

# Sequence Alignment in Linear Space and Quadratic Time with Piecewise-Linear Approximation of Gap Penalty: Theory and Practice

Ofer H. Gill<sup>1</sup> and Bud Mishra<sup>1,2</sup>

<sup>1</sup> Courant Institute, New York University,  
251 Mercer Street,  
New York, NY 10012, USA,  
{gill,mishra}@cs.nyu.edu,

WWW home page: <http://www.cs.nyu.edu/cs/faculty/mishra/>

<sup>2</sup> Watson School of Biological Sciences, Cold Spring Harbor Laboratory,  
Demerec Building, 1 Bungtown Road,  
Cold Spring Harbor, NY 11724, USA

**Abstract.** The unprecedented growth of biological sequence databases provides great challenges and opportunities for molecular and computational biologists. As more and more biologists use these accumulated data in their attempt to discover the biological functions of genes and the proteins they encode, it has become quite important to improve the space and time complexity of these algorithms as well as their fidelity to the underlying biology.

Using a general statistical model, based on the probability distributions of point mutations, point insertions, point deletions and arbitrary size gaps, one can formulate an accurate approach to compare two biological sequences through their Levenshtein distance (or a cost function based on negative log-likelihood) and suggest a dynamic programming solution through Hamilton-Jacobi-Bellman equation. One solution (Needleman-Wunsch) based on general assumptions was proposed but was found to be computationally inefficient. As a result, an approximate solution (Smith-Waterman using “affine gap model”) has found a wider application, instead. Here, we propose a better approximation solution (“piece-wise affine gap model” allowing arbitrarily closer approximation for exponential, power-law and other reasonable distributions for gap lengths); this solution does not entail sacrificing the asymptotic computational complexity of Smith-Waterman.

## 1 Introduction

Even a reasonable implementation of localized pairwise protein (or nucleotide) sequence alignment for an arbitrary gap penalty function can prove impractical as a result of their time and space complexity, in spite of an efficient dynamic programming model [4, 7]. Making certain simple approximating assumptions

about the gap penalty function allows the time and space to be significantly reduced, as is shown in this paper.

This paper focuses on using piecewise-linear gap penalty functions for pairwise local sequence alignment. From a mathematical point of view, piecewise-linear functions are derived from linear functions, but can be constructed to resemble general functions with little loss of accuracy. Therefore, piecewise-linear functions allow us to achieve the fast memory-saving results of linear functions dynamic programming, while maintaining close fidelity to general gap functions (and hence the underlying biology).

This paper is organized as follows: First a discussion of the typical dynamic programming models for general gap and linear gap formulas are discussed. Next, a formulation of the dynamic programming model for two-part piecewise-linear gap formulas is discussed. We then generalize the formulation for general piecewise-linear gap formulas. Finally, we conclude by proposing how to reduce the space used from quadratic to linear when considering piecewise-linear gap penalty functions.

## 2 General Gap Formula

The problem of local pairwise sequence alignment can be described as follows:

Given two sequences  $X$  and  $Y$ , of sizes  $m$  and  $n$ , respectively, we wish to find an alignment to best emphasize the blocks of  $X$  and  $Y$  have in common, and the blocks where  $X$  and  $Y$  differ.

For the sake of simplicity, assume that we are explicitly given a general gap penalty function  $w(\cdot)$  and a matching function  $s(\cdot, \cdot)$ . Score function  $s(\cdot, \cdot)$  takes two values as input and returns a score based on their match or mismatch. Gap function  $w(\cdot)$  takes in one parameter  $i$ , and  $w(i)$  denotes the nonnegative penalty given to a gap of length  $i$ . We can create a dynamic programming model as follows:

- Let  $V(i, j)$  denote the best score obtainable for the prefixes  $X[1 \dots i]$  and  $Y[1 \dots j]$ .
- Let  $G(i, j)$  denote the best score obtainable assuming we align  $X[i]$  with  $Y[j]$  (regardless of if  $X[i]$  and  $Y[j]$  match).
- Let  $E(i, j)$  denote the best score obtainable assuming we align  $Y[j]$  against a gap.
- Let  $F(i, j)$  denote the best score obtainable assuming we align  $X[i]$  against a gap.

Note further that, for the sake of exposition, we make in the rest of this paper the following simplifying assumption regarding  $s(\cdot, \cdot)$ :

$$s(X[i], Y[j]) = \begin{cases} 1, & \text{if } X[i] = Y[j]; \\ 0, & \text{otherwise.} \end{cases}$$

Our goal is to optimize a global cost function  $V(m, n)$  and determine the alignment corresponding to it, where  $V$  can be described inductively as follows:

$$V(0, 0) = 0$$

$$V(i, 0) = E(i, 0) = -w(i)$$

$$V(0, j) = F(0, j) = -w(j)$$

$$V(i, j) = \max\{E(i, j), F(i, j), G(i, j)\}$$

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

$$E(i, j) = \max_{0 \leq k \leq j-1} [V(i, k) - w(j - k)]$$

$$F(i, j) = \max_{0 \leq k \leq i-1} [V(k, j) - w(i - k)]$$

Intuitively, we typically wish to discourage short gaps, yet encourage long gaps. Therefore,  $w(i)$  is commonly chosen so that it monotonically increases as  $i$  increases, but the rate of increase slows down with increasing values of  $i$ ; in other words,  $w(i)$  has a positive derivative, but a negative double-derivative. In some cases, the function  $w(i)$ , for large values of  $i$ , asymptotically becomes constant-valued, i.e., eventually  $w(i)$  “flattens out” with increasing  $i$ .

Also, in the same time needed to compute a max or min functions, we can save the location where the optimal function argument came from. Therefore we can, without affecting the asymptotic runtime, store in each entry of the dynamic-programming table  $V$  “trace-backs” to previously computed results, making it trivial to backtrack our table to get the alignment we found for sequences,  $X$  and  $Y$ .

The dynamic programming model described earlier for general gap-penalty function  $w(\cdot)$  uses  $O(mn)$  space. Furthermore, to compute  $V(m, n)$ , for each entry, we need to consider  $m + n + 1 = O(m + n)$  previously computed entries. Hence, our lookbehind size<sup>3</sup> for  $V(i, j)$  is  $O(m + n)$ . Therefore, the overall runtime is  $O(mn \times (m + n)) = O(m^2n + mn^2)$ . Hence, the most general algorithm for biological sequence comparison takes cubic time and quadratic space. Thus, these algorithms are usable for sequences of length of few *kb*'s but not larger.

Thus a natural question for computer scientists is the following: *Can we do better?* If  $w(\cdot)$  is restricted, the answer is yes.

### 3 Linear Gap Formula

When  $w(i)$  is a linear formula of form  $w_c i + w_o$ , where  $w_c$  is the rate of increase and  $w_o$  is the initial cost, we can reduce our runtime by reducing the number of

<sup>3</sup> *Lookbehind* is defined as the number of previously computed table entries we have to look at in order to correctly compute the current table entry.

lookbehinds. The reason for this is that, since the gap penalty always increases by  $w_c$  once the gap is longer than one, we need not worry where a gap begins—only whether it already began, or a new gap has just started.

Thus, the dynamic programming recurrence equation for  $V(\cdot, \cdot)$  becomes the following:

$$V(0, 0) = 0$$

$$V(i, 0) = E(i, 0) = -w_o - i \cdot w_c$$

$$V(0, j) = F(0, j) = -w_o - j \cdot w_c$$

$$V(i, j) = \max\{F(i, j), E(i, j), G(i, j)\}$$

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

$$E(i, j) = -w_c + \max\{E(i, j - 1), V(i, j - 1) - w_o\}$$

$$F(i, j) = -w_c + \max\{F(i - 1, j), V(i - 1, j) - w_o\}$$

In this case, each entry of  $V(\cdot, \cdot)$  is computed using only  $5 = O(1)$  lookbehinds. Therefore, overall runtime is  $O(mn)$ . Also, the space used here is still  $O(mn)$ . Hence, while we are still using quadratic space, we have reduced our runtime to quadratic time.

While this affine gap penalty assumption makes the solution practical for reasonably long biological sequences of interest, it unnecessarily restricts us to very small classes of gap models. Furthermore, as we learn more about the dynamics of genome evolution [11], it is becoming very apparent that models supported by Smith-Waterman may be unrealistic and introduce distortions into the underlying biology. In particular, linear and affine gap-penalty functions sacrifice our ability to decrease the rate of gap penalty increase as our gap penalty gets larger and thus unnaturally force the algorithm to select smaller gaps. Piecewise-linear functions serve a simple heuristic to introduce minimal distortions into the underlying biology while delivering algorithms that are as practical as Smith-Waterman. In particular, we will show next, how one can get the same runtime and space results as we got for simple linear and affine functions, but over general piecewise-linear functions. In order to keep the exposition accessible, we start with a “two-part piece-wise linear gap penalty formula.”

## 4 Two-Part Piecewise-Linear Gap Formula

Before examining the model for general piecewise-linear formulas, it will help as a stepping stone to understand the dynamic programming algorithm for the simpler case involving two-part piecewise-linear gap formula. Furthermore, this simpler model is adequate to alleviate most of the serious problems with Smith-Waterman and is likely to be the one that finds wide usage.

A two-part piecewise-linear gap formula  $w(i)$  will take the form of:

$$w(i) = \begin{cases} w_{c[0]} \cdot i + w_o, & \text{if } i < k; \\ w_{c[1]} \cdot (i - k) + w_{c[0]} \cdot k + w_o, & \text{if } i \geq k. \end{cases}$$

Here, as before,  $w_o$  is the cost of starting a gap;  $w_{c[0]}$  is the rate of penalty increase for any gap below size  $k$ , and  $w_{c[1]}$  is the rate of increase for any gap of size  $k$  or larger<sup>4</sup>.

In this case, we will use five tables to derive the  $V$  table (instead of just three tables as in Smith-Waterman). These five tables are denoted  $G$ ,  $E_0$ ,  $E_1$ ,  $F_0$ , and  $F_1$ .

- $G(i, j)$  and  $V(i, j)$  behave the same way as before.
- $E_0(i, j)$  denotes the score if we align  $Y[j]$  against a gap of length less than  $k$ .
- $E_1(i, j)$  denotes the score if we align  $Y[j]$  against a gap of length  $k$  or larger.
- $F_0(i, j)$  and  $F_1(i, j)$  behave similarly, but for aligning  $X[i]$  against a gap.
- In this model, for  $E_0$  and  $F_0$ , the gap penalty always increases by  $w_{c[0]}$  once the gap is longer than one but smaller than  $k$ , hence we need not be concerned as to where a gap begins ( $E_1$  and  $F_1$  will account for if the gap exceeds size  $k$ ). So for  $E_0$  and  $F_0$ , we only need “remember” if a gap has already begun, or a new gap is started.
- Also, for  $E_1$  and  $F_1$ , once the gap is larger than  $k$ , we do not need to “remember” where it began, and the gap penalty always increases by  $w_{c[1]}$ . So, we only need to do book-keeping for the cases where a gap larger than  $k$  was already begun, or where a new gap of size at least  $k$  is started, based on  $E_0$  and  $F_0$ , which are gaps of size at least one.

The dynamic programming recurrence equations for  $V(\cdot, \cdot)$  are as follows:

*Base Cases:*

$$V(0, 0) = 0;$$

$$V(i, 0) = E_0(i, 0) = -w_o - i \cdot w_{c[0]},$$

if  $i < k$ ;

$$V(i, 0) = E_1(i, 0) = -w_o - k \cdot w_{c[0]} - (i - k) \cdot w_{c[1]},$$

if  $i \geq k$ ;

$$V(0, j) = F_0(0, j) = -w_o - j \cdot w_{c[0]},$$

if  $j < k$ ;

$$V(0, j) = F_1(0, j) = -w_o - k \cdot w_{c[0]} - (j - k) \cdot w_{c[1]},$$

if  $j \geq k$ .

---

<sup>4</sup> In one interesting case, we may set  $w_{c[1]}$  to zero to approximate a logarithmic penalty function deriving from a power-law distribution.

*Inductive Cases,  $i > 0 \wedge j > 0$ :*

$$V(i, j) = \max\{F_1(i, j), F_0(i, j), E_1(i, j), E_0(i, j), G(i, j)\};$$

$$G(i, j) = V(i - 1, j - 1) + s(X[i], Y[j]);$$

$$E_0(i, j) = -w_{c[0]} + \max\{E_0(i, j - 1), V(i, j - 1) - w_o\};$$

$$E_1(i, j) = \max\{E_1(i, j - 1) - w_{c[1]}, E_0(i, j - (k - 1)) - (k - 1) \cdot w_{c[0]}\},$$

$$\quad \text{if } j \geq k;$$

$$= -\infty, \quad \text{otherwise};$$

$$F_0(i, j) = -w_{c[0]} + \max\{F_0(i - 1, j), V(i - 1, j) - w_o\};$$

$$F_1(i, j) = \max\{F_1(i - 1, j) - w_{c[1]}, F_0(i - (k - 1), j) - (k - 1) \cdot w_{c[0]}\},$$

$$\quad \text{if } i \geq k;$$

$$= -\infty, \quad \text{otherwise.}$$

Note that, just as in Smith-Waterman, in our two-part piecewise-linear gap model, we perform operations on  $O(1)$  lookbehinds to compute each  $V(i, j)$  entry. Therefore, our runtime is  $O(mn)$ ; also, our space complexity remains  $O(mn)$ .

## 5 General Piecewise-Linear Gap Formula

In the general piecewise-linear gap formula, we assume that we are given a  $(p+1)$ -part piecewise-linear function  $w(\cdot)$  which contains a penalty for starting a gap, denoted  $w_o$ , and  $(p+1)$  different slopes, denoted  $w_{c[0]}, w_{c[1]}, \dots$  and  $w_{c[p]}$ , as well as  $p$  values  $k[1], k[2], \dots, k[p]$  where the slopes change.

Assuming that  $k[0] = 0$  and  $k[p+1] = \infty$ , we can write  $w(i)$  as follows: Let  $u(i) \geq 0$  be an index such that  $u(i) = \max\{u' : k[u'] \leq i\}$ , then

$$w(i) = w_o + \left( \sum_{v=1}^{u(i)} [(k[v] - k[v-1]) \cdot w_{c[v-1]}] \right) + (i - k[u(i)]) \cdot w_{c[u(i)]}.$$

In this case, we use the  $i^{\text{th}}$  and  $j^{\text{th}}$  entries of tables  $E_0, E_1, \dots, E_p$  and  $F_0, F_1, \dots, F_p$  to compute the scores for aligning  $X[i]$  and  $Y[j]$  against gaps. In addition, for each  $u$ ,  $E_u$  and  $F_u$  correspond to the  $u^{\text{th}}$  slope in our gap function. The precise reasoning for constructing these tables is similar to the two-part piecewise-linear gap formula, and is left to the reader as an exercise.

The dynamic programming recurrence equations for  $V(\cdot, \cdot)$  are as follows:

*Base Cases:*

$$V(0, 0) = 0;$$

$$V(i, 0) = E_u(i, 0) = -w(i), \quad \text{if } \exists u(k[u] \leq i < k[u+1]);$$

$$V(0, j) = F_u(0, j) = -w(j), \quad \text{if } \exists u(k[u] \leq j < k[u+1]).$$

*Inductive Cases,  $i > 0 \wedge j > 0$ :*

$$V(i, j) = \max\{F_p(i, j), \dots, F_1(i, j), F_0(i, j), \\ E_p(i, j), \dots, E_1(i, j), E_0(i, j), G(i, j)\};$$

$$G(i, j) = V(i - 1, j - 1) + s(X[i], Y[j]); \\ E_0(i, j) = -w_{c[0]} + \max\{E_0(i, j - 1), V(i, j - 1) - w_o\}; \\ F_0(i, j) = -w_{c[0]} + \max\{F_0(i - 1, j), V(i - 1, j) - w_o\};$$

For  $u > 0$ ,

$$E_u(i, j) = \max\{E_u(i, j - 1) - w_{c[u]}, \\ E_{u-1}(i, j - (k[u] - k[u - 1])) - (k[u] - k[u - 1]) \cdot w_{c[u-1]}\}, \\ \text{if } j \geq k[u]; \\ = -\infty, \quad \text{otherwise}; \\ F_u(i, j) = \max\{F_u(i - 1, j) - w_{c[u]}, \\ F_{u-1}(i - (k[u] - k[u - 1]), j) - (k[u] - k[u - 1]) \cdot w_{c[u-1]}\}, \\ \text{if } i \geq k[u]; \\ = -\infty, \quad \text{otherwise.}$$

Assuming that  $p$  is a parameter to be selected, in order to achieve an appropriate approximation, each  $V(i, j)$  entry uses  $2p + 1 = O(p)$  lookbehinds to compute its value, and we use  $O(mnp)$  memory overall. Our runtime is  $O(mnp)$ . Hence we use cubic time and memory under this assumption. (Essentially, we get the same behavior as the typical global alignment algorithm.)

However, in practice for almost all gap penalty functions of interest,  $p$  is a small constant,  $p = O(1)$ , and we use  $O(mn)$  time and  $O(mn)$  space—quadratic time and space, just as in the linear gap formula model. In our empirical studies, we found that a chosen gap function using  $p \leq 10$  can achieve acceptable approximations while using quadratic time and space, and also, that this algorithm is of enormous practical value.

Now that we have a dynamic programming algorithm for piecewise-linear gap formula alignment in quadratic space, it is natural to ask: *Can we do better?* The next section answers this question.

## 6 Reducing to Linear Space

Hirschberg [2] has described a scheme that can be adapted to our problem to obtain an alignment from the linear gap model and general piecewise-linear gap model in  $O(n)$  space and  $O(mn)$  time. In order to achieve this, we note the following:

For  $X$  and  $Y$ , let  $X^r$  and  $Y^r$  denote the reversals of  $X$  and  $Y$ , respectively, and let  $V^r(i, j)$  denote the score for  $X^r[1 \dots i]$  and  $Y^r[1 \dots j]$ . We can compute  $V^r$  in the same way as  $V$ , and  $V^r(i, j)$  gives us the alignment of the last  $i$  characters of  $X$  with the last  $j$  characters of  $Y$ .

Thus, we can compute  $V(\frac{m}{2}, n)$  and  $V^r(\frac{m}{2}, n)$ . And also,  $V(m, n)$ , assuming that:

$$V(m, n) = \max_{0 \leq q \leq n} \left[ V\left(\frac{m}{2}, q\right) + V^r\left(\frac{m}{2}, n - q\right) \right].$$

This observation implies that computing  $V(\frac{m}{2}, n)$  and  $V^r(\frac{m}{2}, n)$  allows us to find  $V(m, n)$ . Essentially, we split  $X$  into half, and look for a way to split  $Y$  such that the left part of  $Y$  aligns with the first half of  $X$ , and the right part of  $Y$  aligns with the second half. By finding the split of  $Y$  so that our calls to  $V(\frac{m}{2}, n)$  and  $V^r(\frac{m}{2}, n)$  are maximized, we essentially find the character in  $Y$  such that our optimal “trace-back” in the table for  $V(m, n)$  goes through the half-point<sup>5</sup> for  $X$ .

We record whatever alignment data we found from the calls to  $V(\frac{m}{2}, n)$  and  $V^r(\frac{m}{2}, n)$ . If we use linear space in these calls, where  $t$  is a parameter denoting the number of most recently computed columns we save, then we only have the last  $t$  columns from the calls to  $V(\frac{m}{2}, n)$  and  $V^r(\frac{m}{2}, n)$ . Therefore, we only have the ending portion of the alignment from the  $V(\frac{m}{2}, n)$  call, and the starting portion of the alignment from the  $V^r(\frac{m}{2}, n)$  call. Once we found the correct point in  $Y$  to split, we can repeat ourselves recursively on the left parts of  $X$  and  $Y$ , and on the right parts of  $X$  and  $Y$ . This process gives us the alignments for the rest of the fragments of  $X$  and  $Y$ . We can then merge these results to get the optimal alignment for  $X$  and  $Y$  with this “divide-and-conquer” approach. The exact details of Hirschberg’s OPTA algorithm are presented in Appendix A of this paper for the interested reader.

If we need to check at most  $(t - 1)$  locations to the left of a current table entry in order to compute the current table entry, then we get the best alignment for  $X$  and  $Y$  by calling  $\text{OPTA}(1, m, 1, n, t)$ . Note, however, that in saving the  $t$  most recently computed columns, we allow computations that check at most  $(t - 1)$  locations to the left.

If  $T(m, n)$  is the runtime of  $\text{OPTA}(1, m, 1, n, t)$  over  $X$  and  $Y$ , then we can express  $T(m, n)$  as follows:

$$\begin{aligned} T(1, 1) &= O(1), \\ T(1, n) &= O(n), \\ T(m, 1) &= O(m), \\ T(m, n) &\leq \max_{0 \leq q \leq n} \left[ T\left(\frac{m}{2}, n - q\right) + T\left(\frac{m}{2}, q\right) \right] + O(mn). \end{aligned}$$

The solution to the above recurrence equation yields,  $T(m, n) = O(mn)$ .

Hence, we have found an optimal alignment in the linear and piecewise-linear gap models in  $O(mn)$  time and  $O(n)$  space—quadratic time and linear space.

<sup>5</sup> Give or take 1.

Note that the parameter,  $t$ , does not affect the runtime, but in truth, we do use  $O(tn)$  space. But if  $t$  is constant, we achieve linear space complexity.

In the two-piece piecewise linear gap function, we will need to check  $(k - 1)$  entries to the left, so we must be sure this is not deleted from memory. Calling  $\text{OPTA}(1, m, 1, n, k)$  using the two-piece piecewise linear gap function will make sure this does not happen.

Next, assume  $k[0] = 0$ . For arbitrary piecewise linear gap functions, let  $d = \max_{1 \leq u \leq p} [k[u] - k[u-1]]$ . We will need to check at most  $d$  entries to the left, so we must be sure this is not deleted from memory. Thus, calling  $\text{OPTA}(1, m, 1, n, d+1)$  using the arbitrary piecewise linear gap function will make sure this does not happen. As before, assuming that  $d$  is bounded by a constant, (and so also  $p$ ), we achieve  $O(mn)$  runtime in  $O(n)$  space. Hence, we do get efficient piecewise-linear gap alignment in linear space, as desired.

However, there are two aspects of these algorithms that we must be careful about, while dealing with piecewise linear functions that do not affect linear functions.

Firstly, in piecewise linear functions, when two answers give the same result for  $G$ ,  $E_u$ , or  $F_u$ , we prefer that  $X$  be aligned with the longest possible gap, because this gives us a lower penalty-increase when we combine gaps that cross horizontally between the  $V$  and  $V^r$  tables, hence increasing the overall score and improving the alignment. (This is why the max function is implemented in such a manner that in the case of ties, the longest possible gap we can align with  $X$  is the result chosen as winner). We remark in passing that, this is why, for the general piecewise-linear gap model, it is best to set  $V(i, j)$  to

$$\max\{F_p(i, j), \dots, F_0(i, j), E_p(i, j), \dots, E_0(i, j), G(i, j)\},$$

so that in case of ties, the first mentioned max value gets priority.

Next, as it was discussed earlier that, assuming:

$$V(m, n) = \max_{0 \leq q \leq n} [V(\frac{m}{2}, q) + V^r(\frac{m}{2}, n - q)],$$

we do get the optimal alignment for  $X$  and  $Y$ .

With linear gap functions, the extra gap cost for extending a gap is always the same value, regardless of how big the gap is. Consequently, we actually achieve an optimal alignment in linear space for linear gap functions. In contrast, in piecewise-linear gap functions, the extra gap cost of extending a gap decreases for larger gaps. Therefore, in the OPTA algorithm, if we happen to have a gap that starts in the  $V$  portion, and ends in the  $V^r$  portion, (and this gap is of length  $a$  in the  $V$  portion, and of length  $b$  in the  $V^r$  portion), then the max function for finding  $q^*$  in OPTA should “know” that this is one gap of size  $a + b$ , not two gaps of sizes  $a$  and  $b$ .

Remember that  $w(a + b) \leq w(a) + w(b)$  when the rate of increase for  $w(\cdot)$  slows down as the gap length grows, so we might have  $w(a+b) < w(a)+w(b)$ , and we need to compensate for this. One way to improve the situation is to, for each  $j$  and  $j'$ , after computing  $V(h, j)$  and  $V^r(h, j')$  in OPTA, create compensation functions  $c$  and  $c'$ .

For each  $u$ , we record the gap length  $a$  for the choice of  $F_u(h, j)$  (and we can add gap length tracing information into all the entries of  $V$  without affecting the asymptotic runtime), then we set  $c(u, j) = a$ . Similarly, we can compute  $c'$  from  $V'$ .

Hence, after computing  $V(\frac{m}{2}, n)$  and  $V^r(\frac{m}{2}, n)$ , we can attempt to correct for a gap that stretches from  $V$  to  $V^r$  by instead finding a  $q^*$  with the computation below:

$$\begin{aligned} & \max_{0 \leq q \leq n} [V(\frac{m}{2}, q) + V^r(\frac{m}{2}, n - q), \\ & \max_{0 \leq u \leq p, 0 \leq v \leq p} [F_u(\frac{m}{2}, q) + F_v^r(\frac{m}{2}, n - q) \\ & \quad + w(c(u, q)) + w(c'(v, n - q)) - w(c(u, q) + c'(v, n - q))]]. \end{aligned}$$

Therefore, we use the max function shown above in order to select  $q^*$ . This helps to account for “bridges” between the left and right alignments. If  $p$  is a constant, this correction does not affect the asymptotic runtime.

However, the heuristic described above is not sufficient to “correct” for all piecewise-linear gap functions, as seen by the somewhat artificial counter-example presented in Appendix B of this paper. This means that it is possible to select the wrong value for  $q^*$ .

The main culprit for our problem is that OPTA sets the  $F_u$  values without assuming a bridge between  $V$  and  $V^r$ . Therefore, for general piecewise-linear gap models, the only way to correctly obtain  $q^*$  from  $V$  and  $V^r$  in linear space is to exhaustively inspect each entry of both tables at one moment of time or another. But, this improvement in space usage will be at a cost of overall cubic time complexity.

Thus, the heuristic method for general piecewise-linear gap models achieving linear space and quadratic time complexity is merely an approximation—nonetheless, an empirically excellent approximation. For instance, in several tests performed over mutated proteins in the quadratic time for 3 to 5 part piecewise-linear gap models in quadratic space versus the same model in quadratic time and linear space, the alignments obtained were identical in 90% of all the tests. For the 10% where they differed, the alignments turned out to be practically indistinguishable (differing by at most 5% of the characters for the proteins involved).

## 7 Conclusions

Several problems remain: next natural step is to introduce the BLOSUM-62 and PAM models, and others similar to it, in order to improve the quality of the alignments obtained. We also plan to investigate specific non-linear gap functions, such as logarithmic and fractional-powers, and determine whether they permit linear space and quadratic time dynamic programming solutions. Other problems of interest: better heuristics for general OPTA-like algorithms, aligning more than one proteins, heuristic approaches for whole genome alignments, multiple sequence alignments and a parallel implementation.

## 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] HIRSCHBERG, D. S., “A Linear Space Algorithm for Computing Maximal Common Subsequences.” *Comm. ACM*, **18**: 341–343, 1975.
  - [3] 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.
  - [4] 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.
  - [5] PEARSON, W.R., “Comparison of Methods for Searching Protein Sequence Databases.” *Protein Science*, **4**:1145–1160, 1995.
  - [6] 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.
  - [7] SMITH, T.F., AND WATERMAN, M.S., “Identification of Common Molecular Subsequences.” *Journal of Molecular Biology*, **147**: 195–197, 1981.
  - [8] 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.
  - [9] STATES, D.J., GISH, W., AND ALTSCHUL, S.F., “Basic Local Alignment Search Tool.” *Journal of Molecular Biology*, **215**: 403–410, 1990.
  - [10] 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.
  - [11] ZHOU, Y., AND MISHRA, B., “Models of Genome Evolution,” *Modeling in Molecular Biology*, Lecture Notes in Computer Science, Springer-Verlag2003. (In press).
-

## Appendix A: Hirschberg's OPTA Algorithm

Hirschberg's OPTA algorithm works as follows, assuming that we use at most  $t$  columns:

```

OPTA( $l, l', r, r', t$ )
begin
if ( $l > l'$ ) then
    align  $Y[r \dots r']$  against gaps and return result; (Base case)
else if ( $r > r'$ ) then
    align  $X[l \dots l']$  against gaps and return result; (Base case)
else if ( $1 + l' - l \leq t$ ) then
    compute  $V$  for entries  $X[l \dots l']$  and  $Y[r \dots r']$  and
        trace the result to get an alignment  $A$ . We do not throw
        away any columns doing this, so we can get the full alignment
        for  $X[l \dots l']$  and  $Y[r \dots r']$ . (This is another base case.)
    We return  $A$ ;
else begin
     $h = (l' - l)/2$ ;
    In  $O(r' - r) = O(n)$  space, compute  $V$  on entries  $X[l \dots h]$ 
        and  $Y[r \dots r']$  and compute  $V^r$  on entries
         $X[h + 1 \dots l']^r$  and  $Y[r \dots r']^r$ . Then, find an
        index  $q^*$  such that  $X[l \dots h]$  aligned
        with  $Y[r \dots q^*]$  and  $X[h + 1 \dots l']$  aligned with
         $Y[q^* + 1 \dots r']$  gives the best score. Trace the last  $t$ 
        columns in  $V$  and the last  $t$  columns in  $V^r$  (or first
        depending on how you see it) in order to find the alignments for
         $X[h - (t - 1) \dots h]$  with  $Y[q_1 \dots q^*]$  and
         $X[h + 1 \dots h + t]$  with  $Y[q^* + 1 \dots q_2]$ .
        (Note:  $q_1$  and  $q_2$  are determined from the positions in  $Y$ 
        we are when we can trace no further.)
    We'll call these combined alignments  $L_2$ ;
    Let  $L_1 = \text{OPTA}(l, h - t, r, q_1, t)$ ;
    Let  $L_3 = \text{OPTA}(h + t + 1, l', q_2, r', t)$ ;
    We glue  $L_1$  followed by  $L_2$  followed by  $L_3$  to make an
        alignment  $L$ . We return  $L$ ;
end {if}
end

```

## Appendix B: Counter-Example to the Accuracy of Piecewise-Linear Gap Formulas in Linear Space

As mentioned earlier in this paper, for the Piecewise-Linear Gap Model done in Linear Space, after computing  $V(\frac{m}{2}, n)$  and  $V^r(\frac{m}{2}, n)$ , we create compensation functions  $c$  and  $c'$  based on the gap lengths involved, and use it to attempt to correct for a gap that stretches from  $V$  to  $V^r$ . With this, we then select  $q^*$  from the best possible  $q$  value used in the computation below:

$$\begin{aligned} & \max_{0 \leq q \leq n} [V(\frac{m}{2}, q) + V^r(\frac{m}{2}, n - q), \\ & \max_{0 \leq u \leq p, 0 \leq v \leq p} [F_u(\frac{m}{2}, q) + F_v^r(\frac{m}{2}, n - q) \\ & \quad + w(c(u, q)) + w(c'(v, n - q)) - w(c(u, q) + c'(v, n - q))]]. \end{aligned}$$

The problem with this “corrected” model can be stated as follows:

Let  $w(i) = i$  if  $i < 3$ ,  $(i - 3) * 0.5 + 3$  if  $3 \leq i < 10$ , and  $6.5$  if  $i \geq 10$ , a 3-part piecewise linear function. Next, suppose, for some value  $q$ , that:

$$V(\frac{m}{2} - 4, q) = G(\frac{m}{2} - 4, q) = 5.75, \quad \text{and} \quad V(\frac{m}{2} - 7, q) = G(\frac{m}{2} - 7, q) = 7.$$

Assume, for all  $u$  and all  $i$  besides  $\frac{m}{2} - 4$  and  $\frac{m}{2} - 7$ ,

$$G(i, q) < 1, \quad \text{and} \quad E_u(i, q) < 1.$$

Then,

$$\begin{aligned} F_0(\frac{m}{2}, q) &= V(\frac{m}{2} - 4, q) - 4 \cdot w_{c[0]} = 5.75 - 4 = 1.75 \\ &> 0 = 7 - 7 = V(\frac{m}{2} - 7, q) - 7 \cdot w_{c[0]}, \end{aligned}$$

and

$$\begin{aligned} V(\frac{m}{2}, q) &= F_1(\frac{m}{2}, q) = V(\frac{m}{2} - 4) - (3 * w_{c[0]} + 1 * w_{c[1]}) = 5.75 - 3.5 = 2.25 \\ &> 2 = 7 - 5 = V(\frac{m}{2} - 7) - w(7). \end{aligned}$$

Furthermore, suppose that for all  $i$  and  $u$ , we happen to have  $V(i, q) = V^r(i, n - q)$ ,  $E_u(i, q) = E_u^r(i, n - q)$ ,  $F_u(i, q) = F_u^r(i, n - q)$ , and  $G(i, q) = G^r(i, n - q)$ .

Then, using a similar argument for  $V^r$ , we also have

$$V^r(\frac{m}{2}, n - q) = F_1^r(\frac{m}{2}, n - q) > V^r(\frac{m}{2} - 7) - w(7).$$

Therefore,  $c(0, q) = c'(0, n - q) = 4$ , and  $c(1, q) = c'(1, n - q) = 4$ . As a result, using our heuristic computation for  $q$ , we conclude that the best solution for  $q$  is

$$F_1(\frac{m}{2}, q) + F_1^r(\frac{m}{2}, n - q) + w(4) + w(4) - w(8) = 2.25 + 2.25 + 3.5 + 3.5 - 5.5 = 6.$$

However, in truth, the best solution for  $q$  should be

$$V(\frac{m}{2} - 7, q) + V^r(\frac{m}{2} - 7, n - q) - w(14) = 7 + 7 - 6.5 = 8.5.$$

This example contradicts our assumption that the heuristic chooses the correct value for  $q^*$ .