

Aligning Sequences with Non-Affine Gap Penalty: PLAINS Algorithm, a Practical Implementation, and its Biological Applications in Comparative Genomics

Ofer Gill¹, Yi Zhou³ and Bud Mishra^{1,2}

¹ Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY, USA 10012.
gill@cs.nyu.edu

² Cold Spring Harbor Lab, 1 Bungtown Road, Cold Spring Harbor, NY, USA 11724.

³ Department of Biology, New York University, 100 Washington Square East, New York, NY 10003.

ABSTRACT In this paper, we consider PLAINS, an algorithm that provides efficient alignment over DNA sequences using piecewise-linear gap penalties that closely approximate more general and meaningful gap-functions. The innovations of PLAINS are fourfold. First, when the number of parts to a piecewise-linear gap function is fixed, PLAINS uses linear space in the worst case, and obtains an alignment that is provably correct under its memory constraints, and thus has an asymptotic complexity similar to the currently best implementations of Smith-Waterman. Second, we score alignments in PLAINS based on important segment pairs; optimize gap parameters based on interspecies alignments, and thus, identify more significant correlations in comparison to other similar algorithms. Third, we describe a practical implementation of PLAINS in the Valis multi-scripting environment with powerful and intuitive visualization interfaces, which allows users to view the alignments with a natural multiple-scale color grid scheme. Fourth, and most importantly, we have evaluated the biological utility of PLAINS using extensive lab results; we report the result of comparing a human sequence to a fugu sequence, where PLAINS was capable of finding more orthologous exon correlations than similar alignment tools.

1 INTRODUCTION

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

ing 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 transcribed and translated into proteins can be separated by intergenic regions of few thousands of base pairs that do not get expressed, and perhaps, are subject to strikingly different (or no) selection constraints. Thus these intergenic regions typically vary to a greater extent in one species compared to another. Therefore, we may expect the gap lengths in DNA alignments to be larger, more variable, and have specie-specific distributions. Moreover, these distributions characterizing the gap-lengths may not be memory-less (i.e., exponential distributions). There have been suggestions that power-law distributions may be more appropriate. The evolutionary processes governing the genomes of species, and the log-likelihood of certain indel gaps occurring when comparing one species against another suggest that a logarithmic gap function is more appropriate for DNA sequences. Because of this, the traditional affine (or linear) gap functions used for aligning proteins are unsatisfactory for DNA sequences, as the ultimate results may be biologically misleading.

In order to exploit the fidelity of general non-linear gap functions for DNA sequences, without suffering performance penalties 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 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 [10]), but we present several additional theoretical innovations in terms of worst-case upper-bound memory usage, alignment optimization, and visualization of data. We have the algorithm available in a powerful bioinformatic environment, called *Valis*. Our algorithm uses an innovative learning-heuristic to determine the best score function, a near-optimal gap-penalty model, and a scheme to compute p-values for reporting alignment reliabilities.

As we hope to demonstrate here by an extensive set of experimental results, our algorithm 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. 3)

This paper is structured as follows: The first section introduces the notations used throughout this paper, overviews how PLAINS aligns, and describes how PLAINS does its “log function to piecewise-linear function” approximation, alignment scoring, alignment reliability computations, parameter optimization, and color grid scheme. The second section describes the empirical results found from comparing Plains to similar algorithms, and

overviews the usage of PLAINS. The final section concludes with a discussion of possible future extensions for PLAINS. The appendix gives the specific sequences used for the tests ran on PLAINS and similar alignment tools, as well as describes the PLAINS alignment method in detail, including proofs of its space-bound and correctness.

2 THE PLAINS SCORING SYSTEM

We now explain several aspects of PLAINS: its general notations, its alignment method, and how it approximates a log function using a piecewise-linear function, decides what a “best alignment” is, computes reliabilities of its results, optimizes parameters for alignments, and visually displays alignments using a ColorGrid.

2.1 THE PLAINS ALIGNMENT METHOD

For the remainder of this paper, assume that the two strings to be aligned are denoted⁴ X and Y , and their respective lengths are m and n . Also, assume that the penalty for each mismatch is denoted as ms .

Next, a p -part piecewise-linear function is denoted as $ww(\cdot)$. This function has a y -intercept of wo , and the slopes of the linear functions in successive intervals are wc_1, wc_2, \dots, wc_p , and the x -values at which one interval ends and the next begins are denoted k_1, k_2, \dots, 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 $k_{\bar{p}-1} < i \leq k_{\bar{p}}$, $ww(i)$ is defined as:

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

For all practical purposes, it is sufficient to set p to be at most 10. With our mismatch penalty ms and our piecewise-linear gap penalty function $ww(\cdot)$, and reward per match fixed at 1, PLAINS generates an alignment using a method similar to Miller-Meyers [10], except that because PLAINS exclusively uses piecewise-linear gap functions (as opposed to general gap functions), it is able to take advantage of an algorithm of its very own, and uses $O(np)$ space in the worst-case⁵. Further details of how PLAINS generates an alignment, and the proofs of its $O(np)$ space bound and the correctness of the computed alignment obtained are all explained in appendices B, C, and D.

2.2 PLAINS LOG APPROXIMATION AND PARAMETER OPTIMIZATION

Recall our definition for a p -part piecewise-linear gap function $ww(\cdot)$. For a given piecewise-linear gap function and mismatch penalty, the PLAINS algorithm does find the best alignment for X and Y . When a user asks PLAINS to find the “best set” of gap-mismatch parameters that yield a “best alignment,” PLAINS optimizes over four variables: α , β , d , and ms . The penalty for each mismatch is denoted ms , as in the previous section. If the gaps

⁴ X and Y should not be confused with x and y , as the capital letters denote strings and the small letters, positions within such strings.

⁵ For all practical purposes, $p \leq 20$, hence we can say that PLAINS uses worst-case linear space

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, such a piecewise-linear function assumes an affine function (corresponding to an exponential distribution for gap lengths), it retains the generality for a wide class of distributions.

More specifically, the log gap penalty function over i is denoted as⁶: $\alpha \ln(i + 1) + \beta$. For a given d , α , and β , $ww(\cdot)$ uses k_1, \dots, k_p values set to $d, 2d, \dots, p * d$, 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 u th line⁷, 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 next section, 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 [5]. Empirical runs over PLAINS have shown that Simulated Annealing yields better results, but Genetic Algorithm explores the space of v more thoroughly. However, all of this should come of no surprise, since (1) Monte Carlo related methods are successful in optimizing Hidden Markov Models (which are similar to sequence alignments), and (2) Genetic Algorithms typically consider subsequent solutions in a more random manner than Simulated Annealing. PLAINS is designed so that any algorithm to optimize gap/match-mismatch parameters can easily be plugged in instead of these two methods; for instance, one may search parameters with a somewhat time consuming MCMC approach, or variants such as Gibbs sampler or EM.

2.3 THE PLAINS SCORING SYSTEM

For a fixed set of gap/match-mismatch parameters (α, β, d, ms) , PLAINS creates an alignment and scores it by grabbing segment pairs that yields “good “ scores. The sum of these segment pairs’ scores R is the value PLAINS reports, and not the score obtained from the dynamic table. This is because when we observe an alignment, it is the segment pairs of high scores that are the only true items of significance.

There are many ways to optimally select segment pairs. The method chosen by PLAINS creates the alignment first, and uses the alignment to obtain the segment pairs. The reason

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 p th line, wc_p is computed assuming that $k_p = p * d$, even though k_p is later assumed to be infinity, and the p th line of the piecewise-linear function is assumed to continue off into infinity.

PLAINS does not grab segment pairs from the dynamic table like most other algorithms do is because PLAINS uses linear space in memory, and grabbing segment pairs from the dynamic table would typically require quadratic space in some way or another. Furthermore, PLAINS assumes that both mismatches and gaps are allowed in the segment pairs (though sparingly, since the segment pair has to be considered a “good” one).

Using fixed constants W , ω , and ρ (where W is an integer, and ω and ρ are reals within $[0, 1]$), PLAINS obtains segment pairs from an alignment A of length a in following way:

(1) For all i from 1 to $a - (W - 1)$, PLAINS computes a $pa(i)$ value, which is the percentage of entries in $A[i..i + W - 1]$ where a match has occurred. We then compute μ and σ , the mean and standard deviation of our $pa(\cdot)$ values. Next, we mark⁸ any $pa(\cdot)$ values as “special” if they exceed $\mu + \omega\sigma$.

(2) For each u and u' (with $u \leq u'$), if $pa(u), pa(u + 1), \dots, pa(u')$ are all marked as “special”, but $pa(u - 1)$ and $pa(u' + 1)$ are not, then we create a segment pair that ranges from $A[u..u' + W - 1]$ (i.e., a segment pair that starts at the leftmost entry $pa(u)$ represents, and ends at the rightmost entry that $pa(u')$ represents).

(3) We trim each segment pair P so that it starts and ends at a position in the alignment where a match occurred. Next, we merge together any segment pairs that overlap. (I.e., if a segment pair P_1 starts at position u_1 and ends at position v_1 , and a segment pair P_2 starts at position u_2 and ends at position v_2 , and $u_1 < u_2$, and $v_1 \geq u_2$, we merge the two segment pairs into a new segment pair P which begins at u_1 and ends at $\max(v_1, v_2)$.)

(4) With all segment pairs representing non-overlapping regions, we then proceed to give a score to each segment pair. (I.e., if a segment pair P starts at position u and ends at position v , then we evaluate the score of subalignment $A[u..v]$ using our given gap/mismatch penalties ww and ms . We continue to give 1 point per match. Note that all segment pairs kept at step (4) must begin and end at a point in the alignment where a match occurs.)

(5) For each score S given to each segment pair P , we compute the Karlin-Atschul probability $p = 1 - \exp(-Kmn e^{-\lambda S})$ as outlined in [7]. We delete any segment pairs whose $p > \rho$, since this signifies a segment pair that “occurs by chance” and does not represent a significant relation between X and Y . The choice of selection of K and λ values is described later.

(6) The r segment pairs kept at this step are considered the “good” ones. We now compute R , the sum of the scores of the “good” segment pairs. (I.e., if the i th “good” segment pair has score S_i , then we compute $R = \sum_{i=1}^r [S_i]$.)

We also compute ζ , the Karlin-Atschul probability for the r segments. Here, if $S'_i = \lambda S_i - \ln(Kmn)$, and $T_r = S'_1 + \dots + S'_r$, then as mentioned in [8], $T_r = \sum_{i=1}^r [S'_i] = \sum_{i=1}^r [\lambda S_i - \ln(Kmn)] = \lambda \sum_{i=1}^r [S_i] - r \ln(Kmn) = \lambda R - r \ln(Kmn)$, and $\zeta = \text{Prob}(T_r \geq x) \approx \frac{e^{-x} x^{r-1}}{r!(r-1)!}$. This acts as an overall probability for the “good” segment pairs.

Setting $W = 50$, $\omega = 0.5$, $\rho = 0.5$ empirically yields segment pairs that are reasonably long, and have significantly higher matches than the alignment “background”. With these values for W and ω , we ran PLAINS over 900 pairs of randomly generated sequences, each with lengths ranging from 500 thru 4000, and observed the “good” segment pairs computed. (Note: We temporarily ommitted step 5 for these runs, the step where we eliminate

⁸ The choice of using $\mu + \omega\sigma$ as the cutoff value instead of a fixed constant gives PLAINS the flexibility to catch the important regions in two sequences, regardless of how homologous they are to each other

segment pairs based on the ρ values). From this empirical study, it was estimated that $K = 3.31 * 10^{-4}$ and $\lambda = 0.0762$.

Using our known $K, \lambda, W, \omega, \rho$ values, PLAINS can take any prespecified gap/mismatch parameters ms and ww , and report for sequences X and Y the overall alignment A obtained, along with all the “good” segment pairs, their scores and p values, and the R score and ζ value. In the event that $r = 0$, then $R = 0$ and PLAINS will report that no “good” segments were obtained.

2.4 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 $X[1, \dots, m/M]$ in the alignment; we set spot 2 to a color based on the match percentage found in $X[m/M + 1, \dots, 2m/M]$ in the alignment; we set spot i to a color based on the match percentage found in $X[(i-1)m/M + 1, \dots, im/M]$ in the alignment; and so on. The coloring grid for Y works in a similar way. Figures 3 and 3 are examples of this, with bright colors such as red, orange, yellow, and green signifying 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.

3 EMPIRICAL RESULTS

Two set of experiments were performed and used to compare PLAINS to similar alignment tools. The first set involved related sequences. The second set involved unrelated sequences. Tables 1 and 2 explain details regarding both sets of test runs, and appendix A elaborates further.

We have chosen to compare PLAINS against the similar localized DNA alignment tools of EMBOSS, LAGAN, and LALIGN. LAGAN uses piecewise-linear gap functions just like PLAINS, whereas EMBOSS and LALIGN use linear gap functions.

We made PLAINS optimize the approximate best gap/mismatch parameters based on the pair of species aligned, and the nature of the sequence. This is resemblant of LAGAN’s techniques to account for the nature of certain species in performing its alignments.⁹ In contrast, EMBOSS and LALIGN each use a fixed set of gap/mismatch parameters for all

⁹ 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.

Name	Species	Lengths	Sequence Nature	Large Gaps?	Identity Percentage in Homologous Regions
humanHomo1_15	Human vs. Mouse	8K vs.	coding	no	86%(<i>nt</i>)
humanHomo1_16		400-3000			90%(<i>nt</i>)
MousePseudo1	Mouse vs. Pseudogene	2.4K-9.6K	noncoding	yes	62%(<i>nt</i>)
MousePseudo2		vs.			55%(<i>nt</i>)
MousePseudo3	400-500	56%(<i>nt</i>)			
HumanPseudo1	Human vs. Pseudogene	1.5K-11.2K	noncoding	yes	83%(<i>nt</i>)
HumanPseudo2					vs.
HumanPseudo3		vs.			85%(<i>nt</i>)
HumanPseudo4		400-4000			74%(<i>nt</i>)
HumanPseudo5		400-4000			75%(<i>nt</i>)
HFugu2r	Human vs. Fugu	6K-12K	noncoding	yes	58%(<i>aa</i>)
HFortho1					vs.
HFortho2		vs.			52%(<i>aa</i>)
HFortho3		1.8K-3.6K			64%(<i>aa</i>)
HFortho4		1.8K-3.6K			52%(<i>aa</i>)
HFortho5		1.8K-3.6K			73%(<i>aa</i>)

Table 1. Sequence Descriptions for the Related Experiments Ran. All the sequences are retrieved from ENSEMBL database [www.ensembl.org]. Note for the rightmost column: (*nt*) indicates match percentage of nucleotides, (*aa*) indicates match percentage of amino acids after the two DNA sequences are transcribed into proteins.

Name	Species	Lengths	Sequence Nature
HFncd1	Human vs. Fugu	10K-20K	noncoding
HFncd2		vs.	
HFncd3		6K-10K	
FFcd1	Human, Fugu, and Mouse (All six combinations)	1.5K-4.8K	coding
HFcd1			
HHcd1		vs.	
HMcd1		1.5K-4.8K	
MFcd1		1.5K-4.8K	
MMcd1		1.5K-4.8K	

Table 2. Sequence Descriptions for the Unrelated Experiments Ran. All the sequences are retrieved from ENSEMBL database [www.ensembl.org].

species. We present a figure 3 showing the piecewise-linear gap functions that PLAINS came up with for each species pair¹⁰.

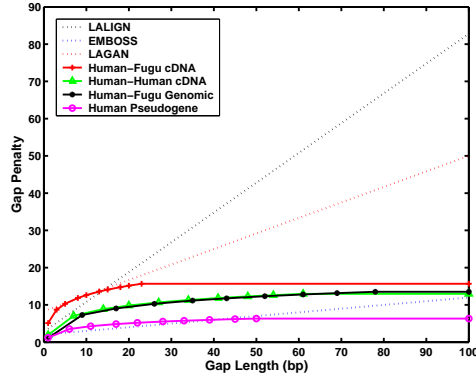


Fig. 1. The Piecewise-Linear Gap Functions that PLAINS came up with in optimizing the R score for different species pairs, along with the rescaled gap-parameters the other tools use. Note that the LAGAN gap parameters shown here are its default parameters. LAGAN uses a number of unspecified gap parameters in aligning on a species by species basis.

All scores for all alignment tools were obtained directly from the alignments and not the dynamic programming table, as outline earlier in “The PLAINS Scoring System” section. In tables 3 and 4, we see respectively the R , r , and ζ' values obtained from each run (where $\zeta' = -\log(\zeta)$). As explained earlier, r is the number of “good” segment pairs, R is the total score of all those segment pairs, and ζ is the probability of having our “good” segment pairs sum up to R . We choose to display the ζ' values instead of the ζ values, because ζ' values are easier to compare. Note that more reliable alignments correspond to smaller ζ values, and hence, larger ζ' values. When no “good” segment pairs are found, an X is placed as the ζ' value. In the HFncd2 and HFncd3 experiments, an X is placed as the R , r , and ζ' values for EMBOSS because EMBOSS ran out of memory aligning those experiments.

As these results show, PLAINS yielded the highest R and r values for the most part. However, because different alignment algorithms use different gap/mismatch parameters in different cases, it is actually the ζ' value which holds the most significance (with higher ζ' indicating that the alignment is less likely to happen “by chance”). Many of the genomic alignments yielded by the four tools have caught exons in the alignment, but most of these exons caught aren’t included in the “good” regions of the alignment, because the Karlin-Atschul probability concluded that their results are unreliable. Figure 3 is an example of this, since here, both PLAINS and LAGAN identify most of the exons in the human sequence, but we only count the exons that the tools identify as lying within the “good” regions.

The MousePseudo (alignments of Mouse genes against corresponding pseudogenes), and humanHomol (alignment of Human genes against homologous Mouse genes) runs were, for the most part, a relatively close competition between the four alignment tools,

¹⁰ 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 R scores reported in this paper are also divided by a tool’s reward-per-match value.

Test Name	PLAINS			EMBOSS			LAGAN			LALIGN		
	R	r	ζ'	R	r	ζ'	R	r	ζ'	R	r	ζ'
humanHomol_15	116.871	1	0.894	110.000	1	0.686	112.000	1	0.745	110.000	1	0.686
humanHomol_16	2184.956	1	68.343	1751.000	2	48.256	2064.917	1	64.370	1568.200	4	32.441
MousePseudo1	267.956	2	3.168	265.200	2	3.087	268.000	2	3.169	166.000	1	2.965
MousePseudo2	106.319	1	0.562	203.200	2	0.784	109.667	1	0.658	109.400	1	0.650
MousePseudo3	0.000	0	X	0.000	0	X	0.000	0	X	0.000	0	X
HumanPseudo1	281.992	3	1.881	255.200	3	1.437	83.250	1	0.470	84.400	1	0.502
HumanPseudo2	281.851	3	1.372	0.000	0	X	89.000	1	0.332	0.000	0	X
HumanPseudo3	1782.646	8	21.827	1066.700	5	11.998	1383.583	8	16.858	439.200	2	5.415
HumanPseudo4	110.620	1	0.538	0.000	0	X	0.000	0	X	0.000	0	X
HumanPseudo5	446.425	3	2.909	138.000	1	0.901	142.667	1	1.046	131.000	1	0.689
HFugu2r	322.784	2	2.523	0.000	0	X	0.000	0	X	0.000	0	X
HFortho1	282.933	2	1.381	0.000	0	X	0.000	0	X	0.000	0	X
HFortho2	452.657	2	6.158	234.600	1	3.808	215.667	1	3.182	0.000	0	X
HFortho3	627.357	4	3.894	0.000	0	X	0.000	0	X	0.000	0	X
HFortho4	737.478	4	6.071	0.000	0	X	0.000	0	X	0.000	0	X
HFortho5	217.274	1	3.328	0.000	0	X	0.000	0	X	0.000	0	X

Table 3. Related Run Scores for PLAINS, EMBOSS, LAGAN, and LALIGN.

Test Name	PLAINS			EMBOSS			LAGAN			LALIGN		
	R	r	ζ'	R	r	ζ'	R	r	ζ'	R	r	ζ'
HFncd1	0.000	0	X	0.000	0	X	0.000	0	X	0.000	0	X
HFncd2	0.000	0	X	X	X	X	0.000	0	X	0.000	0	X
HFncd3	0.000	0	X	X	X	X	0.000	0	X	0.000	0	X
FFcd1	0.000	0	X	0.000	0	X	0.000	0	X	0.000	0	X
HFcd1	0.000	0	X	0.000	0	X	0.000	0	X	0.000	0	X
HHcd1	139.604	1	1.568	0.000	0	X	0.000	0	X	0.000	0	X
HMcd1	341.872	2	3.517	0.000	0	X	0.000	0	X	0.000	0	X
MFcd1	0.000	0	X	0.000	0	X	0.000	0	X	0.000	0	X
MMcd1	0.000	0	X	0.000	0	X	0.000	0	X	0.000	0	X

Table 4. Unrelated Run Scores for PLAINS, EMBOSS, LAGAN, and LALIGN.

in terms of the actual alignments obtained, especially between PLAINS and LAGAN. This shows either the difference of linear gap functions over piecewise-linear gap functions, or the difference of using general-case gap parameters over using customized gap parameters per species, or possibly both.

For most of the HumanPseudo (alignments of Human genes against corresponding pseudogenes) runs, PLAINS and LAGAN yielded alignments with many similar correlations, but the results reported by PLAINS are more reliable (higher ζ' values). One illustration of this is figure 3, which compares the results of PLAINS to LAGAN for HumanPseudo5 in further detail. Furthermore, in HumanPseudo4, PLAINS was the only tool to catch any significant correlation.

The CD1 and HFncd experiments involved unrelated nonrandom sequences. This was why, for the most part, no correlation was caught by PLAINS, EMBOSS, LAGAN, and LALIGN in any of these experiments.¹¹ The correlations that PLAINS caught in the Human-Human and Human-Mouse CD1 experiments are protein codon homologies, most of them being a short stretch of perfect matches located relatively close to each other. Although these runs were meant to check how PLAINS behaves with unrelated sequences, the correlations PLAINS caught could ironically hold some sort of importance that has been usually ignored.

However, the most interesting results obtained were in the HFugu2r and HFortho runs, the runs involving genomic Human and Fugu sequences. Since the evolutionary distance between the human and fugu species is significantly long, one expects even the most conserved exon regions of the orthologous gene in the two genomes to have diverged quite a lot (despite the protein sequences still sharing high homology). Furthermore, 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 PLAINS shows over the other alignment tools is no small matter.

In all of these alignments except HFortho2, PLAINS is the only tool that catches significant matches. For HFortho2, PLAINS recieved the most reliable results, followed by EMBOSS, and then LAGAN. Here, not only is the ζ' value for PLAINS better than that of EMBOSS, but PLAINS also caught more common exons between the two related genomic sequences. Therefore, as PLAINS currently stands, it holds promise of becoming a tool of choice for aligning several thousand nucleotide DNA sequences, and possibly also for identifying exons between two genomes as diverged as human and fugu. Figure 3 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)

¹¹ Although this isn't reported in the paper, PLAINS EMBOSS, LAGAN, and LALIGN all catch no correlations when given randomly generated DNA sequences of lengths up to 8000.

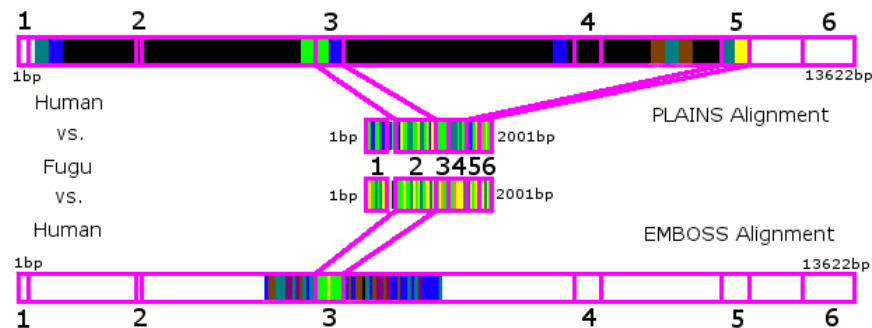


Fig. 2. Match Ratio Color Lines in the HFOOrtho2 test for PLAINS and EMBOSS. Here, the Human and Fugu sequence used have six exon regions that correspond to each other (though not necessarily in order, as exon region 2 in the Fugu sequence corresponds to exon region 3 in Human sequences for example). Here, both PLAINS and EMBOSS correctly identify the correlation of exon region 3 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.

there is constant extra overhead in using Linked-List Assistance (mentioned earlier) to help create an alignment.

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. More information can be found at <http://bioinformatics.nyu.edu/~gill>

4 CONCLUSIONS AND FUTURE WORK

PLAINS able to catch more important correlations than its competition, especially in sequences of distant relation like Human and Fugu. And, PLAINS is also capable of distinguishing unimportant regions from important ones (since PLAINS catches no important regions for randomly generated DNA sequences, and DNA sequences from Human/Mouse/Fugu with no protein codon homologies). All of this makes the results yielded by PLAINS quite satisfying.

However, it has become apparent that some upwards scaling is essential if PLAINS is to be run over sequences with more than 8000 nucleotides. Plains, as it stands, essentially performs localized alignment (since it discards leftmost and rightmost nucleotides that would be aligned against gaps), and therefore, it is only fit for sequences of up to 8000 nucleotides.

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

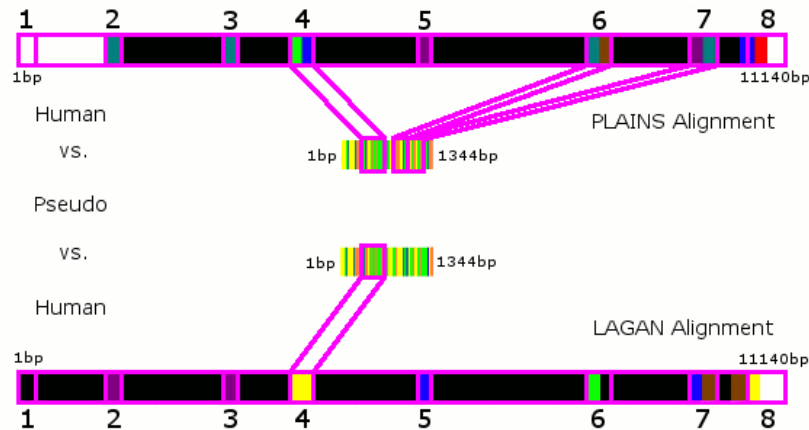


Fig. 3. Match Ratio Color Lines in the HumanPseudo5 test for PLAINS and LAGAN. Here, the Human sequence has 8 exon regions that are similar to areas of the pseudo-sequence used, and alignments of PLAINS and LAGAN for these cases are similar, even by eyegance of the ColorGrids. Note that although PLAINS and LAGAN catch most of these regions in their alignments, we're only counting the exon regions that participated in "good" segment pairs. With this in mind, PLAINS and LAGAN both identify exon region 4 as important, but PLAINS also deems exon regions 6 and 7 in the Human sequence as important, which LAGAN misses.

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

Further extension includes development of better statistics taking into account the base-pair compositions of the sequences (e.g., Nucleotide-bias, CG content, dinucleotides distribution, codon bias, etc.). Plains, at the moment, assigns a score of 1 to a perfect match, and a score of ms (specified by user) to any mismatch. It may be useful to have a scoring matrix to assign different scores to different types of matches/mismatches. (For example, if aligning a C against a T is more common than aligning at C against a G , then perhaps we can penalize the $C-T$ mismatch less than the $C-G$ mismatch when performing an alignment, etc.)

References

1. Altschul, S.F., Boguski, M.S., Gish, W., and Wooton, J.C., "Issues in Searching Molecular Sequence Databases." *Nature Genetics*, **6**:119–128, 1994.
2. Michael Brudno, Chuong Do, Gregory Cooper, Michael F. Kim, Eugene Davydov, Eric D. Green, Arend Sidow, Serafim Batzoglou, "LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA," *Genome Research*, **13**(4):721-31, 2003 Apr.
3. Craig G. Nevill-Manning, Cecil N. Huang, Douglas L. Brutlag, "Pairwise protein sequence alignment using Needleman-Wunsch and Smith-Waterman algorithms," Personal communication (<http://motif.stanford.edu/alion/>), 1997.
4. 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.
5. Hromkovic J, "Heuristics." *Algorithms for Hard Problems, Second Edition*, **6**:439-467, 2003.
6. X. Huang and W. Miller, *Advanced Applied Mathematics*, **12**:373-381, 1991.
7. 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.

8. 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.
9. 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.
10. Miller, W., and Myers E.W., "Sequence Comparison with Concave Weighting Functions" *Bulletin of Mathematical Biology*, **50**:97–120, 1988.
11. Miller, W., and Myers E.W., "Optimal Alignments in Linear Space" *CABIOS*, **4**:11–17, 1988.
12. 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.
13. 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.
14. Pearson, W.R., "Comparison of Methods for Searching Protein Sequence Databases." *Protein Science*, **4**:1145–1160, 1995.
15. 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.
16. 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.
17. Rice P, Longden I, Bleasby A., "EMBOSS: the European Molecular Biology Open Software Suite" *Trends Genetics*, **Jun 16(6)**:276-7, 2000.
18. Smith, T.F., and Waterman, M.S., "Identification of Common Molecular Subsequences." *Journal of Molecular Biology*, **147**: 195–197, 1981.
19. 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.
20. States, D.J., Gish, W., and Altschul, S.F., "Basic Local Alignment Search Tool." *Journal of Molecular Biology*, **215**: 403–410, 1990.
21. 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.
22. 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.

Appendix

A NAMES OF THE SEQUENCES USED

Tables 5 and 6 list the specific sequences used in the experiments ran on PLAINS, EMBOSS, LAGAN, and LALIGN.

Name	First Sequence	Second Sequence
humanHomol_15	ENST00000263253	ENSMUST00000050968
humanHomol_16	ENST00000263253	ENSMUST00000068387
MousePseudo1	ENSMUSG00000016720	pseudogene
MousePseudo2	ENSMUSG00000004038	pseudogene
MousePseudo3	ENSMUSG00000034321	pseudogene
HumanPseudo1	ENSG00000087086	pseudogene
HumanPseudo2	ENSG00000164104	pseudogene
HumanPseudo3	ENSG00000079432	pseudogene
HumanPseudo4	ENSG00000135486	pseudogene
HumanPseudo5	ENSG00000101210	pseudogene
HFugu2r	ENSG00000111845	SINFRUG00000137119 (reverse-complement)
HFortho1	ENSG00000183628	SINFRUG00000128815
HFortho2	ENSG00000099937	SINFRUG00000140660
HFortho3	ENSG00000142168	SINFRUG00000132716
HFortho4	ENSG00000138764	SINFRUG00000152968
HFortho5	ENSG00000057757	SINFRUG00000123004

Table 5. Sequence Details for the Related Experiments Ran. All the sequences are retrieved from ENSEMBL database [www.ensembl.org].

Name	First Sequence	Second Sequence
HFncd1	Human NCBI34:1:190164774:190174772	FUGU2:scaffold_5343:1:5999:1
HFncd2	Human NCBI34:22:32724006:32744004:1	FUGU2:scaffold_3421:1:9999:1
HFncd3	Human NCBI34:10:56721585:56731583:1	FUGU2:scaffold_1415:1:6999:1
FFcd1	SINFRUT00000127255	SINFRUT00000165154
HFcd1	ENSG00000150967.3	SINFRUT00000127255
HHcd1	ENSG00000150967.3	ENST00000259748
HMcd1	ENSG00000150967.3	ENSMUST00000025930
MFcd1	ENSMUST00000025930	SINFRUT00000165154
MMcd1	ENSMUST00000031354	ENSMUST00000024034

Table 6. Sequence Details for the Unrelated Experiments Ran. All the sequences are retrieved from ENSEMBL database [www.ensembl.org].

B HOW PLAINS GENERATES AN ALIGNMENT

Recall from earlier that the two strings to be aligned are denoted X and Y , and their respective lengths are m and n , and the penalty for each mismatch is denoted as ms .

Also recall our p -part piecewise-linear function $ww(\cdot)$, where, for some value i such that $k_{\bar{p}-1} < i \leq k_{\bar{p}}$, $ww(i)$ is defined as:

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

Note that, for any i from 1 to m , a simple hashtable can help us figure out in $O(1)$ time¹² which line of our piecewise-linear curves that an x -value of i is at. This, combined with explicitly saving to memory $ww(k_u)$ for all u from 0 to $p - 1$, lets us compute $ww(i)$ for any given i from 1 up to m in $O(1)$ time.

Using a given p -part piecewise-linear gap function $ww(\cdot)$ and mismatch penalty ms , PLAINS generates an alignment for X and Y of lengths m and n 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 $X[1 \dots i]$ with $Y[1 \dots j]$. 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). Because we are dealing with simple alignment of DNA sequences, assume $s(c, d)$ gives a score of 1 when $c = d$, and score of $-ms$ when $c \neq d$.

With all of this, dynamic programming scoring model for an alignment used by PLAINS is:

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

If p , the number of lines used in the $ww(\cdot)$ function is set to 1, the scoring model mentioned above resembles Smith-Waterman¹³. If p is set to m , then the scoring model mentioned above resembles Needleman-Wunsch.

B.1 Linked-List Assistance

Plains uses a Linked-List Assistance technique similar to that of Miller and Myers [10]. This technique involves considering possible solutions for the $E(\cdot, \cdot)$ and $F(\cdot, \cdot)$ entries before we explicitly compute them. To gain an intuition into this technique, first suppose that j is fixed in order to keep the discussion simple for the moment. Next, let $eval(k, i) = V(k) - ww(i - k)$, and let $cand_k(i)$ denote the k' value, where $k' \leq k$, such that $eval(k', i)$

¹² Since the length of a gap is at most m , we will never need to compute $ww(i)$ for any i larger than m .

¹³ Except that Smith-Waterman modifies $E(\cdot, \cdot)$ and $F(\cdot, \cdot)$ to improve runtime specifically for the case of affine gap functions

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 i th 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 k th iteration of our algorithm, we know that, for some a, b, q and for all i' in $[a, b]$, $q = cand_k(i')$. Then we can represent this fact with one data structure, instead of $b - a + 1$ of them.
- (S2) In all practical cases, our gap function ww is convex (meaning that $ww(i)$ 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 i th iteration, we were to scan the $cand_i(\cdot)$ values in the order: $cand_i(i + 1), cand_i(i + 2), \dots, cand_i(m)$, then we would see that the $cand_i(\cdot)$ entries are nonincreasing (each next $cand_i(\cdot)$ entry is either smaller or equal to the previous one).

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

winner = i	winner = 3	winner = 2
v = 56	v = 12	v = 7
lwb = i+1	lwb = x+1	lwb = r+1
upb = x	upb = r	upb = q

Furthermore, from (S2), we know that on the i th 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 i th 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 i th iteration, there is an element in L is such that $cand_i(lwb) = i$, but $cand_i(upb) \neq i$ and we need to know which index within the group is the largest a such that $cand_i(a) = i$, then, we simply take the element's previous winner value of k , its v value, and consider for x the plots of $v - ww(x - k)$ versus $V(i) - ww(x - i)$. We seek the point where the first plot intersects the second one. The two plots involve p -part piecewise-linear curves, and binary search over the lines of these curves (which takes $O(\log p)$ time)

tells us which lines on these curves contain the intersection. At this point, we can get the intersection point itself in $O(1)$ time. From this, we get the largest a such that $cand_i(a) = i$, with $lwb \leq a \leq upb$, and then update the elements of L accordingly.

This algorithm uses list L to obtain solutions for $F(\cdot)$ in $O(m \log p)$ time. At any time, PLAINS always has $O(p)$ elements in L (and $p \ll m$ typically). The proof for the main algorithmic difference over the Miller-Myers method: the $O(p)$ worst-case space complexity of L , is presented in appendix D.

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, \dots, L_n and each list is updated in the manner explained earlier), and for each column i , we compute $E(i, \cdot)$ using the help of a list R (which gets updated in a manner similar to that of L for the F entries, except that when we finish computing a column of our table, we empty R so it can be reused when proceeding to the next column). Clearly, the updates for R and each L_j list are interweaved. This implies $O(mn \log p)$ time complexity, and further implies that all lists combined take up $O(np)$ space, assuming each list uses $O(p)$ space.

See appendix sections C and D for an explanation of how PLAINS uses $O(n)$ space from the $V(\cdot, \cdot)$, $E(\cdot, \cdot)$, $F(\cdot, \cdot)$, and $G(\cdot, \cdot)$ tables, and a proof of correctness for the scheme used.

C TABLE SPACE REDUCTION

PLAINS uses only $O(n)$ space for its dynamic programming tables V , E , F , and G because it exploits a method, generalizing several ideas first described by Hirschberg, and later Miller-Meyers[11]. This 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 .¹⁴ We maintain lists L_j^r to assist in computing $F^r(\cdot, j)$, and list R^r to assist in computing each i th column of $E^r(i, \cdot)$ in a manner similar to the way we used L_j for $F(\cdot, j)$ and R for $E(i, \cdot)$. However, PLAINS saves only the t most recently computed columns of V , E , F , G , V^r , E^r , F^r , and G^r , where t is some fixed constant¹⁵.

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 $Y[1..n]$), PLAINS uses a “maximum criteria” explained below to obtain a “middle” subalignment from the saved portions of V , E , F , G , and V^r , E^r , F^r , G^r . With this “middle” subalignment known, and just as in Hirschberg’s algorithm, PLAINS makes two recursions, one to obtain a subalignment for the left characters of X and Y , and another to obtain a subalignment for the right characters of X and Y . We glue all of these subalignments together to get the overall alignment for X and Y .

¹⁴ 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 .

¹⁵ We can correctly maintain the L_j , L_j^r , R and R^r lists and correctly compute all table entries while only saving the t most recent columns, since any earlier important V and V^r values we need to look at are saved in the lists themselves.

This method of using $O(n)$ space for the tables to get an alignment increases overall runtime by at most a constant factor compared to the intuitive $O(mn)$ space method of saving all columns of all tables. Hence, the overall runtime for PLAINS in creating an alignment is $O(mn \log p)$.

To explain the “maximum criteria” selection, 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 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 proven by Hirschberg. Therefore, when $p = 1$, it is satisfactory to select our “maximum criteria” to select a k such that $gr(k)$ is maximized, then use $V(m/2, k)$ and $V^r(m/2, n - k)$ to obtain two subalignments from the saved columns of V and V^r based on this. Then, we glue these subalignments to make a “middle” subalignment (and this is essentially the subalignment that uses middle bits of X).

When $p > 1$, each L_j list is computed assuming the indices i can range from 0 to m , not 0 to $m/2$ (even though that may be all we need for the V table). Then, by the end of computing the V table, we will use L_j while computing the V^r table and L_j^r lists¹⁶ to obtain $rc(j)$ values¹⁷ 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 $V^r(m - k', n - k) + eval_k(cand_{m/2}^L(k', k), k')$, 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 $cand_{m/2}^L(rc(k), k)$ and $rc(k)$ the right and left endpoints to this gap, and we will use this to construct a subalignment with this gap, and use whatever we saved of V and V^r to obtain any additional subalignment parts for characters left and right of this gap. All of this combined gives us our “middle” subalignment.

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

The proof of correctness for our selection of k in this manner is deferred to the next section.

D PROOFS TO THE NONTRIVIAL 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

¹⁶ Note that while computing V^r , we are only going to read entries from L_j , not make any changes to it.

¹⁷ 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.

in the worst-case scenario. Second, under its given space and runtime bounds, PLAINS obtains the correct alignment based on the given piecewise-linear function $ww(\cdot)$.

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

D.1 PROOF FOR THE $O(np)$ SPACE BOUND, THE PLAINS INNOVATION

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

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

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

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

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

PROOF: In the beginning, when $i = 0$, L starts with one element. Later on, in some i th iteration, after we just finished computing $V(i)$, if we split an element of L with winner k and value v (where $v = V(k)$), then this implies that, for some x , the k th plot of $v - ww(x - k)$ intersects the i th plot of $V(i) - ww(x - i)$ (i.e., $V(i) - ww(x - i) = v - ww(x - k)$ for some x). However, both of these plots are identical in shape. By translating one horizontally and vertically, this plot can fit perfectly into the other.

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

Hence, if list L has p elements, this implies having found $p - 1$ intersections, each from a different plot. This means that we have $cand(\cdot)$ values taken from the ur_1 th line of some q_1 plot intersecting the ul_2 th line of some q_2 plot, and the ur_2 th line of the q_2 plot intersects the ul_3 th 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 , $ul_h = wr_h$ 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_h th 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 i th iteration, we have p elements in L , then the i plot of $V(i) - ww(x - i)$ will have lines of the same slopes as those corresponding to lines ul_1 through ul_p . Therefore, if the p' th-line of the i th plot intersects some $q_{h'}$ plot (with $h' \geq 2$), then all elements of L derived from q_h plots with $ul_h \leq p'$ will be discarded (i.e., at least one element of L will definitely get discarded, and one new element with i as its $cand(\cdot)$ value is created, implying that total number of elements in L overall in this i th iteration will either stay the same or decrease). Note that it is impossible for the i -curve's p' th-line to intersect the q_1 plot.

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

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

D.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 constant¹⁸ at least 1.

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

Suppose also that while computing V , list L maintains $cand.(i)$ 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).

¹⁸ 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.

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 $V(cand_{m/2}^L(i, j), j) - ww(i - cand_{m/2}^L(i, j)) + V^r(m - i, n - j) = eval_j(cand_{m/2}^L(i, j), i) + V^r(m - i, n - j) = er(j, i)$ is maximized.

D.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) occurs¹⁹, it is feasible to align $X[1..m/2]$ against Y separately from aligning $X[m/2..m]$ against Y . Furthermore²⁰, 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 $i = cand_{m/2}^L(i', j)$, 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 $i = cand_{m/2}^L(i', j)$ maximizes $V(i, j) + V^r(m - i', n - j) - ww(i' - i) = V(cand_{m/2}^L(i', j), j) - ww(i' - cand_{m/2}^L(i', j)) + V^r(m - i', n - j) = eval_j(cand_{m/2}^L(i', j), i') + V^r(m - i', n - j) = er(j, i')$. 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.

¹⁹ This is essentially what the V and V^r tables do.

²⁰ Therefore for this k' value, $gr(k') > er(k', rc(k'))$.