Finding Your Match: Techniques for Improving Sequence Alignment in DNA and RNA

Ofer Hirsch Gill

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

May 2006

Bud Bhubaneswar Mishra

"You cannot determine peoples' destinies, they find it out for themselves."

- *Kevin Sorbo*

Dedicated to my family and friends, who never stopped believing in me.

Acknowledgements

I would like to thank my advisor, Bud Bhubaneswar Mishra, for his continuous patience during my studies, as well as my thesis review board members Laxmi Parida, Narendra Ramakrishnan, and Avi Goldstein, for their comments and advice. Further thanks go to fellow labmates Toto Paxia, Raoul Sam-Daruwalla, Yi Joey Zhou, Bing Sun, and Venkatesh Mysore for their help and collaboration throughout the research process. I would not have been able to carry out this research without all of you.

Abstract

In Bioinformatics, finding correlations between species allows us the better understand the important biological functions of those species and trace its evolution. This thesis considers sequence alignment, a method for obtaining these correlations. We improve upon similar sequence alignment tools with PLAINS, an algorithm than uses piecewise-linear gap functions and parameter-optimization to obtain correlations in remotely-related species pairs such as human and fugu using reasonable amounts of memory and space on an ordinary computer. We also explore SEPA, a tool that uses p-value estimation based on exhaustive empirical data to better emphasize key results from an alignment with a measure of reliability. Using SEPA to measure the quality of an alignment, we proceed to compare PLAINS against similar alignment tools, emphaisizing the interesting correlations caught in the process.

Contents

Dedication **v**

Acknowledgements **vi**

Abstract **vii**

List of Figures **x**

List of Tables **xv**

List of Appendices **xvi**

1 Introduction **1**

2 Literature Survey **5**

2.1 Alignment Overview 5

2.2 p-Value Methods 13

3 Overview **15**

4 Plains **16**

4.1 The Plains Alignment Method 16

4.2 Plains Log Approximation and Parameter Optimization 20

5 SEPA **24**

5.1 Obtaining High-Scoring Strips from an Alignment 26

5.2 Methods: Analyzing Segment Pairs 28

6 Colorgrid and DNA Results **34**

6.1 The Plains ColorGrid Method 34

6.2 Empirical Results 35

7 Conclusions and Open Problems **42**

Appendices **44**

A.1 Proof for the $O(np)$ Space Bound 45

A.2 Details for the Maximum Criterion Selection 47

B.1 Segment Pair Analysis in Further Detail 51

B.2 Sequence Details 57

Bibliography **59**

List of Figures

4.1 The Piecewise-Linear Gap Functions that PLAINS came up with in optimizing the score for different species pairs, along with the rescaled gap-paramters the other tools use. Note that the LAGAN gap paramters shown here are its default paramters. LAGAN uses a number of unspecified gap parameters in aligning on a species by species basis. 23

5.1 Shown above are the mean length-to-score ratio and mean segment scores observed in the strips from aligning randomly generated DNA sequences. In the plots shown above, a unique line is plotted corresponding to each value of n in the thousand lengths ranging from 1000 to 8000. For these plots, x represents the m value divided by 1000, and y represents the mean observed for that particular m and n, and the left plots illustrate mean length-to-score ratio for the segment pairs, while the right plots illustrate mean segment pair scores. 30

5.2 Shown here is a plot of segment scores to frequency for randomly generated sequences using our assumption that segment score is length-independent. The x axis represents segment score, and the y axis represents frequency. The tail of this plot is an exponential distribution of form $P(S = x) = Ke^{-\lambda x}$, where we have approximated $K = 8.69 \times 10^{-2}$ and $\lambda = 3.26 \times 10^{-2}$. This curve is at its highest when $x = 30$, and by emprical observation, we have noticed that strips scoring less than 30 are generally unimportant portions of an alignment. 31

5.3 From our alignments over the randomly generated sequences, after adjusting the number of segments r and the total score t for length-dependent average and deviation behavior, we chose to plot the frequency of observing certain r and t values. The figure shown here is a surface plot of this, where lighter spots indicate higher frequencies. From it, we observe that the majority of the data is concentrated in one area. This area approximates to $e^{c}e^{-a_t T^2 + b_t T + c_t}e^{-a_r R^2 + b_r R + c_r}$, where $c = -183.90, a_t = 10.1, b_t = 9070, c_t = -2.04 \times 10^6, a_r = 0.241, b_r = 4.71, c_r = -27.5$. 33

6.1 In this figure, we observe the unadjusted r and t values produced by PLAINS, LAGAN, and EMBOSS from the human-mouse.3 – 9 experiment where we vary the ρ variable used to filter our segment pairs. On each curve, we observed the t and r values of each tool when varying ρ over various values from 0.1 till 0.9. Recall from table 6.1 that PLAINS performed poorly in terms of ζ' values for $\rho = 0.5$ for the human-mouse.3 – 9 experiments. However, note from this plot that for any fixed r where PLAINS is comparable to a different tool, PLAINS receives the highest t value, and therefore if we designed SEPA using a fixed r value over all alignment tools, then PLAINS would have the highest t value, and hence the highest ζ' value (i.e., the best result). Many other experiments from table 6.1 have a similar plot to this one. 38

6.2 Match Ratio Color Lines in the HFOrtho2 test for PLAINS and EMBOSS. Here, the Human and Fugu sequence used have six exon regions that correspond to each other (though not necessarily in order, as exon region 2 in the Fugu sequence corresponds to exon region 3 in Human sequences for example). Here, both PLAINS and EMBOSS correctly identify the correlation of exon region 2 in Fugu with exon region 3 in Human, but only PLAINS identifies the correlation of exon region 5 in Fugu with exon region 5 in Human. 40

6.3 Match Ratio Color Lines in the HumanPseudo5 test for PLAINS and LAGAN. Here, the Human sequence has 8 exon regions that are similar to areas of the pseudosequence used, and alignments of PLAINS and LAGAN for these cases are similar, even by eyeglance of the ColorGrids. Note that although PLAINS and LAGAN catch most of these regions in their alignments, we're only counting the exon regions that participated in "good" segments according to SEPA. With this in mind, PLAINS and LAGAN both identify exon region 4 as important, but PLAINS also deems exon regions 6 and 7 in the Human sequence as important, which LAGAN misses. 41

B.1 Shown above are the mean and variance plots for the segment pair length-to-score ratio from aligning randomly generated DNA sequences. A unique line is plotted corresponding to each value of n in the thousand lengths ranging from 1000 to 8000. For these figures, and others that follow, x represents the m value divided by 1000, and y represents the mean or variance value obtained for that particular m and n. 53

B.2 Shown here are the mean and variance plots for segment scores from aligning randomly generated DNA sequences. 54

B.3 Shown here are the mean and variance plots for r, the number of segment pairs obtained from aligning randomly generated DNA sequences. 55

B.4 The plots shown here are the mean and deviation plots for t, the total score of all segment pairs from aligning randomly generated DNA sequences. Because the variance plot was difficult to quantify in terms of m and n, we instead model the deviation for total score in terms of d and i, where $i = \min(m, n)$ and $d = \|m - n\|$. The lower figure shows the deviation plot, with each curve corresponding to a unique d value, and the x-axis reprsenting i in units of thousands. 56

List of Tables

6.1 Shown here for PLAINS, EMBOSS, and LAGAN are the r, t, and ζ' values obtained from aligning genomic DNA sequences of lengths between 0.5 Kb and 12 Kb within human, mouse, dog, and fugu, where the pairs are biologically related and mainly non-coding DNA with expected large gaps and low homology regions. 36

B.1 Sequence Details for the Biologically Related Alignments Ran. All the sequences are retrieved from ENSEMBL database [www.ensembl.org]. 58

List of Appendices

Appendix A 44

Proofs for Non-Trivial Portions of PLAINS

Appendix B 51

SEPA Details

Chapter 1

Introduction

Since biological sequences like DNA, RNA, and amino acid sequences, did not arise *ab initio*, but share a common ancestry and similar selection constraints, a key focus in bioinformatics has been to enhance our ability to compare large number of these sequences against each other. An effort of this kind can ultimately catalogue elements that are conserved, motifs that are repeated, regions that are hyper-mutated or deleted, and segments that are inserted and reinserted over and over. This process starts with aligning two or more sequences with an algorithm that optimizes an alignment score, and often ends with organizing a set of sequences in a global tree structure where the tree-distances roughly correspond to the evolutionary distances. Both the score and distance functions are determined by the underlying stochastic processes modeling genome evolution, and must be represented in a flexible manner in order to be faithful to biology. But this sort of generality often implies a loss of computational efficiency. This dilemma is resolved through reliance on simple algorithms, quasi-local cost functions (e.g., linear gap penalty), and by applying these algorithms only on short subsequences after most unlikely candidates have been discarded.

To a rough approximation, DNA sequence alignment problem differs marginally from protein sequence alignment problem. (For instance, at a superficial level, one may note that DNA alignment is over an alphabet of 4 letters whereas protein alignment is over an alphabet of 20 letters). However, two key differences are that (1) there are 3 bp DNA code per amino acid, and that (2) genes in DNA sequences that ultimately get transcripted and translated into proteins can be separated by intergenic regions of few thousands of base pairs that do not get expressed, and perhaps, are subject to strikingly different (or no) selection constraints. Thus these intergenic regions typically vary to a greater extent in one species compared to another. Therefore, we may expect the gap lengths in DNA alignments to be larger, more variable, and have specie-specific distributions. Moreover, these distributions characterizing the gap-lengths may not be memory-less (i.e., exponential distributions). There have been suggestions that power-law distributions may be more appropriate. The evolutionary processes governing the genomes of species, and the log-likelihood of certain indel gaps occurring when comparing one species against another suggest that a logarithmic gap function is more appropriate for DNA sequences. Because of this, the traditional affine (or linear) gap functions used for aligning proteins are unsatisfactory for DNA sequences, as the ultimate results may be biologically misleading.

In order to exploit the fidelity of general non-linear gap functions for DNA sequences, without suffering performance penalites associated with them, we have chosen to use piecewise-linear gap functions modeled to approximate the gap functions in a dynamic programming approach. Here, we present an implementation of an alignment algorithm, PLAINS, that uses reasonable amount of memory, avoids a major shortcoming associated with generalized gap penal-

ties, and only demands a loss of constant factor (of ≤ 5.6) in time complexity compared to the best algorithm using an affine-gap model. There have been other algorithms that also proposed piece-wise linear gap model (see Miller-Myers [13]), but PLAINS presents several additional theoretical innovations in terms of worst-case upper-bound memory usage, alignment optimization, and visualization of data. We have PLAINS available in a powerful bioinformatic environment, called *Valis*. Our algorithm uses an innovative learning-heuristic to determine the best score function and near-optimal gap-penalty model.

In addition, to draw our attention very quickly to the most pertinent similar subsequences, it is necessary to compare the important areas of alignments and rank them in order of their relevance. For instance, by comparing alignments in related sequences to those of unrelated sequences with no common biological function, we may derive, for any alignment, the probability that its important areas occur by mere coincidence. This probability measure is also known as a p-value, and low p-values relate to high relevance rank.

Many p-value estimation techniques have been suggested and examined previously, for instance, Karlin-Altschul [9] and Siegmund-Yakir [21], but none have proven completely satisfactory. Hence, we discuss SEPA (Segment Evaluator for Pairwise Alignments), which focuses on using empirical results in its p-value approximation. We will emphasize alignments typically dealt with in PLAINS, which is those of noncoding nucleotide sequences of lengths varying from .5 Kb to 12 Kb, with expected large gaps and low similarities.

We will demonstrate how SEPA selects and scores important segments pairs. Furthermore, for random sequences, we also empirically characterize how various alignment statistics, such as the segment pair lengths, scores, and magnitudes, distribute as a function of sequence lengths. From this analysis, the

parameters for a p-value approximation are estimated, and used to demonstrate the method of sensitivity in distinguishing important homologies from unimportant chance occurrences of subalignments within sequences. Furthermore, SEPA is non-subjective, since it can easily be applied to any alignment tool. We will illustrate this advantage by using it to compare the results of PLAINS with LAGAN, EMBOSS. Because of these strengths and despite its empirical foundation, SEPA fulfills a practical computational need by speeding up the core search processes in comparative genomics.

As we hope to demonstrate here by an extensive set of experimental results, PLAINS works satisfactorily for DNA sequences, and can better reveal the underlying biological significances than other existing algorithms (e.g., needle, swat, emboss, etc.). As a concrete example, we present our alignment results for the genomic sequences of a pair of orthologous genes in Human and Fugu. While all the alternative alignment algorithms either fail by mis-aligning the exons in the Fugu sequence, or by not identifying important correlations, PLAINS is able to recover the orthologous relation between exons in the Fugu and Human sequences with good reliability. (See Fig. 6.2)

Chapter 2

Literature Survey

2.1 Alignment Overview

2.1.1 Dynamic Programming Intro

Suppose there are two strings to be aligned denoted as X and Y, and their respective lengths are m and n with $m \geq n$. We can generate an alignment for X and Y by maximizing $V(m, n)$, where $V(\cdot, \cdot)$ is a two-dimensional scoring function such that $V(i, j)$ denotes the best score for aligning $X[1 : i]$ with $Y[1 : j]$ (i.e., characters 1 thru i in X, and characters 1 thru j in Y). For now, we will assume we score m_a for each match, m_s for each mismatch, and w_c for each gapped character (corresponding to an insertion or deletion in transforming X to Y), where m_a is a reward and everything else is a penalty. Also, suppose $s(c, d)$ yields m_a when $c = d$, and $-m_s$ when $c \neq d$.

We hence compute $V(m, n)$ as follows:

$$V(0,0) \quad = \quad 0$$

$$
\begin{aligned}
V(i,0) &= w_c \cdot i \\
V(0,j) &= w_c \cdot j \\
V(i,j) &= \max\{V(i-1,j-1) + s(X[i],Y[j]), V(i-1,j) - w_c, V(i,j-1) - w_c\}
\end{aligned}
$$

The $V(i,0)$ case corresponds to $X[1:i]$ being aligned against a gap. The $V(0,j)$ case corresponds to $Y[1:j]$ being aligned against a gap. In the $V(i,j)$ recursion, the $V(i-1,j-1) + s(X[i],Y[j])$ case corresponds to $X[i]$ aligned to $Y[j]$, $V(i-1,j) - w_c$ corresponds to $X[i]$ being aligned to a gap, and $V(i,j-1) - w_c$ corresponds to $Y[j]$ being aligned to a gap. Computing $V(m,n)$ takes $O(mn)$ time, and requires $O(mn)$ space. We can then backtrace from $V(m,n)$ to obtain the alignment.

2.1.2 Smith-Watermann Formula

The Smith-Waterman formula is similar to the previous formula, except that instead of representing a gap of length i with penalty $w_c \cdot i$, we'd like to represent it with penalty $w_o + w_c \cdot i$. That is, we wish to include a w_o penalty for opening a gap, which is typically much larger than the w_c penalty for extending a gap. This encourages fewer gaps, but allows for a gap to be more easily extended once it already exists.

To assist the computation of $V(\cdot,\cdot)$, we will use functions $E(\cdot,\cdot)$, $F(\cdot,\cdot)$, and $G(\cdot,\cdot)$, where $E(i,j)$ is the score for aligning $X[1:i]$ with $Y[1:j]$ when we end with the "solid" character at the end of Y's suffix aligned against a gap, $F(i,j)$ is the score for aligning $X[1:i]$ against $Y[1:j]$ when we end with the "solid" character at the end of X's suffix aligned against a gap, and $G(i,j)$ is the score for aligning $X[1:i]$ against $Y[1:j]$ when we end with the "solid" characters at

the end of the suffixes of X and Y aligned against each other (and they can be matched or mismatched). Then:

$$\begin{aligned}
V(0,0) &= 0 \\
V(i,0) &= E(i,0) = -w_o - i \cdot w_c \\
V(0,j) &= F(0,j) = -w_o - j \cdot w_c \\
V(i,j) &= \max\{E(i,j), F(i,j), G(i,j)\} \\
G(i,j) &= V(i-1,j-1) + s(X[i], Y[j]) \\
E(i,j) &= \max\{E(i,j-1) - w_c, V(i,j-1) - w_c - w_o\} \\
F(i,j) &= \max\{F(i-1,j) - w_c, V(i-1,j) - w_c - w_o\}
\end{aligned}$$

For $E(i,j)$, we get two cases. The $E(i,j-1) - w_c$ case corresponds to continuing a gap that is already existing. The $V(i,j-1) - w_c - w_o$ case corresponds to opening up a brand new gap. A similar idea goes for the cases in $F(i,j)$. Computing $V(m,n)$ and our overall alignments takes $O(mn)$ time and $O(mn)$ space.

2.1.3 Hirschberg Table Space Reduction

This Hirschberg space-optimal approach, in addition to using X and Y to compute tables V, E, F, and G, also uses X^r and Y^r (the reversed strings of X and Y) to compute tables V^r, G^r, E^r, and F^r.[1] We save only the t most recently

[1]Here, $V^r(i,j)$ denotes the score for the first i entries of X^r and the first j entries of Y^r—in other words, the last i entries of X and the last j entries of Y. And, $E^r(i,j)$, $F^r(i,j)$, and $G^r(i,j)$ behave similar to $E(i,j)$, $F(i,j)$, and $G(i,j)$ over the first i entries of X^r and the first j entries of Y^r.

computed columns of V, E, F, G, V^r, E^r, F^r, and G^r, where t is some fixed constant.

In this manner by computing V, E, F, G for $X[1..m/2]$ and $Y[1..n]$, and V^r, E^r, F^r, G^r for $X^r[1..m/2]$ and $Y^r[1..n]$ (which are really $X[(m/2)+1...m]$ and $Y[1..n]$), we can use a "maximum criteria" to obtain a "middle" subalignment from the saved portions of V, E, F, G, and V^r, E^r, F^r, G^r. Let $gr(k)$ denote $V(m/2,k)+V^r(m/2,n-k)$. Note that $V(m,n)=\max_k[gr(k)]$, so our "maximum criteria" is to select a k such that $gr(k)$ is maximized. We then trace the $V(m/2,k)$ solution t columns until $V(m/2-t,l)$; where $l \leq k$, saving the alignment portion M_l obtained thus far. We also trace the $V^r(m/2,n-k)$ table t columns until $V^r(m/2-t,r)$, where $r \leq n-k$, saving the reversed alignment portion M_r obtain thus far. We glue M_l and M_r to make our middle alignment M. We proceed recursively to align $X[1:m/2-t]$ with $Y[1:l]$ to obtain the left alignment L, and recursively over $X[m/2+t:m]$ and $Y[n-r:n]$ to right the right alignment R. We then glue together L and M and R to get our alignment A of $X[1:m]$ with $Y[1:n]$.

If $T(m,n)$ represents the runtime being performed by the Hirschberg process, then we can quantify it as:

$$\begin{aligned} T(1,1) &= 1 \\ T(m,n) &= O(mn)+T(m/2,n-k)+T(m/2,k) \end{aligned}$$

Where k may or may not be constant. From this, we see that $T(m,n) = O(mn)$. Hence, this Hirschberg method of using $O(n)$ space for the tables to get an alignment increases overall runtime by at most a constant factor compared to

the intuitive $O(mn)$ space method of saving all columns of all tables. Hence, the overall runtime for Smith-Waterman in creating an alignment with Hirschberg reduction is $O(mn)$, but the space used is reduced from $O(mn)$ to $O(n)$.

2.1.4 Needleman-Wunsch Formula

Next, suppose that we wish to assign a gap of length i a penalty $w(i)$, where $w(\cdot)$ is some arbitrary mathematical function (hence, $w(i)$ need not necessarily be $w_o + i \cdot w_c$). The Needleman-Wunsch formula answers this issue as follows:

$$\begin{aligned}
V(0,0) &= 0; \\
V(i,0) &= E(i,0) = -w(i), \\
V(0,j) &= F(0,j) = -w(j); \\
V(i,j) &= \max\{E(i,j), F(i,j)G(i,j)\}, \\
G(i,j) &= V(i-1,j-1) + s(X[i], Y[j]), \\
E(i,j) &= \max_{0 \le k \le j-1} [V(i,k) - w(j-k)], \\
F(i,j) &= \max_{0 \le k \le i-1} [V(k,j) - w(i-k)].
\end{aligned}$$

Because the slope of increase for gap penalty from $w(i)$ to $w(i+1)$ is not contsant in Needleman-Wunsch like it was with Smith-Watermann, computing $E(i,j)$ requires us to inspect $V(i,k)$ for all $k < j$ in order to fairly compute the best alignment for X and Y, and similarly for $F(i,j)$. This means at each cell in our tables, we make $O(m+n)$ lookups to previously computed values. Therefore, computing $V(m,n)$ takes $O(mn \cdot (m+n)) = O(m^2 n)$ time, and $O(mn)$ space[2]. Needleman-Wunsch has the flexibility to model any gap function, but

[2]The $O(m+n)$ lookups to previous rows and columns implies we can't take advantage

the runtime and memory usage is too large.

2.1.5 Miller-Meyers Linked-List Assistance

Miller and Myers [13] uses the same formula as Needleman-Wunsch, but they assume $w(\cdot)$ is convex (meaning that is has a negative double-derivative). This lets them take advantage of a Linked-List Assistance technique to cut down on runtime. This technique involves considering possible solutions for the $E(\cdot,\cdot)$ and $F(\cdot,\cdot)$ entries before we explicitly compute them. To gain an intuition into this technique, first suppose that j is fixed in order to keep the discussion simple for the moment. Next, let $eval(k,i) = V(k) - w(i-k)$, and let $cand_k(i)$ denote the k' value, where $k' \leq k$, such that $eval(k',i)$ is maximized, and let $cand(i)$ denote the k' value, where $k' < i$, such that $eval(k',i)$ is maximized. (Note that $cand_{i-1}(i) = cand(i)$.)

Then, on the i'th iteration, with $i' < i$, once we figure out what $V(i')$ is, we can simply take $k' = cand_{i'-1}(i)$, and compare $eval(k',i)$ with $eval(i',i)$, and whichever of these two values is greater dictates $cand_{i'}(i)$. When $i' = i-1$, this gives us $cand(i)$, and thus on the ith iteration, we know $F(i)$ (and subsequently $V(i)$) in $O(1)$ time without needing to look backwards at previous $V(\cdot)$ entries. Next, note that:

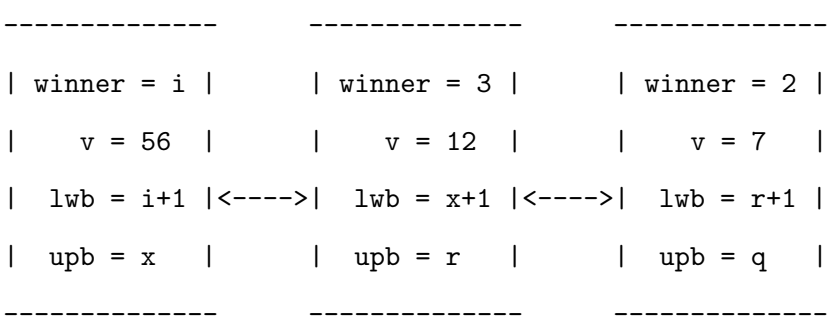
- (S1) If by the kth iteration of our algorithm, we know that, for some a, b, q and for all i' in $[a,b]$, $q = cand_k(i')$. Then we can represent this fact with one data structure, instead of $b-a+1$ of them.

- (S2) In all practical cases, our gap function w is convex (meaning that

of the Hirschberg reduction to cut down on space, unless we can make certain assumptions about the gap function $w(\cdot)$.

$w(i)$ increases as i gets larger, but the rate of increase itself decreases as i gets larger). In this situation, we know that if for some $i' > i$, $eval(i, i') < eval(cand(i'), i')$, then for all $i'' > i'$, we also know that $eval(i, i'') < eval(cand(i''), i'')$. Therefore, if at the end of the ith iteration, we were to scan the $cand_i(\cdot)$ values in the order: $cand_i(i{+}1), cand_i(i{+}2), \ldots, cand_i(m)$, then we would see that the $cand_i(\cdot)$ entries are nonincreasing (each next $cand_i(\cdot)$ entry is either smaller or equal to the previous one).

From these facts, we can coalesce adjacent indices with the same $cand_i(\cdot)$ values into a single group. We can maintain one element per group in a data structure. Each group can be represented by a single element. This element will contain the $winner = cand_i(\cdot)$ value for all indices represented by the group, as well the value $v = V(winner)$, and the leftmost and rightmost indices of the group, *lwb* and *upb*. The elements will be listed in order from leftmost to rightmost indices in this list L. Clearly, we will have to add or delete elements from L to correspond to groups being split off or merged when we go from the ith iteration to the $(i+1)$th iteration. See example below:



Furthermore, from (S2), we know that on the ith iteration, if $cand_i(i{+}1) \neq i$, then for all $i' > i$, $cand_i(i') \neq i$. Supposing that there exists an a such that for

all $\tilde{i}$ such that $i{+}1 \leq \tilde{i} \leq a$, $cand_i(\tilde{i}) = i$, and $cand_i(a{+}1) \neq i$, then for all $i' > a$, $cand_i(i') \neq i$. Hence, on the ith iteration, we can proceed on the elements of L from left to right to find the rightmost value a such that $cand_i(a) = i$, delete any element of previous winner entries, and add in a single leftmost element to L with i as its winner, and lwb and upb set accordingly.

In the case that on the ith iteration, there is an element in L is such that $cand_i(lwb) = i$, but $cand_i(upb) \neq i$ and we need to know which index within the group is the largest a such that $cand_i(a) = i$, then, we simply take the element's previous winner value of k, its v value, and consider for x the plots of $v - w(x - k)$ versus $V(i) - w(x - i)$. We seek the point where the first plot intersects the second one. A binary search over these curves takes $O(\log m)$ time to find the intersection. From this, we get the largest a such that $cand_i(a) = i$, with $lwb \leq a \leq upb$, and then update the elements of L accordingly.

Note that if for some i, we inspect q elements of L, then we must delete the left $q - 1$ elements of L. Also, the number of elements in L is $O(m)$. With this in mind, the number of elements we inspect in L over all iterations of i is $O(m)$. When we combine this with the binary intersection, we hence use list L to obtain solutions for $F(\cdot)$ in $O(m \log m)$ time.

For now, suppose that we are to return to our two-dimensional computational model, but use it in the manner outlined here. PLAINS computes entries to the V, E, F, and G tables column by column. For each row j, we compute $F(\cdot, j)$ with the help of a list L_j (so we maintain lists $L_0, L_1, \ldots, L_n$ and each list is updated in the manner explained earlier), and for each column i, we compute $E(i, \cdot)$ using the help of a list R (which gets updated in a manner similar to that of L for the F entries, except that when we finish computing a column of our table, we empty R so it can be reused when proceeding to the next column).

Clearly, the updates for R and each L_j list are interweaved.

This implies $O(mn \log m)$ time complexity, and further implies that all lists combined take up $O(mn)$ space, since each L_j list uses $O(m)$ space and the R list uses $O(n)$ space. Hence, the Miller-Meyers Linked-List Assistance uses the same space as Needleman-Wunsch, but uses up less time.

2.2 p-Value Methods

2.2.1 Karlin-Altschul

The Karlin-Altschul approach in [9] is to p-value Estimation is motivated by the desire to identify the biological relevance of a generated alignment instead of just creating an arbitrary alignment with a set of "good" segments. Their methods provide a way to approximate reliability without requiring excessive biological information from our two sequences X and Y.

Their method works on gapless local alignments as follows: Suppose for each letter i that p_i is the probability of observing letter i in sequence X, and for each letter j that p'_j is the probability of observing letter j in sequence Y, and that the score for pairing letter i with j is s_{ij}. We may suppose that for a random pair of sequences, the expected alignment score $\Sigma_{i,j} p_i p'_j s_{ij}$ is negative; and nonetheless, it is possible to generate a positive score. Also, suppose each high-scoring segment is found independently of each other.

Then, for some $\lambda > 0$, we have that $\Sigma_{i,j} p_i p'_j e^{\lambda s_{ij}} = 1$, and the average strip score is $\frac{\ln mn}{\lambda}$. We also have that for some constant K, the probability of a strip having score S' after adjusting for average score is $P(x = S') = Ke^{-\lambda S'}$. And since $S' = S - \frac{\ln mn}{\lambda}$, we get that $P(x = S) = Ke^{-\lambda S'} = Ke^{-lambda(S - \frac{\ln mn}{\lambda})} =$

$Kmne^{-\lambda S}$

From this, we can Poisson-approximate $P(x \leq S)$ as $exp(-P(x = S))$, and hence, our p-value of $P(x \geq S)$, which is the probability of finding one or more strip of score at least S, becomes:

$P(x \geq S) = 1 - exp(-P(x = S)) = 1 - exp(-Kmne^{-\lambda S})$

2.2.2 Multiple Karlin-Altschul

Building on the previous p-value formula, Karlin-Altschul[10] approximate the probability of getting r or more strips of score at least S as:

$P(x_r \geq S) = 1 - exp(-P(x = S))\Sigma_{k=0}^{r-1}\frac{P(x=S)^k}{k!} = 1 - exp(-Kmne^{-\lambda S})\Sigma_{k=0}^{r-1}\frac{Kmne^{-k\lambda S}}{k!}$

Furthermore, if there are r strips, and the ith strip has score S_i, then let the adjusted score $S_i' = \lambda S_i - \ln(Kmn)$.

If m and n are large, we can approximate the joint density function for $S_1', S_2', \ldots, s_r'$ as:

$f(x_1, \ldots, x_r) = exp(-e^{-x_r} - \sigma_{k=1}^{r} x_k)$

And hence, if $T_r = S_1' + S_2' + \ldots + S_r'$, then for large m and n, the probability density function for T_r approaches:

$f(t) = \frac{e^{-t}}{r!(r-2)!}\int_0^\infty y^{r-2} exp(-e^{(y-t)/r})dy$

And hence, to find the probability of T_r exceeding some x value becomes:

$P(T_r \geq x) = \int_x^\infty f(t)dt \approx \frac{e^{-x}x^{r-1}}{r!(r-1)!}$

Chapter 3

Overview

The remainder of this paper is structured as follows: Chapter 4 overviews how PLAINS aligns, and describes how PLAINS does its "log function to piecewise-linear function" approximation and parameter optimization. Chapter 5 describes how SEPA works over alignments, and how its p-value approximation is derived. Chapter 6 describes Colorgrid scheme used over alignments, as well as the empirical results found from comparing PLAINS to similar algorithms using SEPA. The final section concludes with a discussion of possible future extensions. The appendix gives the specific sequences used for the tests ran on PLAINS and similar alignment tools, as well as describes the PLAINS alignment method in detail, including proofs of its space-bound and correctness. It also gives further details for the derivation of the p-value scheme used in SEPA.

Chapter 4

Plains

We now explain several aspects of PLAINS: its general notations, its alignment method, and how it approximates a log function using a piecewise-linear function, decides what a "best alignment" is, and optimizes parameters for alignments.

4.1 The PLAINS Alignment Method

We will denote a p-part piecewise-linear function as $ww(\cdot)$. This function has a y-intercept of w_o, and the slopes of the linear functions in successive intervals are $wc_1, wc_2, \ldots, wc_p$, and the x-values at which one interval ends and the next begins are denoted $k_1, k_2, \ldots, k_{p-1}$, and k_u is the x-value where the u_{th} linear function of slope wc_u ends. Also, assume that $k_0 = 0$, and that the p_{th} function of slope wc_p never ends (i.e., extends off into infinity). Then, for some value i such that $k_{\tilde{p}-1} < i \leq k_{\tilde{p}}$, $ww(i)$ is defined as:

$$ww(i) = wo + [wc_{\tilde{p}}(i - k_{\tilde{p}-1})] + \Sigma_{u=1}^{\tilde{p}-1}[wc_u(k_u - k_{u-1})].$$

This p-part piecewise-linear function $ww(\cdot)$ can be modeled to emulate any general gap function $w(\cdot)$. For practical purposes, we will assume $w_o \geq wc_1 \geq wc_2 \geq \ldots \geq wc_p$, which makes $ww(\cdot)$ a convex function. Also, it is sufficient to set p to be at most 10. With our reward per match m_a fixed at 1, and a mismatch penalty m_s, and our piecewise-linear gap penalty function $ww(\cdot)$ substituted for $w(\cdot)$, PLAINS generates an alignment similar to Miller-Meyers [13], which is valid since PLAINS uses convex piecewise-linear gap functions. However, because PLAINS exclusively uses piecewise-linear gap functions, it is able to run faster and use less memory. It uses $O(mnlogp)$ time and $O(np)$ space.

4.1.1 Modified Linked-List Assistance

PLAINS uses the same Linked-List Assistance technique mentioned in Chpater 2.1.5, except that because it is exclusely using convex piecewise-linear gap functions, it is able to do two things differently.

First, instead of considering the plots of $v - w(x-k)$ versus $V(i) - w(x-i)$, we instead consider the plots of $v - ww(x-k)$ versus $V(i) - ww(x-i)$. What this now means is that we now seek the point where the first plot intersects the second one, where both plots are p-part piecewise-linear curves. We find this intersection by binary searching over the lines of these curves, instead of binary searching over the points on this curve, and this takes $O(\log p)$ time instead of the $O(\log m)$ time mentioned in Chapter 2.1.5. This reduces the overall runtime from $O(mn \log m)$ to $O(mn \log p)$.

Second, because we're using p-part piecewise-linear convex functions, each list L used has $O(p)$ elements in it at any point in time. An explicit proof of this

is deferred to appendix A.1. This means all of list our lists combined take up $O(np)$ space. In Chapter 4.1.2, we show how it is possible to use $O(n)$ space in the dynamic programming tables and still obtain the correct alignment. Using these two space reductions together means PLAINS uses $O(np)$ space.

4.1.2 Table Space Reduction

The Table-Space Reduction technique used in PLAINS is similar to that of the Hirschberg reduction mentioned in Chapter 2.1.3, except that the "maximum criteria" used is different.

In the case of PLAINS, the use of Linked-Lists of form L_j to assist in computing $F(\cdot)$ turns out to come in useful for Table-Space reduction. Let $cand_k^L(i,j)$ denote the $cand_k(i)$ derived from list L_j, and let $eval_j(k,i)$ denote $V(k,j) - ww(i-k)$ (essentially, $eval_j(k,i)$ is the two-dimensional version of $eval(k,i)$, and we're using similar notation to the previous section as well as Chapter 2.1.5). And let $gr(k)$ denote $V(m/2,k) + V^r(m/2,n-k)$. Also, let $er(k,k')$ denote $V^r(m-k',n-k) + eval_k(cand_{m/2}^L(k',k),k')$.

When p is 1, $V(m,n) = \max_k[gr(k)]$, as mentioned earlier. Therefore, when $p = 1$, it is satisfactory to select our "maximum criteria" to select a k such that $gr(k)$ is maximized, then use $V(m/2,k)$ and $V^r(m/2,n-k)$ to obtain two subalignments from the saved columns of V and V^r based on this. Then, we glue these subalignments to make a "middle" subalignment (and this is essentially the subalignment that uses middle bits of X).

When $p > 1$, each L_j list is computed assuming the indices i can range from 0 to m, not 0 to $m/2$ (even though that may be all we need for the V table). Then, by the end of computing the V table, we will use L_j while computing

the V^r table and the L_j^r lists[1] to obtain $rc(j)$ values[2] for each j, denoting an endpoint for X against a gap that uses row j of our tables. If we let $er(k, k')$ denote $V^r(m-k', n-k)+eval_k(cand^L_{m/2}(k', k), k')$, then our "maximum criteria" becomes to select a k that maximizes

$$k^* = \arg\max_k \{gr(k), er(k, rc(k))\}.$$

For our chosen k, in the event that $er(k)$ is larger, then we know our optimal alignment uses X against a gap, with this gap starting in the first half of X and ending in the second half of X, and we have $cand^L_{m/2}(rc(k), k)$ and $rc(k)$ the right and left endpoints to this gap, and we will use this to construct a subalignment with this gap, and use whatever we saved of V and V^r to obtain any additional subalignment parts for characters left and right of this gap. All of this combined gives us our "middle" subalignment.

Similarly, for our chosen k, in the event that $gr(k)$ is larger, then we know our optimal alignment does not involve X against a gap with this gap starting in the first half of X and ending in the second half of X. Therefore, we can simply trace the subalignments from the appropriate points in the V and V^r tables the same way we would for the $p = 1$ case to get our "middle" subalignment.

The proof of correctness for our selection of k in this manner is deferred to the appendix. With this in mind, saving $rc(j)$ for all j while computing the tables allows us to align using $O(n)$ space from the tables and $O(np)$ space from the Linked-Lists, which means we use $O(np)$ space overall.

[1]Note that while computing V^r, we are only going to read entries from L_j, not make any changes to it.

[2]Formally, $rc(j)$ is the i value in range $[m/2, m]$ such that $er(j, i)$ is maximized. A deeper intuition for $rc(j)$ is explained in the next section.

4.2 PLAINS Log Approximation and Parameter Optimization

Recall our definition for a p-part piecewise-linear gap function $ww(\cdot)$. For a given piecewise-linear gap function and mismatch penalty, the PLAINS algorithm does find the best alignment for X and Y. When a user asks PLAINS to find the "best set" of gap-mismatch parameters that yield a "best alignment," PLAINS optimizes over four variables: α, β, d, and ms. The penalty for each mismatch is denoted ms, as in the previous section. If the gaps follow a power-law distribution, then the best gap penalty function, determined by the log-likelihood, follows a log gap function. We have found that such gap functions give the best alignments. Since piecewise-linear functions can be modeled to resemble convex general functions (with some controllable degree of accuracy), the PLAINS optimization models piecewise-linear functions to approximate the continuous logarithmic function. In the extreme case, where $p = 1$, such a piecewise-linear function will assume an affine function (corresponding to an exponential distribution for gap lengths). Hence, it retains the generality for a wide class of distributions.

More specifically, the log gap penalty function over i is denoted as[3]: $\alpha \ln(i+1) + \beta$. For a given d, α, and β, $ww(\cdot)$ uses k_1, ..., k_p values set to d, $2d$, ..., $p * d$, and for each u from 0 to p, $ww(k_u) = \alpha \log(k_u + 1) + \beta$, and from this, we can calculate the slope wc_u for each uth line[4], and wo is set to β.

[3]Note: ln is $\log_e$ using base e, and $\ln(i+1)$ is used instead of $\ln(i)$, with the result that the function takes the value $= \beta$ for $i = 0$.

[4]Note that for the pth line, wc_p is computed assuming that $k_p = p * d$, even though k_p is later assumed to be infinity, and the pth line of the piecewise-linear function is assumed to

Computational exploration reveals that varying any of ms, α, β, and d results in different alignments. Each alignment is given a score "adaptively" (i.e., the score given to each alignment is not the same score found in the dynamic programming table) in a way explained in the chapter 5, and among this collection of alignments, the one with the highest score is considered "the best."

One can envision the gap/match-mismatch parameters (α, β, d, ms) as a vector v, and its corresponding score as a scalar $= f(v)$, where f maps each vector to its corresponding ratio score. So, for a given vector v', we can find $f(v')$ by performing an alignment using parameters specified by v'. Hence, the problem PLAINS works over now becomes one of finding a vector v to maximize $f(v)$, which is a numerical optimization problem.

At the user's request, PLAINS can find the v to optimize $f(v)$ using either Simulated Annealing or Genetic Algorithm. Both are explained in [6]. Empirical runs over PLAINS have shown that Simulated Annealing yields better results, but Genetic Algorithm explores the space of v more thoroughly. However, all of this should come of no surprise, since (1) Monte Carlo related methods are successful in optimizing Hidden Markov Models (which are similar to sequence alignments), and (2) Genetic Algorithms typically consider subsequent solutions in a more random manner than Simulated Annealing. PLAINS is designed so that any algorithm to optimize gap/match-mismatch parameters can easily be plugged in instead of these two methods; for instance, one may search parameters with a somewhat time consuming MCMC approach, or variants such as Gibbs sampler or EM.

We have chosen to use SEPA, explained in Chapter 5 to compare the results

continue off into infinity.

of PLAINS against the similar alignment tools LAGAN, EMBOSS, and LALIGN. We made PLAINS optimize the approximate best gap/mismatch parameters based on the pair of species aligned, and the nature of the sequence. This is resemblant of LAGAN's techniques to account for the nature of certain species in performing its alignments.[5] In contrast, EMBOSS and LALIGN each use a fixed set of gap/mismatch parameters for all species. We present a figure 4.2 showing the piecewise-linear gap functions that PLAINS came up with for each species pair[6]. A comparison of the alignment results from PLAINS and the other alignment tools can be found in Chapter 6.

PLAINS can easily align a pair of sequences, each with nucleotides of up to $8Kb$. It can either (1) seek the best gap-mismatch parameters for a given pair of sequences and align with those parameters, or (2) use a user-specified set of gap-match parameters to align the pair of sequences. In (1), the runtime typically ranges from 30 minutes to 2 hours. In (2), the runtime typically ranges from 10 seconds to 1 minute. Plains can either be used via commandline, or as part of the Valis tool set.

[5]LAGAN has special gap-parameters for Human and Mouse, but not Fugu. For the runs using Human and Fugu sequences, the parameters where LAGAN got the best results was used.

[6]Note that all gap parameters are normalized by dividing by the reward-per-match value of an alignment tool. This is done in order to fairly compare one tool to another. Also, for the same reason, all scores reported in this paper are also divided by a tool's reward-per-match value m_a.

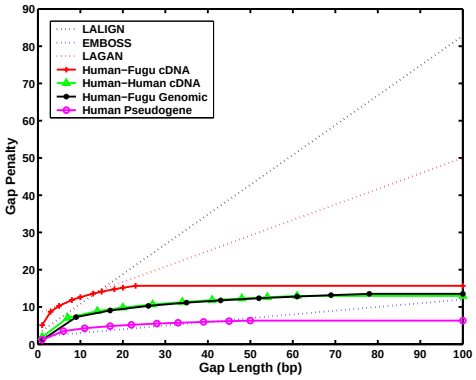


Figure 4.1: The Piecewise-Linear Gap Functions that PLAINS came up with in optimizing the score for different species pairs, along with the rescaled gap-paramters the other tools use. Note that the LAGAN gap paramters shown here are its default paramters. LAGAN uses a number of unspecified gap parameters in aligning on a species by species basis.

Chapter 5

SEPA

Recalling the notation used in Chapter 4, assume the sequences to be aligned are X and Y, and their respective lengths are m and n. Let us suppose that aligning X and Y with some arbitrary alignment tool produces an alignment A of length a, where $m \leq a \leq m+n$. We will represent an alignment A as follows: For each i, $A[i]$ denotes the i^{th} position in alignment A, and it is represented as a pair of index coordinates (u, v) taken from X and Y, and this corresponds to $X[u]$ and $Y[v]$ being aligned to each other at position i in A if $u > 0$ and $v > 0$, or one of $X[u]$ or $Y[v]$ being aligned against a gap if either $v \leq 0$ or $u \leq 0$.

Next, let $A[i:j]$ denote the portion of alignment $A[i], A[i+1], \ldots, A[j]$. We will refer to $A[i:j]$ as a *strip* or *segment* from position i to position j.

Reintroducting more notation from Chapter 4, let $ww(i)$ denote the penalty for a gap of length i. $ww(\cdot)$ can be any arbitrary function, but for this paper, we will assume it is a p-part piecewise-linear function where each successive slope is smaller than the previous one. A more specific version of this score-function is where $p = 1$, which is the affine function used in the Smith-Watermann algorithm.

With this, let $S(i, j)$ denote the score for strip $A[i : j]$ where the score is computed by adding following values: m_a is a score for each match, m_s is the penalty for each mismatch, and $ww(\cdot)$ is used to penalize the gaps. To compute $S(i, j)$ from $A[i : j]$, each match and mismatch within it is added or deducted from the score individually, while each region of X against a gap and Y against a gap is penalized as a whole using $ww(\cdot)$ based on the length of that region. Please note that $S(i, j)$ is computed after $A[i : j]$ is already found, which contrasts the scoring method mentioned in Chapter 4, where a score is computed in the dynamic table and used to generate an alignment.

Suppose we have a scheme that marks r non-overlapping strips as important. Suppose that the endpoints for these strips are denoted as $(i_1, j_1), (i_2, j_2), \ldots, (i_r, j_r)$. For each k, we wish to measure in some way how strip $A[i_k : j_k]$ provides a meaningful correlation between X and Y. One common mathematical approach is to, given a certain null hypothesis, compute the p-value of $Pr(x \geq s)$ where $s = S(i_k, j_k)$. This p-value is known as the coincidental probability of obtaining a strip with score at least s. For this paper, we will assume the null-hypothesis is the behavior of important strips taken from pairwise-aligning randomly generated DNA sequences. Also, if the total scores of all strips is $t = \Sigma_{k=1}^{r} S(i_k, j_k)$, then $\zeta = Pr(x \geq t, y \leq r)$, the probability of obtaining at least a total score of t using at most r strips.

One should note that coincidental probabilities of the segments (both p-values and ζ) are dictated by the scheme used to determine the segments as important. One scheme might deem strip $A[i : j]$ as important, but SEPA might not, and instead SEPA may consider a possibly overlapping strip $A[i' : j']$ as important. As a result, the formula for the p-values and ζ value could differ from one scheme to the other. For instance, in the method used to obtain important

segments mentioned in Karlin-Altschul [9], $Pr(x \geq s) = 1 - exp(Kmne^{-\lambda s})$ holds. However, as argued later in this paper, for the way SEPA obtains the segments from an alignment A, we approximate the p-value as $Pr(x \geq s) = \frac{K}{\lambda}e^{-\lambda s}$.

5.1 Obtaining High-Scoring Strips from an Alignment

Given an alignment A produced from sequences X and Y, we produce important strips as follows: Given fixed constants W and ω, and ρ (where W is an integer, and ω and ρ are real numbers in the range $[0, 1]$), let W denote the window size to be used, ω denote the value used to prevent portions of A of lowest match percentage from becoming considered as important strips, and ρ denote the value used to filter away areas of A that have too low of a p-value. We obtain our segment pairs in the following steps:

(1) For all i from 1 to $a - (W - 1)$, we compute $p_a(i)$, the percentage of entries in $A[i : i + W - 1]$ where a match has occurred. Let μ and σ denote the mean and standard deviation of our $p_a(\cdot)$ values. Next, for each i, we mark[1] $p_a(i)$ values as "special" if they exceed a threshold value of $\mu + \omega\sigma$. Hence, we filter away $A[i : i + W - 1]$ if it fails to meet this threshold value.

(2) For each u and u' (with $u \leq u'$), if $p_a(u)$, $p_a(u + 1)$, ..., $p_a(u')$ are all marked as "special", but $p_a(u - 1)$ and $p_a(u' + 1)$ are not, then we consider the strip $A[u : u' + W - 1]$ as important (i.e., we consider as important the

[1]The choice of using $\mu + \omega\sigma$ as the cutoff value instead of a fixed constant gives us the flexibility of catching important regions in the two sequences, regardless of how homologous they are to each other.

strip starting the leftmost entry repsented by $p_a(u)$, up till the rightmost entry represented by $p_a(u')$).

(3) For each strip $A[i:j]$ deemed important, we trim it so that it starts and ends at a position in the alignment where a match occurred. Thus, if i' is the smallest value such that $i' \geq i$ and $A[i']$ is a match position, and j' is the largest value such that $j' \leq j$ and $A[j']$ is a match position, then we trim strip $A[i:j]$ into strip $A[i':j']$.

(4) Next, we merge together any important strips that overlap. Namely, if we have two strips $A[i:j]$ and $A[k:l]$ such that $i \leq k \leq j$, then we merge these strips into one larger strip $A[i:\max(j,l)]$.

(5) With all strips now representing non-overlapping regions, we then proceed to give each strip $A[i:j]$ its corresponding score $S(i,j)$, as well as its p-value. We delete $A[i:j]$ if its p-value exceed ρ, since that indicates that $A[i:j]$ may be coincidental. We can optionally also collect other information at this point, such as the length of each strip.

(6) The r strips kept at this step are considered the "good" ones. We now compute t, the sum of the scores of the these strips. Using this value, we can compute ζ, coincidental probability for all r strips obtained.

Based on empirical experimentation, setting $W = 50$, $\omega = 0.5$, and $\rho = 0.5$ yields segment pairs that are reasonably long, non-coincidental, and have significantly higher matches than the alignment "background". We reasoned that since our method of obtaining segment pairs differs from that of Karlin-Altschul, then the method for computing p-values for each segment pair cannot build upon their assumptions.

5.2 Methods: Analyzing Segment Pairs

In order to approximate an appropriate p-value estimation for SEPA, we analyzed segment pairs behavior over our assumed null hypothesis of alignments for randomly generated nucleotide sequences. For length values ranging from 1000 bp to 8000 bp, we generated 25 random sequences. We also generated 25 random sequences of length 500 bp. For each combination of these length pairs, we ran all 625 possible pairwise alignments using PLAINS, and analyzed results using SEPA where $\rho = 1$ (to avoid filtering any segments out due to low p-value). The results for mean length-to-score and mean mean segment scores are shown in Fig. 5.1. These plots indicate that, for small n values, the average length-to-score ratio and average score decrease with increasing m. However, asymptotically (for large n) the average length-to-score ratio and average segment scores stay roughly constant in terms of m (at 3.1 and 45 respectively) and don't stray too far. This leads us to infer that length-to-score ratio can be well-approximated by a constant, and that segment scores are independent of m and n. From this, we infer that both average length-to-score ratio and average segment scores are uniform in terms of m and n. In the appendix, Figures B.1 and B.2 elaborate further.

For our random sequences, we also observed the average and variance behaviors for r and t in terms of m and n, where r is the number of segment pairs observed, and t is the total score of all the segment pairs. Furthermore we found that the mean for r, variance for r, and mean for t all scale roughly to $k_0 \ln(k_1 mn + k_2(m+n) + k_3)$, and the deviation for t scales roughly to $\max(k_0, k_1 i \cdot d + k_2 i + k_3 d + k_4)$, where $i = \min(m, n)$, $d = \|m - n\|$, and

k_0, k_1, k_2, k_3, k_4 are constants[2]. Figures B.3 and B.4 in the appendix illustrate further how all of this was derived.

Since the average ratio of segment lengths to score is almost uniform in these plots, it suggests that the gap penalty used to score the strips can be treated as if it is a differently-weighted mismatch. Also, note that the p-values computed with the model studied by Siegmund-Yakir[21] differs mildly from the model using the simplifying assumption that gaps are differently-weighted mismatches. For this reason, it is common for tools to ignore the effects of gaps in generating their p-values, much like BLAST[3]. Thus, we may similarly treat our piecewise-linear gap penalty $ww(\cdot)$ as differently-weighted mismatches in approximating the p-value. Fig. 5.2 shows a plot of segment scores to frequency from which we derive our p-value approximation. Using it, we approximate that $P(x = s) = Ke^{-\lambda s}$, with $K = 8.69 \times 10^{-2}$ and $\lambda = 3.26 \times 10^{-2}$. Our p-value of $P(x \geq s)$ is therefore:

$$P(x \geq s) = \int_s^\infty Ke^{-\lambda x} dx = \frac{K}{\lambda} e^{-\lambda s}$$

And notice that by this construction, $P(x \geq 30) = \frac{K}{\lambda} e^{-30\lambda} \approx 1$. We have designed our p-value estimation this way since strip scores below 30 are empirically observed to be unimportant.

Our next natural step, after obtaining p-values for each segment pair, is to

[2]For average r, $k_0 = 10^3$, $k_1 = 7.95 \times 10^{-10}$, $k_2 = 1.54 \times 10^{-7}$, $k_3 = 1.01$. For variance of r, $k_0 = 10^3$, $k_1 = 1.93 \times 10^{-10}$, $k_2 = 1.97 \times 10^{-7}$, $k_3 = 1.00$. For average t, $k_0 = 10^5$, $k_1 = 4.29 \times 10^{-10}$, $k_2 = 1.33 \times 10^{-8}$, and $k_3 = 1.00$. For deviation of t, $k_0 = 100$, $k_1 = -5.54 \times 10^{-5}$, $k_2 = 4.63 \times 10^{-1}$, $k_3 = 1.04 \times 10^{-2}$, and $k_4 = -65.01$.

[3]The main reason we did not use BLAST in comparing alignment results is because BLAST was unable to align most of the sequences mentioned in table 6.1.

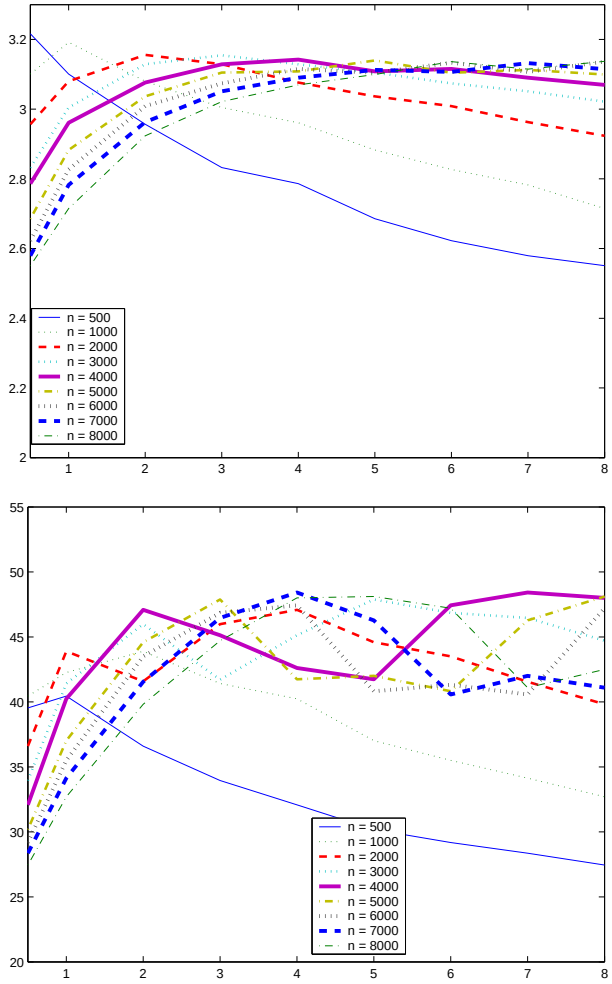


Figure 5.1: Shown above are the mean length-to-score ratio and mean segment scores observed in the strips from aligning randomly generated DNA sequences. In the plots shown above, a unique line is plotted corresponding to each value of n in the thousand lengths ranging from 1000 to 8000. For these plots, x represents the m value divided by 1000, and y represents the mean observed for that particular m and n, and the left plots illustrate mean length-to-score ratio for the segment pairs, while the right plots illustrate mean segment pair scores.

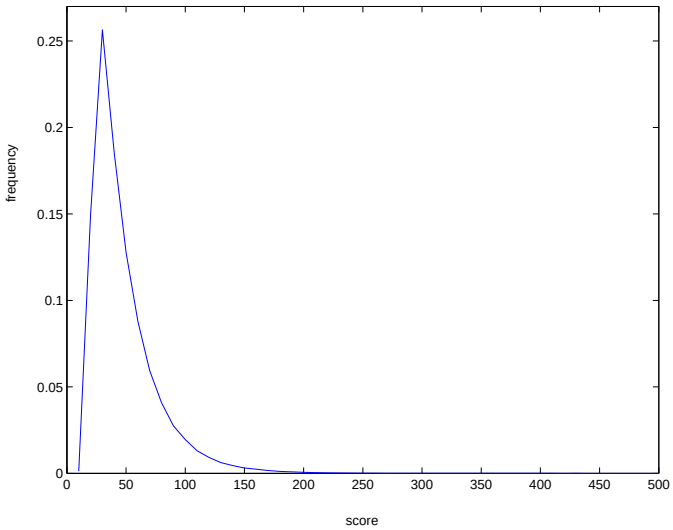


Figure 5.2: Shown here is a plot of segment scores to frequency for randomly generated sequences using our assumption that segment score is length-independent. The x axis represents segment score, and the y axis represents frequency. The tail of this plot is an exponential distribution of form $P(S = x) = Ke^{-\lambda x}$, where we have approximated $K = 8.69 \times 10^{-2}$ and $\lambda = 3.26 \times 10^{-2}$. This curve is at its highest when $x = 30$, and by empirical observation, we have noticed that strips scoring less than 30 are generally unimportant portions of an alignment.

provide a p-value estimate ζ for coincidental probability for the whole alignment, determined by the strips found. As mentioned earlier, we have learned that both r and t depend on sequence lengths m and n. Hence, if R and T are supposed to be the number of segment pairs and the total score of the segment pairs after adjusting for mean and variance based on sequence length, then the coincidental probability $\zeta = P(x \geq T, y \leq R)$. More specifically, ζ is the coincidental probability of seeing a total score of at least T using at most R segment pairs.

Figure 5.3 shows the distribution of r and t values observed from randomly generated sequences after adjusting for mean and variance. From it, we approximate for T and R that $P(x = T, y = R) = e^c e^{-a_t T^2 + b_t T + c_t} e^{-a_r R^2 + b_r R + c_r}$, where $c = -183.90, a_t = 10.1, b_t = 9070, c_t = -2.04 \times 10^6, a_r = 0.241, b_r = 4.71, c_r = -27.5$. This gives us for *zeta* that[4]:

$$\begin{aligned}
\zeta &= P(x \geq T, y \leq R) = \\
&= \int_T^\infty \int_0^R e^c e^{-a_t x^2 + b_t x + c_t} e^{-a_r y^2 + b_r y + c_r} dy dx = \\
&= \frac{\pi e^{c + c_t + c_r + \frac{b_t^2}{4a_t} + \frac{b_r^2}{4a_r}}}{4\sqrt{a_t a_r}} \Big(1 - Erf(\frac{-b_t + 2a_t T}{2\sqrt{a_t}})\Big)\Big(Erf(\frac{-b_r + 2a_r R}{2\sqrt{a_r}}) - Erf(\frac{-b_r}{2\sqrt{a_r}})\Big)
\end{aligned}$$

[4]Note that $Erf(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-x^2} dx$

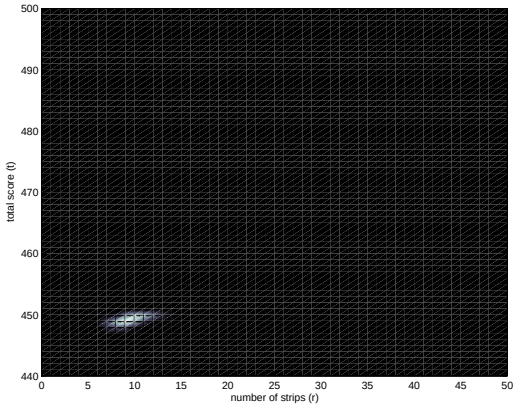


Figure 5.3: From our alignments over the randomly generated sequences, after adjusting the number of segments r and the total score t for length-dependent average and deviation behavior, we chose to plot the frequency of observing certain r and t values. The figure shown here is a surface plot of this, where lighter spots indicate higher frequencies. From it, we observe that the majority of the data is concentrated in one area. This area approximates to $e^{c}e^{-a_tT^2+b_tT+c_t}e^{-a_rR^2+b_rR+c_r}$, where $c = -183.90, a_t = 10.1, b_t = 9070, c_t = -2.04 \times 10^6, a_r = 0.241, b_r = 4.71, c_r = -27.5$.

Chapter 6

Colorgrid and DNA Results

Before describiong the results generated by SEPA and PLAINS, we will describe the Colorgrid method used to visualize the results.

6.1 The PLAINS ColorGrid Method

For visualization of the computed alignments, the PLAINS program ported in Valis uses a coloring grid to summarize high and low matched areas for X found in the alignment. It works as follows: For some M (different from N), we color in a grid with at most M spots. We set color spot 1 based on the match percentage found in $X[1, \ldots, m/M]$ in the alignment; we set spot 2 to a color based on the match percentage found in $X[m/M + 1, \ldots, 2m/M]$ in the alignment; we set spot i to a color based on the match percentage found in $X[(i-1)m/M + 1, \ldots, im/M]$ in the alignment; and so on. The coloring grid for Y works in a similar way. Figures 6.2 and 6.2 are examples of this, with bright colors such as red, orange, yellow, and green signifiying high-match areas, and dark colors such as blue, purple, brown, and black signifying low-

match areas. White signifies any nucleotides of X or Y on the left/right sides that were unaligned.

Notice how here, the number of match percentages found is a fixed size. The color computations in this way has many advantages, such as how it handles the limited resolution of the computer screen compared to the sizes of X and Y.

In addition to visualizing color grids for all of X and Y, users also have the option to view portions of X or Y by specifying a substring range for either X or Y, with the Colorgrid of the unspecified sequence automatically resized to represent the corresponding area in the specified sequence's substring.

6.2 Empirical Results

Furthermore, table 6.1 shows a comparison of alignments for biologically related sequences in terms of unadjusted r and t values, and ζ' values, all using $\rho = 0.5$. Note that $\zeta' = -\ln(\zeta)$. The conversion from ζ to ζ' was carried out for convenience in comparing lab results, where higher ζ' indicates results that are less coincidental. We chose to use $\rho = 0.5$ in all data shown in this table because with it, SEPA successfully filters away all segment pairs when aligning randomly generated DNA sequences, while retaining important segment pairs when aligning biologically related noncoding sequences, even when they have expected high gaps and low similarity regions. Also, please note the loss of precision involved in reporting ζ' values. Hence, if for a paricular alignment, PLAINS and LAGAN receive ζ' values that differ by less than 1×10^2, then their ζ' values would "appear" equal in this table. For further information regarding the sequences used, see Table B.1 in the appendix.

Test Name	PLAINS			LAGAN			EMBOSS		
	t	r	ζ'	t	r	ζ'	t	r	ζ'
HumanPseudo1	356.71	4	7.37	340.32	4	6.00	340.19	4	5.99
HumanPseudo2	285.75	3	3.96	281.84	3	3.94	238.30	3	3.87
HumanPseudo3	2181.50	14	47.18	441.58	6	22.98	1708.51	10	18.49
HumanPseudo4	511.99	7	3.85	2172.40	14	-Inf	296.84	4	4.59
HumanPseudo5	792.64	7	7.29	775.74	7	7.29	176.73	1	13.04
MousePseudo1	389.84	4	13.97	386.88	4	13.40	388.88	4	13.78
MousePseudo2	461.68	6	8.88	453.64	6	7.77	206.02	2	5.56
MousePseudo3	72.19	1	6.75	72.19	1	6.75	83.34	1	6.75
fugu2r	534.14	5	11.15	360.22	3	13.05	151.39	2	14.07
HFortho1	734.82	7	10.94	349.33	4	14.18	374.35	5	13.05
HFortho2	600.22	4	16.78	555.61	4	16.78	327.91	1	20.18
HFortho3	637.52	7	14.53	259.44	3	19.05	409.99	5	16.71
HFortho4	1004.97	10	21.74	529.16	5	-0.00	367.86	4	-0.00
HFortho5	739.71	7	11.07	450.93	5	13.07	453.61	5	13.07
human_mouse.1_1	676.29	10	8.46	52.36	1	18.29	186.98	2	17.00
human_mouse.1_3	552.55	6	15.14	406.79	6	15.14	429.51	6	15.14
human_mouse.3_9	1260.69	15	15.47	432.25	7	24.23	801.15	12	18.44
human_mouse.3_16	218.47	3	5.71	x	x	x	180.05	2	6.77
human_mouse.4_3	262.19	3	15.44	74.91	1	17.79	176.83	2	16.59
human_mouse.4_5	421.71	6	7.35	221.57	3	10.47	401.71	5	8.32
human_mouse.6_17	986.89	12	23.00	240.10	3	-0.00	260.66	4	-0.00
human_mouse.7_11	594.32	8	9.06	164.10	2	15.44	476.71	7	9.99
human_mouse.17_11	608.75	7	13.93	171.96	3	18.57	451.60	6	15.02
human_mouse.x_x	1302.49	18	17.20	636.82	9	-0.00	568.46	9	-0.00
human_dog.6_1	1239.35	14	18.99	424.59	6	-0.00	688.81	8	26.81
human_dog.6_12	1284.79	14	13.88	548.19	7	21.23	394.04	6	22.44
human_dog.6_34	1488.26	16	-0.00	496.14	6	-0.00	900.73	12	-0.00
human_dog.7_16	1042.19	13	10.45	128.07	2	22.40	309.03	4	19.84

Table 6.1: Shown here for PLAINS, EMBOSS, and LAGAN are the r, t, and ζ' values obtained from aligning genomic DNA sequences of lengths between 0.5 Kb and 12 Kb within human, mouse, dog, and fugu, where the pairs are biologically related and mainly noncoding DNA with expected large gaps and low homology regions.

Also, PLAINS does not always yield the results of least coincidental probability in this table, and this anomaly has a simple explanation. Note that the nature of PLAINS is to capture the biology faithfully even when the sequences have expected large gaps and low similarities. Thus it tries to aggressively align as many regions as possible, and hence in these situations, it produces r and t values that tend to be higher than those from other tools, even though its high r causes its overall result to appear more coincidental in spite of the compensating higher t. However, it turns out that when we fix r for all the tools, PLAINS yields higher t and hence better ζ' results. In other words, for any given r, each of the r segment pairs generated by PLAINS have smaller individual coincidental probabilities than the best r segment pairs generated by other tools. Figure 6.1 explains the details further.

Many of the genomic alignments yielded by the four tools have caught exons in the alignment, but most of these exons caught aren't included in the "good" regions of the alignment, because SEPA removed them for having a ρ value that was too high. Figure 6.2 is an example of this, since here, both PLAINS and LAGAN identify most of the exons in the human sequence, but we only count the exons that the tools identify as lying within the "good" regions.

The MousePseudo (alignments of Mouse genes against corresponding pseudogenes), and humanHomol (alignment of Human genes against homologous Mouse genes) runs were, for the most part, a relatively close competition between the four alignment tools, in terms of the actual alignments obtained, especially between PLAINS and LAGAN. This shows either the difference of linear gap functions over piecewise-linear gap functions, or the difference of using general-case gap parameters over using customized gap parameters per species, or possibly both.

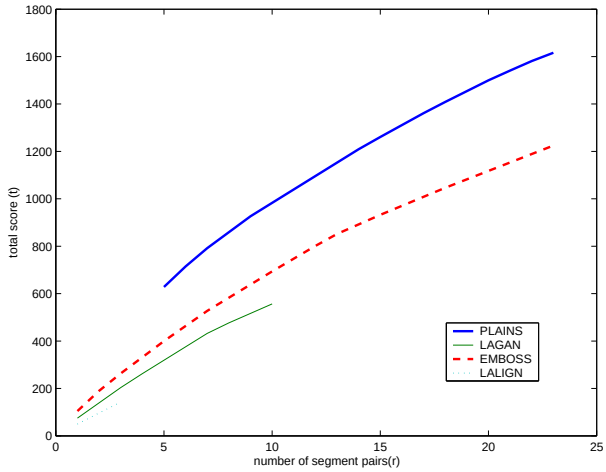


Figure 6.1: In this figure, we observe the unadjusted r and t values produced by PLAINS, LAGAN, and EMBOSS from the human-mouse.3 – 9 experiment where we vary the ρ variable used to filter our segment pairs. On each curve, we observed the t and r values of each tool when varying ρ over various values from 0.1 till 0.9. Recall from table 6.1 that PLAINS performed poorly in terms of ζ' values for $\rho = 0.5$ for the human-mouse.3 – 9 experiments. However, note from this plot that for any fixed r where PLAINS is comparable to a different tool, PLAINS receives the highest t value, and therefore if we designed SEPA using a fixed r value over all alignment tools, then PLAINS would have the highest t value, and hence the highest ζ' value (i.e., the best result). Many other experiments from table 6.1 have a similar plot to this one.

For most of the HumanPseudo (alignments of Human genes against corresponding pseudogenes) runs, PLAINS and LAGAN yielded alignments with many similar correlations, but more exons were caught by PLAINS. One illustration of this is figure 6.2, which compares the results of PLAINS to LAGAN for HumanPseudo5 in further detail.

However, the most interesting results obtained were in the HFugu2r and HFortho runs, the runs involving genomic Human and Fugu sequences. Since the evolutionary distance between the human and fugu species is significantly long, one expects even the most conserved exon regions of the orthologous gene in the two genomes to have diverged quite a lot (despite the protein sequences still sharing high homology). Futhermore, the two genomes have very different gene structures — the genes in the Fugu genome have very short introns, while the introns in the Human genome are usually very long. Hence, the results caught by the alignment tools here is no small matter.

For HFortho2, not only is the ζ' value for PLAINS better than that of EMBOSS, but PLAINS also caught more common exons between the two related genomic sequences. Therefore, as PLAINS currently stands, it holds promise of becoming a tool of choice for aligning several thousand nucleotide DNA sequences, and possibly also for identifying exons between two genomes as diverged as human and fugu. Figure 6.2 shows more details.

Each run of PLAINS to optimize gap/mismatch parameters on a pair of species took 30 minutes to 2 hours. The relatively long time taken by PLAINS is due to its need for computing several hundred alignments under various gap/mismatch parameters before deciding which gap/mismatch parameters are the most optimal. When ran using fixed-set gap-mismatch parameters, PLAINS ran in just under a minute, a constant factor of at most 5.6 times slower than

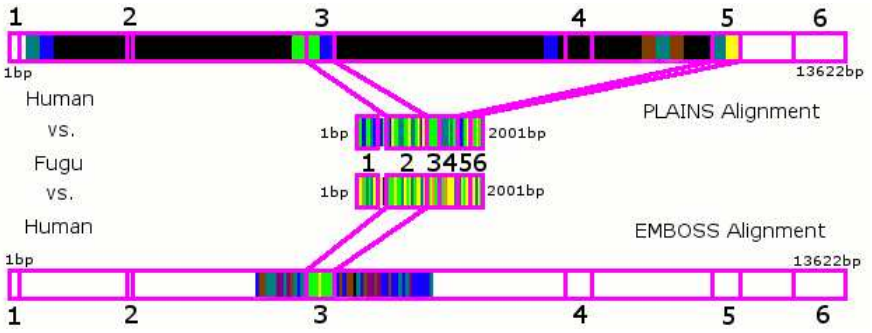


Figure 6.2: Match Ratio Color Lines in the HFOrtho2 test for PLAINS and EMBOSS. Here, the Human and Fugu sequence used have six exon regions that correspond to each other (though not necessarily in order, as exon region 2 in the Fugu sequence corresponds to exon region 3 in Human sequences for example). Here, both PLAINS and EMBOSS correctly identify the correlation of exon region 2 in Fugu with exon region 3 in Human, but only PLAINS identifies the correlation of exon region 5 in Fugu with exon region 5 in Human.

`EMBOSS`. The reason for this slowdown is manifold: (1) PLAINS uses a linear space table instead of the quadratic space typical of dynamic programming, and (2) there is constant extra overhead in using Linked-List Assistance (mentioned earlier) to help create an alignment.

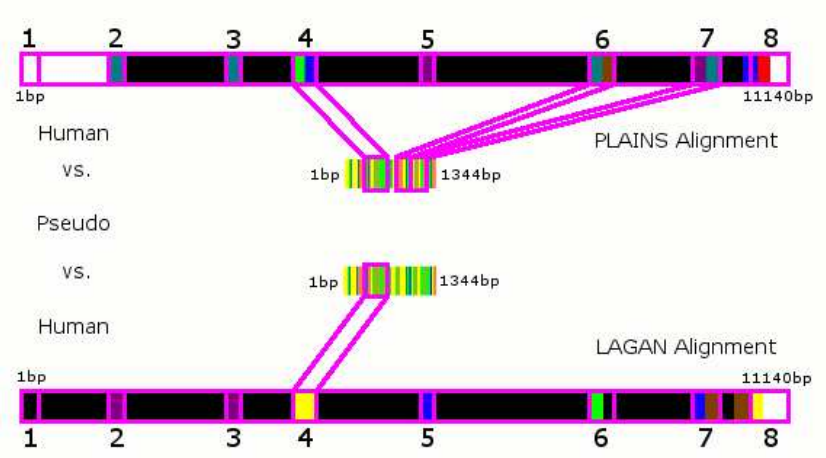


Figure 6.3: Match Ratio Color Lines in the HumanPseudo5 test for PLAINS and LAGAN. Here, the Human sequence has 8 exon regions that are similar to areas of the pseudosequence used, and alignments of PLAINS and LAGAN for these cases are similar, even by eyeglance of the ColorGrids. Note that although PLAINS and LAGAN catch most of these regions in their alignments, we're only counting the exon regions that participated in "good" segments according to SEPA. With this in mind, PLAINS and LAGAN both identify exon region 4 as important, but PLAINS also deems exon regions 6 and 7 in the Human sequence as important, which LAGAN misses.

Chapter 7

Conclusions and Open Problems

PLAINS is able to catch more important correlations than its competition, especially in sequences of distant relation like Human and Fugu. And, the SEPA filtering is capable of distinguishing unimportant regions from important ones. In addition, our empirical analysis leads us to the conclusion that the SEPA-based p-value technique models coincidental probabilities quite accurately. Furthermore, we note that aggressively incorporating too many segment pairs into an alignment can corrupt the overall result with false positives, in spite of an apparent improvement in the total score or in identified exon regions caught, as illustrated by PLAINS. However, SEPA can modify the overall alignment to select only the best r segments from an alignment while keeping the confidence in the final result high. It is here that the strength of PLAINS becomes obvious, since its r segments are less coincidental than its competition, and have higher scores, and hence better ζ' values.

It has become apparent that some upwards scaling is essential if PLAINS is to be run over sequences with more than 8000 nucleotides. Plains, as it stands, essentially performs localized alignment (since it discards leftmost and

rightmost nucleotides that would be aligned against gaps), and therefore, it is only fit for sequences of up to 8000 nucleotides.

Possible future extensions include scaling Plains to search large sequences (of mega bases of nucleotides) for smaller areas where localized alignments can be performed, and then combining localized alignments to globalized alignments. PLAINS could become the ideal tool for aligning EST sequences to a genome.

Another possible extension involves adding a model to learn expected alignments over various species, as opposed to just merely approximating the best gap/mismatch parameters.

We can improve SEPA by using random portions of DNA from Human, Mouse, and Fugu instead of randomly generated DNA sequences. In that case, our concern shifts from the coincidental probability of a segment's score from aligning random DNA, to the coincidental probability of a segment's score from aligning unrelated random regions of organisms under comparison. Further extension includes development of better statistics that realistically capture the base-pair and coding/noncoding distributions within the sequences, as well as the effects of secondary and tertiary structures.

In addition, PLAINS at the moment assigns a score of $m_a = 1$ to a perfect match, and a score m_s (specified by user) to any mismatch. It may be useful to have a scoring matrix to assign different scores to different types of matches/mismatches. (For example, if aligning a C against a G is more common than aligning at C against a T, then perhaps we can penalize the C-G mismatch less than the C-T mismatch when performing an alignment, etc.)

Appendix A

Proofs for Non-Trivial Portions of PLAINS

In creating an alignment from a given set of parameters, there are two features PLAINS has that makes it stand out from the literature it derives from. First, PLAINS uses $O(np)$ space in the worst-case scenario. Second, under its given space and runtime bounds, PLAINS obtains the correct alignment based on the given piecewise-linear function $ww(\cdot)$.

These two features of PLAINS also happen to be nontrivial and tedious, which was why they were not elaborated and proven earlier when we were describing how PLAINS creates an alignment. For the second of these two features, note that proving the correctness of alignment obtained by PLAINS boils down to proving the correctness of the "maximum criteria" selection employed by PLAINS when using V and V^r tables to help create our alignment. We will now proceed with these nontrivial proofs.

A.1 Proof for the $O(np)$ Space Bound

Earlier, we stated that we are using $O(np)$ space worst-case when using p-part piecewise-linear function $ww(\cdot)$ (and when $p \ll m$, as is in PLAINS, then this results in a substantial improvement over the quadratic space complexity). This, in fact, is the main innovation of PLAINS over the original intuitions of Miller-Myers.

As explained earlier, PLAINS uses $O(n)$ space from the tables because it saves the t most recently computed columns of all tables, and uses recursion to obtain unknown portions of the alignment. The space taken up by the recursions is $O(\log m)$, however in practice, m and n are assumed to differ from each other by a constant factor, and hence the $O(\log m)$ space used by recursion is less than the $O(n)$ space used by the tables.

What uses the most space in the PLAINS algorithm is not the tables, but the lists of form L_j, L_j^r, R and R^r used to compute the tables. We will now prove that each list used in PLAINS uses $O(p)$ space.

Suppose for simplicity that we fix j so that we are dealing for all i with $V(i)$ and $F(i)$, and we use linked-list L to obtain the much-needed solutions for F and V. (I.e., $V(i) = V(i, j)$ and $F(i) = F(i, j)$ and L is how we get values for V and F.)

CLAIM: *For p-part function $ww(\cdot)$, L will always have at most p elements in it.*

PROOF: In the beginning, when $i = 0$, L starts with one element. Later on, in some ith iteration, after we just finished computing $V(i)$, if we split an element of L with winner k and value v (where $v = V(k)$), then this implies that, for some x, the k_{th} plot of $v - ww(x-k)$ intersects the i_{th} plot of $V(i) - ww(x-i)$

(i.e., $V(i) - ww(x - i) = v - ww(x - k)$ for some x). However, both of these plots are identical in shape. By translating one horizontally and vertically, this plot can fit perfectly into the other.

Therefore, in considering the lines from both plots that intersect (assuming p_1th line from the i_{th} plot and the p_2th line from the k_{th} plot intersect), since $k < i$, the p_1th line from the i_{th} plot must have a higher downward slope than that of the p_2th line from the k_{th} plot. Therefore $p_1 < p_2$. (So, a given line from the i_{th} plot intersects a later line from the k_{th} plot.)

Hence, if list L has p elements, this implies having found $p-1$ intersections, each from a different plot. This means that we have $cand(\cdot)$ values taken from the ur_1th line of some q_1 plot intersecting the ul_2th line of some q_2 plot, and the ur_2th line of the q_2 plot intersects the ul_3th line of the q_3 plot, and so on up to the q_p plot, and therefore:

$$ur_1 < ul_2 \leq ur_2 < ul_3 \leq ur_3 \cdots ur_{p-1} < ul_p.$$

However, all of these plots have exactly p lines. Therefore, in this case, for each h from 1 to p, $ul_h = ur_h$ must be true. Hence for all h from 1 to p, the ul_h values correspond to all the lines of our ww function (meaning $ul_h = h$ for all h).

Hence in this case, for each element g in L, if g uses the q_h plot for some value h, then only one line from the q_h plot, the ul_hth line, can give the best solution for indices from the $[g_l, g_r]$ interval (g_l and g_r are the lwb and upb values for element g in list L).

Therefore, if during the ith iteration, we have p elements in L, then the i plot of $V(i)-ww(x-i)$ will have lines of the same slopes as those corresponding to lines ul_1 through ul_p. Therefore, if the p'th-line of the i_{th} plot intersects some

$q_{h'}$ plot (with $h' \geq 2$), then all elements of L derived from q_h plots with $ul_h \leq p'$ will be discarded (i.e., at least one element of L will definitely get discarded, and one new element with i as its $cand(\cdot)$ value is created, implying that total number of elements in L overall in this i_{th} iteration will either stay the same or decrease). Note that it is impossible for the i-curve's p'th-line to intersect the q_1 plot.

Hence, it is never possible to increase the number of elements in L from p to $p+1$. So L always has $O(p)$ elements in it.

Therefore, in returning to our 2-dimensional model, this argument implies that our n different linked lists of form L_j and L_j^r each use $O(p)$ space, and similarly, R and R^r also each use $O(p)$ space. Hence, total space used by all of the lists is $O(np)$. QED

A.2 Details for the Maximum Criterion Selection

A.2.1 Definition of $rc(j)$

Before the next section, where we prove the correctness of the "maximum criteria" selection rule used by PLAINS, it may help to gain an intuition of the definition of $rc(j)$.

Suppose for the moment that we fix the index j used by the algorithm in the V and V^r tables so that we flatten to one dimension in order to keep the arguments simple. Hence, assume $V(i) = V(i,j)$, $F(i) = F(i,j)$, $V^r(i) = V^r(i, n-j)$, and $F^r(i) = F^r(i, n-j)$. We are thus saving the most recently

found t entries from V and V^r, where t is some constant[1]at least 1.

During the computations of V and V^r, we use a linked list L to maintain solutions for F, and a linked list L^r to maintain solutions for F^r. Next, assume that we compute all the F and V entries before starting on the F^r and V^r entries, and we look at L while computing V^r (but we do not modify L while computing V^r).

Suppose also that while computing V, list L maintains $cand_{\cdot}(i)$ for all i from 0 to m, (even though we only need indices of i from 0 through $m/2$ to complete computations for F and V), and therefore, when we are done computing V and L, list L now has the $cand_{m/2}(i)$ values for all i from $m/2$ to m. Hence, after computing F and V, we know that:

For any i in range $[(m/2)+1, m]$: If $i' = cand_{m/2}(i)$, then i' is the number in range $[0, m/2]$ such that $V(i') - ww(i'-i)$ is maximized.

Note that, during the process of computing the V^r entries, one possible best alignment solution in combining both the V and V^r tables could be $V(i') - ww(i'-i)) + V^r(m-i)$ (a solution with a gap starting in the first half of X and ending in the second half of X).

So, now suppose we have some extra variable rc equal to the i in range $[(m/2)+1, m]$ such that $V(cand_{m/2}(i)) - ww(i - cand_{m/2}(i))) + V^r(m-i) = eval(cand_{m/2}(i), i) + V^r(m-i)$ is maximized. We can figure out the value for rc while computing the entries for V^r using the list L. In considering all possible alignments that have a gap starting in the first half of X and ending in the second half of X, we know that rc and $cand_{m/2}(rc)$ give us the coordinates in the right and left halves of X of the gap for the best-scoring alignment of this

[1]Saving the t most recently computed entries for $V(\cdot)$ and $V^r(\cdot)$ corresponds to saving the t most recently computed columns of $V(\cdot,\cdot)$ and $V^r(\cdot,\cdot)$ in our two-dimensional model.

type.

Switching over to using all rows in computing our F, V, F^r, and V^r tables, we will have, for each row j from 0 to n, a value $rc(j)$ which is equal to the i in range $[(m/2)+1, m]$ such that $V(cand^L_{m/2}(i,j),j) - ww(i - cand^L_{m/2}(i,j))) + V^r(m-i, n-j) = eval_j(cand^L_{m/2}(i,j), i) + V^r(m-i, n-j) = er(j,i)$ is maximized.

A.2.2 Proof of Correctness in "Maximum Criteria" Selection

As mentioned earlier, in the PLAINS computation of the V and V^r tables, the "maximum criteria" selection is used to find a k that maximizes $\max\{gr(k), er(k, rc(k))\}$. Below we give a proof for the correctness of this method.

In the alignment of X against Y, two general cases may occur.

1. We may have X aligned against a gap of a type starting in the first half of X, and ending in the second half of X.

2. We do not see X aligned against a gap of a type starting in the first half of X, and ending in the second half of X.

When case (2) occurs[2], it is feasible to align $X[1..m/2]$ against Y separately from aligning $X[m/2..m]$ against Y. Furthermore[3], there exists a k' such that for all i and i' and j, $V(m/2, k') + V^r(m/2, n-k') > V(i,j) + V^r(m-i', n-j) - ww(i'-i)$.

Hence, we will obtain the correct alignment by selecting a k that maximizes $gr(k)$. Therefore, selecting a k such that $\max\{gr(k), er(k, rc(k))\}$ is maximized gives us the correct alignment in this case.

[2]This is essentially what the V and V^r tables do.

[3]Therefore for this k' value, $gr(k') > er(k', rc(k'))$.

When case (1) occurs, then there exists an i, i', and j such that for all k', $V(i,j) + V^r(m-i', n-j) - ww(i'-i) > V(m/2, k') + V^r(m/2, n-k')$. Furthermore, for a fixed pair of i' and j, note that $i = cand^L_{m/2}(i', j)$, the $cand_{m/2}(i')$ value from the L_j list, maximizes $V(i,j)+V^r(m-i', n-j)-ww(i'-i)$. Next, note that if j is fixed, setting $i' = rc(j)$ and hence $i = cand^L_{m/2}(i', j)$ maximizes $V(i,j)+V^r(m-i', n-j)-ww(i'-i) = V(cand^L_{m/2}(i', j), j)-ww(i'-cand^L_{m/2}(i', j)))+V^r(m-i', n-j) = eval_j(cand^L_{m/2}(i', j), i')+V^r(m-i', n-j) = er(j, i')$. Therefore, we obtain the highest scoring alignment by selecting a k' that maximizes $er(k', rc(k'))$. Therefore, by selecting a k that maximizes $\max\{gr(k), er(k, rc(k))\}$, the algorithm computes the correct alignment in this case.

Appendix B

SEPA Details

B.1 Segment Pair Analysis in Further Detail

In order to approximate an appropriate p-value estimation for SEPA, we analyzed segment pairs behavior over our assumed null hypothesis of alignments for randomly generated nucleotide sequences. For length values ranging from 1000 bp to 8000 bp, we generated 25 random sequences. We also generated 25 random sequences of length 500 bp. For each combination of these length pairs, we ran all 625 possible pairwise alignments using PLAINS, and analyzed results using SEPA where $\rho = 1$ (to avoid filtering any segments out due to low p-value), and recorded the results in fig. B.1, B.2, B.3, and B.4.

From fig. B.1, we observe for segment length-to-score ration that, for the most part, the mean takes a constant value at 3.1, and the variance remains below 0.4, leading us to infer that length-to-score ratio can be well-approximated by a constant.

From fig. B.2, we infer that, although for small n values, the average segment score decreases with increasing m, asymptotically (for large n) the it stays

roughly constant in terms of m, while the variance fluctuates wildly around a constant value. Hence, for our scoring method, we model the segment scores as independent of m and n.

From fig. B.3, we estimate that the average and variance for r (the number of segment pairs) scale roughly with $\Theta(log(mn))$. More specifically, we approximate the mean of r and the variance of r, called $r_a(m,n)$ and $r_v(m,n)$ respectively, to scale roughly as $k_0 \ln(k_1 mn + k_2(m+n) + k_3)$ where k_0, k_1, k_2, and k_3 are empirically determined constants. In the case of $r_a(m,n)$, we observe that $k_0 = 10^3$, $k_1 = 7.95 \times 10^{-10}$, $k_2 = 1.54 \times 10^{-7}$, $k_3 = 1.01$, and in the case of $r_v(m,n)$, we observe that $k_0 = 10^3$, $k_1 = 1.93 \times 10^{-10}$, $k_2 = 1.97 \times 10^{-7}$, $k_3 = 1.00$.

From fig. B.4, we estimate that the average total segment score scales roughly with $\Theta(log(mn))$, and the deviation for total segment score scales roughly with $\Theta(i \cdot d)$ (but never declines below 100), where $i = \min(m,n)$ and $d = \|m - n\|$. More specifically, we approximate the average total score $t_a(m,n)$ to scale roughly as $k_0 \ln(k_1 mn + k_2(m+n) + k_3)$, and the deviation for total score $t_D(m,n)$ to scale roughly as $\max(k_0, k_1 i \cdot d + k_2 i + k_3 d + k_4)$, where k_0, k_1, k_2, k_3, and k_4 are empirically estimated constants (and the variance $t_v(m,n) = t_D(m,n)^2$). In the case of $t_a(m,n)$, we observe that $k_0 = 10^5$, $k_1 = 4.29 \times 10^{-10}$, $k_2 = 1.33 \times 10^{-8}$, and $k_3 = 1.00$, and in the case of $t_v(m,n)$, we observe that $k_0 = 100$, $k_1 = -5.54 \times 10^{-5}$, $k_2 = 4.63 \times 10^{-1}$, $k_3 = 1.04 \times 10^{-2}$, and $k_4 = -65.01$.

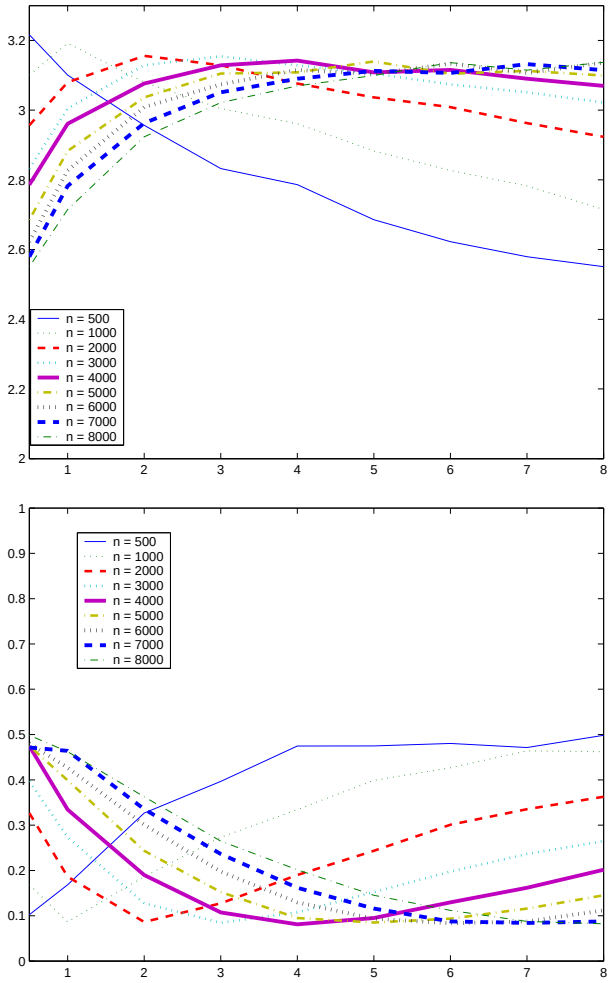


Figure B.1: Shown above are the mean and variance plots for the segment pair length-to-score ratio from aligning randomly generated DNA sequences. A unique line is plotted corresponding to each value of n in the thousand lengths ranging from 1000 to 8000. For these figures, and others that follow, x represents the m value divided by 1000, and y represents the mean or variance value obtained for that particular m and n.

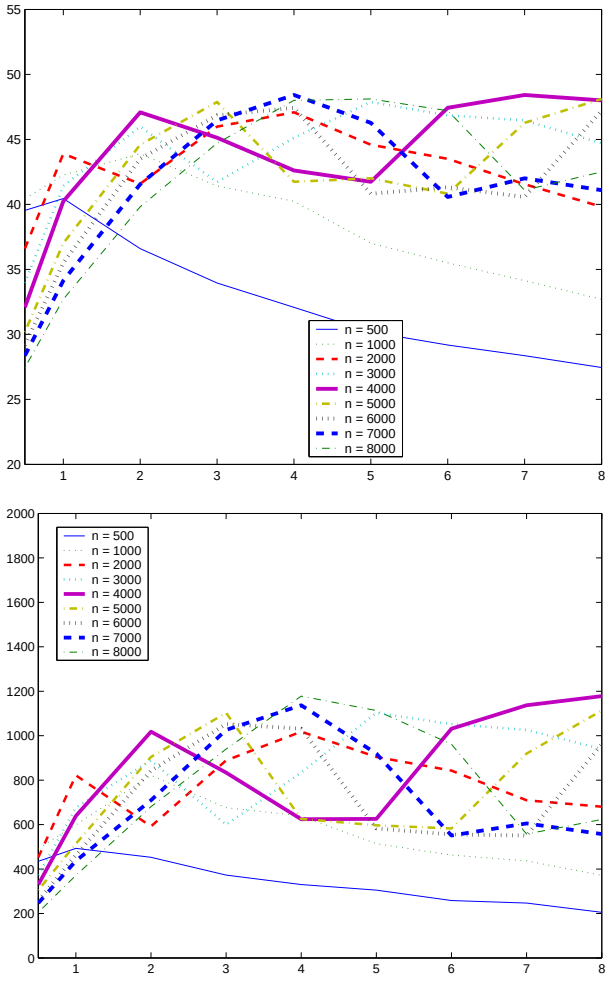


Figure B.2: Shown here are the mean and variance plots for segment scores from aligning randomly generated DNA sequences.

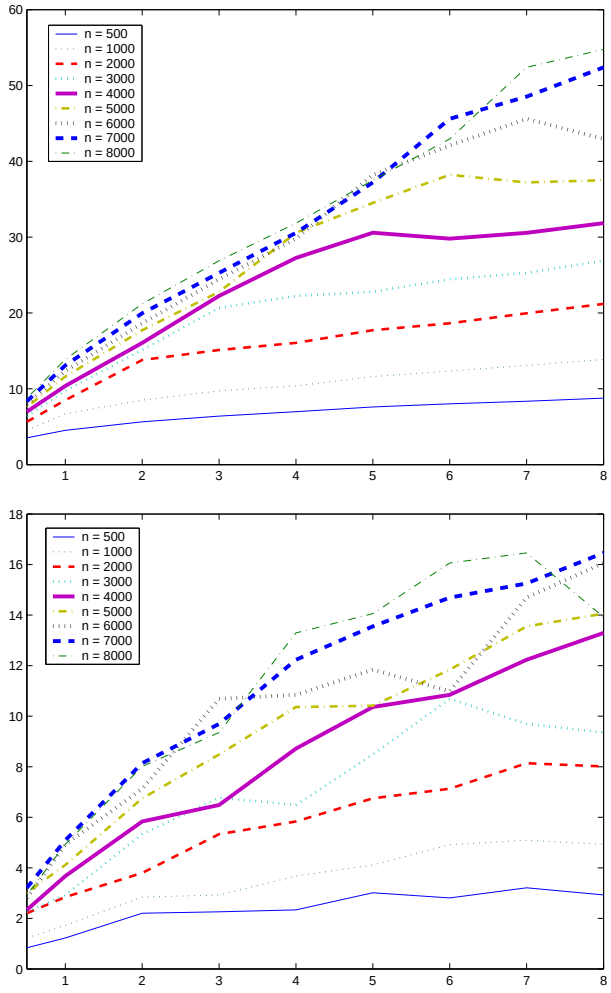


Figure B.3: Shown here are the mean and variance plots for r, the number of segment pairs obtained from aligning randomly generated DNA sequences.

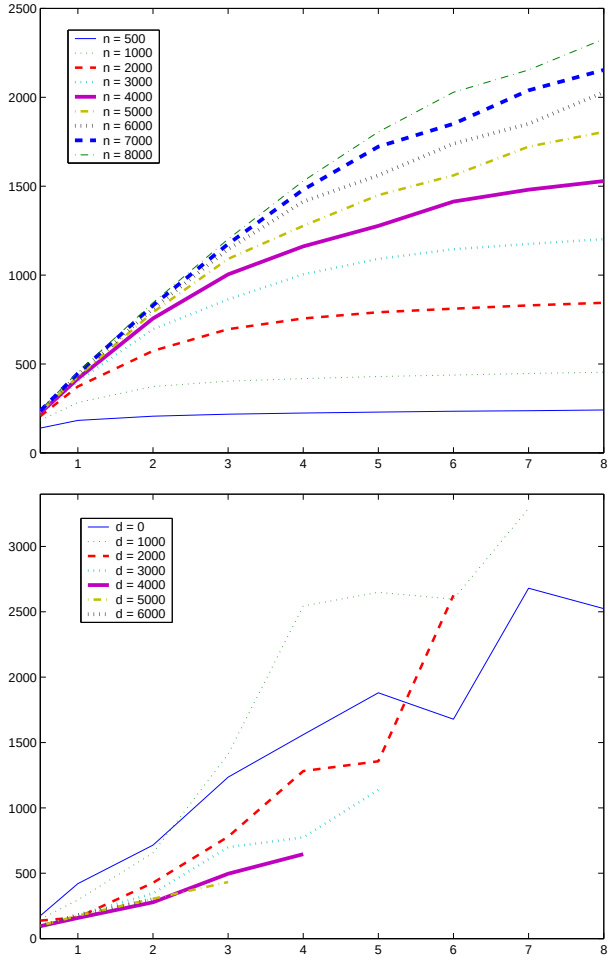


Figure B.4: The plots shown here are the mean and deviation plots for t, the total score of all segment pairs from aligning randomly generated DNA sequences. Because the variance plot was difficult to quantify in terms of m and n, we instead model the deviation for total score in terns of d and i, where $i = \min(m, n)$ and $d = \|m - n\|$. The lower figure shows the deviation plot, with each curve corresponding to a unique d value, and the x-axis reprsenting i in units of thousands.

B.2 Sequence Details

Shown in Table B.1 are further details for the sequences used to compare PLAINS against LAGAN, EMBOSS, and LALIGN. Please note that sequences are expressed in their regular format unless they end with a ":-1" or "-" symbol, which indicates that they have been reverse-complemented prior to performing any alignments.

Name	First Sequence	Second Sequence
HumanPseudo1	chr1 8257472 8257969 +	NCBI34:19:54160379:54161804:1
HumanPseudo2	chr1 163548408 163549002 +	NCBI34:4:174948678:174951482:-1
HumanPseudo3	chr1 212839737 212843396 +	NCBI34:19:47480657:47491789:1
HumanPseudo4	chr2 215849936 215850977 −	NCBI34:12:52960755:52965297:1
HumanPseudo5	chr3 154761512 154762855 −	NCBI34:20:62845714:62856853:-1
MousePseudo1	chr1 6930250 6930693 +	NCBIM32:4:116062392:116064688:1
MousePseudo2	chr10 34897773 34898331 +	NCBIM32:3:111151293:111157009:1
MousePseudo3	chr1 101195551 101195966 +	NCBIM32:19:41974653:41984383:1
fugu2r	NCBI34:6:10803176:10817954:1	FUGU2:scaffold_3266:7199:8502:1
HFortho1	NCBI34:22:17268346:17274146:1	FUGU2:scaffold_115:304567:308251:1
HFortho2	NCBI34:22:19452941:19466562:1	FUGU2:scaffold_385:130429:132429:1
HFortho3	NCBI34:21:31952480:31961633:1	FUGU2:scaffold_492:107025:110089:-1
HFortho4	NCBI34:4:78536922:78549607:1	FUGU2:scaffold_1018:38886:42563:-1
HFortho5	NCBI34:1:23574363:23584195:1	FUGU2:scaffold_2020:1332:3570:1
human_mouse.1_1	hg17 chr1:1045045-1049199	mm6 chr1:58087808-58093089 −
human_mouse.1_3	hg17 chr1:109911-115784	mm6 chr3:108302834-108307402 +
human_mouse.3_9	hg17 chr3:920975-927750	mm6 chr9:13034270-13040751 −
human_mouse.3_16	hg17 chr3:40927-45344	mm6 chr16:36425494-36426630 +
human_mouse.4_3	hg17 chr4:1016348-1026634	mm6 chr3:43806778-43808958 +
human_mouse.4_5	hg17 chr4:33206-37263	mm6 chr5:116454347-116457564 −
human_mouse.6_17	hg17 chr6:1515792-1522464	mm6 chr17:5319541-5327318 +
human_mouse.7_11	hg17 chr7:253979-256656	mm6 chr11:47406997-47414401 −
human_mouse.17_11	hg17 chr17:203511-209188	mm6 chr11:46304241-46308929 −
human_mouse.x_x	hg17 chrX:928373-936336	mm6 chrX:100457186-100463788 +
human_dog.6_1	hg17 chr6:48183-58637	canFam1 chr1:66683762-66688436 −
human_dog.6_12	hg17 chr6:791946-797744	canFam1 chr1:58385127-58391875 +
human_dog.6_34	hg17 chr6:1248975-1255904	canFam1 chr34:40546832-40556432 −
human_dog.7_16	hg17 chr7:40725-45009	canFam1 chr16:22868000-22875215 +

Table B.1: Sequence Details for the Biologically Related Alignments Ran. All the sequences are retrieved from ENSEMBL database [www.ensembl.org].

Bibliography

[1] Altschul, S.F., Boguski, M.S., Gish, W., and Wooton, J.C., "Issues in Searching Molecular Sequence Databases." *Nature Genetics*, **6**:119–128, 1994.

[2] Michael Brudno, Chuong Do, Gregory Cooper, Michael F. Kim, Eugene Davydov, Eric D. Green, Arend Sidow, Serafim Batzoglou, "LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA," *Genome Research*, **13(4)**:721-31, 2003 Apr.

[3] Craig G. Nevill-Manning, Cecil N. Huang, Douglas L. Brutlag, "Pairwise protein sequence alignment using Needleman-Wunsch and Smith-Waterman algorithms," Personal communication (`http://motif.stanford.edu/alion/`), 1997.

[4] Gill, O., Zhou, Y., Mishra, B.: Aligning Sequences with Non-Affine Gap Penalty: PLAINS Algorithm, a Practical Implementation, and its Biological Applications in Comparative Genomics. Series in Mathematical Biology and Medicine **8** (2005). An unabridged version can be found at: `http://bioinformatics.nyu.edu/~gill/index.shtml`

[5] Gu X, Li WH., "The size distribution of insertions and deletions in human and rodent pseudogenes suggests the logarithmic gap penalty for sequence alignment." *J Mol Evol.*, **40(4)**:464-73, 1995 Apr.

[6] Hromkovic J, "Heuristics." *Algorithms for Hard Problems, Second Edition*, **6**:439-467, 2003.

[7] X. Huang and W. Miller, *Advanced Applied Mathematics*, **12**:373-381, 1991.

[8] Iglehart, D.L.: Extreme Values in the GI/G/1 Queue. The Annals of Mathematical Statistics **43 (2)** (1972) 627–635

[9] Karlin S, Altschul S.F., "Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes" *Proc. Natl. Acad. Sci. USA*, **87**:2264–2268, March 1990.

[10] Karlin S, Altschul S.F., "Applications and statistics for multiple high-scoring segments in molecular sequences" *Proc. Natl. Acad. Sci. USA*, **90**:5873–5877, June 1993.

[11] Karlin, S., Dembo, A., Kawabata, T.: Statistical Composition of High-Scoring Segments from Molecular Sequences. The Annals of Statistics **18 (2)** (1990) 571–581

[12] Lipman, D.J., Altschul, S.F., and Kececioglu, J.D., "A Tool for Multiple Sequence Alignment." *Proceedings of the National Academy of Sciences USA*, **86**:4412–4415, 1989.

[13] Miller, W., and Myers E.W., "Sequence Comparison with Concave Weighting Functions" *Bulletin of Mathematical Biology*, **50**:97–120, 1988.

[14] Miller, W., and Myers E.W., "Optimal Alignments in Linear Space" *CABIOS*, **4**:11–17, 1988.

[15] Needleman, S.B., and Wunsch, C.D., "A General Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins." *Journal of Molecular Biology*, **48**: 443–453, 1970.

[16] Ophir R, Graur D., "Patterns and rates of indel evolution in processed pseudogenes from humans and murids." *Gene.*, **205(1-2)**: 191–202, 1997 Dec 31.

[17] Pearson, W.R., "Comparison of Methods for Seqrching Protein Sequence Databases." *Protein Science*, **4**:1145–1160, 1995.

[18] Pearson, W.R., "Searching Protein Sequence Libraries: Comparison of the Sensitivity and Selectivity of the Smith Waterman and FASTA algorithms." *Genomics*, **11**: 635–650, 1991.

[19] Press W.H., Flannery B.P., Teukolsky S.A., Vetterling W.T., "Downhill Simplex Method in Multidimensions." *Numerical Recipes: The Art of Scientific Computing*, **10.4**: 289–293, 1986.

[20] Rice P, Longden I, Bleasby A., "EMBOSS: the European Molecular Biology Open Software Suite" *Trends Genetics*, **Jun 16(6)**:276-7, 2000.

[21] Siegmund, D., Yakir, B.: Approximate p-Values for Local Sequence Alignments. The Annals of Statistics **28 (3)** (2000) 657–680

[22] Smith, T.F., and Waterman, M.S., "Identification of Common Molecular Subsequences." *Journal of Molecular Biology*, **147**: 195–197, 1981.

[23] Shpaer, E., Robinson, M., Yee, D., Candlin, J., Mines, R., and Hunkapiller, T., "Sensitivity and Selectivity in Protein Similarity Searches: A Comparison of Smith-Waterman in Hardware to BLAST and FASTA." *Genomics*, **38**: 179–191, 1996.

[24] States, D.J., Gish, W., and Altschul, S.F., "Basic Local Alignment Search Tool." *Journal of Molecular Biology*, **215**: 403–410, 1990.

[25] Waterman, M.S., and Eggert, M., "A New Algorithm for Best Subsequence Alignments with Applications to tRNA -rRNA Comparisons." *Journal of Molecular Biology*, **197**: 723–728, 1987.

[26] Zhang Z, Gerstein M, "Patterns of nucleotide substitution, insertion and deletion in the human genome inferred from pseudogenes." *Nucleic Acids Res.*, **31(18)**: 5338-48, 2003 Sep 15.