Algorithms for Local Alignments with Length Constraints

Abdullah N. Arslan and Ömer Eğecioğlu *
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106 USA
$\{arslan, omer\}@cs.ucsb.edu$

Abstract

The local sequence alignment problem is the detection of similar subsequences in two given sequences of lengths $n \geq m$. Unfortunately the common notion of local alignment suffers from some well-known anomalies which result from not taking into account the lengths of the aligned subsequences. We introduce the *length restricted local alignment problem* which includes as a constraint an upper limit T on the length of one of the subsequences to be aligned. However, obvious extensions of dynamic programming formulations for the solution of the length restricted local alignment problem result in expensive computations: $O(Tnm)$ time and $O(Tm)$ space. We propose an efficient approximation algorithm, which finds a solution satisfying the length bound, and whose score is within difference Δ of the optimum score for any given positive integer Δ . The algorithm runs in time $O(nmT/\Delta)$ using $O(mT/\Delta)$ space. We also introduce the *cyclic local alignment* problem and show how our idea can be applied to this case as well. This is a dual approach to the well-known cyclic edit distance problem. These results generalize directly to the case of affine gap penalties.
Keywords: Local sequence alignment, edit distance, cyclic edit distance, approximation algorithm.

1 Introduction

Local sequence alignment problem in computational molecular biology aims to reveal similar regions in a given pair of sequences X and Y whose lengths are n and m respectively with $n \geq m$. The Smith-Waterman algorithm finds the local alignment by searching for two segments with maximum similarity score by discarding poorly conserved initial and terminal fragments. Since it is not designed to exclude non-similar internal fragments, an alignment returned by the algorithm may contain a mosaic of well-conserved fragments artificially connected by poorly conserved or even unrelated fragments. As a result, a local alignment with score 1,000 and length 10,000 (*long alignment*) may be chosen over a local alignment with score 998 and length 1,000 (*short alignment*), although the latter one is probably more important biologically. This deficiency causes two forms of anomalies: "mosaic effect" and "shadow effect". Mosaic effect in an alignment is observed when a very poor region is sandwiched between two regions with high similarity scores. Shadow effect is

*Supported in part by NSF Grant No. CCR–9821038.

observed when a biologically important short alignment is not detected because it overlaps with a significantly longer yet biologically inadequate alignment with only a slightly higher score.

We consider *length restricted local alignment* ($LRLA$) problem in which we search for substrings I and J that maximize the score $s(I, J)$ among all substrings I and J with $|J| \leq T$ where T is a given upper limit on the length of J. This is yet another attempt to eliminate problems associated with local alignment. The objective is similar to that of normalized local alignment algorithms [5], in that we aim to circumvent the undesirable mosaic and the shadow effects. The degree of fragmentation in this new problem is controlled by varying the bound T.

Length restricted local alignment can be solved by extending the dynamic programming formulation of local alignment problem. However the resulting time complexity is $O(Tnm)$, which may not be practical for large values of n, m, and T. In this paper, we propose two approximation algorithms for $LRLA$. The first one is a simple $\frac{1}{2}$-approximation algorithm, and its complexity is the same as that of the local alignment problem (Algorithm *HALF*, section 4). The second algorithm (Algorithm APX-$LRLA$, section 4) returns a score guaranteed to be within difference Δ of the optimum for a given $\Delta \geq 1$. The time complexity of this algorithm is $O(nmT/\Delta)$, with $O(mT/\Delta)$ space. These two approximation algorithms can also be used to approximately solve the *cyclic local alignment* problem (CLA) of maximizing $s(I, J)$ where I is a substring of X, and J is a substring of a cyclic shift of Y. Both algorithms can easily be implemented, without increasing the complexity, for the case of affine gap penalties.

The outline of this paper is as follows. In section 2, we discuss related previous work in the literature on algorithms for restricted versions of local alignment, and cyclic edit distance. In section 3 we present the requisite notions and give the notation we use. Section 4 contains the description of the approximation algorithms *HALF* and APX-$LRLA$. Conclusions are presented in section 5.

2 Previous Work

The anomalies of mosaic effect and shadow effect that exist in the ordinary formulation of local alignment and the Smith-Waterman algorithm lead to problems in comparison of long genomic sequences and comparative gene prediction, as recently pointed out by Zhang et al., 1999 [18]. They proposed to decompose a discovered local alignment into sub-alignments that avoid the mosaic effect. However, the post-processing approach may miss the alignments with the best degree of similarity if the Smith-Waterman algorithm missed them in the first place. As a result, highly similar fragments may be ignored if they are not parts of longer alignments dominating other local similarities. Another approach to fixing the problems with the Smith-Waterman algorithm is based on the notion of X-*drop*, a region within an alignment that scores below X. The alignments that contain no X-drops are called X-*alignments*. Although X-alignments are expensive to compute in practice, Altschul et al., 1997 [4] and Zhang et al., 1998 [17] used some heuristics for searching databases with this approach.

It is well-known that the statistical significance of the local alignment depends on both its score and length (Altschul and Ericson, 1986 [2], 1988 [3]). Alexandrov and Solovyev, 1998 [1] proposed to normalize the alignment score by its length and demonstrated that this new approach leads to better protein classification. Arslan et al., 2001 [5] defined the *normalized local alignment problem* where the goal is to find substrings I and J that maximize $s(I,J)/(|I|+|J|)$ among all substrings I and J with $|I|+|J| \geq t$, where $s(I,J)$ is the score, and t is a threshold for the minimal overall length of I and J. Because of the cubic time complexity of the exact algorithm as an approximation to the original problem they proposed a solution to the maximization of $s(I,J)/(|I|+|J|+L)$ for a given parameter L. This can be done in time $O(nm \log n)$ and using $O(m)$ space [5].

The length restricted local alignment ($LRLA$) problem considered in this paper is essentially another attempt to eliminate problems associated with local alignment by an approximation algorithm that runs in reasonable time and which allows for a control over the length of the optimal local alignment sought. The limit is placed on only the substring J of Y. We believe that the underlying scoring scheme should limit the length of the other substring involved in an optimal alignment automatically, and therefore having two limits, one for $|I|$ and another for $|J|$ is redundant. An application of length restricted local alignment is the formulation of the *cyclic local alignment* problem as a special case of $LRLA$. The cyclic local alignment is the problem of maximizing $s(I,J)$ over all I and J, where I is a substring of X, and J is a substring of a *cyclic shift* of Y. A cyclic shift of a string s is a string obtained by concatenating symbols of s in a circular fashion. For example *ccdab*, *cdabc* are cyclic shifts of string *abccd*. The cyclic local alignment problem is the length restricted local alignment problem with strings X and YY, and limit $T = |Y|$. The approximation algorithms we develop can readily be used for this problem. Moreover it defines a dual approach to well-known *cyclic edit distance problem* which aims to find the minimum edit distance between string X and a cyclic shift of Y over all possible cyclic shifts of Y.

Cyclic edit distance problem appears in many applications, and there is extensive literature on the subject. Bunke and Bühler, 1993 [6] presented cyclic edit distance as a method for two-dimensional shape recognition. Uliel et al., 1999 [14] suggested using it for detecting circular permutations in proteins. There are many algorithms for the problem. Fu and Lu [8] presented an $O(nm^2)$-time algorithm extending the dynamic programming edit distance algorithm of Wagner and Fisher, 1974 [15]. Maes [12] proposed an algorithm with $O(nm \log m)$ time and $O(nm)$ space complexity, and described how to reduce the space complexity to $O((n+m)\log m)$. There are also $O(nm)$-time suboptimal algorithms developed by Gorman et al., 1988 [9], Bunke and Bühler, 1993 [6], and Uliel et al., 1999 [14]. Gregor and Thomason, 1996 [10] presented a data-dependent algorithm whose time complexity ranges from $O(nm)$ to $O(nm^2)$. There are faster algorithms for the case of unit-cost edit operations in which each edit operation has weight 1 except for a match (substitution of the same symbol, or no operation) whose weight is 0. For this case, Chung, 1998 [7] proposed an algorithm for a generalized version of the problem called *banded cyclic string-to-string correction problem* whose time complexity ranges between $O(nm)$ and $O(nm \log m)$ for cyclic edit distance computation. Landaue et al., 1998 [11] described an algorithm for *incremental string comparison* which can be used to solve cyclic edit distance problem with unit costs in time $O(m^2)$.

Since cyclic local alignment problem is a special case of length restricted local alignment, the approximation algorithms we describe in this paper can also be used to solve the cyclic local alignment problem with the same efficiency. This makes cyclic local alignment an alternative to cyclic edit distance in applications using cyclic string comparison. In addition, our algorithms can easily be implemented, without increasing the complexity, for the case of *affine gap penalties* in which total cost associated with each block (called a *gap*) of indels (insertions or deletions) is a linear function of a gap open penalty and the number of individual indels in the gap.

3 Preliminaries and Definitions

Given two strings $X = x_1x_2 \dots x_n$ and $Y = y_1y_2 \dots y_m$ with $n \geq m$, the *Alignment Graph* $G_{X,Y}$ (*Edit Graph* in the context of string editing) is used to represent all possible *alignments* between X and Y. It is a directed acyclic graph having $(n+1)(m+1)$ lattice points (u,v) as vertices for $0 \leq u \leq n$, and $0 \leq v \leq m$ (Figure 1). An *alignment path* for substrings $x_i \cdots x_k$, and $y_j \cdots y_l$ is a directed path from the vertex $(i-1, j-1)$ to (k,l) in $G_{X,Y}$ where $i \leq k$ and $j \leq l$. Horizontal and vertical arcs correspond to insert and delete operations respectively. The diagonal arcs correspond to substitutions which are either matching (if the corresponding symbols are the same), or mismatching (otherwise). If we trace the arcs of an alignment path for substrings I and J and perform the indicated edit operations in the given order on I, we obtain J.

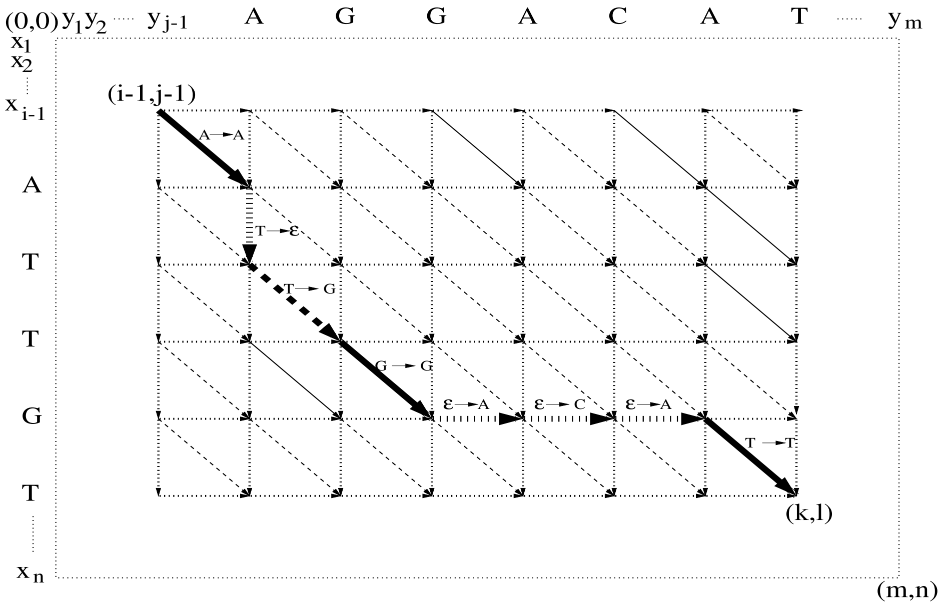

Figure 1: Alignment graph $G_{X,Y}$ where $x_i \cdots x_k = ATTGT$ and $y_j \cdots y_l = AGGACAT$. Matching diagonal arcs are drawn as solid lines while mismatching diagonal arcs are shown by dashed lines. Dotted lines are used for horizontal and vertical arcs. An example alignment path is shown. Labels of the arcs on this path are the corresponding edit operations where ϵ denotes the null string.

The objective of sequence alignment is to quantify the similarity between two strings. There are various scoring schemes for this purpose. In the *basic scoring scheme*, the arcs of $G_{X,Y}$ are assigned weights determined by non-negative reals δ (*mismatch penalty*) and μ (*indel* or *gap penalty*). We assume that $s(x_i, y_j)$ is the similarity score between the symbols x_i, and y_j which is normally 1 for a match ($x_i = y_j$) and $-\delta$ for a mismatch ($x_i \neq y_j$). We will use the terms alignment and alignment path interchangeably.

The following is the classical dynamic programming formulation ([16]) to compute the maximum local alignment score $\mathcal{S}_{i,j}$ ending at each vertex (i,j):

$$\mathcal{S}_{i,j} = \max\{0,\ \mathcal{S}_{i-1,j} - \mu,\ \mathcal{S}_{i-1,j-1} + s(x_i, y_j),\ \mathcal{S}_{i,j-1} - \mu\} \tag{1}$$

for $1 \le i \le n$, $1 \le j \le m$, with the boundary conditions $\mathcal{S}_{i,j} = 0$ whenever $i = 0$ or $j = 0$.

Let $\subseteq$ indicate the substring relation. The *local alignment* (LA) problem seeks for substrings $I \subseteq X$ and $J \subseteq Y$ with the highest similarity score. The optimal value $LA^*(X,Y)$ for this problem is given by

$$LA^*(X,Y) = \max\{s(I,J) \mid I \subseteq X, J \subseteq Y\} = \max_{i,j} \mathcal{S}_{i,j} \tag{2}$$

where $s(I,J)$ is the best alignment score between I and J. LA^* can be computed using the Smith-Waterman algorithm [13] in time $O(nm)$. The space complexity is $O(m)$ because only $O(m)$ entries of the dynamic programming matrix need to be stored. In what follows, for any optimization problem $\mathcal{P}$, we denote by $\mathcal{P}^*$ its optimum value, and sometimes drop the parameters from the notation when they are obvious from the context.

As in [5] the objective of the *normalized local alignment* problem ($NLAt$) is:

$$NLAt^*(X,Y) = \max\{s(I,J)/(|I| + |J|) \mid I \subseteq X, J \subseteq Y, |I| + |J| \ge t\} \tag{3}$$

Figure 2 includes examples where optimal alignments for LA and $NLAt$ may be different. In each case, the long alignment has the highest ordinary score whereas the shorter alignments have higher *normalized scores*. The normalized score of an alignment is obtained by dividing its score by its length. The length of an alignment is defined as the sum of the lengths of the substrings involved in the alignment. If we use ordinary scores as the similarity measure then the long alignments in Figure 2 are optimal. If we use normalized scores then the desired alignments depend on the value of t . In this case $t = 200$ is a separating value in determining the optimality of short and long alignments. The need to have control over the alignment lengths becomes apparent when we use normalized scores. Without controlling the desired alignment lengths, with normalized scores short alignments overshadow the true long alignments causing yet another anomaly. However for normalized local alignments an effective and efficient such control is yet to be described. Arslan et al., 2001 [5] changed the objective to $s(I,J)/(|I| + |J| + L)$ by introducing parameter L aiming that it provides a control over the alignment length and the length constraint can be dropped. This gave rise to an efficient algorithm which runs in time $O(nm \log n)$ and using $O(m)$ space. However an adequate control over the length through parameter L is difficult to describe as the behavior of normalized scores as a function of L is not well understood.

Given a limit T, we define the *length restricted local alignment* ($LRLA$) score between X and Y as

$$LRLA^*(X,Y,T) = \max\{s(I,J) \mid I \subseteq X, J \subseteq Y, \text{ and } |J| \le T\} \tag{4}$$

Figure 3 illustrates the length constraint schematically. In $LRLA$ problem, the horizontal lengths of the resulting alignments are controlled by upper limit T on one of the substrings which in practice will be determined by biological considerations. For the alignments in Figure 2, setting $T = 100$ or $T = 300$ changes the optimality of the short and long alignments when the ordinary scores are used.

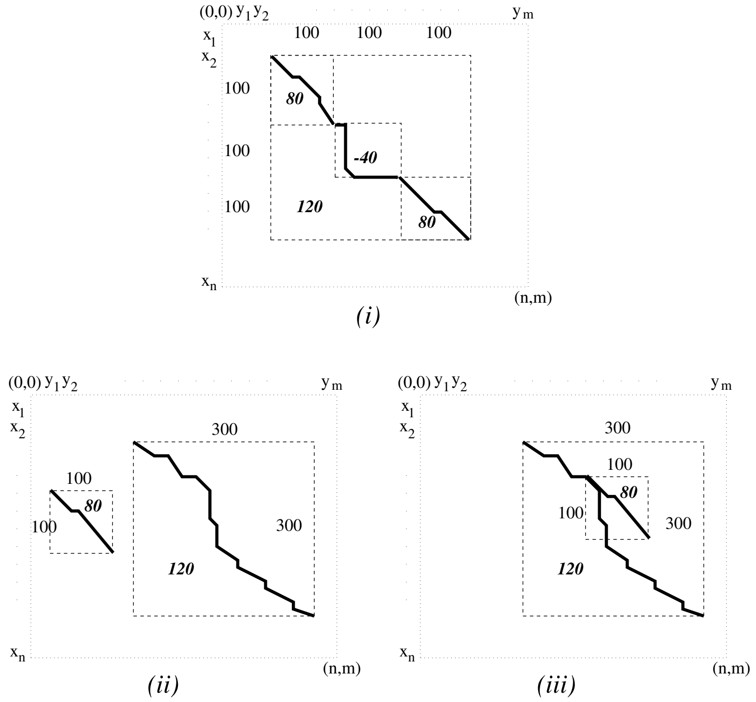

Figure 2: Some example alignments. The numbers written in italic are the ordinary scores of the alignments. The normalized score of the shorter alignment(s) is $80/200 = 0.4$ while that of the longer alignment is $120/600 = 0.2$.

To solve *LRLA* problem, we can extend the dynamic programming formulation in (1) by adding another dimension to store at each entry of the dynamic programming matrix optimum scores for all possible horizontal lengths up to T. This increases the time complexity to $O(Tnm)$.

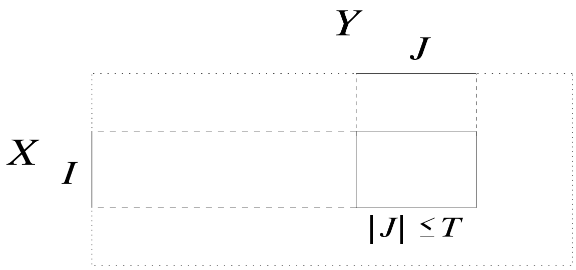

Figure 3: Candidates for I and J in the computation of $LRLA^*(X, Y, T)$.

The *cyclic edit distance* (*CED*) between X and Y is the minimum edit distance between X and any cyclic shift of Y:

$$CED^*(X, Y) = \min\{ed(X, \sigma^k(Y)) \mid 0 \le k < m\} \tag{5}$$

where ed denotes the edit distance, and $\sigma^k(Y)$ is the cyclic shift of Y by k which is defined as follows: $\sigma^0(Y) = Y$, and for $0 < k < m$, $\sigma^k(Y) = y_{k+1} \cdots y_m y_1 \cdots y_k$.

Maes' algorithm [12] computes $CED^*(X, Y)$ in $O(nm \log m)$ steps. For any k, $ed(X, \sigma^k(Y))$ is the cost of the shortest (least-cost) path $P(k)$ between the vertices $(0, k)$ and $(n, m+k)$ in $G_{X,YY}$ as shown in Figure 4. Maes' idea is to make use of the "non-crossing" property of shortest paths, which restricts the candidate $P(k)$ to be squeezed between $P(i)$ and $P(j)$ where $i < k < j$ as illustrated in the figure. However this idea cannot be generalized to the case of *affine gap penalties* in which the total cost of a gap of size a, i.e. a block of a insertions (or deletions), is $\alpha + (a-1)\mu$ where α is the gap open penalty. It can easily be seen that $P(i)$ and $P(j)$ may be crossing each other in this case.

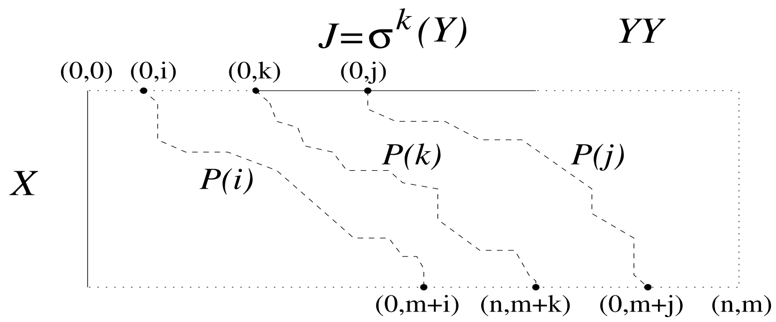

Figure 4: Finding $P(k)$ between $P(i)$ and $P(j)$ as a step towards the computation of $CED^*(X,Y)$.

As a dual approach to *CED* we define *cyclic local alignment* (*CLA*) problem by expressing its objective in the form

$$CLA^*(X,Y) = \max\{s(I,J) \mid I \subseteq X,\ J \subseteq \sigma^k(Y) \text{ for some } k,\ 0 \le k < m\} \tag{6}$$

Note that *CLA* is a special case of *LRLA*. More specifically

$$CLA^*(X,Y) = LRLA^*(X, YY, |Y|)$$

4 Approximation Algorithms for *LRLA*

We first give a simple $\frac{1}{2}$-approximation algorithm *HALF* for the *LRLA* problem. Clearly, we can assume $m > T$, for otherwise we can run the local alignment algorithm without alteration on $G_{X,Y}$ and obtain the exact solution to *LRLA* in time $O(nm)$. Let $u = \lceil m/T \rceil$ and put

$$Y_j = y_{(j-1)T+1} \cdots y_{jT}$$

for $1 \le j < u$ with $Y_u = y_{(u-1)T+1} \cdots y_m$. Thus $Y = Y_1 Y_2 \cdots Y_u$, and the Y_j partition Y into blocks of length T each (except possibly for Y_u, which may be shorter). Let

$$HALF^* = \max_{1 \le j < u} \{s(I,J) \mid I \subseteq X,\ J \subseteq Y_j Y_{j+1}\} \tag{7}$$

Finding an optimal alignment a for *HALF* requires solving $u-1$ ordinary local alignment problems with strings X and $Y_j Y_{j+1}$ for $1 \le j < u$ as schematically described in Figure 5 . Total time taken for this is $O(nm)$, since each Y_j needs to be considered at most twice during the computations. The space complexity of *HALF* is $O(m)$.

Let a be an optimal alignment found by *HALF*. We obtain an approximation to $LRLA^*$ from a as follows. Suppose $a \subseteq Y_j Y_{j+1}$ for some j. Let a_1, a_2 denote the two halves of a that lie in

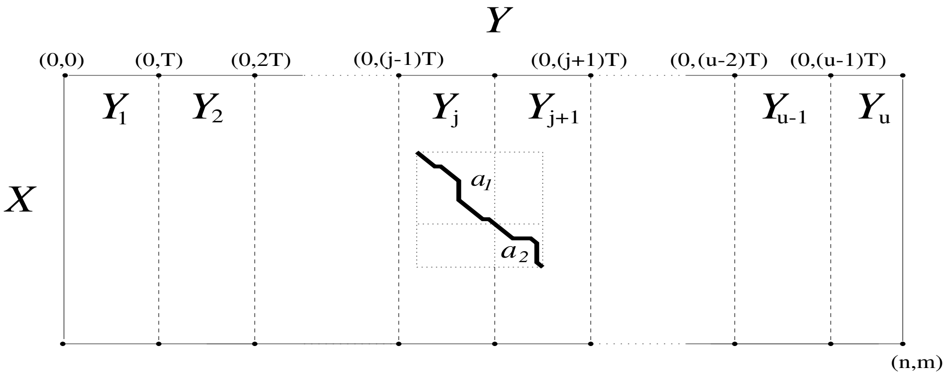

Figure 5: Regions of alignment graph explored in computing $HALF^*$, and parts a_1 and a_2 of an optimal alignment a.

Y_j and Y_{j+1}, respectively as shown in Figure 5. Without loss of generality, assume that a_1 is the one with the higher score, say $\widehat{S}$. Then $2\widehat{S} \geq LRLA^*$, and therefore $\frac{1}{2}LRLA^* \leq \widehat{S} \leq LRLA^*$. Clearly the horizontal length of a_1 is $\leq T$. The same approximation and complexity results hold for the cases of arbitrary scoring matrices and affine gap penalties. To solve the individual local alignment problems, Algorithm HALF uses the existing algorithms in the underlying scoring scheme.

We give next an approximation algorithm $APX\text{-}LRLA$, which computes a local alignment score $\widehat{S} \leq LRLA^*$ within a prescribed difference Δ from $LRLA^*$, i.e.

$$LRLA^* - \Delta \leq \widehat{S} \leq LRLA^*.$$

If desired, the position of an alignment achieving $\widehat{S}$, and consequently the substrings $I \subseteq X$ and $J \subseteq Y$ with $|J| \leq T$ achieving this score can be identified. The complexity of the algorithm is $O(nmT/\Delta)$ time and $O(mT/\Delta)$ space.

For simplicity, we assume a basic scoring scheme: i.e. score between the symbols x_i, and y_j is 1 for a match ($x_i = y_j$) and $-\delta$ for a mismatch ($x_i \neq y_j$), and the indel score is $-\mu$. We argue later that the algorithm can be easily modified within the same complexity results for the cases of arbitrary scoring matrices, and also affine gap penalties.

Our approximation idea is that instead of a single score, we maintain at each node (i,j) of $G_{X,Y}$, a list of scores with the property that at least one element of the list lies within Δ of the score of any optimal alignment ending at (i,j) and starting within a past horizontal window of size T of (i,j). We show that the dynamic programming formulation can be extended to preserve this property through the nodes. In particular, an alignment with maximum score $\widehat{S}$ in the list of scores computed in one of the nodes (i,j) will be between $LRLA^* - \Delta$ and $LRLA^*$. We assume that Δ is integral, otherwise we use the the largest integer smaller than the given value for Δ.

Similar to the case of $HALF$, we imagine the columns of the graph $G_{X,Y}$ as grouped into vertical slabs of $\Delta + 1$ columns each, starting with the leftmost column (i.e. $j = 0$). Two consecutive slabs share a column which we call a *boundary*. The *left* and the *right boundaries* of the slabs are defined as the leftmost and rightmost column positions in the slab. We agree that a slab does not contain the vertical edges among the vertices on the left boundary. Now to a given column j in $G_{X,Y}$, we associate a number of slabs as follows. Let *slab 0* with respect to j is the slab that contains column j. We order the consecutive slabs to the left of *slab 0* with respect to j as *slab 1, slab 2, ...*. This orders the slabs weakly to the left of column j, with respect to j. In other words, *slab k* with respect to column j is the subgraph of $G_{X,Y}$ composed of vertices placed inclusively between columns $\lfloor j/\Delta \rfloor$ and j if $k = 0$, and between columns $(\lfloor j/\Delta \rfloor - k)\Delta$ and $(\lfloor j/\Delta \rfloor - k + 1)\Delta$, otherwise. Figure 6 includes sample slabs with respect to column j, and alignments ending at some node (i, j) . From the dynamic programming formulation (1), we note the following observation for optimal alignments with the basic scoring scheme assumed: Any optimal alignment (with positive score) ending at a given node (i, j) has to start with a match since only the matches have positive scores.

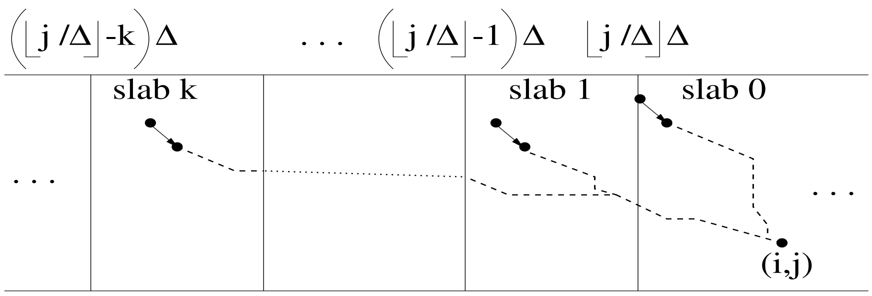

Figure 6: Slabs with respect to column j, and alignments ending at node (i, j) starting at different slabs.

Let $\mathcal{S}_{i,j,k}$ represent the optimum score achievable by any alignment ending at node (i, j) and starting at slab k with respect to column j. For *LRLA*, we are only interested in alignments with horizontal length not exceeding T. A single slab can contribute at most Δ to the score of any alignment. Consequently, we need to store at each node at most $\lfloor T/\Delta \rfloor$ scores $\mathcal{S}_{i,j,k}$, for $0 \leq k \leq \lfloor T/\Delta \rfloor - 1$ corresponding to the $\lfloor T/\Delta \rfloor$ slabs that include (i, j) and span a past horizontal window of length T. Figure 7 shows the steps of our approximation algorithm *APX-LRLA*. The processing is done row-by-row starting with the top row $(i = 0)$ of $G_{X,Y}$.

We can modify the Smith-Waterman algorithm such that it breaks the ties in scores by selecting alignments with smaller horizontal lengths. This modified algorithm can be used in Step 1 to check if the maximum score over all the alignments is achieved by an alignment whose horizontal length does not exceed T . Step 2 of the algorithm performs the initialization of the lists of the nodes in the top row $(i = 0)$. Step 3 implements computation of scores as dictated by the dynamic programming formulation in (1):

- If the current node (i, j) is not on the first column after a boundary then nodes $(i - 1, j)$, $(i-1, j-1)$ and $(i, j-1)$ share the same slabs with node (i, j), In this case $\mathcal{S}_{i,j,k}$ is calculated by using $\mathcal{S}_{i-1,j,k}$, $\mathcal{S}_{i-1,j-1,k}$, and $\mathcal{S}_{i,j-1,k}$ as

$$\mathcal{L}_{i,j,k} = \max\{0, \mathcal{L}_{i-1,j,k} - \mu, \mathcal{L}_{i-1,j-1,k} + s(x_i, y_j), \mathcal{L}_{i,j-1,k} - \mu\}.$$

- If the current node is on the first column following a boundary (i.e. $j \bmod \Delta = 1$) then the slabs for the nodes involved in the computations for node (i, j) differ as shown in Figure 8. In this case slab $k-1$ for the nodes at column $j-1$ is slab k for node (i, j). Moreover any alignment ending at (i, j) starting at slab 0 for (i, j) can either include only one of the edges $((i-1, j),(i, j))$, $((i-1, j-1),(i, j))$, or $((i, j-1),(i, j))$, or extend an alignment from node $(i-1, j)$. The edges $((i-1, j),(i, j))$ and $((i, j-1),(i, j))$ both have negative weight $-\mu$. Therefore, $\mathcal{S}_{i,j,0}$ is set to $\max\{0, s(x_i, y_j), \mathcal{S}_{i-1,j,0} - \mu\}$. For slab $k > 0$, $\mathcal{S}_{i,j,k}$ is calculated by

$$\mathcal{L}_{i,j,k} = \max\{0, \mathcal{L}_{i-1,j,k} - \mu, \mathcal{L}_{i-1,j-1,k-1} + s(x_i, y_j), \mathcal{L}_{i,j-1,k-1} - \mu\}$$

During these computations, the running maximum score is also updated whenever a newly computed score is larger than the current maximum, and the final value is returned in Step 3. The alignment position achieving this score may also be desired. This can be done by maintaining a start and end position information for an optimal alignment in addition to its score. In this case in addition to the running maximum score, the start and end positions of a maximal alignment should be stored and updated.

If there is an alignment with the maximum score and with horizontal length not exceeding T then the algorithm returns this score in Step 1. Otherwise, we first show that for any node (i, j) and slab k, $\mathcal{S}_{i,j,k}$ calculated by the algorithm is the optimum score achievable over the set of all the alignments ending at node (i, j) and starting at slab k with respect to column j . This claim is proved by induction. If we assume that the claim is true for nodes $(i-1, j)$, $(i-1, j-1)$ and $(i, j-1)$, and for their slabs, then we can easily see by following Step 3 of the algorithm that the claim holds for node (i, j) and its slabs. Consider the alignments with horizontal length at most T. If there is an optimal alignment with score $LRLA^*$ and with length at most $T - \Delta$, then this alignment is captured during the calculations at its right end point. In this case, by the previous claim, the algorithm returns the optimum score $LRLA^*$.

If all optimal alignments with score $LRLA^*$ have horizontal length $> T - \Delta$, we show that during the computations the algorithm observes a score which does not differ more than Δ from the optimum score $LRLA^*$. To see this, let an optimal alignment start at node (i', j') and end at node (i, j). We know that its horizontal length is larger than $T - \Delta$. Let (i'', j'') be the node the alignment crosses at the boundary $(\lfloor T/\Delta \rfloor - 1)\Delta$. This is the right boundary of the slab (i', j') lies in, or equivalently, the left boundary of slab $k\lfloor T/\Delta \rfloor - 1$ relative to column j as shown in Figure 9. Note that the score of the part of the alignment between nodes (i', j') and (i'', j'') is at most Δ according to our basic scoring scheme. Since $(i', j)'$ is one of the nodes in slab k relative to j $\mathcal{S}_{i,j,k}$ does not differ from the score of the given optimal alignment by more than Δ. Thus

$$LRLA^* - \Delta \leq \mathcal{S}_{i,j,k}$$

Note that it is likely that a score higher than the guaranteed lower bound $LRLA^* - \Delta$ is returned frequently in practice as the algorithm explores all the nodes as possible end points.

Algorithm *APX-LRLA* essentially implements the dynamic programming formulation (1). It is similar to the Smith-Waterman algorithm except at each node instead of a single score, $\lfloor T/\Delta \rfloor$ scores are stored and manipulated. Therefore the resulting complexity exceeds that of the Smith-Waterman

```
Algorithm APX-LRLA(δ,μ)

1. Run a modified Smith-Waterman algorithm.  If the maximum score is
   achieved within horizontal length ≤ T then return this score and exit
2. Initialization:
   set LRLA* = 0
   set L_{0,j,k} = 0 for all j,k,  0 ≤ j ≤ m, and 0 ≤ k ≤ ⌊T/Δ⌋ − 1
3. Main computations :
   for i = 1 to n do
   {
    set L_{i,0,k} = 0 for all k, 0 ≤ k ≤ ⌊T/Δ⌋ − 1
    for j = 1 to m do
    {
      if (j mod Δ = 1) then
       {
        set L_{i,j,0} = max{0, s(x_i,y_j), L_{i−1,j,0} − μ}, and update LRLA*
        set L_{i,j,k} = max{0, L_{i−1,j,k} − μ, L_{i−1,j−1,k−1} + s(x_i,y_j), L_{i,j−1,k−1} − μ}
          for all k,  1 ≤ k ≤ ⌊T/Δ⌋ − 1 and update LRLA*
       } else
         {
           set L_{i,j,k} = max{0, L_{i−1,j,k} − μ, L_{i−1,j−1,k} + s(x_i,y_j), L_{i,j−1,k} − μ}
             for all k,  0 ≤ k ≤ ⌊T/Δ⌋ − 1 and update LRLA*
         }
    }
   }
3. Return LRLA*
```

Figure 7: Algorithm APX-$LRLA$ which approximates $LRLA^*$ within difference Δ.

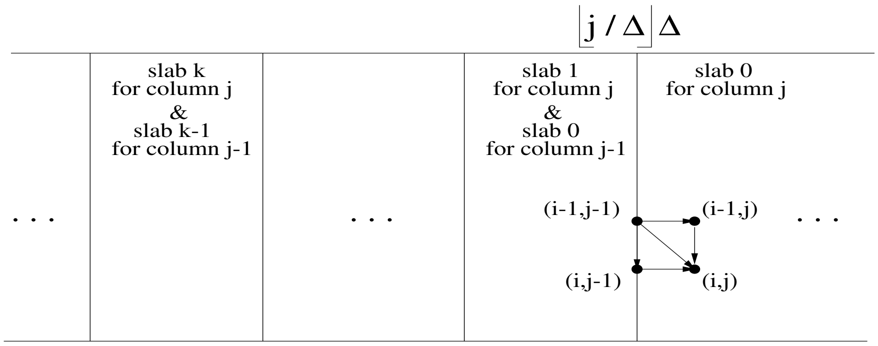

Figure 8: Left boundary of slab 0 with respect to column j, and numbering of slabs with respect to columns j and $j-1$.

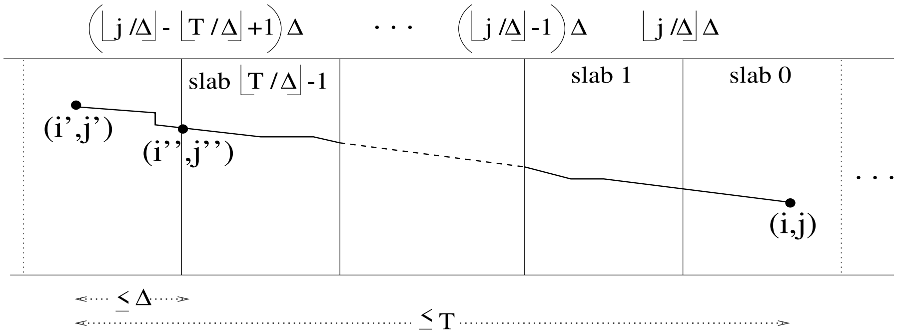

Figure 9: Orientation of an optimal alignment ending at node (i, j) when its horizontal length is larger than $T - \Delta$.

algorithm by a factor of $\lfloor T/\Delta \rfloor$. That is, the time complexity of *APX-LRLA* is $O(nmT/\Delta)$. The algorithm requires $O(mT/\Delta)$ space since we need the entries in the previous and the current row to calculate the entries in the current row.

Algorithm *APX-LRLA* can easily be generalized to other common scoring schemes with simple modifications. For example, varying penalties (or scores) can easily be incorporated for the computation of optimum scores at each node for arbitrary scoring matrices. Affine gap penalties require a slightly different dynamic programming formulation than the one given for basic scoring scheme (1). It can be described as follows ([16]) : Let $\mathcal{E}_{i,j} = \mathcal{F}_{i,j} = \mathcal{S}_{i,j} = 0$ when i or j is 0, and if α is the gap open penalty, then define

$$\begin{aligned} \mathcal{E}_{i,j} &= \max\{\mathcal{S}_{i,j-1} - \alpha,\ \mathcal{E}_{i,j-1} - \mu\}, \\ \mathcal{F}_{i,j} &= \max\{\mathcal{S}_{i-1,j} - \alpha,\ \mathcal{F}_{i-1,j} - \mu\}, \\ \mathcal{S}_{i,j} &= \max\{0,\ \mathcal{S}_{i-1,j-1} + s(x_i, y_j),\ \mathcal{E}_{i,j},\ \mathcal{F}_{i,j}\} \end{aligned} \tag{8}$$

Affine gap penalties do not increase the complexity of the local alignment problem, i.e. the problem can be solved in time $O(nm)$ and using $O(m)$ space. Figure 10 shows algorithm *APX-LRLA-AFFINE* which is a variation of our algorithm *APX-LRLA* for affine gap penalties. It uses the same idea that at each entry of the dynamic programming matrix, instead of a single score a number of scores are maintained and manipulated as dictated by the above dynamic programming formulation, and by using the technique employed in algorithm *APX-LRLA*.

5 Conclusion

We considered the length restricted local alignment problem *LRLA* to circumvent the undesirable mosaic and the shadow effects of the ordinary local alignment algorithm. In length restricted local alignment we search for substrings I and J that maximize the score $s(I, J)$ among all $I \subseteq X$ and $J \subseteq Y$ with $|J| \leq T$. *LRLA* can be solved by extending the dynamic programming formulation of local alignment problem. However the resulting time complexity is $O(Tnm)$, which may not be practical for large values of n, m, and T. In this paper, we proposed two approximation algorithms

```
Algorithm APX-LRLA-AFFINE(δ, α, μ)

1. Run a modified Smith-Waterman algorithm.  If the maximum score is
   achieved within horizontal length ≤ T then return this score and exit
2. Initialization:
   set LRLA* = 0
   set E_{0,j,k} = F_{0,j,k} = S_{0,j,k} = 0 for all j,k,  0 ≤ j ≤ m, and 0 ≤ k ≤ ⌊T/Δ⌋ − 1
3. Main computations :
   for i = 1 to n do
   {
    E_{i,0,k} = F_{i,0,k} = S_{i,0,k} = 0 for all k, 0 ≤ k ≤ ⌊T/Δ⌋ − 1
    for j = 1 to m do
   {
     if (j mod Δ = 1) then
      {
       set E_{i,j,0} = 0
       set E_{i,j,k} = max{S_{i,j−1,k−1} − α, E_{i,j−1,k−1} − μ} for all k, 1 ≤ k ≤ ⌊T/Δ⌋ − 1
       set F_{i,j,k} = max{S_{i−1,j,k} − α, F_{i−1,j,k} − μ} for all k, 0 ≤ k ≤ ⌊T/Δ⌋ − 1
       set S_{i,j,0} = max{0, s(x_i, y_j), S_{i−1,j,0} − α, F_{i−1,j,0} − μ} and update LRLA*
       set S_{i,j,k} = max{0, S_{i−1,j−1,k−1} + s(x_i, y_j), E_{i,j,k−1}, F_{i,j,k}}
        for all k, 1 ≤ k ≤ ⌊T/Δ⌋ − 1 and update LRLA*
      } else
        {
         set E_{i,j,k} = max{S_{i,j−1,k} − α, E_{i,j−1,k} − μ} for all k, 0 ≤ k ≤ ⌊T/Δ⌋ − 1
         set F_{i,j,k} = max{S_{i−1,j,k} − α, F_{i−1,j,k} − μ} for all k, 0 ≤ k ≤ ⌊T/Δ⌋ − 1
         set S_{i,j,k} = max{0, S_{i−1,j−1,k} + s(x_i, y_j), E_{i,j,k}, F_{i,j,k}}
          for all k, 0 ≤ k ≤ ⌊T/Δ⌋ − 1 and update LRLA*
        }
    }
   }
3. Return LRLA*
```

Figure 10: Algorithm $APX\text{-}LRLA\text{-}AFFINE$: Algorithm for affine gap penalties which approximates $LRLA^*$ within difference Δ.

for $LRLA$. The first algorithm is a simple $\frac{1}{2}$-approximation algorithm which has the same time and space complexity as the ordinary local alignment algorithm. The second algorithm $APX\text{-}LRLA$ obtains a score within difference Δ from the optimum for any given positive integer Δ, and runs in time $O(nmT/\Delta)$.

A special case of the length restricted local alignment problem is cyclic local alignment. This gives rise to a dual approach to the cyclic edit distance problem. Within the same complexity bounds of our algorithms and the same approximation guarantee, we can compute

$$CLA^*(X,Y) = \max_{0 \le k < m} \{s(I,J) \mid I \subseteq X,\ J \subseteq y_{k+1} \dots y_m y_1 \dots y_k\ \}.$$

The algorithms can be generalized to the more common scoring scheme of affine gap penalties which is widely used in practice. We present an algorithm $APX\text{-}LRLA\text{-}AFFINE$ which generalizes $LRLA$ to the affine case with the same performance guarantee as $LRLA$.

The algorithms are simple to implement and provably efficient. The degree of approximation can be controlled with a reasonable trade-off of optimality versus time and space. There is also reason to believe that the approximate score returned is on the average much closer to the actual optimum than the worst case error bound of Δ.

References

[1] N. N. Alexandrov and V. V. Solovyev. Statistical significance of ungapped alignments. *Pacific Symposium on Biocomputing (PSB-98), (eds. R. Altman, A. Dunker, L. Hunter, T. Klein)*, pages 463–472, 1998.

[2] S. F. Altschul and B. W. Erickson. Locally optimal subalignments using nonlinear similarity functions. *Bulletin of Mathematical Biology*, 48:633–660, 1986.

[3] S. F. Altschul and B. W. Erickson. Significance levels for biological sequence comparison using nonlinear similarity functions. *Bulletin of Mathematical Biology*, 50:77–92, 1988.

[4] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped Blast and Psi-Blast: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[5] A. N. Arslan, Ö. Eğecioğlu, and P. A. Pevzner. A new approach to sequence comparison: Normalized local alignment. *Bioinformatics*, 17(4):327–337, 2001.

[6] H. Bunke and U. Bühler. Applications of approximate string matching to 2d shape recognition. *Pattern Recognition*, 26(12):1797–1812, 1993.

[7] K-L. Chung. An improved algorithm for solving the banded cyclic string-to-string correction problem. *Theoretical Computer Science*, 201:275–279, 1998.

[8] K. S. Fu and S. Y. Lu. Size normalization and pattern orientation problems in syntactics clustering. *IEEE Trans. Systems Man Cybernet*, 9:55–58, 1979.

[9] J. W. Gorman, O. R. Mitchell, and F. P. Kuhl. Partial shape recognition using dynamic programming. *IEEE Trans. Pattern Anal. Mech. Intell. PAMI-10*, pages 257–266, 1988.

[10] J. Gregor and M. G. Thomason. Efficient dynamic programming alignment of cyclic shifts by shift elimination. *Pattern Recognition*, 29(7):1179–1185, 1996.

[11] G. M. Landau, E. W. Myers, and J. P. Schmidt. Incremental string comparison. *Siam J. Comput.*, 27(2):557–582, 1998.

[12] M. Maes. On a cyclic string-to-string correction problem. *Information Processing Letters*, 35:73–78, 1990.

[13] T. F. Smith and M. S. Waterman. The identification of common molecular subsequences, *J. of Molecular Biology*, 147:195–197, 1981.

[14] S. Uliel, A. Fliess, A. Amir, and R. Unger. A simple algorithm for detecting circular permutations in proteins. *Bioinformatics*, 15(11):930–936, 1999.

[15] R. A. Wagner and M. J. Fisher. The string-to-string correction problem. *J. Assoc. Comput. Mach.*, 21(1):168–173, 1974.

[16] M. S. Waterman. *Introduction to computational biology.* Chapman & Hall, 1995.

[17] Z. Zhang, P. Berman, and W. Miller. Alignments without low-scoring regions. *J. Comput. Biol.*, 5:197–200, 1998.

[18] Z. Zhang, P. Berman, T. Wiehe, and W. Miller. Post-processing long pairwise alignments. *Bioinformatics*, 15:1012–1019, 1999.