algorithm to evaluate the nearest 200 bp. Specifically, our algorithm produces a *local segmentation*, dividing sequence into a related and unrelated part. If we use Viterbi instead of the forward algorithm, our local segmentation becomes a short local alignment. We address sequence variation by allowing each fragment to be scored with more than one set of alignment parameters. We find fragments using a recursive GLASS like procedure. For each recursive step, we use random spaced seeds and filter the set of hits by the score of an ungapped alignment of the length of the seed. We choose the seed weight and threshold to minimize the number of false hits given the length and background distribution of the region being anchored. CAPE also uses a mixed global/greedy approach to choosing alignments. If there are many high-scoring seed hits, we use use the optimal chain of fragments as anchors. However, if we are in a region with

no high score hits, than we use a greedy approach to choose the single best fragment as an anchor.

FEAST is a local aligner that replaces Viterbi extensions, after seeding, with forward extensions. Forward extensions use the x-drop heuristic to constrain the dynamic programming matrix, but instead of finding the optimal path through the constrained space, we sum all possible paths. Since the forward algorithm returns no alignment, our forward extensions drop approximate *anchors* every 100 bases. These anchors do not necessarily lay on the optimal path but can be used to roughly constrain a final alignment step. We note that while our implementation currently uses Viterbi to produce a final alignment, the framework in FEAST allows one to use any alignment algorithm in the final step. One may even skip the final step altogether in cases where an alignment is not needed. For example, in situations where local alignments are used as anchors in a multiple alignment procedure, or when only large scale homologies are needed. We seed our local extensions using a set of six spaced seeds. Three of these are optimized for coding regions and three for intergenic sequence. We filter hits with ungapped alignments in a traditional BLAST-like framework.

Both CAPE and FEAST use a new pair-HMM model of alignment. This model describes a fixed number of sub-models, each of which represents traditional alignment under a specific scoring scheme. The alignment is allowed to jump between sub-models, but incurs a probabilistic penalty to do so. In CAPE, this algorithm is used as the final alignment step. In FEAST, it is used as the model of alignment throughout, for ungapped extensions, gapped extensions, and the final alignment. In addition to giving more accurate alignments, when we use this model with the Viterbi alignment we obtain an additional annotation describing which sub-model produced each column of the alignment.

Finally, we describe a method for training alignment parameters for two genomes using FEAST. This method includes an expectation maximization procedure that infers parameters for our new pair-HMM. In the next chapter, we apply this method to the human and mouse genomes. Afterwards, we describe several hypotheses and experiments on both FEAST and CAPE.

Chapter 5

Sensitive genomic alignment: experiments

In this chapter we make several hypotheses about the methods described in Chapter 4 and test them with experiments on real and synthetic genomic DNA sequences. In the first section we infer new alignment parameters for aligning human and mouse sequences that describe regions of weak homology and regions of strong homology. In the second section, we explore the sensitivity and specificity of our new local extension algorithm using both synthetic data and real biological data. Finally, in the last section we explore the limits of alignment. Using synthetic sequences, we explore how much mutation we can tolerate before simulated sequences become unalignable. We also evaluate our anchoring technique in the situation where the underlying alignment parameters are unknown.

5.1 Inferring parameters for human and mouse local alignments

In order to use our new model of alignment on real data, we must first train parameters. We focus on aligning human and mouse genomic data. Human and mouse alignments have been studied by many researchers, and the default parameters for popular alignment programs such as BLASTZ, and LAGAN target these two species [25]. Alignments between human and mouse are also important to biologists; they are often used to identify new functional sequence in humans. Thus, these species are a good target for our study as we can compare our results against previous work.

Before we start training, we must decide on the number of sub-models we wish to use in our new model of alignment. Human and mouse sequences contain strongly conserved, functional, sections and weakly conserved, non-functional sections. This observation is the basis for comparative genomics, a widely used technique for finding functional sections of a target genome. As such, choosing two sub-models, one that targets highly similar regions, and one that targets weakly similar regions is justified. It is not clear if using more than two sub-models is appropriate, although there are many possible scenarios that one can think of. For example, sub-models could target different background nucleotide distributions, regions of various simple repeat structures, or even regions of moderate conservation such as those that contain promoters that are more tolerant of mutation.

Despite these possibilities, we limit our current study to the most obvious and justified extension, using only two sub-models: one for strongly similar regions and one for weakly similar regions. We also train a traditional model of alignment on the same data so that we may better contrast our new model of alignment with the standard model. We hypothesize that by using a more realistic model of evolution describing strongly and weakly conserved sequence, we can obtain alignment parameters that allow for better detection of homology.

5.1.1 Selecting training regions

We select positions uniformly from UCSC's 2009 `hg19` release of the human genome. Thus, longer chromosomes are more likely to be selected than shorter chromosomes. We include only the main chromosome sequence from UCSC, discarding mitochondria and unassembled DNA. We expand each selected position to a 500 kb region centred on the selected point. For points near chromosome ends, we shift the region such that our selected point is as close to centred as possible while our region is still within the chromosome boundaries.

Since the `hg19` genome was not mapped to the mouse genome at the time of this study, we obtain homologous mouse co-ordinates for each `hg19` region by using UCSC's `liftOver` tool. We map to the 2007 `mm9` version of the mouse genome. For each region, we first find the corresponding human co-ordinates in release `hg18`. Since `hg18` is mapped to `mm9`, we find the homologous region in the `mm9` release of the mouse genome. We discard any `hg19` region that does not map cleanly to a single `mm9` region. It is possible for `mm9` regions that map to more than one `hg19` region to be included, although the opposite situation is prevented.

Table 5.1 shows our final set of training regions. To compare our training methods with that of Chiaromonte, Yap, and Miller [25], we include the `HOXD` region and the `CFTR` region

	Human (hg19)			Mouse (mm9)		
Name	Chr.	Location (bp)	Size (kb)	Chr.	Location (bp)	Size (kb)
R1	21	23,595,894	500	6	82,173,962	270
R2	13	32,486,650	500	5	150,972,003	411
R3	1	72,510,494	500	3	156,056,509	368
R4	6	110,764,533	500	10	39,987,554	317
R5	5	163,928	500	13	74,169,974	332
R6	10	93,246,124	500	19	36,674,643	384
R7	3	192,789,712	500	16	29,050,609	510
R8	6	12,749,983	500	13	42,805,180	401
R9	13	69,114,337	500	14	95,580,184	371
R10	13	93,438,259	500	14	116,816,297	553
CFTR	7	117,072,842	283	6	18,066,103	257
HOXD	2	176,937,507	139	2	74,486,575	134

Table 5.1: Homologous regions of the human and mouse genomes. Regions R1 through R10 are randomly selected, and the CFTR and HOXD regions are manually selected.

in addition to our ten randomly selected regions. Table 5.2 gives a description of known genes in each training region.

5.1.2 Heuristic parameters for training

Our local aligner FEAST contains several heuristic parameters beyond those of our new alignment pair-HMM. These parameters are our choice of seeds, our ungapped alignment x-drop and score filter, and our gapped alignment x-drop and score filter. These parameters are very difficult to formally train, and thus we set them manually by performing numerous test alignments, and test parameter training, on our training data.

Aside from our seeds, which we take from Sun and Buhler [77], we started with conservative values for each heuristic parameter chosen so that we see little noise in several initial test alignments. We use an ungapped x-drop of 10 bits, and an ungapped filter threshold of 10 bits. For gapped alignments, we use an x-drop of 25 bits and a score filter of 50 bits. As we see in later experiments, our choice of gapped score threshold and x-drop are extremely conservative. This is appropriate for training, as we do not want to train using incorrect data.

Name	Chr.	Full Genes	Partial Genes
R1	21	None	None
R2	13	FRY, ZAR1L, BRCA2	EEF1DP3, N4BP2L1
R3	1	None	NEGR1
R4	6	CDC2L6, AMD1	SLC22A16
R5	5	LOC389257, CCDC127, SDHA, PDCD6, AHRR, C5orf55, EXOC3, LOC25845, SLC9A3, CEP72	PLEKHG4B, TPP
R6	10	PPP1R3C, TNKS2, FGFBP3 HECTD2, BTAF1	LOC100188947,
R7	3	HRASLS, MGC2889, ATP13A5, ATP13A4	None
R8	6	None	PHACTR1
R9	13	None	None
R10	13	None	GPC5, GPC6
CFTR	7	CFTR	CTTNBP2
HOXD	2	EVX2, HOXD13, HOXD12, HOXD11, HOXD10, HOXD9, HOXD8, HOXD4, HOXD3, HOXD1	None

Table 5.2: Training regions for human and mouse alignments. Regions R1 through R10 are random selections that each span 500 kb of human sequence. Regions R1 and R9 have a few EST hits, but are otherwise gene deserts.

5.1.3 Training a single conservation level

Chiaromonte, *et al.* [25], use a BLOSUM like approach to training a traditional model of alignment. That is, they exclude any alignment regions that are more similar than a given threshold and train on the remaining weakly similar regions. This technique is based on the idea that the traditional model of alignment does not accurately model real genomic alignments, a problem that we specifically address in our work. In contrast to the training of Chiaromonte, our intent in training a traditional model of alignment is to demonstrate how the model performs if treated as though it is correct. We do not exclude any training data in our training procedure, and optimize our single sub-model parameters in the same

way as we optimize our two sub-model pair-HMM describing highly similar and weakly similar alignment regions.

Before training each region, we initialize our new alignment model with a single sub-model corresponding to alignments with 0.4 substitutions per site. We then train parameters using our expectation maximization algorithm with Baum-Welch estimation on each iteration. Table 5.3 summarizes the results. We omit the sub-model selection probability and sub-model length parameters as they have little affect on produced alignments.

Name	%ID	TTR	GC	GOP	GEP	MGL
R1	71	0.85	31	6.04	0.54	3.22
R2	72	0.61	38	5.85	0.38	4.32
R3	69	0.75	33	5.69	0.58	3.04
R4	74	0.60	40	5.52	0.40	4.14
R5*	78	0.45	54	6.93	0.58	3.00
R6	75	0.60	40	5.44	0.58	3.00
R7	69	0.64	40	5.94	0.42	3.97
R8	69	0.64	40	5.73	0.41	4.08
R9	70	0.83	32	5.95	0.50	3.42
R10	70	0.76	34	5.86	0.44	3.82
CFTR	70	0.68	35	5.68	0.41	4.04
HOXD	79	0.65	50	6.15	0.42	3.95

Table 5.3: Training results for a traditional model of alignment describing a single mutation rate. We summarize the match/mismatch matrix with percent identity (%PID), the transversion-transition ratio (TTR), and background percent GC (GC). We express parameter α_i in terms of a gap open penalty (GOP) and β_i in terms of the gap extension penalty (GEP) and mean gap length (MGL). In region R5, we use DUST [4] with parameter $v = 10$ due to excessive repeats.

From the results, we see that some of the trained parameter sets appear similar. To better understand the results of our training, we compare the parameter sets by using hierarchical clustering. For each region, we represent the trained parameters as a vector of model probabilities and use Euclidean distance to represent the distance between two vectors. We cluster the parameters with the `hclust` command from GNU R [68] using the complete linkage method. At each step, the `hclust` algorithm recomputes distances between clusters with the Lance-Williams dissimilarity update formula. In this process all model probabilities are weighted equally and in the range $(0, 1)$. The resulting dendrogram

in Figure 5.1 shows at least three clear groups.

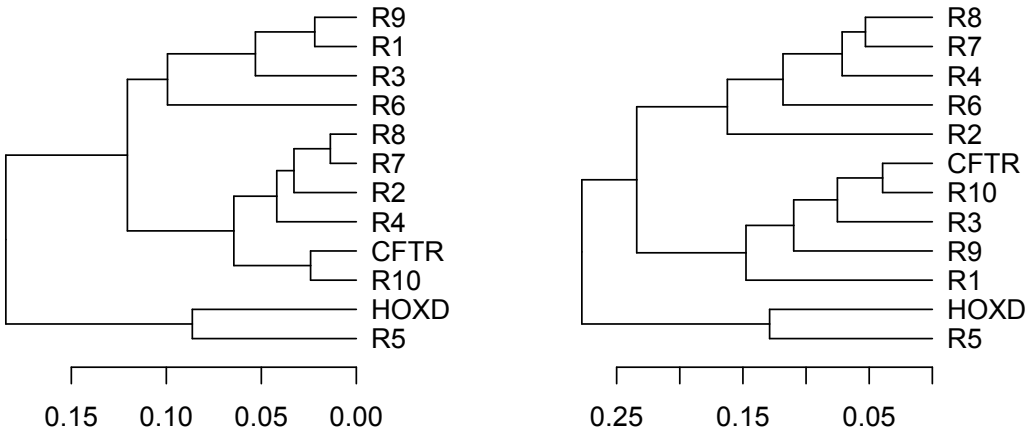

Figure 5.1: Left: dendrogram of trained parameters for the traditional alignment model. Right: dendrogram of trained parameters for a model of alignment describing regions of strong and weak homology. Most pairings remain the same in both trees, but there are some changes in the top three groups. The two model parameters have a wider distribution of distances.

5.1.4 Training two conservation levels

We now expand our alignment model to describe alignments with regions of two types: weakly similar sequences and strongly similar sequences. We initialize our two sub-models to match our goal: one corresponds to alignments with 0.2 substitutions per site, and the other to alignments with 0.4 substitutions per site. As long as our initialization roughly separates the two types of alignment regions, we expect our EM training procedure to converge to the correct local maxima.

In this case, we use Viterbi training instead of Baum-Welch as the last step of each EM iteration. Initially we attempted to train with the Baum-Welch algorithm as we did for our traditional model of alignment. However, we found that the parameters τ_1 and τ_2 became unrealistically small, causing training to fail. We hypothesize that the reason for this is due to the fact that our pair-HMM allows an alignment to jump from a sub-model

S_i back to itself. There are a very large number of valid paths that do this, far more than the number of paths that never jump from a sub-model back to the same sub-model. Since Baum-Welch training evaluates all possible paths, it incorrectly favours extremely small values of τ_i.

Viterbi training avoids the problem because jumping from S_i to S_i incurs an extra probability cost and is thus always worse than simply staying in S_i. A better solution would be to prevent jumps from a sub-model S_i to itself by changing the pair-HMM. However, since our current alignment implementation is hand optimized for performance, modifying the model requires much work. While we settle for Viterbi training in the final alignment step, we still use the forward algorithm for discovering local homologies. Thus we maintain sensitivity in choosing appropriate training pairs.

Table 5.4 summarizes our trained parameter for the strongly similar sub-model and the weakly similar sub-model. While region lengths described by τ_i vary, the gap and conservation rates have many similarities across regions. As before, we cluster our results in order to group similar parameter sets. We follow the techniques we used for the single sub-model. We represent each parameter set as a vector of model probabilities, but exclude the values $\Pr[S_i]$ and τ_i due to their high variability and relatively low impact on output. As with the single sub-model results, the resulting dendrogram on the right side of Figure 5.1 shows at least three main groups.

5.1.5 Selecting good general purpose parameters

Training parameters for every pair of sequences one wants to align is very time intensive. Although it is done in FSA [11], it is much more common to select a single set of parameters that has good performance over a variety of sequence content. In this sub-section our goal is to select a single parameter set for the pair-HMM from the twelve training regions that has good performance over all twelve training regions.

Choosing a performance metric for real alignments, especially ones in which there may be new, not yet known, homologies, is very difficult. While the human and mouse genomes are well studied, current annotations are based on existing techniques and model parameters. Since our goal is the derivation of new parameters, and possibly the identification of new homologies, we desire a metric that is not based on existing results.

For these reasons, we choose to use the amount of human sequence aligned as our performance metric. This metric has a danger of preferring sensitivity over specificity, and thus we continue to use very conservative settings for our heuristics in an attempt to control

Strongly similar sub-model							
Name	Pr.	Len.	%ID	TTR	GOP	MGL	%GC
R1	0.61	334	74	0.75	6.68	4.44	32
R2	0.27	193	85	0.49	7.77	7.13	45
R3	0.40	340	77	0.71	6.41	3.94	34
R4*	0.53	136	87	0.48	6.96	3.66	40
R5*	0.57	128	84	0.41	9.40	3.18	57
R6*	0.41	151	88	0.43	7.48	5.63	38
R7	0.21	185	84	0.48	7.75	3.73	42
R8*	0.31	168	80	0.55	6.87	3.99	39
R9	0.22	330	84	0.71	7.24	3.26	33
R10	0.22	280	83	0.69	7.15	4.41	39
CFTR	0.19	207	84	0.55	7.34	3.95	39
HOXD	0.37	244	92	0.52	7.98	3.78	49
Weakly similar sub-model							
Name	Pr.	Len.	%ID	TTR	GOP	MGL	%GC
R1	0.39	426	59	1.02	7.13	39.59	29
R2	0.73	355	68	0.63	6.35	10.45	36
R3	0.60	466	62	0.79	6.48	11.88	31
R4*	0.47	266	69	0.70	6.29	9.24	42
R5*	0.43	342	65	0.54	6.76	35.20	53
R6*	0.59	233	70	0.61	6.37	6.82	44
R7	0.79	378	66	0.67	6.46	7.94	39
R8*	0.69	293	67	0.62	6.47	7.62	42
R9	0.78	463	65	0.86	6.43	8.02	31
R10	0.78	429	66	0.78	6.46	8.79	33
CFTR	0.81	434	68	0.70	6.30	8.78	35
HOXD	0.63	375	71	0.72	6.42	9.23	50

Table 5.4: Trained parameters for the strongly similar and weakly similar sub-models. We summarize the match/mismatch matrix with percent identity (%ID) and the transversion-transition ratio (TTR). We express parameter α_i in terms of the gap open penalty (GOP) and β_i in terms of the mean gap length (MGL). In regions R4, R5, R6, and R8, we use DUST with parameter $v = 10$ due to excessive repeats.

false alignments. We also make use of known information by looking at the percentage of known exons that are aligned, in addition to global statistics.

Since many of our trained parameter sets are similar to one another, we save time by using our clustering results to select out three parameters that are significantly different from one another. We do this for both single sub-model parameters and two sub-model parameters.

Based on our clustering results, we select regions R9, R8, and HOXD for further analysis. These cover all three major groups in both dendrograms and represent three distinct region types: R9 is a gene desert, HOXD is gene-dense, and R8 contains the majority of a large gene. We name our parameter sets as *k-name*, where k represents the number of sub-models and *name* the region we trained it from; thus, parameter set 2-R8 gives parameters for 2 sub-models trained from region R8.

All Human Positions		**Human Exon Positions**	
Name	Aligned	Name	Aligned
1-R8	1,425,363	1-R8	111,127
1-R9	1,300,453	1-HOXD	109,993
1-HOXD	958,122	1-R9	107,704
2-R8	1,480,120	2-HOXD	113,173
2-R9	1,475,191	2-R8	112,034
2-HOXD	1,285,032	2-R9	108,137

Table 5.5: Number of aligned positions over all training regions for different parameter sets. There are a total of 5,422,010 human positions and 131,992 known exon positions. Parameter sets in each category are ordered by positions aligned. The 2-R8 parameter set is a good compromise between the overall aligned positions and the aligned exon positions.

Table 5.5 gives the number of aligned human positions and the number of aligned exon positions for our selected parameter sets. Overall, parameter sets trained on R8 find the most aligned positions. For known exons, the parameter sets trained from the HOXD region performed best. Given that the HOXD region is gene-dense, this result is not surprising. However, parameter sets trained from HOXD aligned the fewest bases overall. The parameters trained from R8 appear to provide the best compromise between sensitivity to exons and overall sensitivity. We also observe that overall, parameter sets trained with two sub-models performed better than their counterparts trained with only a single sub-model.

A better single-model parameter set

Chiaromonte, *et al.* [25], point out that strongly conserved sequence aligns well regardless of parameter choice. This hypothesis motivates the BLOSUM approach they use, where they exclude highly similar regions from their training data. We explore this hypothesis with our own training data. While [25] manually select a similarity threshold for filtering training data, our two sub-model pair-HMM naturally finds such a threshold by classifying data into each sub-model.

According to the above hypothesis, we expect that a single model parameter set formed from only the weak sub-model will align nearly the same amount of sequence as our complete two sub-model parameter set. We test this by building 1/2-R8, a single model parameter set consisting of the weakly similar sub-model of 2-R8.

Overall, 1/2-R8 aligns 1,469,228 human bases, compared to 2-R8's 1,480,120 bases. On exons, 1/2-R8 aligns 112,034 bases and 2-R8 aligns 111,987. The two sub-model parameter set performs slightly better overall and nearly identical to the one sub-model parameter set on exons. However, compared to the single sub-model parameter sets in Table 5.5, the 1/2-R8 parameter set performs substantially better than than all of them. Over all human positions, 1/2-R8 aligns an additional 44 kb of positions over the best single model parameter set.

Our technique of training with two sub-models allows us to produce good traditional alignment parameters in addition to parameters for our new richer model of alignment.

5.2 Accuracy of FEAST

In this section we explore how the x-drop and score filters affect the sensitivity and specificity of FEAST alignments. Additionally, we compare our new forward extension algorithm with traditional viterbi extensions. Finally, we compare FEAST as a whole with lastz [40] using its default settings. Our purpose is to give an idea how our methods compare with a popular current local aligner in a typical usage scenario.

5.2.1 Simulated human and mouse sequences

To measure the impact of different values for the x-drop and score filters in FEAST, as well as to compare forward extensions to Viterbi extensions, we use synthetic test data. Unlike real sequences, synthetic data allows us to precisely measure sensitivity and specificity of

alignments. We use simulated human and mouse sequence pairs produced by the methods of Blanchette *et al.* [10].

Their procedure simulates the sequences of several species using a phylogenetic tree. The simulation includes the insertion of real repeat elements as well as large scale rearrangement events. While its intended use is to benchmark multiple alignment programs, we use it to benchmark pair-wise alignments by using only human and mouse sequences from each simulated data set. For each of the following experiments, we use 50 simulated sequence pairs, where each sequence is about 20 kb in length. We use the 2-R8 and 1/2-R8 parameter sets from the previous section to compute alignments with FEAST. To compare Viterbi extensions to forward extensions we create a special version of FEAST that uses Viterbi extensions but is otherwise identical.

We measure the sensitivity, or true positive rate (TPR), as the fraction of alignable bases correctly aligned. We express specificity as the false positive rate (FPR), defined as $1 -$ specificity, where specificity is the fraction of aligned bases that are correct.

We examine the effect of our gapped score filter, described in Section 4.3.4 of Chapter 4, by fixing the x-drop at 25 and disabling our ungapped extension filter by setting it to zero. Figure 5.2 gives ROC curves for the 2-R8 and 1/2-R8 parameter sets with both forward extensions and Viterbi extensions. Both Viterbi extensions and forward extensions behave the same. For filter values from 0 bits to 10 bit, we improve specificity with little impact on sensitivity. Above 10 bits we lose sensitivity with no effect on specificity. Thus, for synthetic sequences it seems that 10 bits is the optimal gapped filter threshold.

Next, we fix the gapped score filter to 10 bits and vary the x-drop. The ROC curves in Figure 5.3 show that our initial choice of 25 for the x-drop is extremely conservative. Lowering the x-drop to 15 causes only very short, insignificant alignments to be produced. Thus, our training setting was the most conservative settings we could reasonably use. For forward extensions, the sensitivity peaks at an x-drop of about 75. Higher x-drop values cause more false base pairings to be included in alignments.

Forward extensions versus Viterbi extensions

Figure 5.3 also includes ROC curves for FEAST with Viterbi extensions. We see that for a fixed false positive rate, forward extensions provide much greater sensitivity. Similarly, for a fixed sensitivity rate, forward extensions give a better specificity. Finally, we see that no matter what x-drop we use with Viterbi extension, we cannot match the sensitivity of forward extensions with an x-drop of 35 or higher. On the other hand, at small x-drop

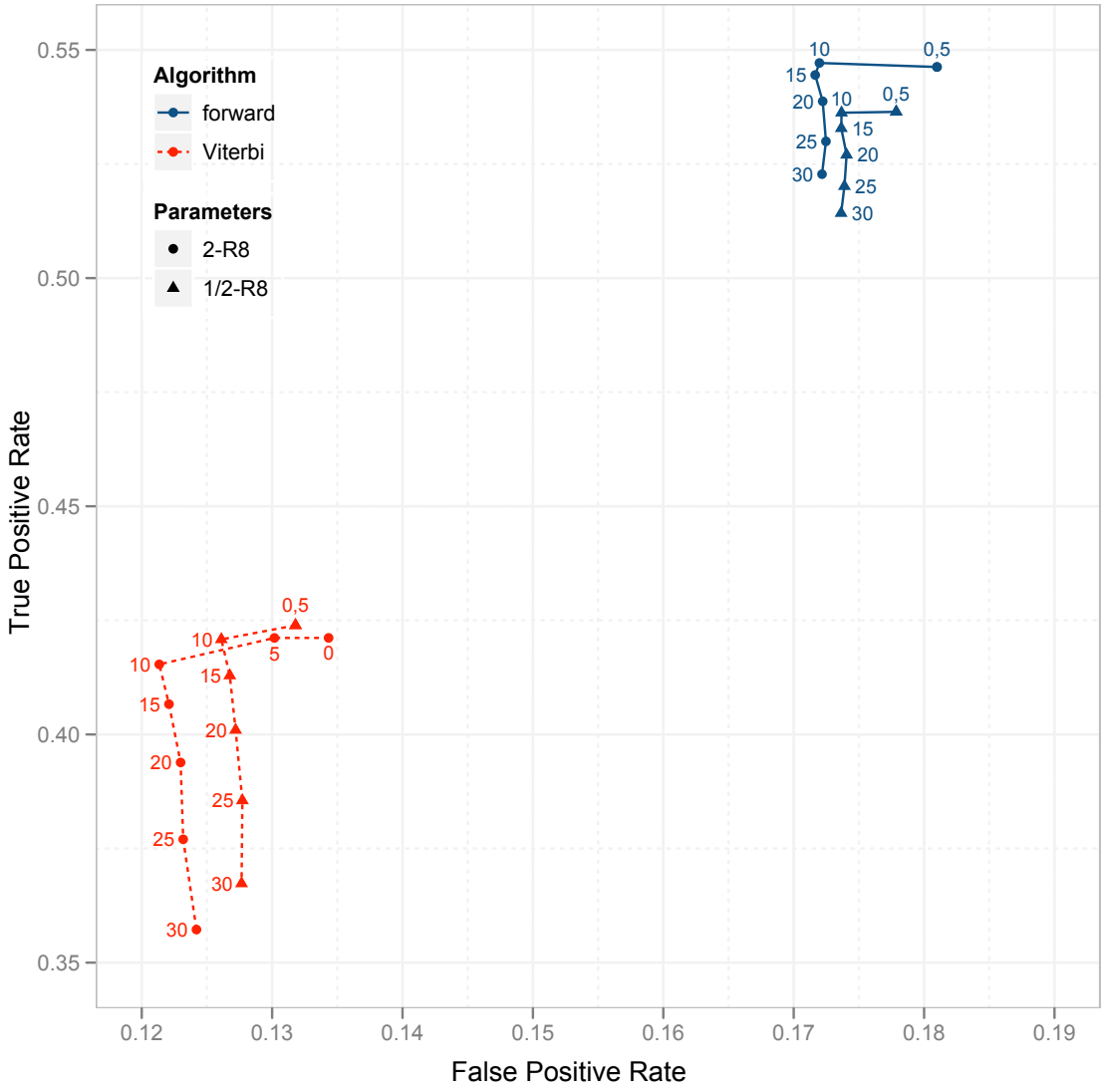

Figure 5.2: ROC curve for the gapped score filter with an x-drop of 25. The filter reduces false positives until it reaches 10 bits. Above 10 bits, we lose sensitivity with no change in specificity. For synthetic sequences, 10 bits is the optimal filter setting.

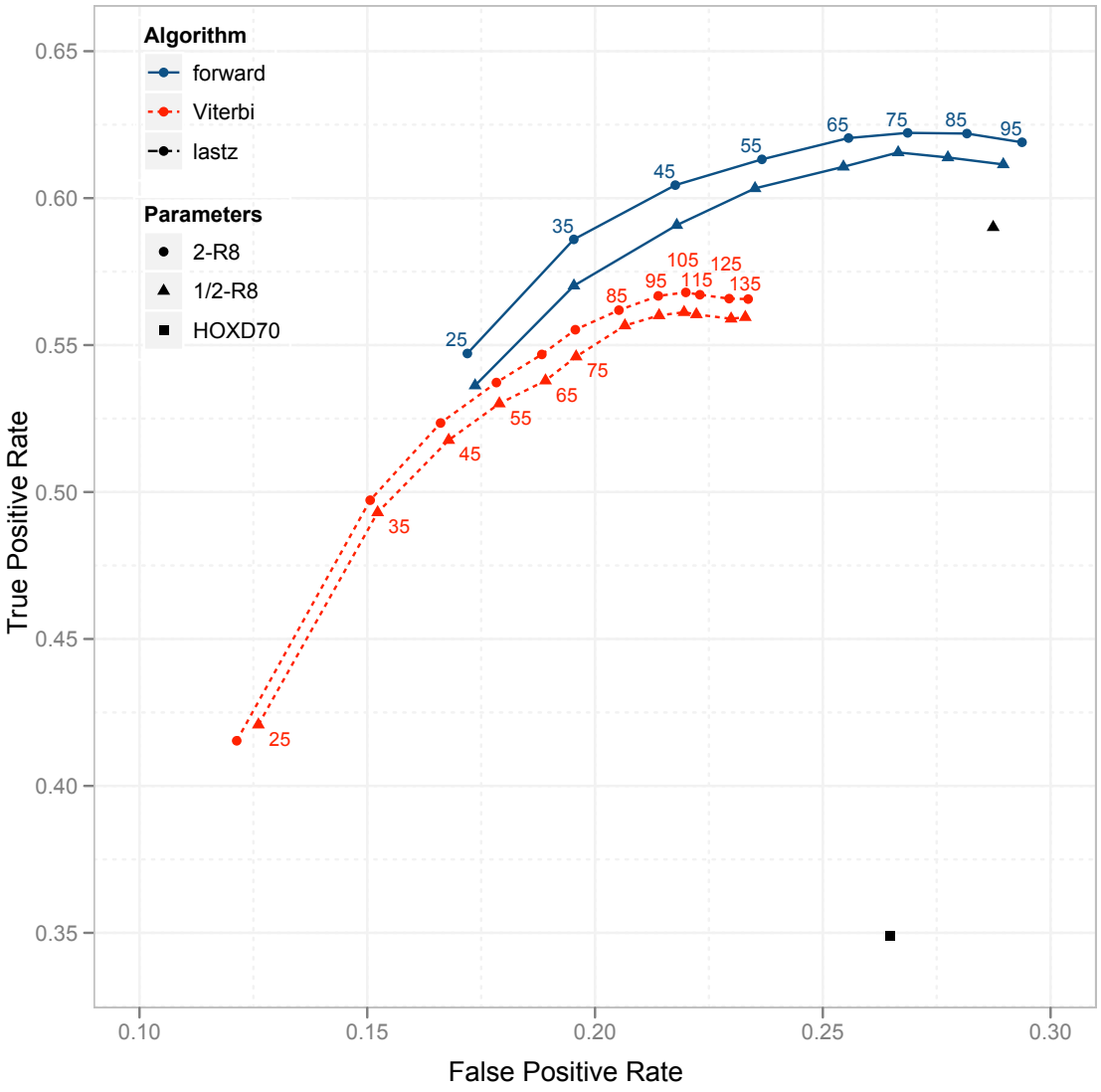

Figure 5.3: ROC curve for the gapped x-drop with score filter of 10. We also plots points corresponding to lastz with default parameters, and lastz using the 1/2-R8 parameter set.

value, Viterbi is less prone to false positives than even the forward algorithm with even the most conservative x-drop.

FEAST compared to lastz

To get a feel for how the techniques in FEAST compare to existing work, we compare sensitivity and specificity to that of lastz. The lastz local aligner is a replacement for BLASTZ [73], a very popular local alignment program that uses a scoring scheme similar to that of Chiaromonte *et al.* [25]. In addition to using Viterbi extensions and a different scoring scheme, lastz also differs from FEAST in its seeding techniques, filter values, and overlap prevention. Our goal in this comparison is not to fully critique all these differences, but rather to give a reasonable idea of how our results compare to alignments by a popular program with typical parameters. Thus, we use lastz with its default settings. Since parameter training is significant contribution of our work, we also test lastz with the 1/2-R8 single model parameter set and an x-drop setting that corresponds to 65 bits.

Figure 5.3 contains points for both lastz and lastz with the 1/2-R8 parameter set. We see that with the default parameter set, lastz has far lower sensitivity than FEAST with any x-drop setting. When we replace the default scoring scheme, trained from the HOXD region, with our new 1/2-R8 scoring scheme, the sensitivity increases substantially. This further supports the results of our parameter training, where we found that parameter sets trained from HOXD had the lowest overall number of alignment bases.

We notice that lastz has a higher number of false positives than FEAST with Viterbi extensions, regardless of parameter set choice. Interestingly, it also has higher sensitivity when using the 1/2-R8 parameter set. We hypothesize that the differences are due to other areas such as seeding or overlap extensions. However, we leave comparisons to future work.

Overall FEAST compares favourably to lastz on synthetic human and mouse alignments, and our 1/2-R8 parameter set improves the sensitivity of lastz substantially on synthetic data.

5.2.2 Performance on real alignments

We now re-align our test regions with new, less conservative settings inspired by our tests on synthetic data. Our goal is to evaluate the techniques in FEAST on real data. We study our alignments in several ways: the number of bases aligned, the number of exons aligned, and alignment dot plots. We align sequences with FEAST using forward extensions and

Viterbi extensions, as well as with lastz using its default settings. We include lastz as a reference point for existing work.

We increase our gapped x-drop value to 65, matching the false positive rate of lastz with default settings in Figure 5.3. Although our synthetic tests show that a gapped extension score threshold of 10 bits is sufficient to filter false alignments, on real sequences, with the 2-R8 parameter set, we find that 10 bits still allows some noise through. Thus we use a gapped extension score threshold of 20 bits instead. For the same reason, we use a score threshold of 15 bits on the ungapped extension around the seed. With these thresholds, dot plots have little noise.

As we note in section 5.1, the main problem with the amount of sequence aligned metric is that it does not distinguish between correct and incorrect aligned data. To better appreciate how this metric behaves with different settings and data, we review performance on our synthetic data with the same parameters we apply to real sequence alignments. Table 5.6 shows that for all setting combinations, FEAST is much more sensitive and has fewer false alignments than lastz with default settings. The added columns in lastz are almost all noise.

According to Figure 5.3 one expects FEAST to have higher sensitivity, than lastz with the 1/2-R8 parameter set and an x-drop equivalent to 65 bits. However, because of the increased gapped and ungapped score filters we find that FEAST with forward extensions and the 2-R8 parameter set only matches the sensitivity of lastz in this case. However, lastz still produced more false alignments. This results in a higher number of overall human bases aligned with the extra aligned bases being false positives. Thus, we see the main limitation of this performance metric.

Program	Parameters	PPA	FPR	TPR
FEAST forward	2-R8	49.7	0.26	0.59
FEAST Viterbi	2-R8	37.0	0.19	0.51
FEAST forward	1/2-R8	48.9	0.26	0.58
FEAST Viterbi	1/2-R8	36.4	0.19	0.50
lastz defaults	HOXD70	28.6	0.26	0.35
lastz 1/2-R8	1/2-R8	52.1	0.29	0.59

Table 5.6: Percent of human positions aligned (PPA), false positive rate (FPR), and true positive rate (TPR), over 50 synthetic human/mouse alignments using tuned FEAST parameters. There are a total of 1,216,219 human positions. The PPA metric is a good indicator of sensitivity.

Despite this, we have no other appropriate measure for real data at this time. One

option is to compare to multiple alignments which, according to synthetic results, have a much higher overall accuracy. However, the current UCSC procedure for producing these starts with pairwise alignments to detect initial candidates regions for multiple alignment. Thus, it is still an ultimately unsatisfying method for measuring performance. For simplicity, we continue to use the number of bases aligned as our performance metric in this section.

Table 5.7 gives the performance with this metric over all our training regions, using the same programs and settings as in Table 5.6. We see the same pattern as we do for our synthetic data, although in this case we cannot produce sensitivity and specificity values as we do not know the true alignment. When we limit our study to known exons, all programs align the majority of positions. Although we cannot claim definitively that these results match our synthetic results, the fact that we see the same pattern in the number of bases aligned metric is promising.

Program	Parameters	PPA All	PPA Exons
FEAST forward	2-R8	35.3	88.3
FEAST Viterbi	2-R8	28.7	87.5
FEAST forward	1/2-R8	34.8	88.2
FEAST Viterbi	1/2-R8	28.6	87.3
lastz defaults	HOXD70	28.3	90.0
lastz 1/2-R8	1/2-R8	39.4	91.6

Table 5.7: Percent of all positions aligned (PPA All) and percent of exon position aligned (PPA Exons), over all training regions . There are a total of 5,422,010 human positions and 131,992 known exon positions. FEAST parameters are tuned with results from synthetic tests. Percentages are relative to the total bases for each category.

From Table 5.7 we see that forward extensions align more bases, and if results on synthetic sequences translate to real sequence alignments, we expect many of the newly aligned bases to represent real homology. To further understand the differences in alignments, we examine dot plots for our test data. A dot plot shows a dot for every aligned base and thus alignments appear as roughly diagonal lines. We note that most of the newly aligned bases occur on the main diagonal of the dot plot. This does not prove these positions represent real homologies, but it is consistent with what we expect real homologies to look like. Figure 5.4 shows a representative example from training region 8. The vertical blue bars indicate aligned human bases found with FEAST using forward extensions, but missing when using FEAST with Viterbi extensions. We see that in some cases, completely new local alignments are found, while in others, alignments extend further or connect two

alignments based on Viterbi extensions.

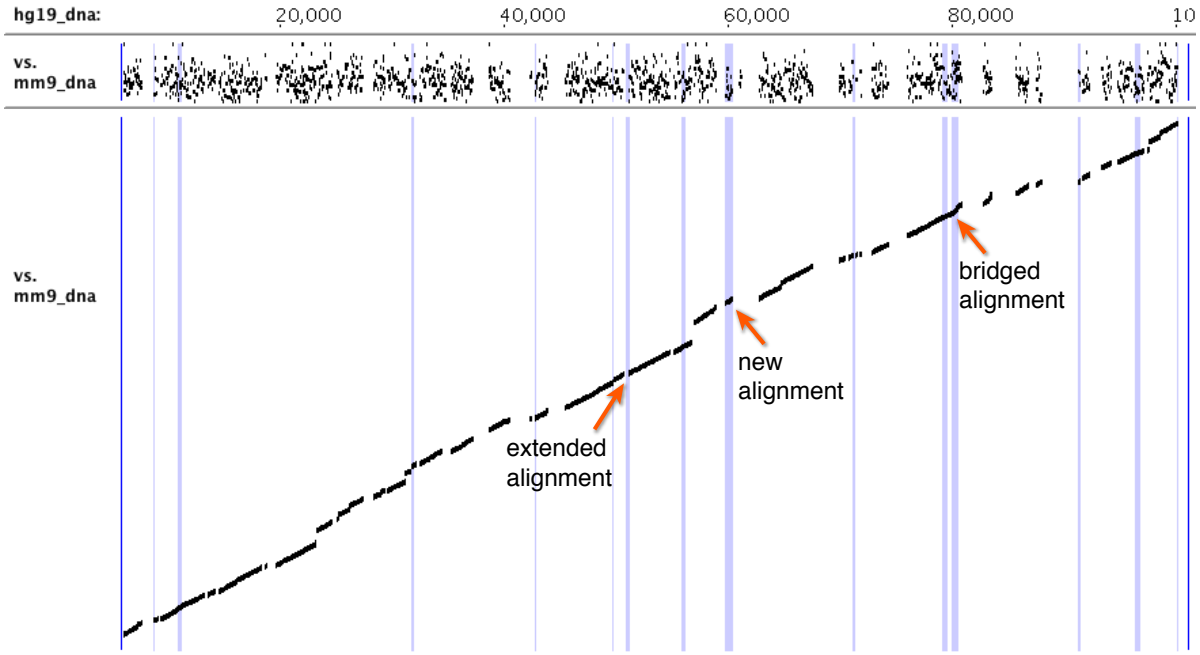

Figure 5.4: Example dot plot for a subset of our alignment of R8. Blue highlighted sections are unique to our forward extension alignments. All other dots are also found with Viterbi extensions.

5.2.3 FEAST's annotation

FEAST provides an annotation for each local alignment that indicates which sub-model the Viterbi path uses for each alignment column. For our two sub-model parameter set, 2-R8, this annotation translates to predictions of weak and strong homology. One possible use for our annotation is to bring immediate attention to alignment sections that may be interesting to researchers. A common use for cross species alignment is to find subsequences that have biological function. This technique, called comparative genomics [78], relies on the fact that functional sections of the genome are less tolerant of change, and thus mutate much slower than the rest of the genome. As a result, if a functional element is shared between two species it is typically more strongly conserved than the surrounding DNA.

We test our annotation with with an experiment based on this idea. We expect that known exons, many of which are shared between human and mouse, have a higher likelihood

of being strongly conserved. Thus, if our annotation is useful, it should often annotate known exons as being strongly conserved.

To test this hypothesis we looked at the number of human exon bases that are aligned and annotated as strongly conserved in FEAST alignments of our training regions. We also look at this same metric for the NSCAN [38] predicted exons in our training regions. We obtain these predictions from UCSC. NSCAN uses a pair-HMM to look for new genes in a target region, given a multiple alignment of that region and one more other species. Thus we expect our annotation to be similarly enriched in human regions predicted by NSCAN to be exons. Finally, we look at the total number of human bases aligned and annotated as strongly conserved. We expect this to be much lower than our other two metrics.

Program	Parameters	PPA All	PPA Exons	PPA NSCAN
FEAST forward	2-R8	5.2	35.7	31.7
FEAST Viterbi	2-R8	5.6	43.0	44.7

Table 5.8: Percent of aligned human positions (PPA) annotated as strongly similar in three categories, over all training regions. There are a total of 5,422,010 human positions, 131,992 known exon positions, and 190,568 NSCAN positions. FEAST parameters are tuned with results from synthetic tests. Percentages are relative to the total bases for each category.

Table 5.8 confirms our hypothesis. Both known exons and NSCAN predicted exons are much more likely to be aligned and annotated as strongly conserved than a randomly chosen human position. Even though FEAST is not designed to find exons or other conserved elements, the annotation it provides may still be useful to researchers.

Table 5.8 also includes values for FEAST with more traditional Viterbi extensions. Unexpectedly, more regions are annotated as highly conserved than when we use forward extensions. A manual inspection of our alignments shows that the additional strongly conserved regions found by Viterbi extensions are also found by forward alignments, but annotated as weakly conserved. This usually occurs in cases where the strongly conserved region ends at the end of a Viterbi extension, but where the corresponding forward extension continues further into weakly conserved sequence. We hypothesize that the extra jump cost required to switch back to the weak sub-model prevents these regions being annotated as strongly conserved in the longer forward extensions. This aspect of FEAST is not a focus of our current study, thus we leave a systematic verification to future work.

5.2.4 Spurious alignment flanks

In 2008, Frith *et al.* [36] point out that local alignments often produce spurious alignment flanks, extra unrelated sequence at the end of an alignment. Because forward extensions are more sensitive than Viterbi extensions, we hypothesize that forward extensions are even more likely to produce spurious alignment flanks. In this section we test our hypothesis by reproducing two experiments of Frith *et al.* using FEAST.

Specifically, we measure spurious alignment flanks by performing local extensions on both synthetic and real sequence. Following the work of Frith *et al.*, we test several typical score based parameter sets by converting them to an equivalent single sub-model pair-HMM parameter set. We describe score based parameter sets in terms of a match and mismatch score, gap open penalty (GOP) and gap extension penalty (GEP). We include three score based schemes: +1/-1 with GOP = 2 and GEP = 1, +2/-3 with GOP = 5 and GEP = 2, and +5/-4 with GOP = 12 and GEP = 4. We also include our new 2-R8 and 1/2-R8 parameter sets. We use a special version of FEAST that assumes a single anchor at the start of each sequence. We disable the ungapped alignment filter and set the x-drop to be so large that the entire dynamic programming space is explored.

Synthetic data

Figure 5.5: A test instance for our synthetic flank test. The first 100 bp in each sequence are identical and the following 200 bp are random. The correct boundary is at the position at 100 bp and we consider any bases beyond the first 100 to be part an over-extension.

Our synthetic test measures an upper bound on the amount of over-extension produced by FEAST with a given parameter set. A test instance consists of a pair 300 bp sequences with a 60% AT background distribution. The first 100 bp of both strings are identical, while the remaining 200 bp are generated independently and randomly. See Figure 5.5 for an example test instance. We consider the first 100 bp to be the homologous section, and the following 200 bp to be the unrelated section. Any extension past the initial 100 bp is thus considered to be spurious. Because the homologous prefix is identical, we never see under-alignment in this test and the flank results represent an upper bound on the true error rate.

	Forward			Viterbi		
Scheme	No over-extension	μ	s	No over-extension	μ	s
+1/-1	0.52	4.15	9.90	0.65	1.00	2.24
+2/-3	0.65	1.18	2.89	0.67	0.86	1.95
+5/-4	0.51	5.19	12.44	0.64	0.97	2.21
2-R8	0.61	1.44	3.73	0.62	1.21	2.83
1/2-R8	0.55	2.67	7.43	0.55	1.96	4.38

Table 5.9: Results of over-extension tests on various scoring schemes for synthetic data. Let x be the number of unrelated bases included in the extension. We list the mean μ, and sample standard deviation s, for the distribution of x, as well as the frequency of seeing no over-extension.

For each parameter set, we perform 10,000 test instances and record the number of unrelated bases from the first sequence included in the extension. We list the mean and sample standard deviation of our data in Table 5.9. The results verify our hypothesis, the forward algorithm is more prone to over-extension than the Viterbi algorithm. However, for most parameter sets, including 2-R8 and 1/2-R8, the increase in over-alignment is relatively small.

Real data

To measure over-extension on real data, Frith *et al.* [36] aligns human nuclear mitochondrial insertions (NUMTs) to mouse, chicken, and fugu mitochondrial DNA. A NUMT is a section of human mitochondrial DNA that is inserted into a position in the normal genomic sequence. Thus, we can be reasonably sure that in an alignment of the genomic sequence to mitochondrial sequence, the NUMT represents truly homologous sequence. Since mitochondrial DNA changes very slowly, human NUMTs will also align against the mitochondria from even distantly related species.

We repeat the experiment here using the same data set. According to the work of Frith *et al.*, their set of NUMTs are derived by aligning human mitochondrial DNA against human genomic sequence with BLAST. For each NUMT, they include 1000 human genomic bases on each side of the NUMT. Alignment into these flanking bases are considered over-extensions. We follow their work and assume that the annotated boundaries of each NUMT are accurate to within 5 bp. In our test, we discard any alignments that include only genomic sequence. Additionally, we omit the large NUMT in chromosome 5 from our test

as it dominates the under-alignment distribution. For this test we use the same version of FEAST and corresponding settings as in our analysis of human mouse alignments.

		All	$\|x\| \leq 5$	$x > 5$		$x < -5$	
Name	Alg.	n	n	n	μ	n	μ
+1/-1	F	86	41	19	26.2	26	-40.5
+1/-1	V	78	38	5	12.0	35	-55.8
+2/-3	F	124	70	9	10.6	45	-42.3
+2/-3	V	130	64	5	9.6	61	-80.2
+5/-4	F	82	32	27	60.1	23	-43.0
+5/-4	V	64	39	5	12.0	20	-39.1
2-R8	F	134	82	14	29.7	38	-36.7
2-R8	V	140	85	9	10.8	46	-65.2
1/2-R8	F	128	77	17	51.3	34	-37.2
1/2-R8	V	130	80	11	16.1	39	-44.3

Table 5.10: Boundary distributions for alignments of human NUMTs against fugu, mouse, and chicken mitochondrial sequence. The alignment deviation x is the number of human bases over (positive) or under (negative) the known boundary. Each alignment has a left and right boundary, n is the number of boundaries in each category, and μ is the mean of x. We compare forward extensions (F) and Viterbi extensions (V) over several parameter sets.

We summarize our boundary statistics in Table 5.10. As with our synthetic test, we observe that forward extensions are more likely to over-extend than Viterbi extensions. However, we also see that forward extension are less likely to under-extend. The 2-R8 parameter set is more likely to get a precise boundary than the 1/2-R8 parameter set. It also finds the most alignments out of all parameter sets. Despite this small trade-off, this experiment shows that forward extensions are not especially more prone to finding false over-alignments than Viterbi extensions. The significant amounts of newly aligned bases in our alignments of human and mouse sequences cannot be attributed to false alignment flanks.

5.3 The limits of alignment

In this section we explore how much mutation we can tolerate before we can no longer align long DNA sequences. Specifically, we are interested in the problem of aligning sequences

that are too long to align without either anchoring or banded alignment.

In the previous chapter, we described a technique for anchoring long genomic sequences that uses the forward algorithm to score potential anchors. We implemented this technique, along with a mixed global greedy anchoring approach, in CAPE, our prototype global aligner. We use this implementation as our example of a sensitive global alignment program.

Local extension as implemented in FEAST may be thought of as a form of banded alignment where the band is allowed to move off the main diagonal as appropriate. In the cases where global alignments are highly similar, local alignment with the x-drop heuristic is capable of finding the correct global alignment. We hypothesize that our forward extensions allow local alignment to be usable even for very weakly similar global alignments.

We explore this idea with tests on synthetic sequences. First, we observe the behaviour of our programs on long synthetic sequences with the correct alignment parameters. Following that we describe experiments which test the limits of alignment when we do *not* know the correct alignment parameters.

5.3.1 Aligning long synthetic sequences with known parameters

In this experiment, we focus on point mutations. In a realistic scenario, we expect both insertions and deletions to increase in addition to point mutations as sequences diverge. However, the exact relationship between the rates of point mutations versus the rates and lengths of indels is not known. Thus, we perform a simpler test and fix the gap open cost to -5 bits and the gap extend cost to -2 bits, the defaults for the "weakly similar sequences" setting at NCBI's BLASTN server [2]. We vary the point mutation rate from 0.5 substitutions per site to 1.2 substitutions per site.

For each rate of mutation, we generate 20 pairs of aligned sequences from the traditional alignment pair-HMM using a 60% G/C background distribution. For each pair, we remove the gaps and align the resulting sequences with FEAST and CAPE. We also include Viterbi versions of both programs. Since we are performing global alignments, we use a special version of FEAST that assumes a single seed hit at the start of the sequence pair. We disable the ungapped alignment filter and use an x-drop of 65 bits and an extension score filter of 20 bits as we did in our tests on human and mouse sequences.

We measure the accuracy of our alignments as the fraction of true homologous bases in our alignment, according to the reference alignment. Because global aligners like CAPE align the entire input, regardless of correct anchoring, specificity is linearly related to sensitivity. Thus, we do not report specificity values.

Subs/Site	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2
FEAST	0.81	0.77	0.70	0.62	0.56	0.48	0.39	0.31
CAPE	0.82	0.77	0.70	0.63	0.54	0.28	0.07	0.03
Viterbi FEAST	0.81	0.68	0.07	0.00	0.00	0.00	0.00	0.00
Viterbi CAPE	0.82	0.77	0.70	0.51	0.15	0.07	0.02	0.01

Table 5.11: Sensitivity of CAPE and FEAST on approximately 20 kb long synthetic sequences using correct alignment parameters.

Table 5.11 shows that FEAST with forward extensions is the most sensitive overall. FEAST with Viterbi extensions is the least sensitive, failing sharply after we pass 0.6 substitutions per site. Both versions of CAPE are much more sensitive than Viterbi FEAST. This supports the traditional view that global alignment is much more sensitive than local alignment. However, our new forward extensions are more sensitive even than CAPE, which also uses the forward algorithm to score fragments.

5.3.2 Aligning long synthetic sequences with unknown parameters

We now relax our experiment assumptions. In a realistic setting, the first time we align new sequence data, we do not know the correct alignment parameters. In order to be able to train parameters, we need at least roughly correct initial alignments. Although our previous test shows that FEAST can align sequences with a great amount of mutation when we know the parameters, it may be that it is not possible to obtain reasonable initial alignments to train those parameters.

In this section we answer two questions: what is the best parameter set, or collection of parameter sets, to use for an initial alignment, and how do the results of the previous sub-section change when we use incorrect parameters.

Choose an initial parameter set

When performing an initial alignment, we must guess at suitable parameters. Since both CAPE and FEAST can support multiple sub-models, we may also consider using a small fixed set of parameter sets for our initial alignment. In a global alignment setting, the overall sensitivity depends mostly on correct anchoring. Without correct anchors, our final alignment cannot possibly be correct. Thus, we answer our first two questions with an

experiment that simulates anchor choosing over a wide range of parameters. We vary the parameters of both the sequence data and the anchor scoring algorithm.

We simplify the anchoring procedure by making the following assumptions. First, we assume that we are producing a global alignment, or in other words, that we are anchoring a pair of completely homologous sequences. Second, given a set of potential anchors, we assume at least one of these is a correct anchor. And finally, while the model parameters for the homologous sequence pair are unknown, we assume they are within a realistic range. Specifically, we define a set of 160 model parameters that cover the parameter space we are interested in. Let this set be

$$Q = \{(d, o, e) | d \in D, o \in O, e \in E\}$$

where D is a set of distances, in substitutions per site,

$$D = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\},$$

O is a set of gap open scores, in bits,

$$O = \{-5, -4, -3, -2\},$$

and E is a set of gap extension scores, in bits,

$$E = \{-2, -1, -0.5, -0.25\}.$$

Our first goal is to find an ordering P of good parameter sets, such that if we only use k parameter sets during fragment scoring in CAPE, the first k items from P will maximize our sensitivity to alignments over the entire range of possible parameters. We form P using a greedy algorithm and the following rank test.

Rank test. Let the inputs to our test be parameter sets $k = (d_1, o_1, e_1)$, $\ell = (d_2, o_2, e_2)$, and a fragment scoring algorithm A. We generate 200 bp homologous sequence pairs (x^1, y^1), (x^2, y^2), $\ldots$, (x^{100}, y^{100}) from 400 column long synthetic alignments generated from parameter set ℓ. To generate a sequence pair given an alignment, we simply take the first 200 non-gap symbols from each sequence in the alignment. We compare each x^i with y^1 through y^{100} using algorithm A with parameter set k and rank the results by score. If the homologous pair (x^i, y^i) ranks first, we count a success. Let $R(k, \ell)$ be the number of successes out of 100 trails.

To greedily form the set P, we define a performance metric based on the above rank test. Given P', a subset of Q, we let $S(P')$ be

$$S(P') = \frac{\sum_{j \in Q} \max_{i \in P'} R(i, j)}{16000}, \tag{5.1}$$

the sensitivity when using the set P' as our set of testing parameters. Here, 16000 is the maximum possible rank test score.

Our greedy approach builds sequence P' by adding a parameter set from Q to P such that we maximize $S(P')$. We note that this algorithm only approximates the anchoring procedure in CAPE. To fully simulate CAPE's anchoring, we would need to merge the scores for all parameter sets in P' and then perform a rank test on the combined results. The amount of time this would take is not practical, thus our greedy procedure makes the assumption that scores of false pairs with any parameter set rarely be higher than the scores of true pairs with a parameter set close to the true values.

After we choose a P', we run a modified rank test that fully simulates our anchoring procedure. Specifically, we merge all the scores for all parameter sets in P', then rank the results. We produce a set P' for both our local segmentation algorithm based on the forward algorithm, as well as a version that uses the Viterbi algorithm. See Table 5.12 for a listing of these sets.

From Figure 5.6 we see that using the forward algorithm for fragment scoring greatly increases sensitivity. The best single parameter set for the forward version of our fragment scoring algorithm is more sensitive than the Viterbi version of our scoring algorithm, no matter how many parameter sets we use.

However, our hypothesis that the results of our rank test with different parameter sets are independent, is wrong. The true sensitivity when we use more than one parameter set is less than the predicted sensitivity assuming that parameter sets do not interfere with one another. This is especially true for the forward algorithm.

From Figure 5.6 we see that when using the Viterbi algorithm, using more parameter sets increases the sensitivity as we hypothesized. After we reach nine parameter sets, however, we see no increase in sensitivity. In fact, we even observe occasional decreases. For the forward algorithm, using more parameter sets can increase sensitivity, but not by a large amount. The results are also much more inconsistent than with the Viterbi algorithm, decreases in sensitivity are more frequent.

Given these results, we conclude that using multiple parameter sets for data exhibiting a constant rate of mutation is useful for the Viterbi algorithm, but not useful for the forward algorithm. The sensitivity of the forward algorithm is very high, even with a single parameter set.

	P' forward			P' Viterbi		
Position	Subs/site	GOP	GEP	Subs/site	GOP	GEP
1	0.7	-3	-1	0.5	-5	-0.5
2	0.8	-5	-0.5	0.2	-4	-0.25
3	0.5	-3	-0.5	0.7	-5	-1
4	1.0	-4	-2	0.4	-5	-0.25
5	0.3	-2	-1	0.5	-5	-1
6	0.6	-4	-0.25	0.1	-2	-2
7	0.5	-2	-2	0.8	-5	-2
8	0.3	-3	-0.25	0.2	-4	-0.5
9	0.9	-4	-1	0.5	-5	-0.25
10	0.9	-3	-2	0.3	-5	-1
11	0.8	-5	-0.25	0.1	-4	-0.25
12	0.2	-2	-0.5	0.2	-3	-2
13	0.6	-2	-1	0.6	-5	-0.5
14	0.7	-4	-0.5	0.5	-5	-2
15	1.0	-5	-2	0.9	-2	-1
16	0.5	-3	-1	0.1	-3	-1
17	0.8	-2	-2	0.3	-4	-2
18	0.9	-5	-0.5	0.6	-5	-1
19	0.9	-5	-1	0.7	-5	-2
20	1.0	-4	-0.5	0.9	-5	-2
21	1.0	-2	-2	0.1	-2	-1
22	0.1	-2	-0.5	0.2	-4	-1
23	0.2	-3	-0.25	0.3	-5	-0.5
24	0.9	-2	-1	0.3	-5	-0.25
25	0.5	-4	-0.5	0.4	-4	-1
26	0.6	-3	-1	0.5	-4	-0.25
27	0.6	-2	-2	0.6	-5	-0.25
28	0.7	-4	-1	0.7	-5	-0.5
29	0.7	-4	-0.25	0.7	-5	-0.25
30	0.8	-4	-2	0.1	-3	-2

Table 5.12: Ordered sets of parameter sets for aligning sequences with unknown parameters. The first column indicates the position within the set.

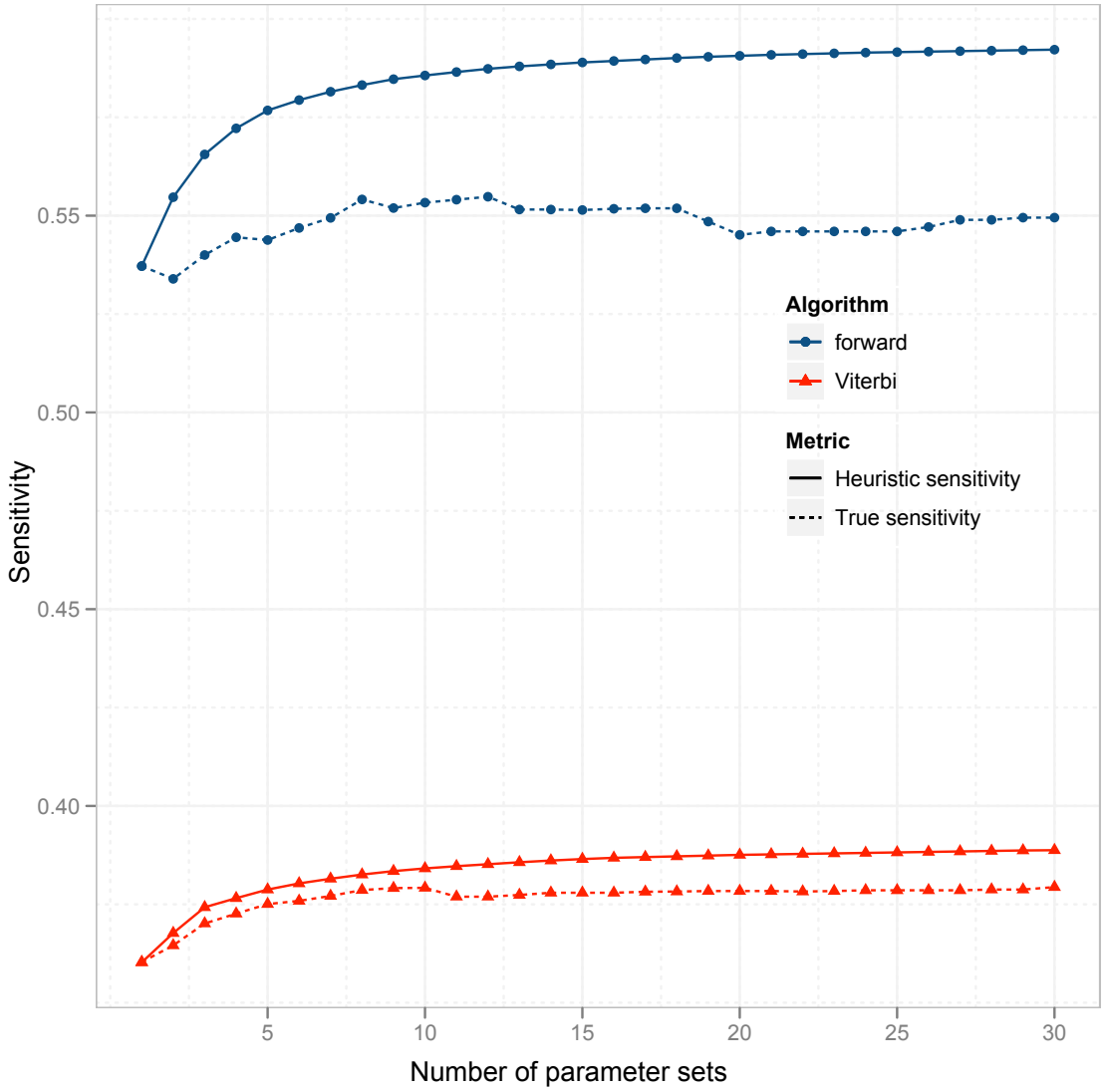

Figure 5.6: Sensitivity of our forward based and Viterbi based local segmentation algorithm. The heuristic sensitivity gives the sensitivity predicted by our greedy parameter set selection algorithm. The true sensitivity is actual sensitivity when using the greedily selected parameter sets together.

Alignments with the best guessed parameter set

We now return to our previous experiment aligning long synthetic sequences. However, instead of using the correct parameter set we now assume that we do not know the underlying parameters. Following the results from our simulation of the anchoring choosing procedure, we use the single parameter set, $(0.7, -3, -1)$ for the forward algorithm and $(0.5, -5, -0.5)$, for Viterbi, which maximized the sensitivity. We hypothesize that we can tolerate less mutation than if we know the correct parameters.

Subs/Site	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2
FEAST	0.82	0.77	0.71	0.64	0.56	0.44	0.02	0.00
CAPE	0.82	0.77	0.71	0.61	0.48	0.11	0.03	0.01
Viterbi FEAST	0.79	0.41	0.00	0.00	0.00	0.00	0.00	0.00
Viterbi CAPE	0.79	0.74	0.68	0.46	0.12	0.03	0.02	0.01

Table 5.13: Sensitivity of CAPE and FEAST on approximately 20 kb long synthetic sequences using incorrect alignment parameters.

From Table 5.13 we see that our hypothesis is correct, although we can still align sequences with a surprising amount of mutation. Despite the fact that our guess parameters are fairly close to the true parameters in this case, the difference is enough to cause FEAST to fail after 1.0 substitutions per site. In contrast, with correct parameters FEAST still achieves a sensitivity of 0.31 even for 1.2 substitutions per site. As before, the forward algorithm continues to provide much better sensitivity than the Viterbi algorithm.

5.4 Summary

In the previous chapter we described two new pair-wise aligners: CAPE, a global aligner, and FEAST, a local aligner. Both algorithms use of the forward algorithm for homology detection rather than the Viterbi algorithm, and use a new more descriptive model of alignment. We also described an expectation maximization (EM) procedure based on FEAST to infer parameters for our new pair-HMM.

In this chapter, we describe a methodology for training alignment parameters for two genomes that makes use of FEAST and our EM parameter inference algorithm. In contrast to previous methods for inferring parameters, our training procedure does not depend on expert knowledge of the input genomes. For example, we do not require a set of pre-annotated genes, nor a set of hand selected training regions. Existing work on parameter

training often uses a heuristic that excludes highly similar, easy to align, regions from training data. In contrast, our approach uses our new model of alignment to naturally find a separation between highly similar regions and weakly similar regions. We apply our procedure to the human and mouse genomes, making only limited use of existing knowledge of these genomes, and arrive at a new set of alignment parameters, 2-R8, that describes highly similar and weakly similar regions. By using only the portion describing weakly similar alignments, we also obtain an new parameter set suitable for existing alignment algorithms. Our experiments show that for detecting truly homologous bases, the parameter set 2-R8 is much more sensitive than the parameter set of Chiaromonte, Yap, and Miller [25] (HOXD70) in alignments of intergenic sequence. The HOXD70 parameter set is widely used, and thus our training results represent a contribution with wide applicability.

In tests on synthetic human and mouse data with FEAST, we find that forward extensions are much more accurate than Viterbi extensions. For a fixed specificity, forward extensions have much better sensitivity. Similarly, for a fixed sensitivity, forward extensions have much better specificity. The maximum attainable sensitivity for forward extensions is much more than that of Viterbi extensions. Depending on the settings, forward extensions can be both more specific and more sensitive than Viterbi extensions. On this same synthetic data, our new 2-R8 parameter set is much more sensitive than the popular HOXD70 parameter set. Combining 2-R8 with forward extensions, we achieve a sensitivity of 0.59 compared to 0.35 achieved by lastz, a popular local aligner, using default settings and the HOXD70 parameter set.

Our alignments of real data show similar trends to our synthetic results using the simple percent sequence aligned metric. If the synthetic results translate to real data, we expect that many of our newly aligned positions represent new homologies. We also find that the sections of our alignments that are annotated as being strongly similar are much more likely to contain known exons or predicted exons than regions annotated as being weakly similar. Although there are much better programs designed specifically for this task, it shows that the annotation returned by FEAST may highlight interesting sections of alignments and thus be an aid to researchers.

Finally, we explore the limits of alignability using FEAST and CAPE on long synthetic sequences. We find that FEAST can tolerate much more mutation than CAPE before it is unable to recover any alignment. On the other hand, when using Viterbi extensions, FEAST performs extremely poorly compared to CAPE, regardless of whether we score fragments in CAPE using the forward or Viterbi algorithms. This verifies the commonly-held assumption that global alignment is more sensitive than local alignment. However, with the introduction of forward extensions, global alignment no longer provides any advantage to sensitivity, and is much more likely to produce false alignments. For

these reasons, despite being a seemingly small change, forward extensions represent an important advance in pairwise alignment.

Chapter 6

Conclusions

We have presented three new algorithms for the pair-wise sequence alignment problem. In Chapter 3 we present, RDA, an algorithm that aligns sequences in segments where we consider all possible evolutionary histories in each segment. Although this algorithm provides interesting and different results than traditional alignment, it is unclear if there are domains where such results are more useful than a traditional alignment. The RDA algorithm is also very slow with a run time of $O(n^2m^2)$ for sequences of length n and m, compared to $O(nm)$ required for traditional alignment. Our heuristic version of RDA gives an improved runtime of $O(k^2nm)$ where larger k gives more accurate results. However, in practice even this version of the algorithm is slow. Thus, we conclude that RDA is an interesting algorithm, but it is not clear if it has a practical value in sequence alignment.

In Chapter 4 we present two algorithms for pair-wise alignment: CAPE, a global alignment program, and FEAST, a local alignment program. Both CAPE and FEAST share a common improvement, the use of the forward algorithm for homology detection rather than the Viterbi algorithm. We also present a new expectation maximization algorithm based on FEAST for inferring parameters from training sequences.

CAPE is a global aligner that scores fragments with a local homology detection algorithm that uses the forward algorithm to evaluate the nearest 200 bp. Specifically, our algorithm produces a *local segmentation*, dividing a sequence into a related and unrelated part. If we use Viterbi instead of the forward algorithm, our local segmentation becomes a short local alignment. We address sequence variation by allowing each fragment to be scored with more than one set of alignment parameters. We find fragments using a recursive GLASS-like procedure. For each recursive step, we use random spaced seeds and filter the set of hits by the score of an ungapped alignment of the length of the seed.

We choose the seed weight and threshold to minimize the number of false hits given the length and background distribution of the region being anchored. CAPE also uses a mixed global/greedy approach to choosing alignments. If there are many high scoring seed hits, we use use the optimal chain of fragments as anchors. However, if we are in a region with no high scoring hits, we use a greedy approach to choose the single best fragment as an anchor.

FEAST is a local aligner that replaces Viterbi extensions with forward extensions. Forward extensions use the x-drop heuristic to constrain the dynamic programming matrix, but instead of finding the optimal path through the constrained space, we sum all possible paths. Since the forward algorithm returns no alignment, our forward extensions drop approximate *anchors* every 100 bases. These anchors do not necessarily lay on the optimal path but can be used to roughly constrain a final alignment step. We note that while our implementation currently uses Viterbi to produce a final alignment, the framework in FEAST allows one to use any alignment algorithm in the final step. One may even skip the final step altogether in cases where an alignment is not needed. For example, in situations where local alignments are used as anchors in a multiple alignment procedure, or when only large scale homologies are needed. We seed our local extensions using a set of six spaced seeds. Three of these are optimized for coding regions and three for intergenic sequence. We filter hits with ungapped alignments in a traditional BLAST-like framework.

Both CAPE and FEAST use a new pair-HMM for alignment. This model describes a fixed number of sub-models, each of which represents traditional alignment under a specific scoring scheme. The alignment is allowed to jump between sub-models, but incurs a probabilistic penalty to do so. In CAPE, this algorithm is used as the final alignment step. In FEAST, it is used as the model of alignment throughout, for ungapped extensions, gapped extensions, and the final alignment. In addition to giving more accurate alignments, when we use this model with Viterbi decoding, we obtain an additional annotation describing which sub-model produced each column of the alignment.

In Chapter 5, we first train new parameters for aligning human and mouse sequences. Then, we propose several hypotheses about FEAST and explore them with experiments on synthetic and real sequence data. Finally, we explore the limits of alignability using FEAST and CAPE. To train parameters for human and mouse, we describe a new training methodology that uses FEAST and our EM procedure for parameter inference. In contrast to previous methods for inferring parameters, our training procedure does not depend on expert knowledge of the input genomes. For example, we do not require a set of pre-annotated genes, nor a set of hand-selected training regions. Existing work on parameter training often uses a heuristic that excludes highly similar, easy to align, regions from training data. In contrast, our approach uses our new model of alignment to naturally

find a separation between highly similar regions and weakly similar regions. We apply our procedure to the human and mouse genomes, making only limited use of existing knowledge of these genomes, and arrive at a new set of alignment parameters, 2-R8, that describes highly similar and weakly similar regions. By using only the portion describing weakly-similar alignments, we also obtain an new parameter set, which we call 1/2-R8, suitable for existing alignment algorithms. Our experiments show that 2-R8 is significantly more sensitive than the parameter set of Chiaromonte, Yap, and Miller [25] (HOXD70) in alignments of intergenic sequence. The HOXD70 parameter set is widely used, and thus our training results represent a contribution wide applicability.

In tests on synthetic human and mouse data with FEAST, we find that forward extensions are much more accurate than Viterbi extensions. For a fixed specificity, forward extensions have much better sensitivity. Similarly, for a fixed sensitivity, forward extensions have much better specificity. The maximum attainable sensitivity for forward extensions is much more than that of Viterbi extensions. Depending on the settings, forward extensions can be both more specific and more sensitive than Viterbi extensions. On this same synthetic data, our new 2-R8 parameter set is much more sensitive than the popular HOXD70 parameter set. Combining 2-R8 with forward extensions, we achieve a sensitivity of 0.59 compared to 0.35 achieved by lastz, a popular local aligner, using default settings and the HOXD70 parameter set.

Our alignments of real data show similar trends to our synthetic results using the simple percent sequence aligned metric. If the synthetic results translate to real data, we expect that many of our newly aligned positions represent new homologies. We also find that the sections of our alignments that are annotated as being strongly similar are much more likely to contain known exons or predicted exons than regions annotated as being weakly similar. Although there are much better programs designed specifically for this task, it shows that the annotation returned by FEAST may highlight interesting sections of alignments and thus be an aid to researchers.

Finally, we explore the limits of alignability using FEAST and CAPE on long synthetic sequences. We find that FEAST can tolerate much more mutation than CAPE before it is unable to recover any alignment. On the other hand, when using Viterbi extensions, FEAST performs extremely poorly compared to CAPE, regardless of whether we score fragments in CAPE using the forward or Viterbi algorithms. This verifies the commonly held assumption that global alignment is more sensitive than local alignment. However, with the introduction of forward extensions, global alignment no longer provides any advantage to sensitivity, and is much more likely to produce false alignments. For these reasons, despite being a seemingly small change, forward extensions represent an important advance in pair-wise alignment.

References

[1] Searching the trace archive with discontiguous MegaBlast. `http://www.ncbi.nlm.nih.gov/Web/Newsltr/FallWinter02/blastlab.html`, 2002. 26

[2] BLAST: Basic Local Alignment Search Tool. `http://blast.ncbi.nlm.nih.gov/Blast.cgi`, September, 2009. 84

[3] National Center for Biotechnology Information. `http://www.ncbi.nlm.nih.gov/`, January 2010. 1

[4] The NCBI C++ Toolkit. `http://www.ncbi.nlm.nih.gov/IEB/ToolBox/CPP_DOC/`, January 2010. xi, 67

[5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990. 16, 26, 55

[6] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, 1997. 16, 26, 55

[7] A. Arribas-Gil, D. Metzler, and J. Plouhinec. Statistical alignment with a sequence evolution model allowing rate heterogeneity along the sequence. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(2):281–95, 2009. 24, 31

[8] S. Batzoglou, L. Pachter, J. P. Mesirov, B. Berger, and E. S. Lander. Human and mouse gene structure: comparative analysis and application to exon prediction. *Genome Research*, 10(7):950–958, 2000. 21

[9] L. E. Baum. An inequality and associated maximization technique in statistical estimatino for probabilistic functions of markov processes. *Inequalities*, 3(1–8), 1972. 30

[10] M. Blanchette, W. J. Kent, C. Riemer, L. Elnitski, A. F. A. Smit, K. M. Roskin, R. Baertsch, K. Rosenbloom, H. Clawson, E. D. Green, D. Haussler, and W. Miller. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Research*, 14(4):708–715, 2004. 73

[11] R. K. Bradley, A. Roberts, M. Smoot, S. Juvekar, J. Do, C. Dewey, I. Holmes, and L. Pachter. Fast statistical alignment. *PLoS Computational Biology*, 5(5):e1000392, May 2009. 15, 23, 25, 31, 69

[12] N. Bray, I. Dubchak, and L. Pachter. AVID: A global alignment program. *Genome Research*, 13(1):97–102, 2003. 17, 22

[13] N. Bray and L. Pachter. MAVID: Constrained ancestral alignment of multiple sequences. *Genome Research*, 14:693–699, 2004. 17

[14] B. Brejova, D. G. Brown, and T. Vinar. Optimal spaced seeds for homologous coding regions. *Journal of Bioinformatics and Computational Biology*, 1(4):595–610, 2004. 18, 46

[15] B. Brejova, D. G. Brown, and T. Vinar. Vector seeds: an extension to spaced seeds. *Journal of Computer and System Sciences*, 70(3):364—-380, 2005. 18

[16] B. Brejova, D. G. Brown, and T. Vinar. The most probable annotation problem in HMMs and its application to bioinformatics. *Journal of Computer and System Sciences*, 73(7):1060–1077, 2007. 35

[17] D. G. Brown. Optimizing multiple seeds for protein homology search. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(1):29–38, 2005. 18

[18] D. G. Brown and A. K. Hudek. New algorithms for multiple DNA sequence alignment. In *Proceedings of WABI 2003 LNCS*, volume 3240, pages 314–324, 2004. 23

[19] M. Brudno, M. Chapman, B. Gottgens, S. Batzoglou, and B. Morgenstern. Fast and sensitive multiple alignment of large genomic sequences. *BMC Bioinformatics*, 4:66, 2003. 18, 22

[20] M. Brudno, C. B. Do, G. M. Cooper, M. F. Kim, E. Davydov, NISC Comparative Sequencing Program, E. D. Green, A. Sidow, and S. Batzoglou. LAGAN and Multi-LAGAN: Efficient tools for large-scale multiple alignment of genomic DNA. *Genome Research*, 13:721–731, 2003. 22

[21] J. Buhler, U. Keich, and Y. Sun. Designing seeds for similarity search in genomic DNA. *Journal of Computer and System Sciences*, 70(3):342–363, 2005. 18

[22] A. Califano and I. Rigoutsos. Flash: a fast look-up algorithm for string homology. *Proc Int Conf Intell Syst Mol Biol*, 1:56–64, 1993. 17

[23] R. A. Cartwright. Problems and solutions for estimating indel rates and length distributions. *Molecular Biology and Evolution*, 26(2):473–80, Feb 2009. 5, 26

[24] W. Chen and W. Sung. On half gapped seed. *Genome Informatics*, 14:176–185, 2003. 18

[25] F. Chiaromonte, V. B. Yap, and W. Miller. Scoring pairwise genomic sequence alignments. *Pacific Symposium on Biocomputing*, pages 115–126, 2002. 30, 59, 63, 64, 66, 72, 76, 91, 95

[26] J. Choi, H. Cho, and S. Kim. GAME: a simple and efficient whole genome alignment method using maximal exact match filtering. *Computational Biology and Chemistry*, 29(3):244–253, 2005. 17, 28

[27] K. P. Choi, F. Zeng, and L. Zhang. Good spaced seeds for homology search. *Bioinformatics*, 20(7):1053–1059, 2004. 18

[28] The ENCODE Project Consortium. The ENCODE (ENCyclopedia Of DNA Elements) Project. *Science*, 306(5696):636–640, 2004. 43

[29] C. Darwin. *On the Origin of Species by Means of Natural Selection*. John Murray, London, 1859. 1

[30] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999. 17, 21

[31] A. L. Delcher, A. Phillippy, J. Carlton, and S. L. Salzberg. Fast algorithms for large-scale genome alignment and comparison. *Nucleic Acids Research*, 30(11):2478–2483, 2002. 17, 21

[32] C. B. Do, M. S. P. Mahabhashyam, M. Brudno, and S. Batzoglou. ProbCons: Probabilistic consistency-based multiple sequence alignment. *Genome Research*, 15:330–340, 2006. 24

[33] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis*. Cambridge University Press, 1998. 6, 7, 10, 12, 30, 47, 52, 53, 60

[34] D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano. Sparse dynamic prgramming II: Convex and concave cost functions. *Journal of the Association for Computing Machinery*, 39(3):546–567, 1992. 21

[35] P. Fariselli, P. L. Martelli, and R. Casadio. A new decoding algorithm for hidden Markov models improves the prediction of the topology of all-beta membrane proteins. *BMC Bioinformatics*, 6 Suppl 4:S12, Dec 2005. 14

[36] M. C. Frith, Y. Park, S. L. Sheetlin, and J. L. Spouge. The whole alignment and nothing but the alignment: the problem of spurious alignment flanks. *Nucleic Acids Research*, 36(18):5863–71, Oct 2008. 81, 82

[37] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, 1982. 4

[38] S. S Gross and M. R. Brent. Using multiple alignments to improve gene prediction. *Journal of Computational Biology*, 13(2):379–93, Mar 2006. 80

[39] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, 1997. 17

[40] R. S. Harris. *Improved Pairwise Alignment of Genomic DNA*. PhD thesis, Pennsylvania State University, 2007. 27, 31, 72

[41] M. Hasegawa, H. Kishino, and T. Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution*, 22(2):160–174, 1985. 49

[42] S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci U S A*, 89(22):10915–9, Nov 1992. 30

[43] I. Holmes and R. Durbin. Dynamic programming alignment accuracy. *Journal of Computational Biology*, 5(3):493–504, 1998. 14

[44] W. Huang, D. M. Umbach, and L. Li. Accurate anchoring alignment of divergent sequences. *Bioinformatics*, 2005. 23

[45] A. K. Hudek. New anchoring techniques for global multiple alignment of genomic sequences. Master's thesis, University of Waterloo, September 2004. 23

[46] L. Ilie and S. Ilie. Multiple spaced seeds for homology search. *Bioinformatics*, 23(22):2969–2977, 2007. 18

[47] U. Keich, M. Li, B. Ma, and J. Tromp. On spaced seeds for similarity search. *Discrete Applied Mathematics*, 138:253–263, 2004. 18, 46

[48] W. J. Kent. BLAT–the BLAST-like alignment tool. *Genome Research*, 12(4):656–664, 2002. 18

[49] W. J. Kent and A. M. Zahler. Conservation, regulation, synteny, and introns in a large-scale C. briggsae-C. elegans genomic alignment. *Genome Research*, 10(8):1115–25, Aug 2000. 17, 24, 27

[50] M. Li, B. Ma, D. Kisman, and J. Tromp. PatternHunter II: highly sensitive and fast homology search. *Journal of Bioinformatics and Computational Biology*, 2(3):417–439, 2004. 46

[51] M. Li, B. Ma, and L. Zhang. Superiority and complexity of the spaced seeds. In *Proceedings of the seventeenth annual ACM-SIAM symposium on discrete algorithms*, pages 444–453, Miami, Florida, 2006. ACM Press. 46

[52] D. J. Lipman and W. R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–41, Mar 1985. 16

[53] R. A. Lippert, X. Zhao, L. Florea, C. Mobarry, and S. Istrail. Finding anchors for genomic sequence comparison. *Journal of Computational Biology*, 12(6):762–776, 2005. 17

[54] G. Lunter, A. Rocco, N. Mimouni, A. Heger, A. Caldeira, and J. Hein. Uncertainty in homology inferences: assessing and improving genomic sequence alignment. *Genome Research*, 18(2):298–309, Feb 2008. 15, 24

[55] B. Ma, J. Tromp, and M. Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18(3):440–445, 2002. 17, 46

[56] D. Mak, Y. Gelfand, and G. Benson. Indel seeds for homology search. *Bioinformatics*, 22(14):e341–e349, 2006. 19

[57] M. Michael, C. Dieterich, and J. Stoye. Suboptimal local alignments across multiple scoring schemes. In Inge Jonassen and Junhyong Kim, editors, *WABI*, volume 3240 of *Lecture Notes in Computer Science*, pages 99–110. Springer, 2004. 24, 52

[58] S. Miyazawa. A reliable sequence alignment method based on probabilities of residue correspondences. *Protein Engineering*, 8(10):999–1009, Oct 1995. 14

[59] B. Morgenstern. DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15(3):211–218, 1999. 25

[60] B. Morgenstern, A. Dress, and T. Werner. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proceedings of the National Acadamy of Sciences USA*, 93:12098–12103, 1996. 25

[61] B. Morgenstern, K. Frech, A. Dress, and T. Werner. DIALIGN: finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290–294, 1998. 25

[62] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970. 4

[63] L. Noe and G. Kucherov. Improved hit criteria for DNA local alignment. *BMC Bioinformatics*, 5:149, Oct 2004. 29

[64] L. Noe and G. Kucherov. YASS: enhancing the sensitivity of DNA similarity search. *Nucleic Acids Research*, 33(Web Server issue):W540–3, Jul 2005. 18, 29

[65] A. Y. Ogurtsov, M. A. Roytberg, S. A. Shabalina, and A. S. Kondrashov. OWEN: aligning long collinear regions of genomes. *Bioinformatics*, 18(12):1703–1704, 2002. 21

[66] L. Pachter and B. Sturmfels. Parametric inference for biological sequence analysis. *Proceedings of the National Academy of Sciences of the USA*, 101(46):16138–16143, 2004. 31

[67] B. Paten, J. Herrero, K. Beal, S. Fitzgerald, and E. Birney. Enredo and Pecan: genome-wide mammalian consistency-based multiple alignment with paralogs. *Genome Research*, 18(11):1814–1828, 2008. 23, 24

[68] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0. 67

[69] M. A. Roytberg, A. Y. Ogurtsov, S. A. Shabalina, and A. S. Kondrashov. A hierarchical approach to aligning collinear regions of genomes. *Bioinformatics*, 18(12):1673–80, Dec 2002. 21

[70] A. Schultz, M. Zhang, T. Leitner, C. Kuiken, B. Korber, B. Morgenstern, and M. Stanke. A jumping profile Hidden Markov Model and applications to recombination sites in HIV and HCV genomes. *BMC Bioinformatics*, 7:265, 2006. 24

[71] A. S. Schwartz. *Posterior Decoding Methods for Optimization and Accuracy Control of Multiple Alignments*. PhD thesis, University of California, Berkeley, 2007. 15

[72] A. S. Schwartz and L. Pachter. Multiple alignment by sequence annealing. *Bioinformatics*, 23(2):e24–9, Jan 2007. 15

[73] S. Schwartz, W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. C. Hardison, D. Haussler, and W. Miller. Human-mouse alignments with BLASTZ. *Genome Research*, 13(1):103–107, 2003. 24, 27, 76

[74] R. Siddharthan. Sigma: multiple alignment of weakly-conserved non-coding DNA sequence. *BMC Bioinformatics*, 7:143, 2006. 25

[75] G. S. Slater and E. Birney. Automated generation of heuristics for biological sequence comparison. *BMC Bioinformatics*, 6:31, 2005. 23, 29

[76] A. R. Subramanian, J. Weyer-Menkhoff, M. Kaufmann, and B. Morgenstern. DIALIGN-T: an improved algorithm for segment-based multiple sequence alignment. *BMC Bioinformatics*, 6:66, 2005. 25

[77] Y. Sun and J. Buhler. Designing multiple simultaneous seeds for DNA similarity search. *Journal of Computational Biology*, 12(6):847–861, 2005. 18, 58, 65

[78] A. Ureta-Vidal, L. Ettwiller, and E. Birney. Comparative genomics: genome-wide analysis in metazoan eukaryotes. *Nat Rev Genet*, 4(4):251–62, Apr 2003. 79

[79] J. Wang, P. D. Keightley, and T. Johnson. MCALIGN2: Faster, accurate global pairwise alignment of non-coding DNA sequences based on explicit models of indel evolution. *BMC Bioinformatics*, 7:292, 2006. 24

[80] J. Xu, D. Brown, M. Li, and B. Ma. Optimizing multiple spaced seeds for homology search. *Journal of Computational Biology*, 13(7):1355–1368, 2006. 18