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
September 2006

Bud 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 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 evolutionary relationships among species allows us to better understand the important biological functions of those species and trace their evolutionary history. This thesis considers sequence alignment, a method for obtaining these inter-relationships. We improve upon sequence alignment tools designed for DNA with Plains (Piecewise-Linear Alignment with Important Nucleotide Seeker), an algorithm than uses piecewise-linear gap functions and parameter-optimization to measure similarities between remotely-related species pairs such as human and fugu, while using reasonable amounts of memory and space on an ordinary computer. We also propose Planar (Piecewise-Linear Alignment for Nucleotides Arranged as RNA), which is similar to Plains, but is designed for aligning RNA, and accounts for secondary structure. We also explore SEPA (Segment Evaluator for Pairwise Alignments), 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 and Planar against similar alignment tools, emphasizing the interesting correlations caught in the process.

Contents

List of Figures

List of Tables

List of Appendices

Chapter 1

Introduction

1.1 PLAINS 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 these discrepancies, 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 penalties, 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 [19]), but PLAINS presents several additional theoretical innovations in terms of worst-case upper-bound on 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.

1.2 PLANAR Introduction

Within an organism, a small percentage of its DNA, particularly the coding regions, transcribe to RNA. A small percentage of that RNA, particularly the mRNA, gets translated to proteins. The transcription occurs in the nucleus cued by transcriptional factors and modulated by enhancers and repressions. The translation occurs in the cytoplasm, and involves ribosomes, initiation factors, miRNA's (microRNA's), miRNA binding sites, and more repression than activation.

Until recently, it was believed that the cellular functions were primarily carried out by the protein coding genes, and most cellular functions found in noncoding RNA (such as rRNA and tRNA) were homologs to coding regions. This assumption was found to be incorrect because there are 30-40K genes in the human genome, which is only twice as many genes as Drosophila (and the

human genome is far longer than just twice as long as that of Drosophila). Hence, by eliminiation, many complex cellular functions in humans must be carried out by noncoding RNA.

The discovery of miRNA's is regarded as a breakthrough. They suggest a new large class of eukaryotic regulatory elements in genes. Evidence for its importance in functional roles can be concluded from phenotypic, expressional, and evolutionary analysis, in spite of the fact that the number of miRNA's with known functions are still fairly small. Understanding its features requires powerful in-vivo/vitro and in-silico methods, and this analysis allows the characterization of regulators and their targets. A few known miRNA functions include leaf development, timing of larval transitions, apoptosis and fat metabolism, and insulin secretion.

The discovery of miRNA's have led to the research and identification of other noncoding RNA, such as tRNA and rRNA. However, identifying correlations in RNA differs from that of DNA or proteins because, with DNA or proteins, the sequence similarities are enough to identify the functional correlations. For RNA, this is not enough. Its functionality is also tied to its secondary structure, and the fact that it has weaker nucleotide bonds than DNA allows for it to have far more flexibility in its structure. Therefore, in order to effectively align RNA sequences, we must account for secondary structures in addition to the sequences themselves.

The secondary structure of RNA features many loops (both internal and external hairpin loops), multi-loops, bulges, and pseudoknots. See Figure 1.1 for details. The hairpins help to stabilize the formation of complexes for coworking elements, such as proteins. In order to reduce the runtime of our alignment algorithms and also to simplify, we will ignore pseudoknots. See Eddy and Rivas [27] for a more detailed discussion of aligning using pseudoknots.

1.3 SEPA **Introduction**

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 [13] and Siegmund-Yakir [28], but none have proven completely satisfactory. Hence, we discuss SEPA (Segment Eval-

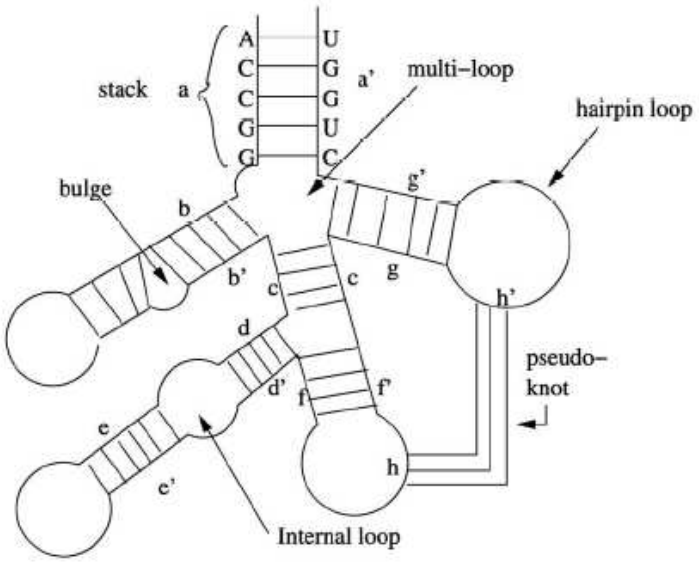


Figure 1.1: Here is an example for a secondary structure of an RNA with bulges, hairpin loops, internal loops, multi-loops, and pseudoknots.

uator 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 and EMBOSS. We will also use SEPA to compare the results of PLANAR with RSEARCH and Vienna. 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. 7.2) Furthermore, PLANAR shows similar promise for RNA secondary structures.

Chapter 2

Literature Survey

2.1 DNA 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:

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

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 would 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

The 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 obtained thus far. We next glue M_l and M_r to make a 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 obtain 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 amount of computational time consumed by the Hirschberg process, then we can quantify it as:

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

Where k need not be a constant. From this, we see that $T(m,n)=O(mn)$. Hence, Hirschberg method, while using $O(n)$ space for the tables to get an alignment, does not increase overall runtime by more than 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:

$$\begin{aligned} V(0,0) &= 0; \\ V(i,0) &= E(i,0)=-w(i), \\ V(0,j) &= F(0,j)=-w(j); \end{aligned}$$

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

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 the runtime and memory usage is unacceptably large for most computational biology applications.

2.1.5 Miller-Meyers Linked-List Assistance

Miller and Myers [19] uses the same formula as Needleman-Wunsch, but they assume $w(\cdot)$ is convex (meaning that it has a negative double-derivative). This assumption enables them to take advantage of a Linked-List Assistance technique to improve on runtime. This technique involves considering possible solutions for the $E(\cdot,\cdot)$ and $F(\cdot,\cdot)$ entries before one explicitly computes 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 $k' \le k$, 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 $cand_{i-1}(i) = cand(i)$.)

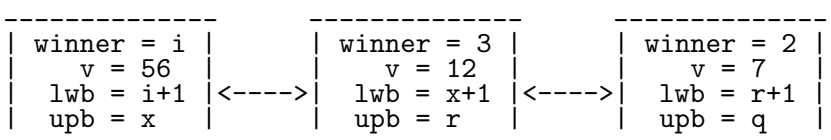
Then, on the i'th iteration, with $i' < i$, once we figure out what $V(i')$ is, we can simply take $k' = cand_{i'-1}(i)$, and compare $eval(k',i)$ with $eval(i',i)$, and whichever of these two values is greater dictates $cand_{i'}(i)$. When $i' = i-1$, this gives us $cand(i)$, and thus on the ith iteration, 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.

[2]The $O(m+n)$ lookups to previous rows and columns implies we can't take advantage of the Hirschberg reduction to cut down on space, unless we can make certain assumptions about the gap function $w(\cdot)$.

- (S2) In all practical cases, our gap function w is convex (meaning that $w(i)$ 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 conclude that 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 as 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 $(i+1)$st 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, the algorithm discovers an element in L such that $cand_i(lwb) = i$, but $cand_i(upb) \neq i$ and needs to know which index within the group is the largest a such that $cand_i(a) = i$, then, it simply takes the element's previous winner value of k, its v value, and considers for x the plots of $v - w(x - k)$ versus $V(i) - w(x - i)$. Thus, 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 search, we discover 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 step with the binary intersection, we are able to 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 observation 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 RNA Alignment Overview

2.2.1 FastR Binarization

Binarization is the process of converting a sequence X of length m into a binary tree based on its secondary structure. Depending on how we binarize X, the alignment result could differ. FastR's Binarization implementation used in [34] works as follows. Note: (i, j) is in S if and only if $X[i]$ and $X[j]$ are bound to each other in X's secondary structure.

```
Node Binarize(i,j) {
    /* Binarize the interval(i,j). */

    if (i == j) {
         /* Return a dotted note with no children. */
         return (new Node(i, j, dotted, NULL));
    }
    else if (i, j) is in S {
         /* Return a solid node with one child v. */
         v = Binarize(i+1, j-1);
         return (new Node(i, j, solid, v));
    }
```

```
    else if for some k where i < k < j,
                            we have (k, j) is in S {
         /* Return a dotted node with two children,
            vl and vr. */
         vl = Binarize(i, k-1);
         vr = Binarize(k, j);
         return (new Node(i, j, dotted, vl, vr));
    }
    else if (i < j) {
         /* Return a dotted node with one child v. */
         v = Binarize(i, j-1);
         return (new Node(i, j, dotted, v));
    }
}
```

Note that in creating the tree T_X from X using the approach mentioned here, solid nodes denote positions where elements of X are bound to each other, and dotted nodes denote unbound positions of X and structural bifurcations (positions in X's secondary structure where the structure branches off in two or more directions). The tree T_X created from X in this manner has at most $2m - 1$ nodes, and this fact can be proven inductively.

Tne method mentioned here is one way to create a binary tree from X. Suppose for some values i and j that $X[i]$ and $X[j]$ are not bound to each other, but are located in the same hairpin. One binary tree T_X could place the nodes containing nodes $X[i]$ and $X[j]$ near each other, but another tree T'_X could place $X[i]$ and $X[j]$ in completely separate branches. See figure 2.1 for more details. Hence, if we use a length-dependent gap function $ww(\cdot)$ to align, much like that used by PLAINS, and we are aligning X to a gap, we can easily bulk $X[i]$ and $X[j]$ together in computing $ww(\cdot)$ for tree T_X (and it might make sense to do this), whereas it is difficult (possibly inefficient algorithm-wise) to achieve the same effect for tree T'_X.

2.2.2 Structural Alignment

Often in this thesis, we will be concerned primarily with aligning X against Y where we only concern ourselves with X's secondary structure. We will adjust the alignment afterwards using Y's secondary structure as needed. The Liu paper [18] contains a discussion of aligning X and Y using both X and Y's secondary structures to generate the original alignment. However, it is achieved by accounting for multi-branches in X and Y with an all-or-nothing style, becoming over-reliant on the secondary structure.

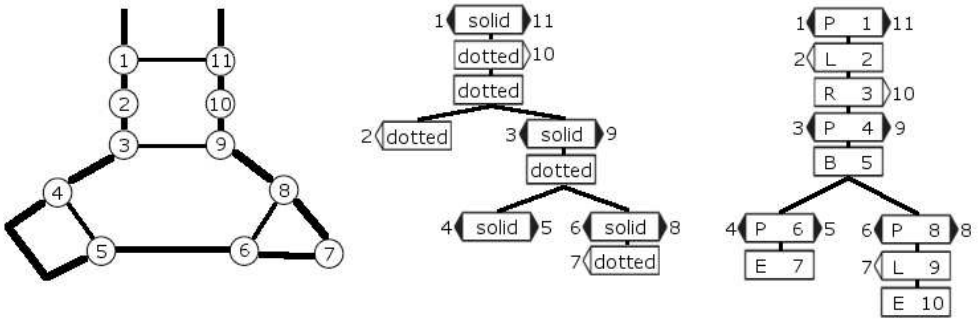


Figure 2.1: In the left drawing, we see a sample secondary structure for an RNA sequence X with indices numbered 1 thru 11, and bonds between node pairs $(1, 11)$, $(3, 9)$, $(4, 5)$, and $(6, 8)$. The middle and right drawings shows how FastR and CMSAA would respectively Binarize X into a tree. The key to notice here is how these two Binarizations create different trees from the same secondary structure, with CMSAA's version making a longer linear chain directly beneath the top node. CMSAA's Binarization bulks adjacent positions of X into the same linear chain more often than FastR. Because doing this allows for an easier implementation of length-dependent gap penalty functions, PLANAR Binarizes in the same manner as CMSAA.

This approach is flawed, since it gives us an incorrect result when X and Y have trees looking like that of figure 2.2.

2.2.3 FastR Alignment

We will begin describing the FastR alignment method by first introducing some notation. Our goal is to align X and Y using just the tree T_X (of size $O(m)$) generated for X by FastR's Binarization. Suppose S' denotes the set of all the nodes in T_X, and S denotes the bound nodes in T_X (i.e., the solid nodes in T_X). Then, $S' - S$ are all the dotted nodes of T_X, i.e., they are the nodes representing either unbound positions of X, or bifurcations, or extra positions.

Next, suppose $A[i, j, v]$ is the best alignment for the subtree of node v (a node within T_X) aligned against $Y[i : j]$ (the substring of Y ranging from position i to j). Let l_v and r_v denote respectively the indices for the left and right characters which node v represents from X. In the case that v only represents one position in X, we will assume that position is r_v. This representation relies on the manner in which T_X was constructed from FastR's Binarization process

mentioned earlier.

Let $b(\cdot,\cdot,\cdot,\cdot)$ denote the bound position score (i.e., match or mismatch score between X and Y for positions of X that are bound), and let $s(\cdot,\cdot)$ denote the unbound position score (i.e., match or mismatch score between X and Y for positions of X that are unbounded).

FastR assumes that the cost of each gapped position is constant. I.e., for a gap of length i, the penalty is $w(i) = d \cdot i$ where d is some constant. This assumption implies that there is no penalty for opening a gap, or a length-dependent change in penalty.

With all of this, the formulation for $A[i,j,v]$ is as follows:

- If v is in S (v represents a solid bound node):

$$A[i,j,v] = \max \begin{cases} A[i+1, j-1, v.child] + b(Y[i], Y[j], X[l_v], X[r_v]) \\ A[i, j-1, v] + s('-', Y[j]) \\ A[i+1, j, v] + s('-', Y[j]) \\ A[i+1, j, v.child] + s(X[l_v], Y[i]) + s(X[r_v], '-') \\ A[i, j-1, v.child] + s(X[l_v], '-') + s(X[r_v], Y[j]) \\ A[i, j, v.child] + s(X[l_v], '-') + s(X[r_v], '-') \end{cases}$$

- If v is in $S' - S$ and v has one child (v represents an unbound dotted node):

$$A[i,j,v] = \max \begin{cases} A[i, j-1, v.child] + s(X[r_v], Y[j]) \\ A[i, j, v.child] + s(X[r_v], '-') \\ A[i, j-1, v] + s('-', Y[j]) \\ A[i+1, j, v] + s('-', Y[i]) \end{cases}$$

- If v is in $S' - S$ and v has two children (v represents a bifurcation node):

$$A[i,j,v] = \max_{i \le k \le j} \{A[i, k-1, v.left] + A[k, j, v.right]\}$$

If m_1 and m_2 denote the number of nodes in T_x with one/two children, where $m_1 + m_2 = O(m)$ and generally, $m_2 << m_1$, then the runtime for FastR is $O(n^2 m_1 + n^3 m_2)$, and the space usage is $O(n^2 m)$. We can reduce speed and memory using banding. The idea for banding is to, for each node v in T_X, limit the number of positions of Y applied based on the width of v's endpoints. If d_m is the size of this banded region, where $d_m << m$, then the runtime and memory reduce respectively to $O(n^2 d_m^2)$ and $O(n^2 d_m)$.

All of this makes FastR fast for most noncoding RNA's of length 100-200 bp. Its accuracy in checking the majority of valid endpoints for Y against T_X in the alignment allows it to predict the secondary structure for Y with good accuracy, especially when Y is a riboswitch.

However, this technique has some disadvantages. A length-dependent gap formula will not correctly work with T_X in the manner built by FastR. Furthermore, length-dependent gap formulas are useful for noncoding regions. Also, narrow-banding fails to correctly align sequences of expected large and varying gap lengths. Therefore, later on, we will present PLANAR, which addresses some of these issues.

2.2.4 CMSAA Binarization

In Eddy's paper [5], we are introduced to the CMSAA (Covariance Model Structural Alignment Algorithm). This algorithm features a few changes from FastR. The first change is in the manner of Binarization for X to make T_X based on its secondary structure. The other change is that, instead of labeling nodes as solid or dotted, we will label them as 'P', 'L', 'R', 'B', or 'E' depending on if the node corresponds to representing a "Pair" of bound positions in X, just the "Left" position in X, just the "Right" position in X, a "Bifurcation" position in X, or an "Endpoint" in X (corresponding to leaf nodes in T_X, and from an algorithmic perspective, this makes exploring T_X simpler). For simplicity purposes, we are omitting the "Start" node, or 'S' node mentioned in the Eddy paper, but the construction of T_X from X remains the same otherwise. Note here that like before, (i, j) is in S if and only if $X[i]$ and $X[j]$ are bound to each other in X's secondary structure. CMSAA's Binarization to make T_X from X works as follows:

```
Node CMSAA_Binarize(i,j) {
    /* Binarize the interval(i,j). */
    if (i > j) {
         /* Return a leaf node with no children. */
         return (new Node(i, j, 'E', NULL));
    }
    else if (i, j) is NOT in S and X[i] is unpaired {
         /* Return an 'L' node with one child v. */
         v = CMSAA_Binarize(i+1, j);
         return (new Node(i, j, 'L', v));
    }
    else if (i, j) is NOT in S and X[j] is unpaired {
         /* Return an 'R' node with one child v. */
```

```
        v = CMSAA_Binarize(i, j-1);
        return (new Node(i, j, 'R', v));
    }
    else if (i, j) is in S {
        /* Return a 'P' node with one child v. */
        v = CMSAA_Binarize(i+1, j-1);
        return (new Node(i, j, 'P', v));
    }
    else if for some k where i < k < j,
                            we have (i, k) is in S {
        /* Return a 'B' node with two children,
           vl and vr. */
        vl = CMSAA_Binarize(i, k);
        vr = CMSAA_Binarize(k+1, j);
        return (new Node(i, j, 'B', vl, vr));
    }
    else if (i <= j) {
        /* Return a leaf node with no children. */
        return (new Node(i, j, 'E', NULL));
    }
}
```

In figure 2.1, we see a comparison of the same secondary structure made into trees using FastR's Binarization versus CMSAA's Binarization. Notice how with CMSAA's version, we bulk adjacent positions of X into the same linear chain more easily, resulting in slightly fewer bifurcations. Also, note that the asymptotic size of T_X constructed from CMSAA's Binarization remains as $O(m)$.

2.2.5 CMSAA Alignment Formula

The CMSAA alignment formula is similar to that of FastR, except that the solid and dotted node cases are substituted with appropriately labeled 'P', 'L', 'R', 'B', and 'E' node cases. Using notation similar to FastR, the formulation for $A[i, j, v]$ for CMSAA is as follows:

- If v's label is 'E', or $i > j$ (these are the base cases), then:

$$A[i, j, v] = 0$$

- If v's label is 'P', then:

$$A[i,j,v] = \max \begin{cases} A[i+1, j-1, v.child] + b(Y[i], Y[j], X[l_v], X[r_v]) \\ A[i, j-1, v] + s('-', Y[j]) \\ A[i+1, j, v] + s('-', Y[j]) \\ A[i+1, j, v.child] + s(X[l_v], Y[i]) + s(X[r_v], '-') \\ A[i, j-1, v.child] + s(X[l_v], '-') + s(X[r_v], Y[j]) \\ A[i, j, v.child] + s(X[l_v], '-') + s(X[r_v], '-') \end{cases}$$

- If v's label is 'L', then:

$$A[i,j,v] = \max \begin{cases} A[i+1, j, v.child] + s(X[l_v], Y[i]) \\ A[i, j, v.child] + s(X[l_v], '-') \\ A[i+1, j, v] + s('-', Y[i]) \\ A[i, j-1, v] + s('-', Y[j]) \end{cases}$$

- If v's label is 'R', then:

$$A[i,j,v] = \max \begin{cases} A[i, j-1, v.child] + s(X[r_v], Y[j]) \\ A[i, j, v.child] + s(X[r_v], '-') \\ A[i, j-1, v] + s('-', Y[j]) \\ A[i+1, j, v] + s('-', Y[i]) \end{cases}$$

- If v's label is 'B' (v represents a bifurcation node), then:

$$A[i,j,v] = \max_{i-1 \leq k \leq j} \{A[i,k,v.left] + A[k+1, j, v.right]\}$$

Ignoring the banding discussion mentioned earlier, the runtime and space usage are $O(n^2 m_1 + n^3 m_2)$ and $O(n^2 m)$ respectively, just like for FastR, where m_1 is the number of nodes with one child, and m_2 is the number of nodes with two children. However, the interesting thing here is how CMSAA extends this alignment method to reduce the space used at runtime.

2.2.6 CMSAA Space Reduction

CMSAA is able to reduce its space from $O(n^2 m)$ to $O(n^2 \log n)$ in a way similar to Hirscherg's space reduction from section 2.1.3, except modified slightly for RNA secondary structure.

The space reduction works as follows. First, we only save the t most recently computed values for nodes v from T_X, where t is a constant ≥ 2. One way of

putting this is that we only save the t most recently computed $n \times n$ "decks". This reduces our space from $O(n^2 m)$ to $O(n^2 t)$. Next, our alignment computation is split into the following methods: INSIDE(), OUTSIDE(), VINSIDE(), VOUTSIDE(), V_SPLIT(), WEDGE_SPLIT(), and GENERIC_SPLIT().

INSIDE($[r, z], [g, q]$) computes $A[i, j, v]$ in the manner mentioned in the previous section for all nodes v from T_X with indices ranging within interval $[r, z]$, and the row-column Y indices i and j ranging within interval $[g, q]$, and we only save the t most recently computed values for v. Later on, we will create an alignment tree T_A that features linear chains of nodes corresponding to single/double match/mismatch positions and gapped positions over T_X and Y, plus bifurcations corresponding to places where T_X bifurcates. Representing an alignment as a tree T_A makes it more convenient to conceptualize how the indices of Y are split at bifurcation points in T_X, and we could always linearize T_A by simple depth-first-search traversal over it later on, if we so desire.

OUTSIDE($[r, z], [g, q]$) computes $A^r[i, j, v]$ where nodes v from T_X proceed in reverse over the interval $[r, z]$, and row-column Y indices i and j proceed in reverse over interval $[g, q]$, and we save only the t most recently computed values for v. We require that nodes r through z form a linear chain in T_X with none of the nodes from r through z being bifurcation nodes.[3] Assuming interval $[r, z]$ saitsfies this requirement, note that $A^r[\cdot, \cdot, \cdot]$ is computed as the perfect reverse of $A[\cdot, \cdot, \cdot]$, and this resembles how table V^r is computed in the reverse-direction of table V for DNA alignments in section 4.1.2.

VINSIDE($[r, z], [g, i'], [j', q]$) works the same way as INSIDE($[r, z], [g, q]$) except that the Y indices proceed differently. For VINSIDE, the Y row-index i iterates over interval $[g, i']$, and the Y column-index j iterates over interval $[j', q]$. This gives us the luxury of saving the computation of z's subtree aligned against Y indices within $[i' + 1, j' - 1]$ for another point in our algorithm where it is needed more.[4] Predictably, VOUTSIDE($[r, z], [g, i'], [j', q]$) works like OUTSIDE($[r, z], [g, q]$) except that the Y row-index i iterates over interval $[i', g]$, and the Y column-index j iterates over interval $[q, j']$. (We still require nodes r through z to form a linear chain in T_X without any of those nodes being bifurcations.)

V_SPLIT($[r, z], [g, i'], [j', q]$) works over a linear-chain of unbifurcated nodes from T_x within interval $[r, z]$, using Y row-index i within interval $[g, i']$ and column-index j within interval $[j', q]$ in the following way: Let w denote the half-point index of interval $[r, z]$. We proceed to align VOUTSIDE($[r, w], [g, i'], [j', q]$) and VINSIDE($[w + 1, z], [g, i'], [j', q]$), saving the t most recently computed

[3] This differs from the computation of INSIDE, which allows bifurcation nodes in its computation.

[4] Note that for INSIDE, both the Y row-index i and the column index j iterate over the same interval $[g, q]$.

decks of A^r and A. Next, of the entries saved, we select (v, i, j) such that $A[i, j, v] + A^r[i, j, v]$ is maximized, and use it to trace backwards over A and A^r as far back as we can, saving the results as the alignment tree T_{M2}. Suppose tracing tables A^r and A backwards got us respectively to entries $(\tilde{v}, \tilde{i}, \tilde{j})$ and $(\tilde{v'}, \tilde{i'}, \tilde{j'})$. We then proceed recursively over V_SPLIT($[r, \tilde{v}], [g, \tilde{i}], [\tilde{j}, q]$) to get an alignment tree T_{M1}, and recursively over V_SPLIT($[\tilde{v'}, z], [\tilde{i'}, i'], [j', \tilde{j'}]$) to get an alignment tree T_{M3}. We combine T_{M1} to T_{M2} and T_{M3} to get an overall alignment tree T_M, which we return.

Overall, V_SPLIT() essentially works by taking an unbifurcated chain in T_X, and aligning its halfpoint nodes against Y and proceeding recursively over the upper and lower portions of this chain and the appropriate indices of Y to get the rest of the alignment.

WEDGE_SPLIT($[r, z], [g, q]$) operates similarly to V_SPLIT. It works over a linear-chain of unbifurcated nodes from T_X within interval $[r, z]$, just like V_SPLIT. However for WEDGE_SPLIT, both the Y row and column indices i and j iterate over interval $[g, q]$, and the rest of its behavior is modified accordingly. This means that calls to VOUTSIDE() and VINSIDE() are respectively substituted with calls to OUTSIDE() and INSIDE(), and T_{M1} and T_{M3} are constructed by respective calls to V_SPLIT() and WEDGE_SPLIT() (instead of two V_SPLIT() calls).

Overall, WEDGE_SPLIT() works exactly like V_SPLIT() except that it also handles the "middle" indices of Y.

GENERIC_SPLIT($[r, z], [g, q]$) is the starting point to the alignment algorithm. It works over nodes from T_X with indices of interval $[r, z]$, and this interval could possibly include nodes with bifurcations (unlike the V_SPLIT and WEDGE_SPLIT cases). Its Y row and column indices i and j range over interval $[g, q]$. It proceeds as follows: If T_X does not contain any bifurcation nodes within the $[r, z]$ interval, then we merely perform WEDGE_SPLIT($[r, z], [g, q]$), since that will solve our alignment for that case. When T_X DOES contain a bifurcation, then suppose u is the index of the highest bifurcation node within $[r, z]$. Note that interval $[r, u-1]$ does not contain any bifurcation nodes (though $[u+1, z]$ might). Therefore, we align OUTSIDE($[r, u-1], [g, q]$) and INSIDE($[u+1, w-1], [g, q]$) and INSIDE($[w, z], [g, q]$), where w represents the leftmost index of u's right subtree, saving the results respectively to tables A^r, A, and B.[5] Note that we are only saving the t most recently computed "decks" of tables A^r, A, and B, and the saved "decks" are all located next to node u. We now select (i, k, j) such that $A^r[u-1, i, j] + A[u+1, i, k] + B[w, k+1, j]$ is maximized. We then use it to backtrace A^r, A, and B as far as we can, recording

[5]We make two calls to INSIDE() here, so we assume the first call records in table A, and the second call records in table B.

all results in the alignment tree T_{M2}. Suppose backtracing A^r, A, and B gets us respectively to indices (r', i', j'), $(\tilde{w}, \tilde{i}, \tilde{k})$, and $(\tilde{w'}, \tilde{k'}, \tilde{j'})$. We then perform V_SPLIT($[r, r'], [g, i'], [j', q]$) to get the alignment tree T_{M1}, and the recursions GENERIC_SPLIT($[\tilde{w}, w-1], [\tilde{i}, \tilde{k}]$) and GENERIC_SPLIT($[\tilde{w'}, z], [\tilde{k'}, \tilde{j'}]$) to get alignment trees T_{M3} and T_{M4}. We then glue T_{M1} to T_{M2} and T_{M3} and T_{M4} to get our overall result T_M, which we return.

In essence, GENERIC_SPLIT() searches for the highest bifurcation node in T_X and records the alignment of all nodes immediately around this node against Y. We then proceed recursively over the unrecorded portions above and below this node with the appropriate indices of Y to get the rest of the alignment.

We perform GENERIC_SPLIT($[1, |T_X|], [1, n]$) to get the overall alignment, where the size of T_X is $|T_X| = O(m)$. The overall runtime for CMSAA is at most a constant factor larger than FastR, which means that it is asymptotically the same $O(n^2 m_1 + n^3 m_2)$, where m_1 is the number of nodes of T_X with one child, and m_2 is the number of nodes with two children. The memory usage for CMSAA is $O(n^2 t \log m)$. The $\log m$ factor is introduced because we make a recursion at each bifurcation node in T_X, as well at each "halfpoint" of the linear chains within T_X. [6] Since t is constant, the memory usage comes out to be $O(n^2 \log m)$.

2.3 p-Value Methods

2.3.1 Karlin-Altschul

The Karlin-Altschul approach in [13] 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;

[6] Note: The worst case runtime and space usage for CMSAA actually turn out to be $O(n^3 m^2)$ and $O(n^2 m)$. However, this is not a serious problem. Changing GENERIC_SPLIT to do its split at a "middle" bifurcation node rather than the highest one in T_X reduces the runtime. And, relabeling the some of the indices of T_X prior to aligning so that we visit right children before left children at certain bifurcation nodes reduces the memory usage. However, in practice, neither of these need to be done for most RNA cases. Further details can be found in [5].

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:

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

2.3.2 Multiple Karlin-Altschul

Building on the previous p-value formula, Karlin-Altschul[14] approximate the probability of getting r or more strips of score at least S as:

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

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:

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

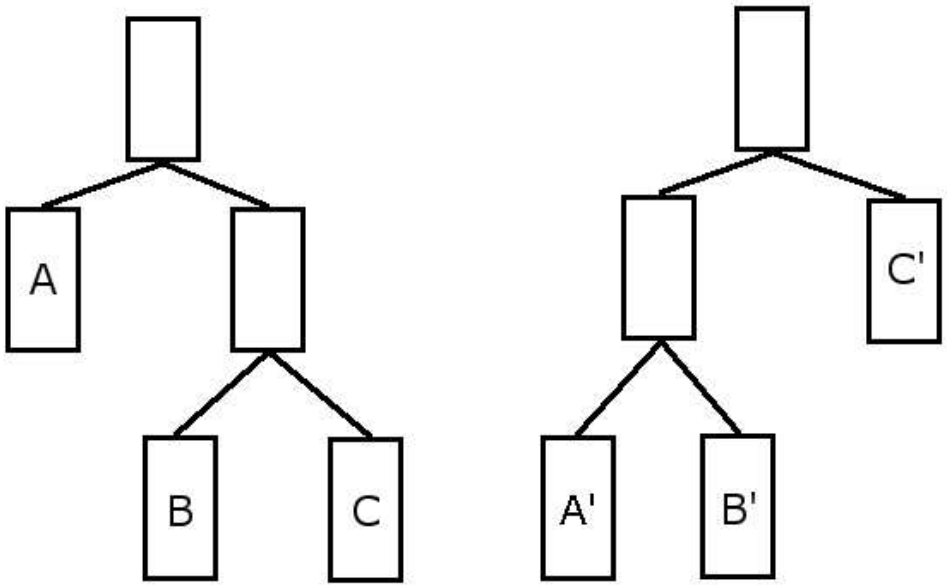


Figure 2.2: Here is an example of two RNA sequences with known secondary structures. Suppose each rectangle is a linear chain of nodes with the bottom-most node being either a bifurcation node or a leaf. Next, suppose that the linear chains corresponding to A and A' have very high homology (such that aligning A against A' would score much higher than aligning A against anything else). And similarly, suppose chains B and B' have high homology to each other, and chains C and C' also have high homology to each other. The optimal alignment should involve A aligned to A', B aligned to B', and C aligned to C'. However, if we align both of these sequences using both of their secondary structures, and approach aligning each left or right branch in an all-or-nothing manner, we would miss the optimal alignment. We could get the optimal alignment if we carefully consider all possible sub-forests for the two sequences, but being able to correctly do this for sequences of complex secondary structures becomes prohibitively expensive in terms of runtime.

Chapter 3

Overview

The remainder of this paper is structured as follows: Chapter 4 overviews how PLAINS aligns, and describes how PLAINS handles its "log function to piecewise-linear function" approximation and parameter optimization. Chapter 5 overviews how PLANAR aligns, and its uses of secondary structures to combine alignments and optimize parameters. Chapter 6 describes how SEPA works over alignments, and how its p-value approximation is derived. Chapter 7 describes Colorgrid scheme used over alignments, as well as the empirical results found from comparing PLAINS and PLANAR to similar algorithms using SEPA. The final chapter 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. 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, proofs for its non-trivial portions, and an explanation for how it approximates a log function using a piecewise-linear function, decides what a "best alignment" is, and optimizes parameters for alignments. PLAINS stands for Piecewise-Linear Alignment with Important Nucleotide Seeker.

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 [19], 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 $O(mn \log p)$ time and $O(np)$ space.

4.1.1 Modified Linked-List Assistance

PLAINS uses the same Linked-List Assistance technique mentioned in Chapter 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 modification implies 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 performing a binary search over the lines of these curves, instead of a binary search 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 technique reduces the overall runtime from $O(mn \log m)$ to $O(mn \log p)$.

Second, because we are 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 fact is deferred to section 4.2.1. This property implies that all of our lists combined take up $O(np)$ space. In section 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 are 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 sufficient for the V table).

Then, at 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 section 4.2.2. 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.

4.2 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, but are done so here. 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

[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.

"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.

4.2.1 Proof for the $O(np)$ Space Bound

Earlier, we stated that we are using $O(np)$ space in the 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 with $V(i)$ and $F(i)$ over all 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 it is seen 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 that we must have found $p-1$ intersections, each from a different plot. Equivalently, we must have had $cand(\cdot)$ 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 intersecting 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 all use $O(p)$ space, and similarly, R and R^r also all use $O(p)$ space. Hence, total space used by all of the lists is $O(np)$. QED

4.2.2 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[3]at least 1.

[3]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.

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 $cand.(i)$ 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 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.

4.2.3 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[4], it is feasible to align $X[1..m/2]$ against Y separately from aligning $X[m/2..m]$ against Y. Furthermore[5], 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.

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.

4.3 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

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

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

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[6]: $\alpha \ln(i+1) + \beta$. 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[7], and wo is set to β.

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 6, 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 [10]. 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 is unsurprising, 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 6 to compare the results 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 method is resemblant of LAGAN's techniques to account for the nature

[6]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 $i = 0$.

[7]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 continue off into infinity.

of certain species in performing its alignments.[8] In contrast, EMBOSS and LALIGN each use a fixed set of gap/mismatch parameters for all species. We present a figure 4.1 showing the piecewise-linear gap functions that PLAINS came up with for each species pair[9]. A comparison of the alignment results from PLAINS and the other alignment tools can be found in Chapter 7.

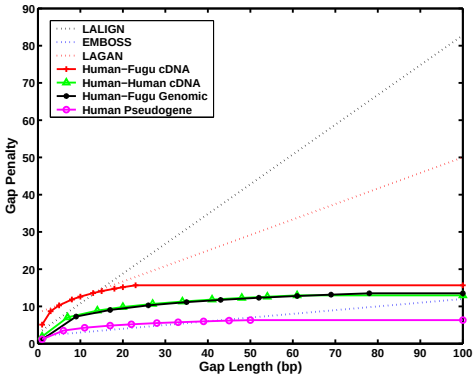


Figure 4.1: The Piecewise-Linear Gap Functions that PLAINS came up with in optimizing the score for different species pairs, along with the rescaled gap-paramters the other tools use. Note that the LAGAN gap paramters shown here are its default paramters. LAGAN uses a number of unspecified gap parameters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

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.

[8]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.

[9]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.

Chapter 5

Planar

We now explain several aspects of PLANAR, its general notations, its alignment method, how it combines two alignments in order to properly account for secondary structures in both sequences, and how it optimizes parameters. PLANAR stands for Piecewise-Linear Alignment for Nucleotides Arranged as RNA.

PLANAR holds two key differences in its design compared to FastR and CMSAA. First, it uses length-dependent piecewise-linear gap functions, not just a linear gap formula without a penalty for opening a gap. Second, after aligning sequences X and Y using X's secondary structure, PLANAR explicitly "corrects" the alignment using Y's secondary structure, if one is supplied.

5.1 Details of the PLANAR Gap Function in Secondary Structures

This section is dedicated to explaining the intuition behind the gap system used over secondary structures by PLANAR. PLANAR uses the same p-part piecewise-linear gap function $ww(\cdot)$ in its alignments as PLAINS. The secondary structure involved with RNA alignments might make it optimal to, at some points, use a small gap, and at other points, use a larger gap. The nature of the piecewise-linear gap functions used in PLAINS accommodates this sort of distribution easily, which is why we have chosen to use it also in PLANAR. We chose to dedicate an entire section to the discussion of the gap usage for RNA by PLANAR, because FastR and CMSAA bypass some of the issues mentioned here due to their choice of a different type of gap formula.

In figure 2.1, we see four nodes bulked together in T_X even in spite of the fact that the derived indices of X involved are far apart (with the pairings from X's secondary structure bringing them together). For this case, it might better bring out homologies between other locations in T_X if all four of these nodes

are aligned against a gap, and this gap be penalized as a reduced $ww(4)$ instead of a higher value. Furthermore, the reason for penalizing this gap as length 4 instead of length 6 (even though the nodes represent 6 base pairs from X) is to better encourage an alignment based on the secondary structure. Therefore, PLANAR penalizes adjacent nodes within linear chains of T_X together based on the number of involved nodes, regardless of if the indices they are derived from in X are far apart or are more numerous than the amount of nodes they are represented by in T_X. [1]

Next, we recall from section 2.2.4 how CMSAA bulks adjacent positions of X into the same linear chain more easily than FastR's Binarization, and as a result, also has slightly fewer bifurcations, with figure 2.1 elaborating. PLANAR binarizes X into T_X the same way as CMSAA, since it allows for easier penalization of adjacent nodes in a linear chain using length-dependent piecewise-linear gap functions. Furthermore, avoiding unneccesary bifurcations help to reduce the overall algorithmic complexity of an alignment, even if only by a constant factor.

However, in spite of all of this, there is still the issue of representing gaps around bifurcation points. Suppose we have a bifurcation node v with q_p nodes above it forming a linear chain, and q_l nodes forming a linear chain in its left subtree, and q_r nodes forming a linear chain in its right subtree, where $q_l \neq q_r$. See figure 5.1 for an example. Now, suppose we wish to consider the alignment of the bottom-most $u_p \leq q_p$ nodes directly above v, and the topmost $u_l \leq q_l$ nodes in v's left subtree, and the topmost $u_r \leq q_r$ nodes in v's right subtree, where u_p, u_l, and u_r are all undetermined. An intuition into the nature of $ww(\cdot)$ would suggest that the most appropriate gap penalty to use here should be $ww(u_p + u_l + u_r)$. However, we would need to iterate all possible values for u_p, u_l, and u_r over the linear chains directly above and below node v, and this would make PLANAR a magnitude of $O(m^3)$ times more complex than it needs to be.[2] Furthermore, taking advantage of X's secondary structure suggests that, when aligning T_X against Y, we should treat the linear chains above and left/right of v independently of each other. Because of this intuition, and for algorithmic simplicity purposes, we will treat each linear chain in T_X independently of each other, and not account for gaps across different linear

[1] As a result of this behavior for $ww(\cdot)$ in PLANAR, when a node in T_X has a label of 'P', its left and right characters will either both align to characters of Y, or both align to a gap (corresponding to the node itself aligning to a gap). We will not see the left character of the node align to a character of Y while the right character aligns to a gap (or vice versa), which could happen in FastR or CMSAA.

[2] One can argue that it is already algorithmically complex enough to find the correct k such that $i < k < j$ in order to split $Y[i:j]$ when aligning it against a bifurcation point in T_X.

chains when aligning at bifurcation nodes. Therefore, the penalty of the case described here will be $ww(u_p) + ww(u_l) + ww(u_r)$, instead of the difficult to find optimal $ww(u_p + u_l + u_r)$.

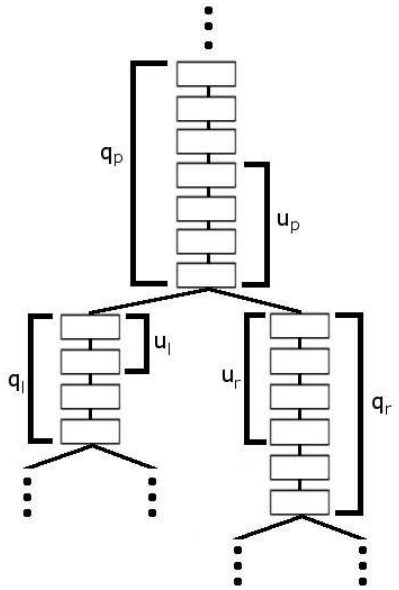


Figure 5.1: Suppose we have a bifurcation in our tree T_X, with q_p nodes above it forming a linear chain, and q_l nodes forming a linear chain in its left subtree, and q_r nodes forming a linear chain in its right subtree, where $q_l \neq q_r$. Suppose the optimal representations of gaps around this bifurcation involves the bottom u_p nodes over the bifurcation, the top u_l nodes to the left of the split, and the top u_r nodes to the right of the split, as pictured above. We would need to consider all valid values of u_p, u_l, and u_r in order to properly know the optimal alignment using a gap around the bifurcation point. This would make the runtime for PLANAR prohibitively more expensive than it needs to be.

5.2 Alignment Formulation

Given two sequences X and Y with X's secondary structure known, we construct a tree T_X using the same method as CMSAA's Binarization. We then proceed with the alignment using notation and conventions similar to PLAINS. Our goal is to align table V. Much like earlier, we will use table G to denote alignments ending at a matched or mismatched position, and table F to denote alignment ending with X (or T_X in this case) aligned against a gap. However, in PLAINS,

we used table E to denote alignments ending with Y aligned against a gap. For PLANAR, we will use two tables for this: D and E. D denotes alignments ending with the left portion of Y aligned against a gap, and E (much like earlier) denotes alignments ending with the right portion of Y aligned against a gap. As shown by the alignment formulas for FastR and CMSAA, we need to consider both ends of Y in the alignment since each node in T_X denotes indices from both ends of X.[3]

We will also make use in the alignment of a p-part piecewise-linear gap function $ww(\cdot)$, as well as $s(\cdot,\cdot)$ and $b(\cdot,\cdot,\cdot,\cdot)$ for unbound and bound position scores for matches and mismatches. Note that unlike FastR and CMSAA, the $s(\cdot,\cdot)$ and $b(\cdot,\cdot,\cdot,\cdot)$ functions are not used for gaps, just strictly for matches and mismatches.

With all of this in mind, let $V(v,i,j)$, denote the alignment for v's subtree aligned against $Y[i:j]$, where v is a node from tree T_X. Also, let $|v|$ denote the number of nodes in v's subtree, including v itself, and let $LCB(u,v)$ be a boolean statement that is true only if, for nodes u and v from T_X, we have that u is in the same linear chain as v, and u is below v. See figure 5.2 for more details. With this preliminary, the alignment formula used for PLANAR can be written as follows:

- Base Cases:
  - If v's label is 'E', then:
$$V(v,i,j) = ww(j-i+1)$$
  - If $i > j$, then:
$$V(v,i,j) = ww(|v|)$$
- Recursive Cases:
  - If v's label is 'B', then:
$$V(v,i,j) = \max_{i-1 \le k \le j} [V(v.left,i,k) + V(v.right,k+1,j)]$$

[3]We could in theory eliminate table D and just use table E to denote alignments of either ends of Y aligned against a gap, reducing the penalties resulting from applying $ww(\cdot)$. Unfortunately, this increases the overall alignment runtime by a factor of $O(n)$ from the need to inspect both the left and right ends of Y simultaneously, and empirically, the alignments do not differ drastically when contrasted to using both D and E. Therefore, the extra overhead in runtime is not justified.

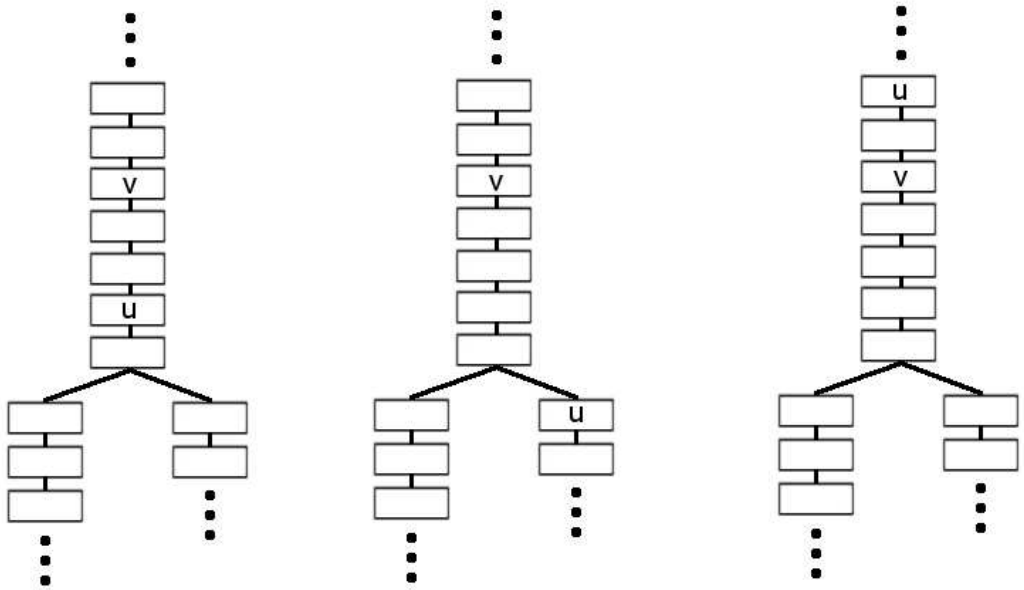


Figure 5.2: $LCB(u, v)$ is a boolean statement that is true only if, for nodes u and v from T_X, we have that u is in the same linear chain as v, and u is below v. In the trees shown above, $LCB(u, v)$ is true for the left tree, but false for the middle and right trees. It is false for the middle tree because u is not in the same linear chain as v. It is false for the right tree because, although u and v are in the same linear chain, u is above v.

– If v's label is NOT 'B', then:

$$\begin{aligned} V(v,i,j) &= \max\{D(v,i,j), E(v,i,j), F(v,i,j), G(v,i,j)\} \\ D(v,i,j) &= \max_{i+1 \le k \le j+1} [V(v,k,j) - ww(k-i)] \\ E(v,i,j) &= \max_{i-1 \le k \le j-1} [V(v,i,k) - ww(j-k)] \\ F(v,i,j) &= \max_{u\_s.t.\_LCB(u,v)} [V(u,i,j) - ww(|v| - |u|)] \end{aligned}$$

– If v's label is 'L', then:

$$G(v,i,j) = s(X[l_v], Y[i]) + V(v.child, i+1, j)$$

– If v's label is 'R', then:

$$G(v,i,j) = s(X[r_v], Y[j]) + V(v.child, i, j-1)$$

– If v's label is 'P', and $i < j$ then:

$$G(v, i, j) = b(X[l_v], X[r_v], Y[i], Y[j]) + V(v.child, i + 1, j - 1)$$

– Otherwise:

$$G(v, i, j) = -\infty$$

We traceback from $V(root, 1, n)$ to get the alignment for all of T_X aligned to $Y[1 : n]$. Using an approach similar to Needleman-Wunsch implies that our runtime is $O((m + n)n^2m_1 + n^3m_2) = O(m^2n^2 + n^3m)$ and the space usage is $O(n^2m)$. The extra runtime compared to FastR's method comes about from inspecting for the optimal k index to compute the D and E tables, as well as inspecting for optimal u to compute the F table. However, using Linked-List Assistance and CMSAA's space reduction improves both the runtime and memory usages for Planar, as discussed in the next section.

5.3 Planar Linked-List Assistance and Space Reduction

In section 2.1.5, we see Miller-Myers Linked-List Assistance applied to reduce the runtime of the Needleman-Wunsch algorithm. Using a similar tactic for the Planar alignment formula, when v is fixed, we will treat each $n \times n$ deck of form $V(v, \cdot, \cdot)$ separately (and similarly for D, E, F, and G). For each row i, we will use a list LD_i to reduce the lookups in computing $D(v, i, \cdot)$. For each column j, we will use a list LE_j to reduce the lookups in computing $E(v, \cdot, j)$. We finish computing each deck before moving on to the next one, and we can empty and reuse each list LD_i and LE_j when changing our v value. For each i and j, we make a list $R_{(i,j)}$ to reduce the lookups in computing $F(\cdot, i, j)$, and each list $R_{(i,j)}$ is updated upon inspecting the next deck. Also, the updates for list $R_{(i,j)}$ can be interweaved with the updates for lists LD_i and LE_j. With this in mind, we are able to properly compute entries for tables D, E, and F by looking up entries from our lists, instead of from previously computed table entires.

Using an argument similar to section 4.1.1, because Planar uses p-part piecewise-linear gap functions, the space used by each list is $O(p)$. Next, for all i and j, the total number of lists of form LD_i, LE_j, and $R_{(i,j)}$ is $O(n^2)$. Therefore, the total space used by all of the lists is $O(n^2p)$. In addition, these lists reduce the overall runtime for the formula mentioned earlier in section 5.2 from $O((m + n)n^2m_1 + n^3m_2)$ to $O((\log p)n^2m_1 + n^3m_2)$ because at the non-bifurcation nodes in T_X, we are only making $O(\log p)$ lookups within our lists

LD_i, LE_j, and $R_{(i,j)}$ instead of the $O(m+n)$ lookups to previously computed table entries.

In addition to all of this, PLANAR only saves the t most recently computed decks of all tables, where t is a constant ≥ 2, and PLANAR computes its final alingment tree in a manner similar to that for CMSAA's GENERIC_SPLIT methodology mentioned in section 2.2.6. This approach implies that the overall space used by PLANAR, which is the space used by the table decks plus the space used by the lists, can be bounded by $O(n^2 t(\log m) + n^2 p) = O(n^2(p + \log m))$, taking advantage of the fact that t is constant. In practice, this is drastically smaller than $O(n^2 m)$.

Furthermore, by using a methodology similar to CMSAA's GENERIC_SPLIT, which increases runtime by only a constant factor while reducing space, PLANAR is able to align with this reduced memory usage in time $O((\log p)n^2 m_1 + n^3 m_2)$. Hence, overall PLANAR uses $O((\log p)n^2 m_1 + n^3 m_2)$ time and $O(n^2(p + \log m))$ space, which is asymptotically identical to CMSAA if p is fixed.

However, there is one difference between the methodology in PLANAR versus that of CMSAA's GENERIC_SPLIT. We need to properly consider the effects of aligning a linear chain of T_X against a gap where this chain stretches from table V to V^r, our reverse table, [4] during execution of the V_SPLIT and WEDGE_SPLIT methods. This issue is resolved in a manner almost identical to that mentioned in section 4.1.2, and does not change the asymptotic runtime or space usage. We use each $R_{(i,j)}$ list during the computation of the V^r tables and $R^r_{(i,j)}$ lists to compute $rc(i,j)$ values, then use $rc(\cdot,\cdot)$ to account for a best solution involving V and V^r and a gap in between. We compare this solution against the alternate solution involving V and V^r without a gap and use whichever is the best.

5.4 Y Secondary Structure Correction

In the event that both X and Y have known secondary structures, PLANAR works as follows: PLANAR obtains T_X based on X's secondary structure, and aligns T_X to Y in the manner mentioned earlier to get the final alignment tree T_{AX}. PLANAR then obtains T_Y based on Y's secondary structure, and aligns T_Y to X to get the final alignment tree T_{AY}, working the exact same way as aligning T_X to Y, except that X and Y are swapped. Using T_{AX}, the alignment obtained using X's secondary structure, and T_{AY}, the alignment obtained using Y's secondary structure, we proceed to combine them to form the final alignment

[4] Note that because PLANAR treats each linear chain of T_X independently of each other, as mentioned earlier, we therefore need only concern ourselves with representing gap penalties accurately over linear chains, and treat bifurcations separately.

A in the following way:

We linearize T_{AX} and T_{AY} into respective linear alignments A_X and A_Y. We can make T_{AX} into A_X recursively by doing, for each node u in T_{AX}, a procedure where we place into A_X the left indices of u, then we recursively visit all of u's children nodes (if u is a bifurcation node, we will visit the left node and then the right node), then we place into A_X the right indices of u. This sequence of steps creates the correct linearized alignment A_X. Converting T_{AY} into A_Y uses the same idea. Our goal is to merge A_X and A_Y into a final alignment A. Figures 5.3 and 5.4 illustrate visually the methodology involved for this. This algorithm is described in more detail in the remainder of this section.

Our first step is to fragment A_X and A_Y into important and unimportant strips. These portions will receive special consideration when we combine A_X and A_Y to form A.

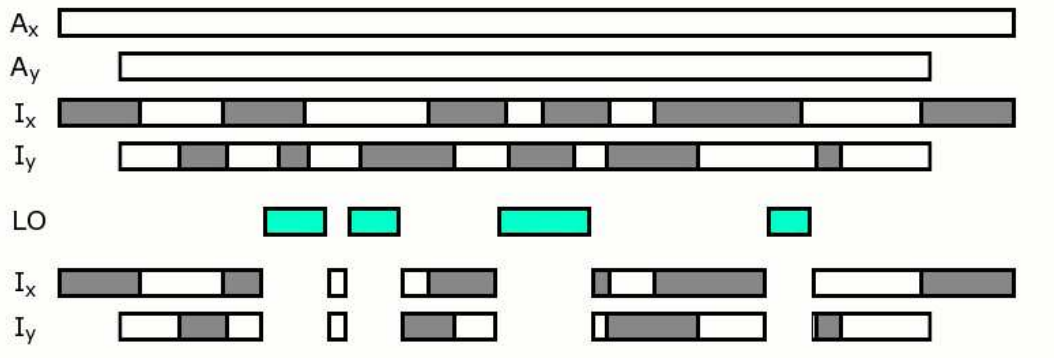


Figure 5.3: Part 1 of how PLANAR "merges" two alignments A_X and A_Y into the final alignment A. Going from top to bottom, first, I_X and I_Y represent respectively A_X and A_Y fragmented into important and unimportant segments based on alignment scores. Below that, LO consists of the identical segments of A_X and A_Y. Next, we trim the segments of I_X and I_Y based on LO.

Suppose alignment A_X is represented as a list of entries of form (x, y), where, if $A_X[i] = (x, y)$ this implies that the i_{th} position in alignment A_X represents $X[x]$ aligned to $Y[y]$. If $x < 0$, this implies that $A_X[i]$ represents $Y[y]$ aligned to a gap. Similarly, if $y < 0$, this implies that $A_X[i]$ represents $X[x]$ aligned to a gap. We do not have double-gapped positions in our alignment, so we will not see both $x < 0$ and $y < 0$ at the same time. The representation for A_Y uses the same idea. Next, suppose that a_X and a_Y are the respective lengths for alignments A_X and A_Y, and let $A_X[i : j]$ denote the subalignment of A_X ranging from indices i thru j, that is $A_X[i : j]$ denotes $A_X[i], A_X[i + 1], \ldots, A_X[j]$, and $A_Y[i : j]$ denotes a similar idea over A_Y.

We fragment A_X and A_Y respectively into sets of strips I_X and I_Y, which is similar to steps (1), (2), (4), and (5) mentioned in section 6.1. Suppose we have some fixed constants q and ω, and ρ all assuming real values in the range $[0, 1]$, let q denote a percentage for the window size to be used, and let ω denote the value used to prevent portions of A_X and A_Y of lowest match percentage from becoming considered as important strips. We use the following steps:

1. For all i from 1 to $a_X - (a_X q - 1)$, we compute the score $s_{ax}(i)$, for subalignment $A_X[i : i + a_X q - 1]$. We use $ww(\cdot)$ to penalize gaps, and $s(\cdot, \cdot)$ to score match and mismatches at positions in A_X corresponding to unbound positions of X. If for some i, $A_X[i]$ corresponds to a bound position of X that is involved in a match or mismatch, we find the i' such that the X indices used in $A_X[i]$ and $A_X[i']$ are bound to each other, and apply the corresponding X and Y characters for $A_X[i]$ and $A_X[i']$ into $b(\cdot, \cdot, \cdot, \cdot)$, and place the score at both positions $A_X[i]$ and $A_X[i']$ as $\frac{1}{2}b(X[A_X[i].x], X[A_X[i'].x], Y[A_X[i].y], Y[A_X[i'].y])$. I.e., the score that would normally be applied to both $A_X[i]$ and $A_X[i']$ gets equally split between both of them. Next, let μ and σ denote the mean and standard deviation of our $s_{ax}(\cdot)$ values. For each i, we mark[5] $s_{ax}(i)$ values as "special" if they exceed a threshold value of $\mu + \omega\sigma$. Hence, we filter away $A[i : i + a_X q - 1]$ if it fails to meet this threshold value.

2. For each u and u' (with $u \leq u'$), if $s_{ax}(u)$, $s_{ax}(u + 1)$, ..., $s_{ax}(u')$ are all marked as "special", but $s_{ax}(u - 1)$ and $s_{ax}(u' + 1)$ are not, then we consider the strip $A[u : u' + a_X q - 1]$ as important (i.e., we consider as important the strip starting the leftmost entry represented by $s_{ax}(u)$, up till the rightmost entry represented by $s_{ax}(u')$).

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

4. With all strips now representing non-overlapping regions, we then proceed to give each strip $A_X[i : j]$ its corresponding score $S(i, j)$. We also know the locations of the unimportant strips, they are merely the subalignments of A_X that do not partcipate in any important strips. For example, $j < k$ and $A_X[i : j]$ and $A_X[k : l]$ are both deemed as important strips with no other important strips between them, then clearly $A_X[j + 1 : k - 1]$ is an unimportant strip. We provide scores for the unimportant strips

[5]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.

as well. At this point, A_X has been fragmented into the important and unimportant strips, and we will let I_X denote this collection of important and unimportant strips.

5. By symmetry, we fragment A_Y into a collection I_Y of important and unimportant strips in a similar manner.

Based on empirical experimentation, setting $q = 0.1$ and $\omega = 0.5$ yields reasonably similar fragments (in terms of their lengths) with important strips scoring significantly higher than unimportant ones.

Next, unrelated to the construction of I_X and I_Y, we compute the LO collection of strips, which consists of the strips that overlap between A_X and A_Y. For alignment positions i and i', if $A_X[i].x = A_Y[i'].x$ and $A_X[i].y = A_Y[i'].y$ are both true, we will say that $A_X[i] = A_Y[i']$, otherwise, we will say that $A_X[i] \neq A_Y[i']$. This corresponds to checking if $A_X[i]$ and $A_Y[i']$ correspond to the same indices for X and Y. [6]

If a strip exists in both A_X and A_Y, we want that in our final result A. Suppose that for strips $A_X[i:j]$ and $A_Y[i':j']$, both of these strips are of the same length, and:

$A_X[i] = A_Y[i'], A_X[i+1] = A_Y[i'+1], \ldots, A_X[j] = A_Y[j']$

But, $A_X[i-1] \neq A_Y[i'-1]$ and $A_X[j+1] \neq A_Y[j'+1]$. Then we place (i, j, i', j') within the LO list. Hence, LO consists of a list of coordinates of endpoints of all locations over A_X and A_Y such that their alignments overlap.

For each entry l in LO, we assign it a score using our match/mismatch and gap functions $s(\cdot,\cdot)$, $b(\cdot,\cdot,\cdot,\cdot)$, and $ww(\cdot)$. We use match/mismatch fucntion $b()$ when a position corresponds to a bounded position of either X or Y, and apply b to the current position and the other position in the alignment it is bound, and use half of it to score the current position. This strategy uses the same approach as the similar cases in step (1) of the methodology that was used to score important/unimportant strips from A_X and A_Y. [7] The manner in which we are using b here may seem a bit unsymmetric, but since we are trying to account for highlights in A_X and A_Y over a linear alignment, and since bound

[6] If $A_X[i]$ corresponds to an index of X aligned against a gap, we will consider $A_X[i] = A_Y[i']$ to be true if and only if $A_X[i].x = A_Y[i'].x$ AND $A_Y[i']$ also corresponds to X aligned against a gap. A similar case applies when $A_X[i]$ corresponds to an index of Y aligned against a gap.

[7] Note that for some indices x and y encompassed by l, and such that $X[x]$ is aligned to $Y[y]$ in l, it might be possible that both of $X[x]$ and $Y[y]$ correspond to different bound positions in X and Y. When this happens, we appropriately apply $b(\cdot,\cdot,\cdot,\cdot)$ twice, once to the alignment position in A_X and its corresponding bound position of the alignment, the second time to the alignment position in A_Y and its corresponding bound position of the alignment, and we will "use" whichever score is higher.

matches/mismatches are scored higher than unbound matches/mismatches in the paramters typical to PLANAR, we argue that this methodology is fair for scoring common linear alignments using two secondary structures simultaneously.

Next, using LO, we trim the fragments in I_X and I_Y generated earlier so that they do not overlap with any entries in LO. We trim the strips of I_X in the following way:

Given an element t from I_X corresponding to strip $A_X[i:j]$ (meaning that its endpoints are (i,j)), suppose that for some element l in LO we see that l's endpoint indices over A_X are (i',j') (which mean that l corresponds to a strip $A_X[i':j']$), and t and l overlap. We get the following cases for this overlap:

- If $i < i' \leq j \leq j'$, then t overlaps l on the left side, so we trim t's endpoints from (i,j) into $(i,i'-1)$
- If $i' \leq i \leq j' < j$, then t overlaps l on the right side, so we trim t's endpoints from (i,j) into $(j'+1,j)$
- If $i < i' \leq j' < j$, then t completely encloses l, so we split t into two entries t' and $\tilde{t}$, where t' has endpoints $(i,i'-1)$, and $\tilde{t}$ has endpoints $(j'+1:j)$.
- If $i' < i \leq j < j'$, then l completely encloses t, so we remove t entirely.

Note that an element t from I_X might overlap with more than one element from LO. When this happens, we check t against each such element l from LO, making appropriate modifications to t (as well as for any splits that come about from t). While these modifications are made, appropriate modifications to the scores of these intervals are also made.

Elements from I_Y are modified based on their overlaps with elements from LO over A_Y in a similar manner.

After these modifications are made to I_X and I_Y, then for each element l from LO, I_X, or I_Y corresponding to either interval $A_X[i:j]$ or interval $A_Y[i':j']$, we determine the first and last X indices represented by this interval (denoted respectively as x_s and x_e), as well as the first and last Y indices represented by this interval (denoted respectively as y_s and y_e). If l has no X indices represented in its interval (meaning that l corresponds to a subalignment that is all Y characters aligned against gaps), then we discard l entirely. Similarly, we discard l if it has no Y indices represented in its interval.

Next, we construct a graph, where each element l from I_X, I_Y, or LO is a node, and for all pairs for nodes u and v, we will direct an edge from u to v if and only if $u.x_e < v.x_s$ and $u.y_e < v.y_s$, i.e., we direct an edge from u to v if it is possible for the subalignment corresponding to u to precede the subalignment

corresponding to v, where either of these subalignments can be either from A_X or A_Y. Furthermore, assuming that the score for the subalignment corresponding to v is $S(v)$ using the $s(\cdot,\cdot)$, $b(\cdot,\cdot,\cdot,\cdot)$, and $ww(\cdot)$ match/mismatch and gap functions mentioned earlier, then we will assign edge (u, v) a value of:

$$S(v) - ww(v.x_s - u.x_e - 1) - ww(v.y_s - u.y_e - 1)$$

This value is the gap penalty for any unused X and Y characters of respective indices between $u.x_e$ and $v.x_s$, and $u.y_e$ and $v.y_s$, followed by the score for the subalignment corresponding to v.

In this graph, we will also add two additional vertices. A start vertex v_s with an edge directed to all other nodes in the graph, and an end vertex v_e where all other nodes direct an edge towards it. For each node u, the weight of edge (v_s, u) is $S(u) - ww(u.x_s - 1) - ww(u.y_s - 1)$, which is gap penalty for unused X and Y characters preceding u followed by the score for u's subalignment. The weight of edge (u, v_e) is $0 - ww(m - u.x_e) - ww(n - u.y_e)$, [8] which is the gap penalty for unused X and Y characters occurring after u's subalignment.

We seek to solve the maximal path from node v_s to node v_e. Note that our graph is a DAG (Directed Acyclic Graph), so this maximal path involves a finite number of edges. Our solution can be obtained by keeping track of the number of in-edges visited thus-far for each node, as well as the predecessor node and maximal path-length for the maximal-path found thus-far for each node. This is like Dijkstra's single-source shortest-path algorithm from [3] in how we store/update predecesors and maximal path-lengths for each node, and also in how when we visit a node, we visit all of its out-edges and update the maximal path for the out-nodes touching these out-edges. However, one key difference from Dijkstra's algorithm (besides the fact that we are computing a maximal path instead of a minimal path) is that we visit a node only after all of its in-edges were visited, and that node's correct predecssor and maximal path-length has already been found[9]. Once this is done, we trace node v_e's predecessor backwards to v_s to get our maximal path.

This maximal path from v_s to v_e corresponds to our final alignment A, where each node u on this path yields a subalignment that is used in A, and unused X and Y indices for an edge (u, v) on this path correspond to ensuring that the ununsed X and/or Y characters aligned against a gap between the subalignments corresponding to u and v. Furthermore, the score $S(A)$ for alignment A is merely the sum of the values for all edges on our maximal path, which in turn is the sum of the scores for each subalignment on this path, minus the gap penalties for ununsed X and Y characters.

[8] Note that m and n are the lengths of X and Y, as mentioned earlier.

[9] And because our graph is a DAG, every node will be visited once.

Note here that for any fixed sequences X and Y with fixed trees T_X and T_Y, varying the match/mismatch scores and gap penalties used for $s(\cdot,\cdot)$, $b(\cdot,\cdot,\cdot,\cdot)$, and $ww(\cdot)$ will result in different results for A_X and A_Y (the alignments using just X's or Y's tree), and therefore give a different result for A, with a possibly different score $S(A)$. The next section uses this approach to develope a procedure for PLANAR to perform parameter optimization.

5.5 PLANAR Parameter Optimization

PLANAR uses its alignment parameters in a similar manner to PLAINS. Its p-part piecewise-linear gap function $ww(\cdot)$ behaves in the same way as PLAINS. Also, for unbound positions, where we use $s(\cdot,\cdot)$, PLANAR merely checks for match or mismatch, giving a fixed reward of 1 for match, and a penalty of ms for mismatch, just like PLAINS. However, for the case of bound positions, where we use $b(\cdot,\cdot,\cdot,\cdot)$, PLANAR checks for equality between the X and Y characters. If there is a mismatch, we give a fixed reward of 1. If there is a match, we give a fixed reward of mb, where $mb > 1$. The idea behind this scoring is that when we wish to encourage bound characters to align (match or mismatch), but we would like to give an extra reward when those characters match. Furthermore, the motivation behind fixing the mismatch reward at bound positons as 1, besides the fact that this is done in other RNA alignment algorithms, has to do with the fact that there is an "equivalent" of an unbounded position match when we align characters of a sequence of unknown secondary structure up with bounded positions of another sequence with known secondary structure, or thought up in a different way, we are "matching" a sequence of unknown structure up to a secondary structure.

PLANAR uses α, β, and d to approximate a logarithmic gap function as a p-part piecewise-linear gap function in the exact same way as done in section 4.3. Therefore, PLANAR has five variables to dictate the gap/match-mismatch parameters used in generating its alignments: $(\alpha, \beta, d, ms, mb)$. This is identical to that mentioned for PLAINS in section 4.3, except for the fifth variable mb that is introduced here. Therefore, as before, all our parameters act as a vector v, and the score of the resulting alignment A from v is quantified as the scalar $f(v)$, and our goal is to find the v that maximizes $f(v)$, which is numerical optimization, and is solved with exactly the same techniques described in section 4.3.

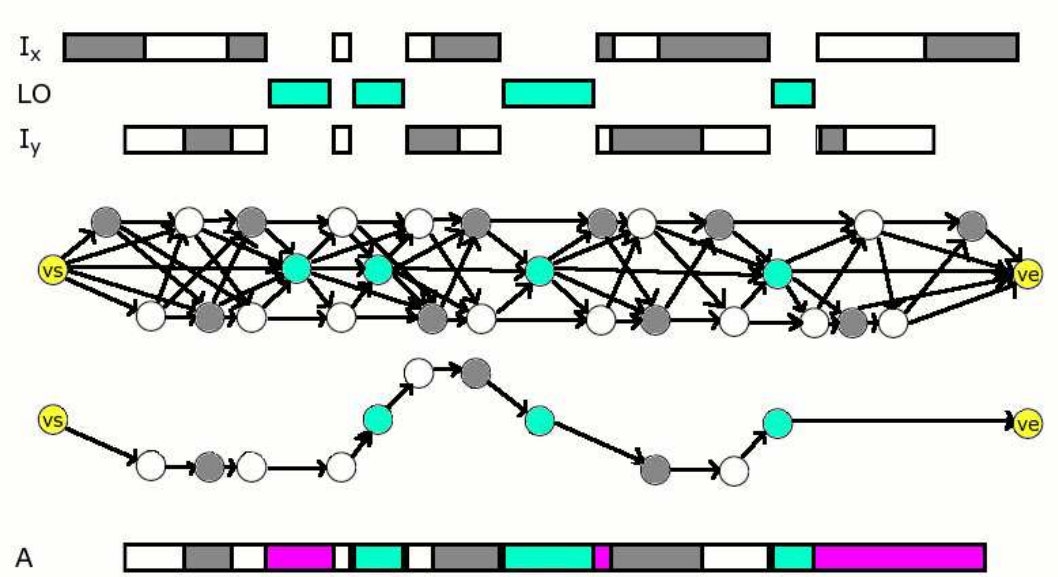


Figure 5.4: Part 2 of how PLANAR "merges" two alignments A_X and A_Y into the final alignment A. Going from top to bottom, we convert the segments of I_X, LO, and I_Y into nodes, directing an edge from node u to node v only if we can create an alignment where u's segment precedes v's segment. There is one detail not shown here: We assign a weight to each edge (u, v) as the score for v's segment minus the gap penalty for any unused X and Y characters between u and v's alignment segments. We also create a dummy source and a sink node (called respectively vs and ve). Note that a few edges have been omitted from this graph for visual simplicity. We solve maximum path algorithm over this graph. Below the graph is a drawing using only the nodes and edges involved in the optimal path. Using the segments corresponding to the nodes on this path, plus gaps for any missing X and Y characters, gives us our alignment A.

Chapter 6

SEPA

SEPA stands for Segment Evaluator for Pairwise Alignments. It takes a pre-computed pairwise alignment, identifies its important segments, and uses a p-value scheme to measure these segments, hence allowing it to "measure" the quality of an alignment.

Recalling the notation used in Chapter 4, assume that 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.

Reintroducing other notations from Chapter 4, let $ww(i)$ denote the penalty for a gap of length i. $ww(\cdot)$ can be any arbitrary function, but for the purpose of the current discussion, we will assume that it is a p-part piecewise-linear function where each successive slope is smaller than the previous one. A more specific and widely-used 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. We note that $S(i, j)$ is computed after $A[i : j]$ is already found, in contrast to

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 also 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[1]. 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 [13], $Pr(x \geq s) = 1 - exp(Kmne^{-\lambda s})$ 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 $Pr(x \geq s) = \frac{K}{\lambda} e^{-\lambda s}$.

6.1 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

[1]For the moment, we are using the same null hypothesis to estimate p-values for DNA and RNA alignments. We can, in the future, account for secondary structures in evaluating RNA alignments.

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[2] $p_a(i)$ 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 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.

6.2 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

[2]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.

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 segment scores are shown in Fig. 6.1 and 6.2. 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 with respect to m (at 3.1 and 45 respectively) and do not stray too far. These observations lead 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. Consequently, 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 A.1, A.2, A.3, and A.4 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[3]. Figures A.5, A.6, A.7, and A.8 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[28] 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[4]. Thus, we may similarly treat our piecewise-linear gap penalty $ww(\cdot)$ as differently-weighted mismatches in approximating the p-value. Fig. 6.3 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:

[3]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 $k_4 = -65.01$.

[4]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 tables 7.1 and 7.2.

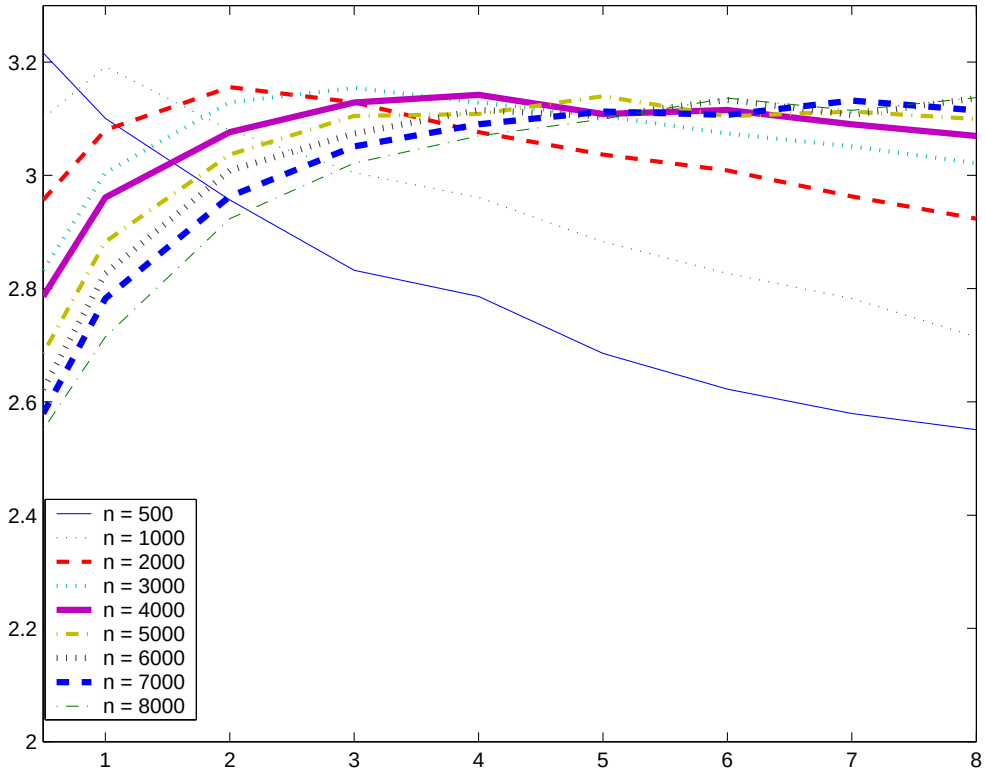


Figure 6.1: Shown above is the mean length-to-score ratio observed in the strips from aligning randomly generated DNA sequences. In the plots shown above, a unique line is plotted corresponding to each value of n in the thousand lengths ranging from 1000 to 8000. For these plots, x represents the m value divided by 1000, and y represents the mean observed for that particular m and n, and the plots illustrate mean length-to-score ratio for the segment pairs.

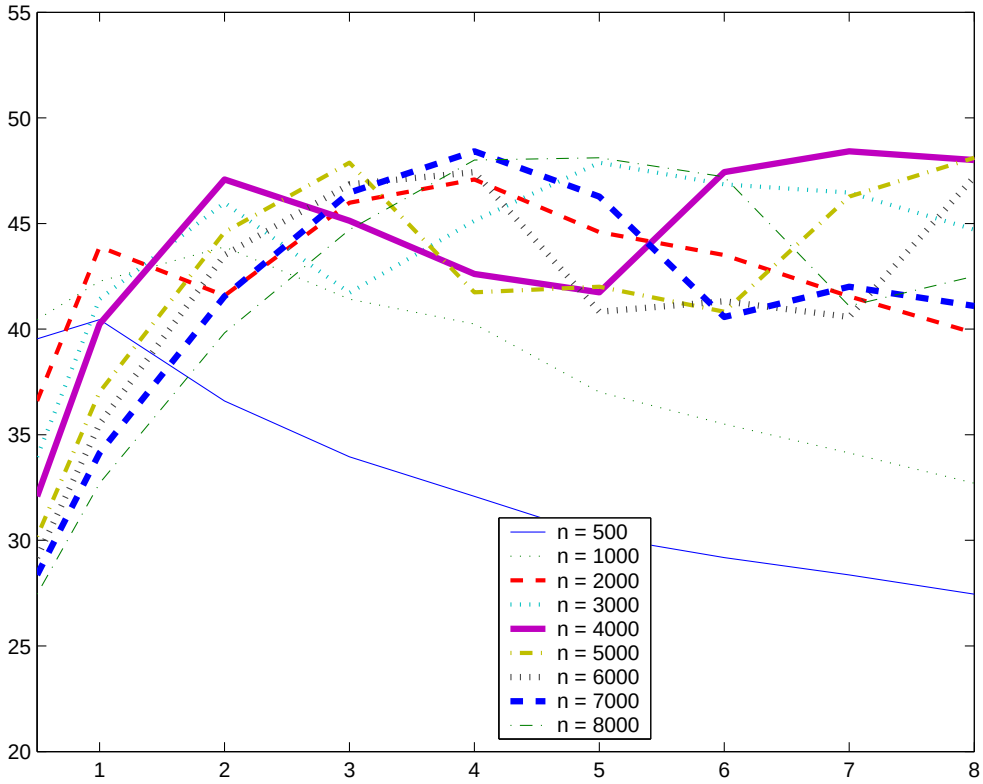


Figure 6.2: Shown above are the mean segment scores observed in the strips from aligning randomly generated DNA sequences. In the plots shown above, a unique line is plotted corresponding to each value of n in the thousand lengths ranging from 1000 to 8000. For these plots, x represents the m value divided by 1000, and y represents the mean observed for that particular m and n, and the plots illustrate mean segment pair scores.

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

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 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 6.4 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[5]:

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

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

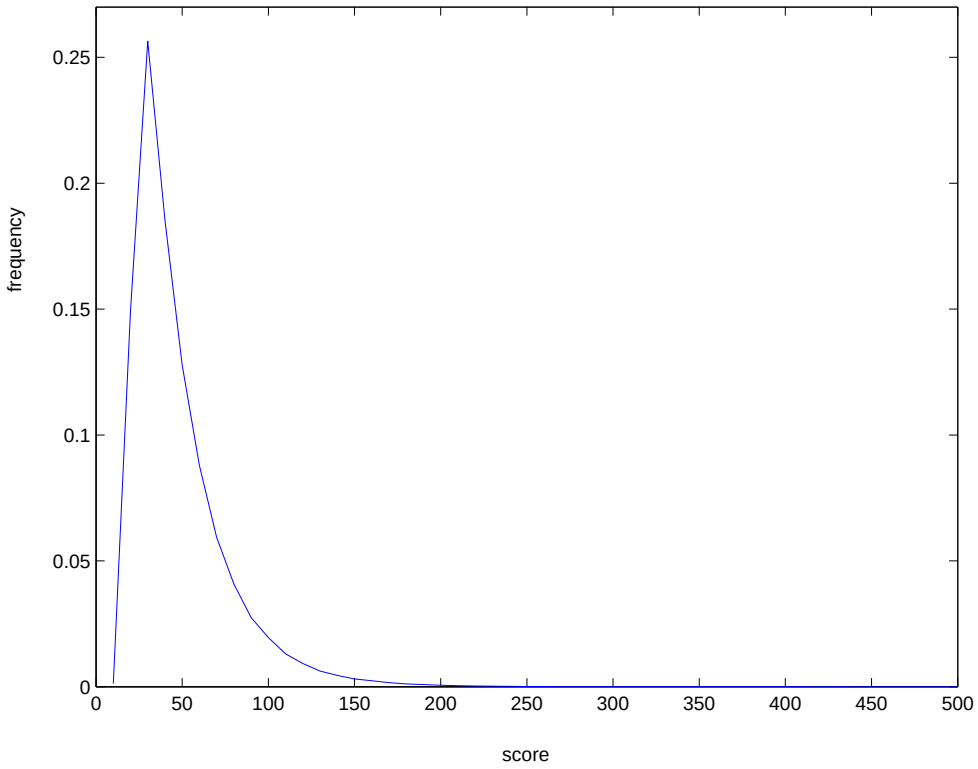


Figure 6.3: Shown here is a plot of segment scores to frequency for randomly generated sequences using our assumption that segment score is length-independent. The x axis represents segment score, and the y axis represents frequency. The tail of this plot is an exponential distribution of form $P(S = x) = Ke^{-\lambda x}$, 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.

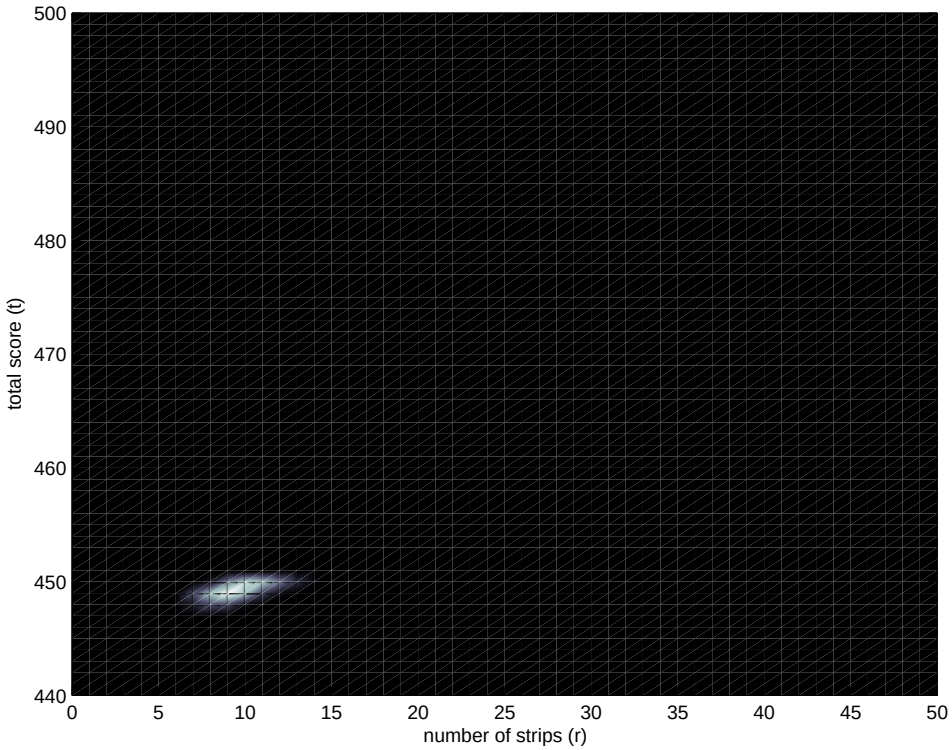


Figure 6.4: From our alignments over the randomly generated sequences, after adjusting the number of segments r and the total score t for length-dependent average and deviation behavior, we chose to plot the frequency of observing certain r and t values. The figure shown here is a surface plot of this, 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 $c = -183.90, a_t = 10.1, b_t = 9070, c_t = -2.04 \times 10^6, a_r = 0.241, b_r = 4.71, c_r = -27.5$.

Chapter 7

Colorgrid and DNA Results

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

7.1 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 7.2 and 7.3 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.

Test Name	PLAINS			LAGAN			EMBOSS		
	t	r	ζ'	t	r	ζ'	t	r	ζ'
Hp1	356.71	4	7.37	340.32	4	6.00	340.19	4	5.99
Hp2	285.75	3	3.96	281.84	3	3.94	238.30	3	3.87
Hp3	2181.50	14	47.18	441.58	6	22.98	1708.51	10	18.49
Hp4	511.99	7	3.85	2172.40	14	-Inf	296.84	4	4.59
Hp5	792.64	7	7.29	775.74	7	7.29	176.73	1	13.04
Mp1	389.84	4	13.97	386.88	4	13.40	388.88	4	13.78
Mp2	461.68	6	8.88	453.64	6	7.77	206.02	2	5.56
Mp3	72.19	1	6.75	72.19	1	6.75	83.34	1	6.75
Hf0	534.14	5	11.15	360.22	3	13.05	151.39	2	14.07
Hf1	734.82	7	10.94	349.33	4	14.18	374.35	5	13.05
Hf2	600.22	4	16.78	555.61	4	16.78	327.91	1	20.18
Hf3	637.52	7	14.53	259.44	3	19.05	409.99	5	16.71
Hf4	1004.97	10	21.74	529.16	5	-0.00	367.86	4	-0.00
Hf5	739.71	7	11.07	450.93	5	13.07	453.61	5	13.07

Table 7.1: 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.

7.2 PLAINS Empirical Results

Furthermore, tables 7.1 and 7.2 show a comparison of alignments for PLAINS, LAGAN, and EMBOSS over biologically related sequences using unadjusted r and t values, and ζ' values, with $\rho = 0.5$. 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 long gaps and low similarity regions. Also, we 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 Tables B.1, B.2, B.3, and B.4 in the appendix.

Also, PLAINS does not always yield the results of least coincidental proba-

Test Name	PLAINS			LAGAN			EMBOSS		
	t	r	ζ'	t	r	ζ'	t	r	ζ'
hm.1_1	676.29	10	8.46	52.36	1	18.29	186.98	2	17.00
hm.1_3	552.55	6	15.14	406.79	6	15.14	429.51	6	15.14
hm.3_9	1260.69	15	15.47	432.25	7	24.23	801.15	12	18.44
hm.3_16	218.47	3	5.71	x	x	x	180.05	2	6.77
hm.4_3	262.19	3	15.44	74.91	1	17.79	176.83	2	16.59
hm.4_5	421.71	6	7.35	221.57	3	10.47	401.71	5	8.32
hm.6_17	986.89	12	23.00	240.10	3	-0.00	260.66	4	-0.00
hm.7_11	594.32	8	9.06	164.10	2	15.44	476.71	7	9.99
hm.17_11	608.75	7	13.93	171.96	3	18.57	451.60	6	15.02
hm.x_x	1302.49	18	17.20	636.82	9	-0.00	568.46	9	-0.00
hd.6_1	1239.35	14	18.99	424.59	6	-0.00	688.81	8	26.81
hd.6_12	1284.79	14	13.88	548.19	7	21.23	394.04	6	22.44
hd.6_34	1488.26	16	-0.00	496.14	6	-0.00	900.73	12	-0.00
hd.7_16	1042.19	13	10.45	128.07	2	22.40	309.03	4	19.84

Table 7.2: Shown here is the additional data for the DNA aligmments over PLAINS, EMBOSS, and LAGAN that did not fit in table 7.1.

bility 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 7.1 explains the details further, and figure 7.3 provides a supporting example.

Many of the genomic alignments yielded by the three tools have caught exons in the alignment, but most of these exons caught are not included in the "good" regions of the alignment, because SEPA removed them for having a ρ value that was too high.

The Mp (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 three alignment tools, in terms of the actual alignments obtained, especially between PLAINS and LAGAN. This data shows either the difference of linear gap func-

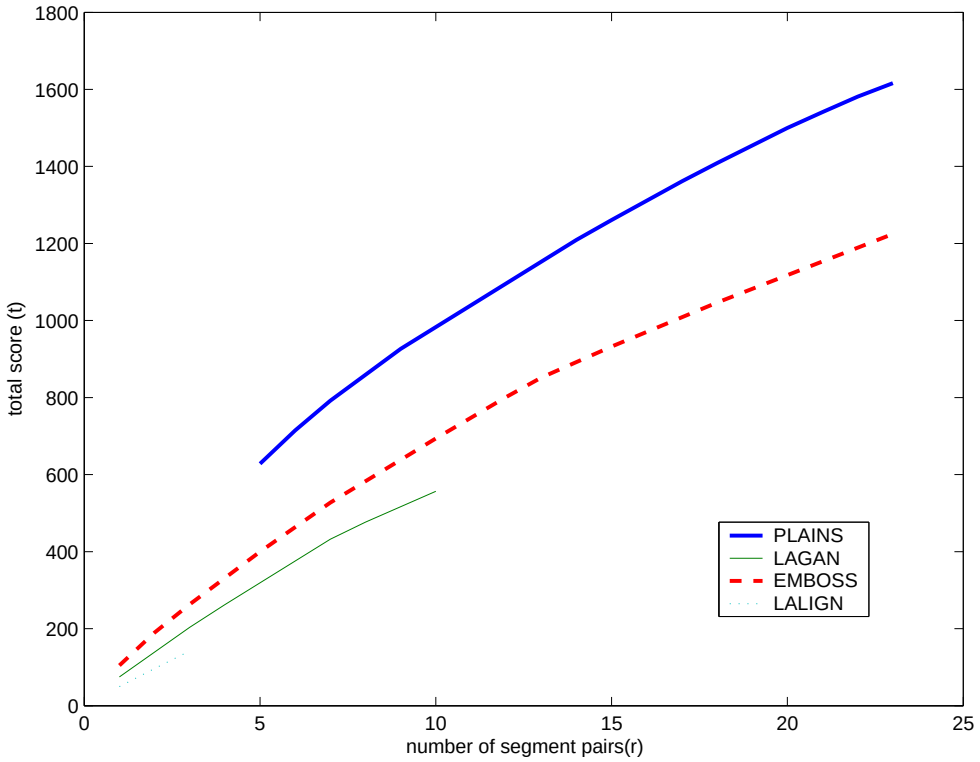


Figure 7.1: In this figure, we observe the unadjusted r and t values produced by PLAINS, LAGAN, and EMBOSS from the hm.3 – 9 experiment where we vary the ρ variable used to filter our segment pairs. On each curve, we observed the t and r values of each tool when varying ρ over various values from 0.1 till 0.9. Recall from table 7.2 that PLAINS performed poorly in terms of ζ' values for $\rho = 0.5$ for the hm.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 tables 7.1 and 7.2 have a similar plot to this one.

tions over piecewise-linear gap functions, or the difference of using general-case gap parameters over using customized gap parameters per species, or possibly both.

For most of the Hp (alignments of Human genes against corresponding pseudogenes) runs, PLAINS and LAGAN yielded alignments with many similar exon correlations, but with PLAINS being able to identify more homologies within those exons. One illustration of this is figure 7.3, which compares the results of PLAINS to LAGAN for Hp5 in further detail.

However, the most interesting results obtained were in the Hf 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 Hf2, although the ζ' value for PLAINS is almost identical to that of EMBOSS, PLAINS caught more common exons between the two related genomic sequences, and even confidently identified correlations not previously known. 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 7.2 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 `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.

7.3 PLANAR Empirical Results

In table 7.3, we see a comparison of alignments for PLANAR and RSMATCH over biologically related sequences using unadjusted r and t values, and ζ' values, with $\rho = 0.9$. Due to the lower primary structure sequence similarities present

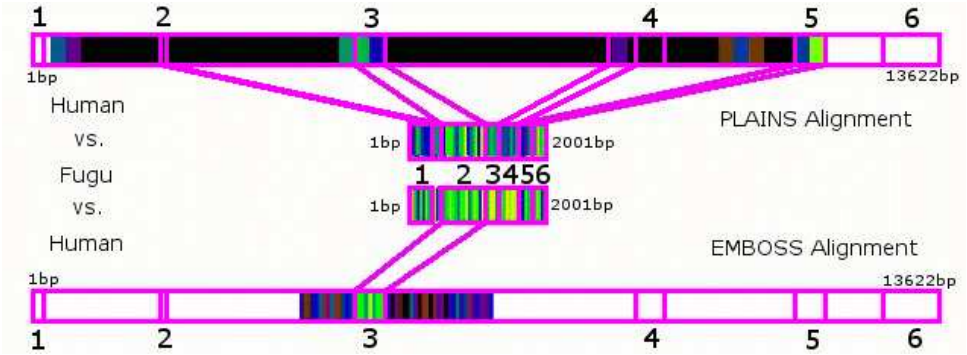


Figure 7.2: Match Ratio Color Lines in the Hf2 test for PLAINS and EMBOSS. Here, the Human and Fugu sequence used have six exon regions that correspond to each other (though not necessarily in order, as exon region 2 in the Fugu sequence corresponds to exon region 3 in Human sequences for example). Here, 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. PLAINS also confidently found two additional correlations between the two sequences not previously known, as shown above.

in these RNA sequences versus the DNA sequences mentioned earlier, we chose to use $\rho = 0.9$ here instead of the $\rho = 0.5$ as used earlier, in order not to risk missing the most important segments.[1] Also as before, there is a loss of precision involved in the reporting of ζ' values, which in some cases "appear" the same for PLANAR and RSMATCH, even if they are not. The rnase experiments use Delta/Epsilon Purple Bacteria RNase P sequences from the RNaseP database, and the telomerase experiments use ribonucleoprotein reverse transcriptase synthesizing telomeric DNA found from the RFAM database with accession number RF00025. For further information regarding the sequences used, see Table B.5 in the appendix.

Note from table 7.3 that, as with PLAINS, PLANAR does not always yield the results of least coincidental probability, and the reasoning for this is similar to PLAINS. Capturing the biology faithfully when sequences have expected large

[1] Although SEPA does not account for secondary structure in its p-values or ζ values, the technique from section 5.4 could fix this issue. However, this requires a separate analysis of the important segments for that case.

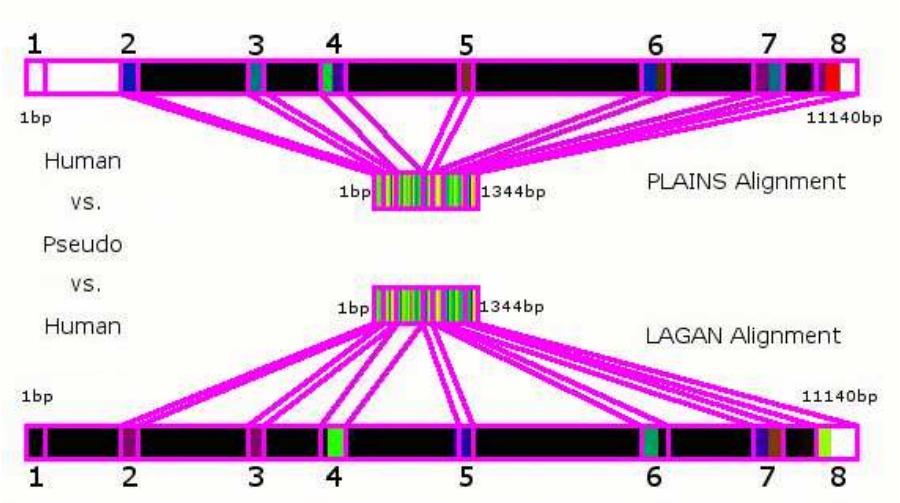


Figure 7.3: Match Ratio Color Lines in the Hp5 test for PLAINS and LAGAN. Here, the Human sequence has 8 exon regions that are similar to areas of the pseudosequence used, and alignments of PLAINS and LAGAN for these cases are similar, even by visual examination of the ColorGrids, and in the fact that they both identified the same 7 exon region correlations. However, when we contrast the homologies (i.e., grid colors) within these exon regions identified, we find that PLAINS found higher correlations within four of these exon regions (regions 2, 3, 7, and 8) and therefore identified these regions with higher confidence. This strengthens our earlier argument of PLAINS being able to identify most correlations with higher confidence.

gaps and low similarities causes PLANAR to aggressively align as many regions as possible, raising its r and t values, with an increase in the r value high enough to adversely affect its ζ' value. As a result, if we fix r for PLANAR and RSMATCH (much like done earlier for PLAINS), the r segments generated by PLANAR will have smaller individual coincidental probabilities.

Using fixed match/mismatch and gap parameters, PLANAR takes roughly a minute, the same amount of time to align a pair of RNA sequences of length 200 bp that PLAINS would use for a pair of DNA sequences of length 12 Kbp. PLANAR runs about 3 times slower than RSMATCH. The longer time used by PLANAR arises from the extra time needed to align with piecewise-linear gap functions (compared to the linear gap functions used in RSMATCH), as well as the time involved in merging two alignments as mentioned in section 5.4.

Test Name	PLANAR			RSMATCH		
	t	r	ζ'	t	r	ζ'
`rnase.1_2`	204.67	3	3.37	118.58	1	5.99
`rnase.1_3`	134.35	2	4.53	85.55	1	5.99
`rnase.3_5`	120.76	2	4.51	42.88	1	5.98
`rnase.4_5`	104.17	1	5.98	93.08	2	4.49
`telomerase.1_2`	29.58	2	4.47	14.89	2	4.47
`telomerase.1_3`	79.23	2	4.48	24.67	2	4.48
`telomerase.2_3`	17.06	1	6.10	2.60	1	6.10

Table 7.3: Shown here for PLANAR and RSMATCH are the r, t, and ζ' values obtained from aligning noncoding RNA sequences of lengths between 100 and 200 bases where the pairs are biologically related, with correlated secondary structures, but poor correlations within their primary structures.

Chapter 8

Conclusions and Open Problems

PLAINS is able to catch more important correlations than its competition, especially between highly divergent sequences such as Human and Fugu. PLANAR holds promise as well, since it was found to be able to achieve a similar effect for RNA sequences. Furthermore, 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 detected, 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, and similarly if PLANAR is to be run over sequences with more than 2000 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. PLANAR is only suitable for sequences of lengths up to 2000 nucleotides due to the higher time and space needed for it to achieve similar effects to PLAINS using secondary structures.

Possible future extensions include aligning large sequences (of megabases of nucleotides), and then using PLAINS or PLANAR to refine alignments over smaller areas, where localized alignments can be performed. For instance, PLAINS can potentially refine the results within the locally identified interval regions found by COMBAT[32] over whole genomes. In addition, PLAINS could

become the ideal tool for aligning EST sequences to a genome, and PLANAR could become the ideal tool for aligning rRNA's or tRNA's to a genome.

Another possible extension involves incorporating 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. Accounting for secondary structures could especially improve the way SEPA estimates p-values for RNA alignments.

In addition, PLAINS at the moment assigns a reward of $m_a = 1$ to a perfect match, and a penalty m_s (specified by user) to any mismatch. Similarly, PLANAR at the moment assigns to unbounded positions a reward of 1 for match and penalty of m_s (specified by user) for mismatch, and assigned to bound positions rewards of 1 for mismatch and m_b (specified by user) for match. It may be useful to have a scoring matrix to assign different scores to different types of matches/mismatches in these cases. (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

SEPA 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. A.1, A.2, A.3, A.4, A.5, A.6, A.7, and A.8.

From fig. A.1 and A.2, 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. A.3 and A.4, we infer that, although for small n values, the average segment score decreases with increasing m, asymptotically (for large n) it stays roughly constant with respect to 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. A.5 and A.6, 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 $k_0 = 10^3$, $k_1 = 1.93 \times 10^{-10}$, $k_2 = 1.97 \times 10^{-7}$, $k_3 = 1.00$.

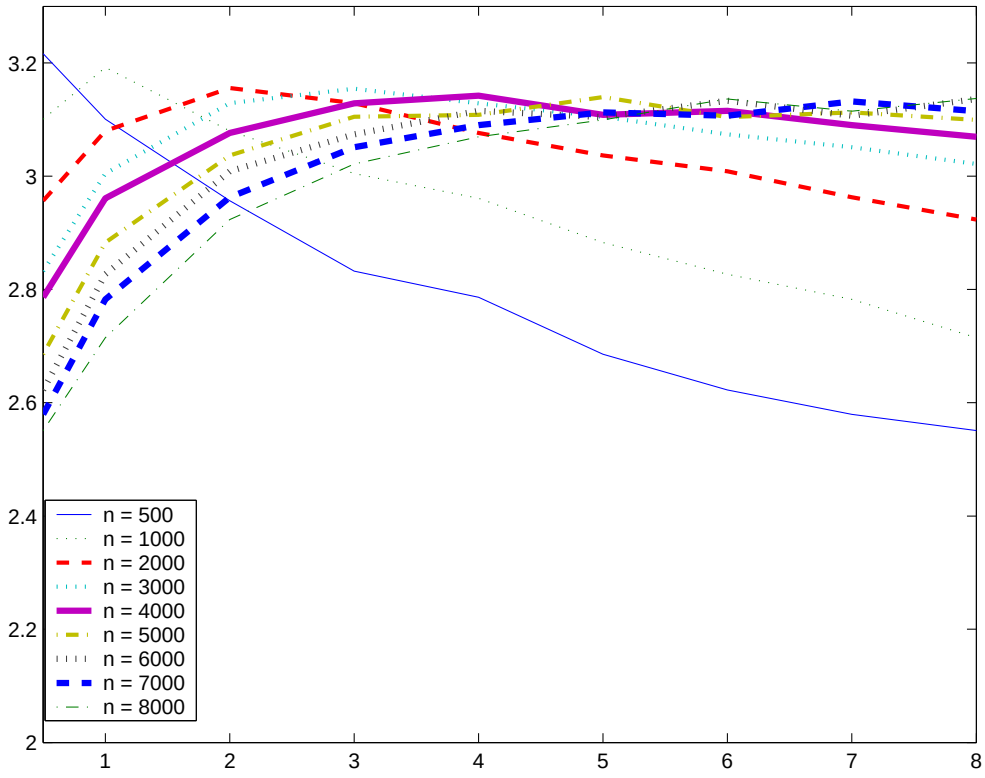


Figure A.1: Shown above are the mean plots for the segment pair length-to-score ratio from aligning randomly generated DNA sequences. A unique line is plotted corresponding to each value of n in the thousand lengths ranging from 1000 to 8000. For these figures, and others that follow, x represents the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

From fig. A.7 and A.8, 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 $k_4 = -65.01$.

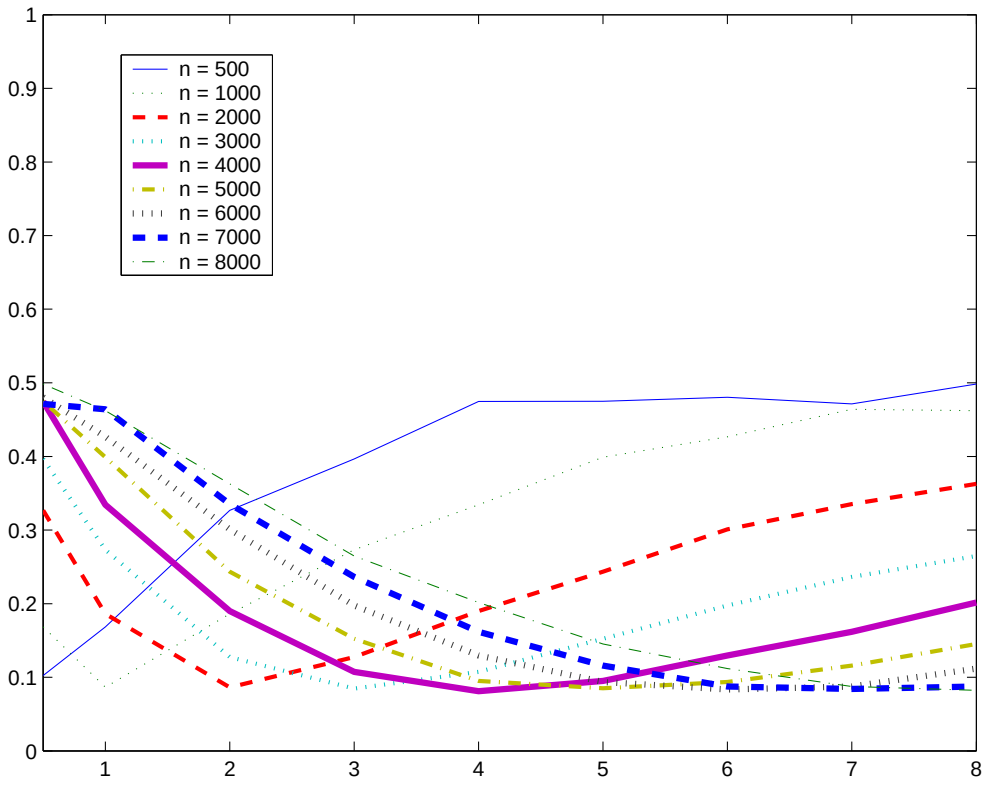


Figure A.2: Shown above are variance plots for the segment pair length-to-score ratio from aligning randomly generated DNA sequences.

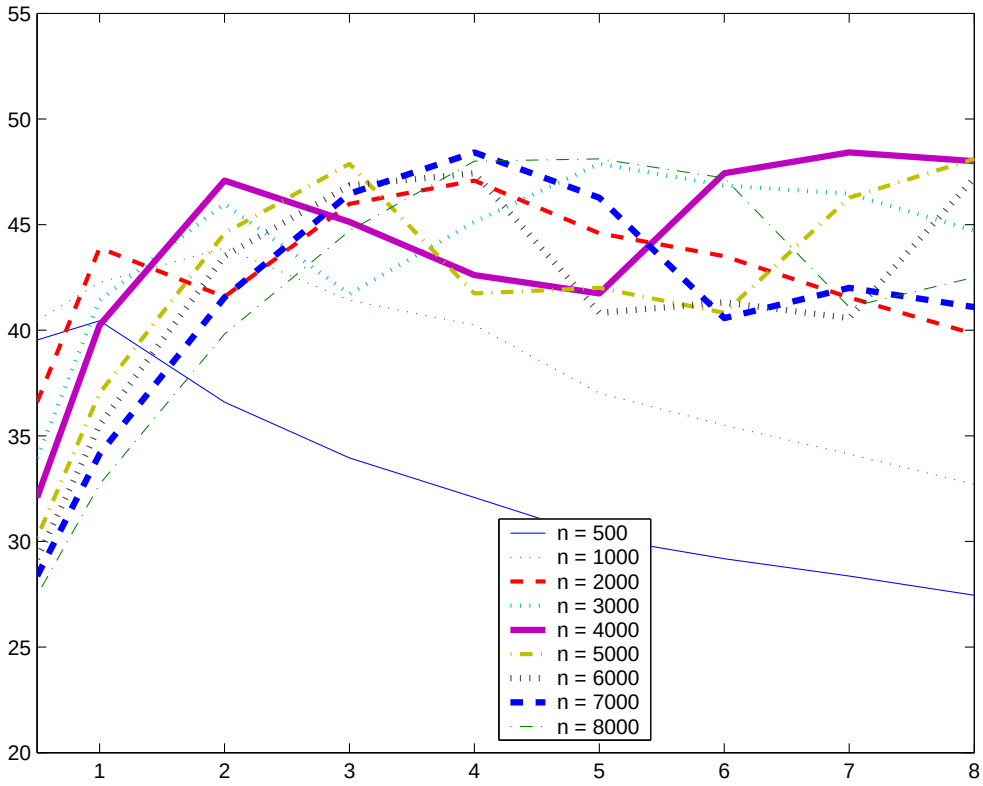


Figure A.3: Shown here are the mean plots for segment scores from aligning randomly generated DNA sequences.

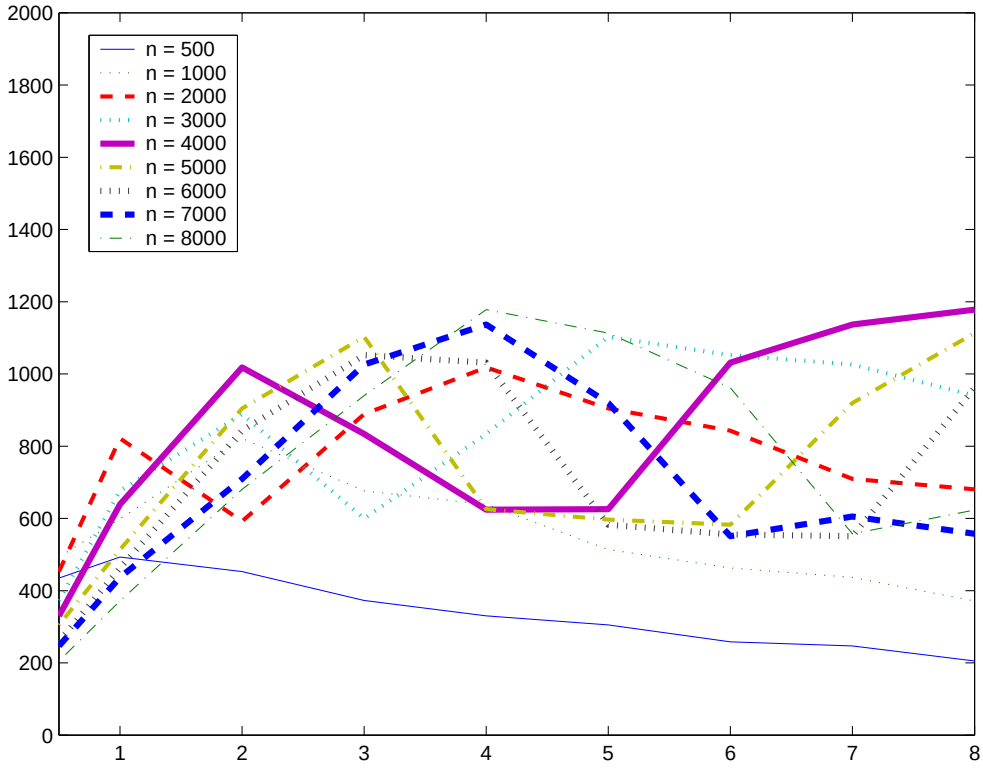


Figure A.4: Shown here are the variance plots for segment scores from aligning randomly generated DNA sequences.

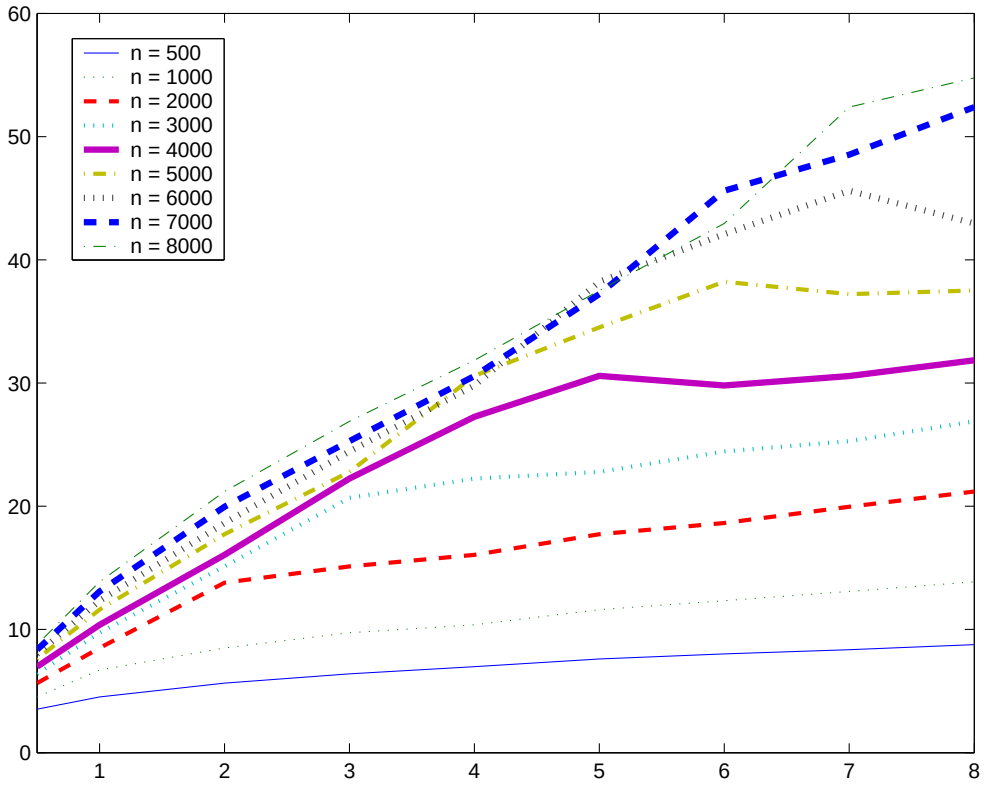


Figure A.5: Shown here are the mean plots for r, the number of segment pairs obtained from aligning randomly generated DNA sequences.

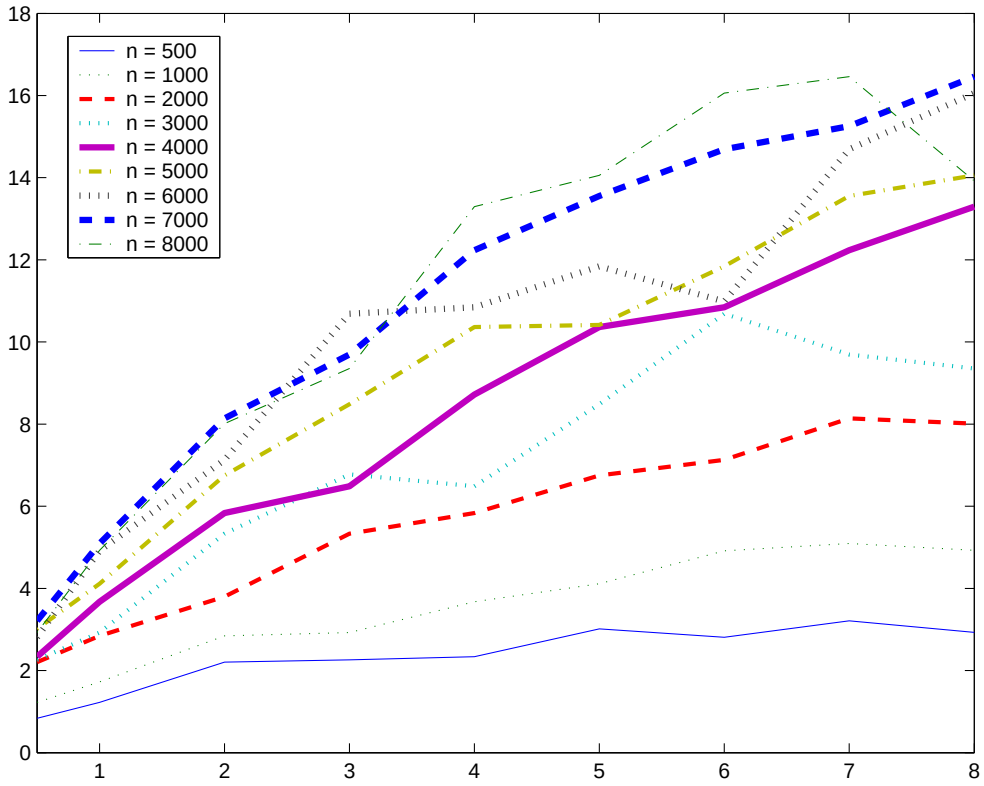


Figure A.6: Shown here are the variance plots for r, the number of segment pairs obtained from aligning randomly generated DNA sequences.

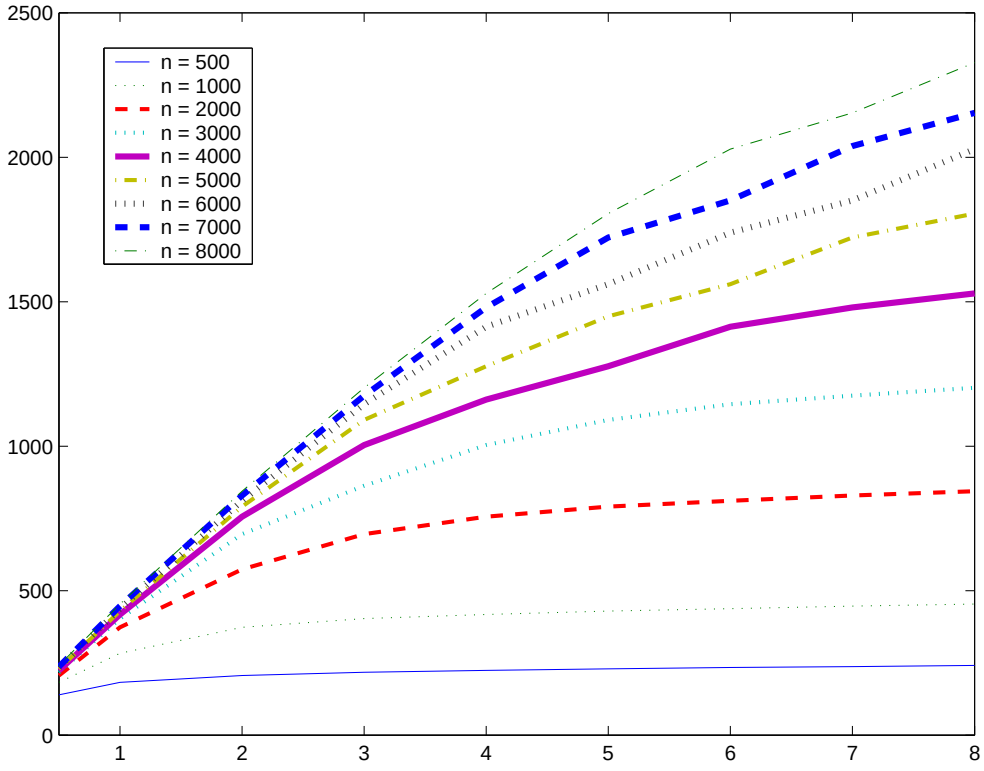


Figure A.7: The plots shown here are the mean plots for t, the total score of all segment pairs from aligning randomly generated DNA sequences.

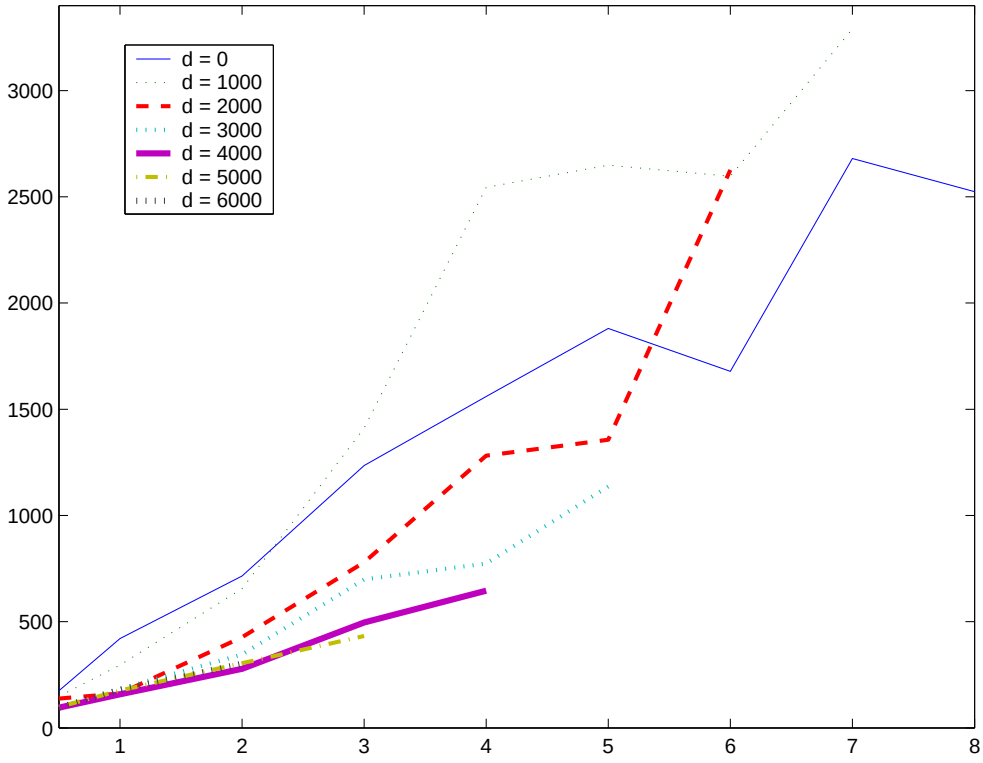


Figure A.8: The plots shown here are the deviation plots for t, the total score of all segment pairs from aligning randomly generated DNA sequences. Because the variance plot was difficult to quantify in terms of m and n, we instead model the deviation for total score in terns of d and i, where $i = \min(m, n)$ and $d = \|m - n\|$. We see here the deviation plot, with each curve corresponding to a unique d value, and the x-axis reprsenting i in units of thousands.

Appendix B

Sequence Details

Shown in Tables B.1, B.2, B.3, and B.4 are further details for the sequences used to compare PLAINS against LAGAN and EMBOSS. 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.

Shown in Table B.5 are further details for the sequences used to compare PLANAR against RSMATCH.

Name	Sequences Used
Hp1	chr1 8257472 8257969 + NCBI34:19:54160379:54161804:1
Hp2	chr1 163548408 163549002 + NCBI34:4:174948678:174951482:-1
Hp3	chr1 212839737 212843396 + NCBI34:19:47480657:47491789:1
Hp4	chr2 215849936 215850977 − NCBI34:12:52960755:52965297:1
Hp5	chr3 154761512 154762855 − NCBI34:20:62845714:62856853:-1
Mp1	chr1 6930250 6930693 + NCBIM32:4:116062392:116064688:1
Mp2	chr10 34897773 34898331 + NCBIM32:3:111151293:111157009:1
Mp3	chr1 101195551 101195966 + NCBIM32:19:41974653:41984383:1

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

Name	Sequences Used
Hf0	NCBI34:6:10803176:10817954:1 FUGU2:scaffold_3266:7199:8502:1
Hf1	NCBI34:22:17268346:17274146:1 FUGU2:scaffold_115:304567:308251:1
Hf2	NCBI34:22:19452941:19466562:1 FUGU2:scaffold_385:130429:132429:1
Hf3	NCBI34:21:31952480:31961633:1 FUGU2:scaffold_492:107025:110089:-1
Hf4	NCBI34:4:78536922:78549607:1 FUGU2:scaffold_1018:38886:42563:-1
Hf5	NCBI34:1:23574363:23584195:1 FUGU2:scaffold_2020:1332:3570:1

Table B.2: Sequence Details for the Biologically Related Alignments Ran, Part 2. All the sequences are retrieved from ENSEMBL database [www.ensembl.org].

Name	Sequences Used
hm.1_1	hg17 chr1:1045045-1049199 mm6 chr1:58087808-58093089 −
hm.1_3	hg17 chr1:109911-115784 mm6 chr3:108302834-108307402 +
hm.3_9	hg17 chr3:920975-927750 mm6 chr9:13034270-13040751 −
hm.3_16	hg17 chr3:40927-45344 mm6 chr16:36425494-36426630 +
hm.4_3	hg17 chr4:1016348-1026634 mm6 chr3:43806778-43808958 +
hm.4_5	hg17 chr4:33206-37263 mm6 chr5:116454347-116457564 −
hm.6_17	hg17 chr6:1515792-1522464 mm6 chr17:5319541-5327318 +
hm.7_11	hg17 chr7:253979-256656 mm6 chr11:47406997-47414401 −
hm.17_11	hg17 chr17:203511-209188 mm6 chr11:46304241-46308929 −
hm.x_x	hg17 chrX:928373-936336 mm6 chrX:100457186-100463788 +

Table B.3: Sequence Details for the Biologically Related Alignments Ran, Part 3. All the sequences are retrieved from ENSEMBL database [www.ensembl.org].

Name	Sequences Used
hd.6_1	hg17 chr6:48183-58637 canFam1 chr1:66683762-66688436 −
hd.6_12	hg17 chr6:791946-797744 canFam1 chr1:58385127-58391875 +
hd.6_34	hg17 chr6:1248975-1255904 canFam1 chr34:40546832-40556432 −
hd.7_16	hg17 chr7:40725-45009 canFam1 chr16:22868000-22875215 +

Table B.4: Sequence Details for the Biologically Related Alignments Ran, Part 4. All the sequences are retrieved from ENSEMBL database [www.ensembl.org].

Name	Sequences Used
rnase.1_2	D.desulfuricans RNase P RNA D.vulgaris RNase P RNA
rnase.1_3	D.desulfuricans RNase P RNA G.sulfurreducens RNase P RNA
rnase.3_5	G.sulfurreducens RNase P RNA H.pylori-26695 RNase P RNA
rnase.4_5	C.jejuni RNase P RNA H.pylori-26695 RNase P RNA
telomerase.1_2	telomerase 1: AF417611/283-441 telomerase 2: U10565/50238
telomerase.1_3	telomerase 1: AF417611/283-441 azeAF417612/231392
telomerase.2_3	telomerase 2: U10565/50238 azeAF417612/231392

Table B.5: Sequence Details for the RNA Alignments Ran. All the sequences are retrieved from CARNAC website [http://bioinfo.lifl.fr/carnac/].

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] Cormen TH, Leiserson CE, Rivest RL, Stein C, "Single-Source Shortest Paths" *Introduction to Algorithms, 2nd Edition*, **24**:580–601 , 2001.

[4] 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.

[5] Eddy S.R., "A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an RNA secondary structure." *BMC Bioinformatics*, **3:18**, 2002.

[6] Gill O, Mishra B, "SEPA: Approximate Non-subjective Empirical p-Value Estimation for Nucleotide Sequence Alignment." *Lecture Notes in Comp. Sci.*, **3992**: 638–645, 2006.

[7] Gill O, Mishra B, "PLANAR: RNA Sequence Alignment with Non-Affine Gap Penalty." *Unpublished work*, 2006.

[8] 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 Math. Bio. and Medicine* **8** (2005). An unabridged version can be found at: `http://bioinformatics.nyu.edu/~gill/index.shtml`

[9] 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.

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

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

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

[13] 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.

[14] 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.

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

[16] Krek A, Grun D, Poy MN, Wolf R, Rosenberg L, Epstein EJ, MacMenamin P, da Piedade I, Gunsalus KC, Stoffel M, Rajewsky N., "Combinatorial microRNA target predictions." *Nature Genetics*, **37(5)**: 495–500, 2005.

[17] 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.

[18] Liu J, Wang JTL, Hu J, Tian B, "A method for aligning RNA secondary structures and its application to RNA motif detection," *BMC Bioinformatics*, **6**:89, 2005.

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

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

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

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

[27] Rivas E, Eddy SR, "A Dynamic Programming Algorithm for RNA Structure Prediction Including Pseudoknots," *J. Mol. Biol.*, **: 285**: 2053–2068, 1999.

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

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

[30] 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.

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

[32] Sun B, Schwartz J, Gill O, Mishra B, "COMBAT: Search Rapidly for Highly Similar Protein-Coding Sequences Using Bipartite Graph Matching." *Lecture Notes in Comp. Sci.*, **3992**: 654–661, 2006.

[33] 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.

[34] Zhang S, Haas B, Eskin E, Bafna V, "Searching Genomes for Noncoding RNA Using FastR." *IEEE/ACM Trans. on Comp. Bio. and Bioinf.*, **2(4)**: 366–379, 2005.

[35] 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.