Dynamic-Programming-Based Approximation Algorithms for Sequence Alignment with Constraints

Abdullah N. Arslan
Department of Computer Science, University of Vermont, Burlington, Vermont 05405, USA, aarslan@cs.uvm.edu

Ömer Eğecioğlu
Department of Computer Science, University of California, Santa Barbara, Santa Barbara, California 93106, USA, omer@cs.ucsb.edu

Given two sequences X and Y, the classical dynamic-programming solution to the local alignment problem searches for two subsequences $I \subseteq X$ and $J \subseteq Y$ with maximum similarity score under a given scoring scheme. In several applications, variants of this problem arise with different objectives and with length constraints on the subsequences I and J. This constraint can be explicit, such as requiring $|I| + |J| \geq t$, or $|J| \leq T$, or may be implicit such as in cyclic sequence comparison, or as in the maximization of *length-normalized* scores, and driven by practical considerations. We present a survey of approximation algorithms for various alignment problems with constraints, and several new approximation algorithms. These approximations are in two distinct senses: in one the constraints are satisfied but the score computed is within a prescribed tolerance of the optimum instead of the exact optimum. In another, the alignment returned is assured to have at least the optimum score with respect to the given constraints, but the length constraints are satisfied to within a prescribed tolerance from the required values. The algorithms proposed involve applications of techniques from fractional programming and dynamic programming.

Key words: local alignment; cyclic sequence comparison; normalized local alignment; length-restricted local alignment; approximation algorithm; dynamic programming; ratio maximization; fractional programming
History: Accepted by Harvey J. Greenberg, Guest Editor; received August 2003; revised February 2004; accepted February 2004.

1. Introduction

Detecting local similarities in two given strings has become an increasingly important computational problem, particularly due to its applications in biological sequence analysis.

Table 1: Local Alignment Problems LA (Waterman 1995), (Section 2), and $ANLA$ (Arslan et al. 2001) (Section 4)

Alignment Problem	Objective	Algorithm	Time	Space	Score Returned
LA	maximize $s(I, J)$	Smith-Waterman	$O(nm)$	$O(m)$	LA^*
$ANLA$	maximize $\frac{s(I,J)}{\|I\|+\|J\|+L}$ for parameter $L \geq 0$	Dinkelbach	$O(nm)$ (experimental)	$O(m)$	$ANLA^*$
		Rational$ANLA$	$O(nm \log n)$	$O(m)$	$ANLA^*$

The objective of locating similar fragments in a given pair of strings can be formulated in several ways. The formulations lead to new optimization problems, some of which invole a length constraint on the fragments. In most cases, there are simple dynamic-programming formulations for the exact version of a given alignment problem with length constraints. However, the resulting algorithms require cubic time which is unacceptably high for practical purposes since the sequence lengths can be on the order of millions. Approaches based on classical algorithms (e.g. Karp's minimum mean-weight cycle algorithm) on general graphs suffer from the same anomalies because they do not readily specialize to highly structured but large graphs used for sequence analysis, and they do not yield algorithms more efficient than naive dynamic-programming algorithms. To cope with high complexity, approximations are considered in both definitions of similarity, and resulting computations.

In this paper we survey constrained alignment problems as summarized in Tables 1 and 2, and present new approximation algorithms for these problems, for which we summarize the results in Table 3.

Given two strings X and Y the *local alignment* (LA) problem seeks substrings $I \subseteq X$, and $J \subseteq Y$ with the highest similarity score $s(I, J)$, where $\subseteq$ indicates the substring relation. We assume that the length of the sequences are $n = |X|$, $m = |Y|$, and $n \geq m$. 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 context. An optimization problem $\mathcal{P}$ is called *feasible* if it has a solution with the given parameters.

A classical algorithm for LA is the well-known Smith-Waterman algorithm that uses dynamic programming. The algorithm essentially discards poorly conserved initial and terminal fragments. Since it is not designed to exclude non-similar internal fragments, an alignment

Table 2: The *LRLA* (Arslan and Eğecioğlu 2002) (Section 5) and the *CLA* (Arslan and Eğecioğlu 2002) (Section 5.1) Problems

Alignment Problem	Objective	Algorithm	Time	Space	Score Returned
LRLA	maximize $s(I,J)$ such that $\|J\| \leq T$	*HALF*	$O(nm)$	$O(m)$	$\geq \frac{1}{2} LRLA^*$
		APX-LRLA	$O(nmT/\Delta)$	$O(mT/\Delta)$	$\geq LRLA^* - 2\Delta$
CLA	*LRLA* with parameters X, YY, and $T = \|Y\|$	The same *LRLA* algorithms, complexity, and results			

Table 3: New Local Alignment Problems *LAt* (Section 6), *Qt* (Section 7), and New Improved Approximation Algorithms (Section 8) for *NLAt* (Arslan et al. 2001) (Section 3)

Alignment Problem	Objective	Algorithm	Time	Space	Returned alignment satisfies
LAt	maximize $s(I,J)$ such that $\|I\| + \|J\| \geq t$	*APX-LAt*	$O(rnm)$	$O(rm)$	score $\geq LAt^*$, length $\geq (1 - \frac{1}{r})t$
Qt	find (I,J) such that $\frac{s(I,J)}{\|I\|+\|J\|} > \lambda$, and $\|I\| + \|J\| \geq t$, for parameter $\lambda > 0$	*APX-LAt*	$O(rnm)$	$O(rm)$	nor.-score $> \lambda$, length $\geq (1 - \frac{1}{r})t$
NLAt	maximize $\frac{s(I,J)}{\|I\|+\|J\|}$ such that $\|I\| + \|J\| \geq t$	Dinkelbach	$O(rnm)$ (experimental)	$O(rm)$	nor.-score $\geq NLAt^*$, length $\geq (1 - \frac{1}{r})t$
		Rational*NLAt*	$O(rnm \log n)$	$O(rm)$	nor.-score $\geq NLAt^*$, length $\geq (1 - \frac{1}{r})t$

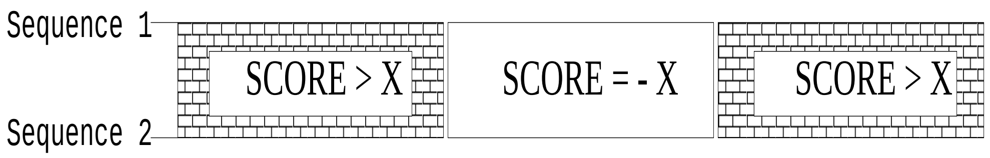


Figure 1: The Inclusion of an Arbitrarily Poor Region in an Alignment (Zhang et al. 1999)

returned may contain a mosaic of well-conserved fragments artificially connected by poorly conserved or even unrelated fragments, as shown in Figure 1. If a region of negative score $-X$ is sandwiched between two regions scoring more than X, then the Smith-Waterman algorithm will join the three regions into a single alignment that may not be biologically adequate.

It is well known that this may cause two forms of anomalies:

- *Mosaic effect* in an alignment is observed when a very poor region is sandwiched between two regions with high similarity scores.

- *Shadow effect* is observed when a biologically important short alignment is not detected because it overlaps with a significantly longer yet biologically inadequate alignment with higher overall score.

These anomalies may lead to uncertainties in comparison of long genomic sequences and comparative gene prediction, and locating coding regions in genes. As a result, applications of the Smith-Waterman algorithm to comparison of related genomes (particularly with short introns as *C.elegans* and *C.briggsae*) may lead to problems (Zhang et al. 1999).

Attempts to fix the problem of mosaic effect undertaken by Goad and Kanehisa (1982) (who introduced alignment with minimal mismatch density) and Sellers (1984) did not lead to successful algorithms and were later abandoned. The mosaic effect was first analyzed by Webb Miller and led to some studies trying to fix this problem at the post-processing stage (Huang et al. 1994, Zhang et al. 1999). Zhang et al. (1999) proposed to decompose a local alignment into sub-alignments that avoid the mosaic effect. Post-processing is also used in determining length-constrained heaviest segments (Lin et al. 2002) . However, the post-processing approach cannot detect the alignments missed by the Smith-Waterman algorithm. As a result, highly similar fragments may be ignored if they are not parts of larger alignments that dominate other local similarities.

Another approach to fixing the problems with the Smith-Waterman algorithm is based on the notion of an X-*drop*, a region within an alignment that scores below X. Alignments that contain no X-drops are called X-*alignments*. Although X-alignments are expensive to compute in practice, Altschul et al. (1997) and Zhang et al. (1998) used some heuristics for searching databases with this approach.

In both the problems of mosaic and shadow effects, the main issue is the ability of the underlying similarity measure to take into account the lengths of the strings matched. For example, if only the scores are considered, a local alignment with score 1,000 and length 10,000 (*long alignment*) is chosen over a local alignment with score 998 and length 1,000 (*short alignment*), although the latter is probably more important biologically. Moreover, if the corresponding alignment paths overlap, the more biologically important "short" alignment will not be detected even by suboptimal sequence alignment algorithm (the shadow effect).

To reflect the length of the local alignment in scoring, score $s(I, J)$ of local alignment involving substrings I and J may be adjusted by dividing $s(I, J)$ by the total length of the aligned regions (*alignment length*), $|I| + |J|$. Arslan et al. (2001) introduced the *normalized local alignment problem*, which aims 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 t is a threshold for the minimal overall length of I and J. The length constraint is necessary because length normalization favors short alignments but the alignments should be sufficiently long to be biologically meaningful.

Arslan et al. (2001) also proposed the *adjusted normalized local alignment* (*ANLA*) problem, which is a variant of the normalized local alignment problem. The objective of the problem is the same as that of the *NLA* problem, which is to obtain sufficiently long alignments with maximum length normalized score. In the *ANLA* problem the objective function is modified: the length constraint is dropped and the lengths of the optimal alignments are controlled by an artificial parameter included in the objective function. This modification allows for fast algorithms based on fractional programming, and Megiddo's search technique. There are two algorithms for the *ANLA* problem (Arslan et al. 2001). The first algorithm is a Dinkelbach algorithm. Experimental results suggest that this algorithm is only three to five times slower on average than the standard Smith-Waterman algorithm. The other algorithm, Algorithm Rational*ANLA*, is based on binary search. This algorithm runs in $O(nm \log n)$ time. We summarize these results in Table 1. Local alignment problems *LA* (Waterman 1995), (Section 2), and *ANLA* (Arslan et al. 2001) (Section 4) have exact solutions. The Smith-Waterman algorithm uses dynamic programming (2). The Dinkelbach algorithm (Figure 4) for *ANLA* uses a fractional-programming technique. Algorithm Rational*ANLA* (Figure 5) is based on Megiddo's search technique. Both *ANLA* algorithms iteratively solve *LA* problems.

Another attempt to eliminate problems associated with local alignment introduced the *length restricted local alignment* (*LRLA*) problem (Arslan and Eğecioğlu 2002), which searches 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. Indirectly, an optimal alignment is forced to have a high normalized score. The limit is placed on only the substring J of Y. 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. That is, the bound T allows for a control over the length of the optimal local alignment sought.

The *LRLA* problem can be solved by extending the dynamic-programming formulation of the local alignment problem. However the resulting time complexity is $O(Tnm)$, which may be impractical for large values of n, m, and T, each of which may be on the order of millions. Two approximation algorithms for *LRLA* have been proposed (Arslan and Eğecioğlu 2002). The first one is Algorithm *HALF*, which returns a score whose difference from the optimum is within half of the optimum, and whose complexity is the same as that of the local alignment problem. The second algorithm is Algorithm *APX-LRLA*. It returns a score guaranteed to be within 2Δ 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 solve approximately the *cyclic local alignment* (*CLA*) problem (Arslan and Eğecioğlu 2002) of maximizing $s(I, J)$, where I is a substring of X. The *CLA* problem was introduced as a dual approach to the well-known *cyclic edit distance*, which has applications in two-dimensional shape recognition, and in detecting circular permutations in proteins. These results are summarized in Table 2. Approximation algorithms *HALF* (Section 5), and *APX-LRLA* (Figure 8) for the *LRLA* problem (Arslan and Eğecioğlu 2002) (Section 5) are based on extending the dynamic-programming formulation of local alignment by using slab decomposition of the alignment graph. The same algorithms, and results are applicable to the *CLA* problem (Arslan and Eğecioğlu 2002) (Section 5.1) since *CLA* is a special case of *LRLA* (12).

In this paper we introduce new local alignment problems with length constraints, and present approximation algorithms by using the ideas in Algorithm *APX-LRLA*. We also use these results to develop approximation algorithms for the *NLAt* problem. All these approximation algorithms return alignments whose scores are at least optimal with respect to the length constraints, but the length of the resulting alignments differ from the desired length only by a prescribed fraction. These results are summarized in Table 3. In the last column of the table, nor.-score $\equiv$ normalized score. New local alignment problems introduced in this paper are *LAt* (Section 6) and *Qt* (Section 7). New improved approximation algorithms for the *NLAt* problem (Arslan et al. 2001) (Section 3) are presented in Section 8. The approximation algorithms for *LAt* (Figures 11 and 14) use the slab-decomposition technique. Problem *Qt* can be approximated by solving an *LAt* problem (Proposition 3). The Dinkelbach algorithm for *NLAt* (Figure 16) and Rational*NLAt* (Figure 15) are similar to the corresponding *ANLA* algorithms except that they iteratively solve *LAt* problems.

The first problem we introduce is the *local alignment with length threshold* (*LAt*), in which the objective is to find a sufficiently long local alignment with a high score, where the length of a given alignment is defined as the sum of the lengths of the subsequences involved in the alignment. We present Algorithm *APX-LAt*, which finds an alignment with ordinary score $\geq LAt^*$, and length $\geq (1-\frac{1}{r})t$ for a given r in time $O(rnm)$ and space $O(rm)$. Although the problem itself is not very interesting, for practical purposes an algorithm for the problem can be used to solve the next problem we introduce, namely the query problem *Qt*, and it also leads to improved approximation algorithms for the *NLAt* problem.

We define Problem *Qt* as finding long alignments with high normalized score. The motivation for the problem can be expressed by the following typical query: “Do X and Y share a (sufficiently long) fragment with more than 70% of similarity?” The problem is feasible if the answer to this query is not empty, i.e., there exists a pair of subsequences I and J with sufficiently large total length (i.e. $|I|+|J| \geq t$ for a given threshold t), and sufficiently high normalized score (i.e. $s(I,J)/(|I|+|J|) > \lambda$ for a given $\lambda > 0$). We show that, for a feasible problem, Algorithm *APX-LAt* can be used to find subsequences with normalized score $> \lambda$ and total length $\geq (1-\frac{1}{r})t$. The approximation ratio is controlled by a free parameter r. The algorithm takes $O(rnm)$ time and $O(rm)$ space.

We present new approximation algorithms for the *NLAt* problem using fractional programming, and applying Algorithm *APX-LAt*. The resulting algorithms are the *Dinkelbach* algorithm for *NLAt* and Algorithm *RationalNLAt*. Both algorithms obtain an alignment whose score is no smaller than *NLAt**, the optimum score of the original *NLAt* problem, and whose length is at least $(1 - \frac{1}{r})t$ for a given r provided that the original *NLAt* problem is feasible. In both resulting algorithms the space complexity is $O(rm)$. Test results suggest that the time complexity of the *Dinkebach* algorithm for *NLAt* is $O(rnm)$. Algorithm *RationalNLAt* has proven time complexity $O(rnm \log n)$.

The outline of the paper is as follows. In Section 2 we give the basic background for sequence comparison. Following this, we describe various alignment problems and corresponding algorithms. Sections 3, 4, and 5 are respectively for normalized local alignment *NLAt*, adjusted normalized local alignment *ANLA*, and length-restricted local alignment *LRLA*. In Sections 6 and 7 we introduce new local alignment problems, respectively, local alignment with length threshold *LAt* problem, and the query problem *Qt*. In Section 8 we present new improved approximation algorithms for the *NLAt* problem. Finally, we make some final remarks in Section 9.

An extended abstract of Sections 6, 7, and 8 of this paper was presented at the 9th International String Processing and Information Retrieval Conference (SPIRE 2002), Portugal, September 2002.

2. Framework for Pairwise Sequence Comparison

Given two strings $X = x_1x_2 \ldots x_n$ and $Y = y_1y_2 \ldots y_m$ with $n \geq m$, we use the *alignment graph* $G_{X,Y}$ to analyze *alignments* between all substrings of X and Y. The alignment graph 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 2 shows an alignment graph for $x_i \cdots x_k = ATTGT$ 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. 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 $j \leq l$. To each vertex there is an incoming arc from each neighbor, if it exists. Horizontal and vertical arcs correspond to insert and delete operations respectively. The diagonal arcs correspond to substitutions that 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.

Blocks of insertions and deletions are also referred to as *gaps* . The alignment in Figure 2

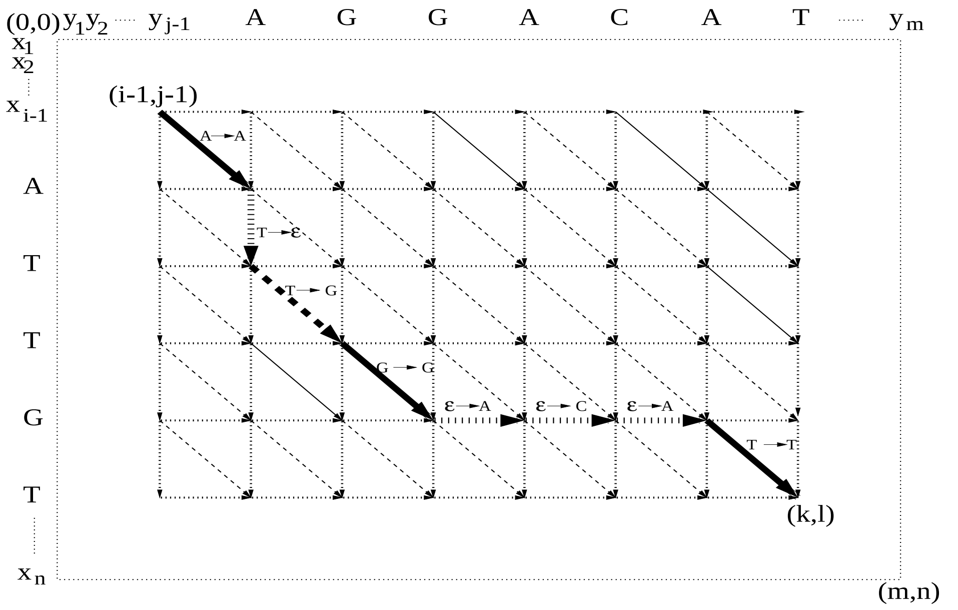


Figure 2: Alignment Graph $G_{X,Y}$ where $x_i \cdots x_k = ATTGT$ and $y_j \cdots y_l = AGGACAT$

includes two gaps with sizes one and three. We will use the terms alignment and alignment path interchangeably.

The objective of sequence alignment is to quantify the similarity between X and Y under a *scoring scheme*. In the *simple 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 one for a match ($x_i = y_j$) and $-\delta$ for a mismatch ($x_i \neq y_j$).

Given two strings X and Y, the *local alignment* (LA) problem seeks substrings $I \subseteq X$ and $J \subseteq Y$ with the highest similarity score, where $\subseteq$ indicates the substring relation. The optimum value $LA^*(X, Y)$ for this problem is given by

$$LA^*(X,Y) = \max\{s(I,J) \mid I \subseteq X, J \subseteq Y\}, \tag{1}$$

where $s(I, J) > 0$ is the best alignment score between I and J.

The following is the classical dynamic-programming formulation (Waterman 1995) to compute the maximum local alignment score $\mathcal{S}_{i,j}$ achieved by an optimal local alignment ending at each vertex (i, j):

$$\mathcal{S}_{i,j} = \max\{0,\ \mathcal{S}_{i-1,j} - \mu,\ \mathcal{S}_{i-1,j-1} + s(x_i, y_j),\ \mathcal{S}_{i,j-1} - \mu\} \tag{2}$$

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

Then

$$LA^*(X,Y) = \max_{i,j} \mathcal{S}_{i,j} \ . \tag{3}$$

Note that LA^* can be computed using the Smith-Waterman algorithm (Smith and Waterman 1981) in time $O(nm)$. The space complexity is $O(m)$ because only $O(m)$ entries of the dynamic-programming matrix need to be stored at any given time.

The simple scoring scheme can be extended such that the scores can vary depending on the individual symbols within the same edit operation type. This leads to arbitrary scoring matrices. In this case there is a dynamic-programming formulation similar to (2).

Affine gap penalties is another common scoring scheme in which the total penalty for a gap of size k, i.e. a block of k insertions (or deletions), is $\alpha + (k-1)\mu$, where α is the *gap open penalty*, and μ is called the *gap extension penalty*. The dynamic-programming formulation for this case can be described as follows (Waterman 1995): $\mathcal{E}_{i,j} = \mathcal{F}_{i,j} = \mathcal{S}_{i,j} = 0$ when i or j is 0, and define

$$\begin{aligned} \mathcal{E}_{i,j} &= \max\{\mathcal{S}_{i,j-1} - \alpha,\ \mathcal{E}_{i,j-1} - \mu\}, \\ \mathcal{F}_{i,j} &= \max\{\mathcal{S}_{i-1,j} - \alpha,\ \mathcal{F}_{i-1,j} - \mu\}, \\ \mathcal{S}_{i,j} &= \max\{0,\ \mathcal{S}_{i-1,j-1} + s(x_i, y_j),\ \mathcal{E}_{i,j},\ \mathcal{F}_{i,j}\} \ . \end{aligned} \tag{4}$$

Affine gap penalties do not increase the asymptotic complexity of the local alignment problem.

We assume that only the matches have nonnegative scores, so on any alignment the score cannot exceed the length.

For some of the techniques explained below it is also useful to express alignment problems as linear optimization problems. We define an *alignment vector* as the vector of edit-operation frequencies such that the scores and the lengths of alignments can be expressed as linear functions over alignment vectors. For example, under the basic scoring scheme, we say that (x, y, z) is an alignment vector if there is an alignment path between subsequences $I \subseteq X$ and $J \subseteq Y$ with x matches, y mismatches, and z indels. In Figure 2, $(3, 1, 4)$ is an alignment vector corresponding to the path shown in the figure. Let AV, under a given scoring scheme, denote the set of alignment vectors. Then $s(I, J)$ can be expressed as a linear function $SCORE$ over AV for the scoring schemes we study, namely, the basic scoring scheme, arbitrary scoring matrices, and affine gap penalties. For example, when simple scoring is used,

$$SCORE(a) = x - \delta y - \mu z \text{ for } a = (x, y, z) \in AV,$$

where x, y, and z of alignment vector a represent the number of matches, mismatches, and indels, respectively.

The local alignment problem LA can be rewritten as follows :

$$LA \quad : \quad maximize\ SCORE(a) \qquad \text{s.t. } a \in AV\ .$$

3. Normalized Local Alignment

Using length-normalized scores in the local alignment is suggested (Arslan et al. 2001) to cope with the mosaic and shadow effects. The degree of similarity is noted in statistics of sequence comparison. For example the similarity between nucleotide sequences of related human and mouse exons is 85% on average, while similarity between introns is 35% on average.

The objective of the *normalized local alignment* ($NLAt$) problem (Arslan et al. 2001) is

$$NLAt^{*}(X, Y) = \max\{s(I, J)/(|I| + |J|) \mid I \subseteq X, J \subseteq Y, |I| + |J| \geq t\}\ . \tag{5}$$

The length of an alignment can appropriately be defined as the sum of the lengths of the substrings involved in the alignment. For an alignment vector $a \in AV$, the length of the corresponding alignment can be expressed as a linear function $LENGTH$. For example when the simple scoring scheme is used,

$$LENGTH(a) = 2x + 2y + z \text{ for } a = (x, y, z) \in AV,$$

where x, y, and z represent the number of matches, mismatches, and indels, respectively.

Let $AVt \subseteq AV$ be a set of alignment vectors corresponding to alignments with length $\geq t$. The normalized local alignment problem $NLAt$ can be rewritten as follows :

$$NLAt \quad : \quad maximize\ \frac{SCORE(a)}{LENGTH(a)} \qquad \text{s.t. } a \in AVt\ .$$

Clearly, optimal alignments for LA and $NLAt$ may be different. When ordinary scores are used, optimal long alignments may include very poor regions (mosaic effect), and they may overshadow important alignments with relatively lower scores. If we use normalized scores, then the desired alignments depend on the value of t. 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 destroy the optimality of important long alignments, which, as a result, are not detected, causing yet another anomaly.

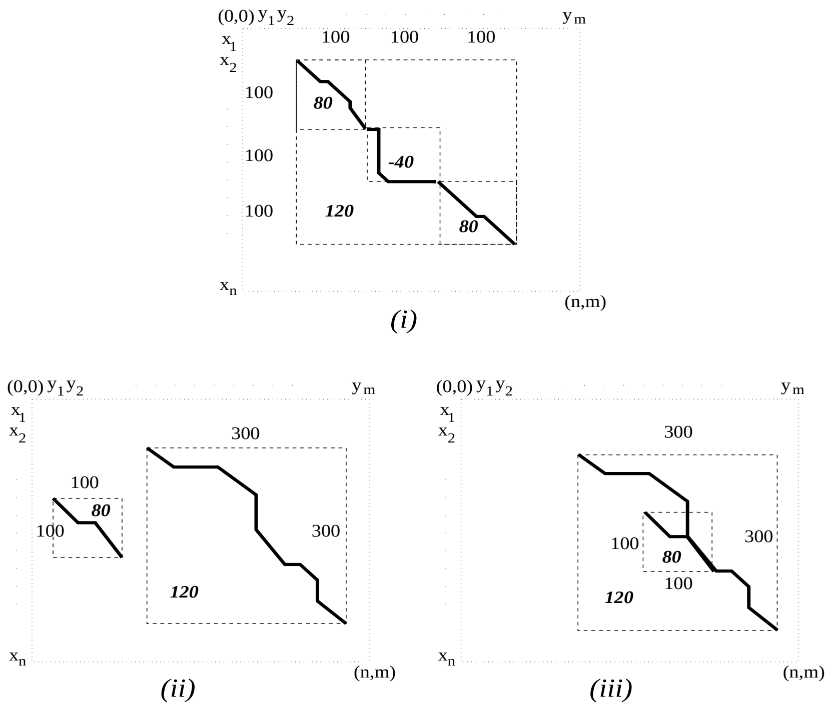


Figure 3: Sample Alignments Showing Mosaic and Shadow Effects. (*i*) Mosaic Effect, (*ii*) Shadow Effect (Non-Overlapping Alignments), (*iii*) Shadow Effect (Overlapping Alignments).

Figure 3 includes examples where optimal alignments for LA and $NLAt$ may be different. The alignments in the figure are only for illustrative purposes; they are not alignments between real biological sequences. Part (*i*) includes an example for the mosaic effect, and parts (*ii*) and (*iii*) have examples for the shadow effect with non-overlapping and overlapping alignments, respectively. Each alignment is identified by a rectangle. The numbers in italics are the ordinary scores of alignments. The unitalicized numbers in the sides of the rectangles are the lengths of the substrings involved in the alignments. The length of each alignment is equal to the sum of the two side lengths of the corresponding rectangles. The *normalized score* of an alignment is obtained by dividing its score by its length, which is defined as the sum of the lengths of the substrings involved in the alignment. The normalized score of the shorter alignment(s) in the figure is $80/200 = 0.4$, while that of the longer alignment is $120/600 = 0.2$. In each part of the figure, the long alignment has the highest ordinary score, whereas the shorter alignments have higher normalized scores. If we use ordinary scores as the similarity measure, then the long alignments in Figure 3 are optimal. If we use normalized scores, then the alignments returned depend on the value of t. For the alignments in the figure, $t = 200$ is a separating value in determining the optimality of short and long alignments.

To solve the *NLAt* problem we can extend the dynamic-programming formulation for the scoring schemes that we address in this paper by adding another dimension. At each entry of the dynamic-programming matrix we can store optimum scores for all possible alignment lengths up to $m + n$. This increases the time and space complexity to $O(n^2m)$ and $O(nm)$, respectively. These are unacceptably high because, in practice, the values of both n and m may be on the order of millions.

It may seem feasible to apply well-known graph algorithms to find long regions with a high degree of similarity. For example, we may formulate an objective with which we aim to minimize a length-normalized weighted edit distance for substrings, and include a length threshold as a lower bound for the desired length. For solving this problem, Karp's $O(|V||E|)$-time minimum mean-weight cycle algorithm (Cormen et al. 2001) seems a natural candidate. Solution requires adding extra edges to cause cycles of a certain minimum length, determined by the given length threshold. For an alignment graph for a pair of strings of length n each, the number of vertices $|V|$ and number of edges $|E|$ (excluding the additional edges) are both $O(n^2)$. This is not more efficient than naive dynamic programming.

There are approximation algorithms for the problem, which we will address in Section 8.

4. Adjusted Normalized Local Alignment

The objective of NLA may be achieved by a reformulation. In the *adjusted normalized local alignment* problem, we can modify the maximization ratio function to drop the length constraint, yet achieve a similar objective: obtain sufficiently long alignments with a high degree of similarity. The adjusted length normalized score of an alignment is computed by adding some $L \geq 0$ to the denominator in the calculation of the quotient of ordinary scores by the length. Thus, the *adjusted normalized local alignment* (*ANLA*) problem (Arslan et al. 2001) is a variant of a normalized local alignment problem in which the length constraint is dropped, and the optimization function is modified by adding a parameter L to the denominator:

$$ANLA^*(X, Y) = \max\{s(I, J)/(|I| + |J| + L) \mid I \subseteq X, J \subseteq Y, L \geq 0\} . \qquad (6)$$

The adjusted normalized local alignment problem *ANLA* can be rewritten as follows:

$$ANLA \quad : \quad maximize \; \frac{SCORE(a)}{LENGTH(a) + L} \qquad \text{s.t. } a \in AV \, .$$

The objective is still to obtain sufficiently long alignments with high length-normalized scores. Parameter L provides some control over the resulting alignment lengths. When $L = 0$, *ANLA* is equivalent to *NLAt* with no constraint on the length, in which case a single match is an optimal alignment. With larger values of L, the optimal alignments are forced to have larger ordinary alignment scores, and they tend to become longer, and yet have smaller length-normalized scores. In each example in Figure 3, the shorter alignment(s) with a score of 80 and length 200 has adjusted normalized score $\frac{80}{200+L}$, and the long alignment with a score of 120 and length 600 has adjusted normalized score $\frac{120}{600+L}$. In these cases, in *ANLA* setting L to a value smaller than 600 distinguishes shorter alignments as optimal; otherwise (for $L \geq 600$), the longer alignments are optimal. Although the optimal alignments for *ANLA* and *NLAt* may be different, to approximate the goal of *NLAt* we may use *ANLA* instead and obtain sufficiently long alignments with high normalized scores, provided that we have chosen proper values for L such that the lengths of the optimal alignments of *ANLA* meet the length constraint in *NLAt*. Using $L = 2000$ in *ANLA* reveals many interesting alignments between orthologous human (GenBank Acc. No. AF030876) and mouse (GenBank Acc. No. AF121351), and in *bli-4* locus in *C.elegans* and *C.briggsae* (Arslan et al. 2001).

For *ANLA*, faster algorithms are possible using a *fractional-programming* technique. The time complexity of *ANLA* is $O(nm \log n)$ using one algorithm. In another algorithm, the test results suggests that the time complexity is $O(nm)$, though this has not been proven. Compared to $O(n^2m)$ time complexity of *NLAt*, *ANLA* can be solved much faster.

One *ANLA* algorithm (Arslan et al. 2001) is a Dinkelbach algorithm, which uses the *parametric method* of fractional programming. The algorithm iteratively solves a so-called *parametric problem* LA_λ, which is the following optimization problem: for a given λ,

$$LA^*_\lambda(X,Y) = \max\{s(I,J) - \lambda(|I| + |J| + L) \mid I \subseteq X, J \subseteq Y\} \; . \tag{7}$$

$LA_{\lambda(X,Y)}$ can also be written as

$$LA(\lambda) \quad : \quad maximize \; SCORE(a) - \lambda \; LENGTH(a) - \lambda L \qquad \text{s.t. } a \in AV \, .$$

A parametric local alignment problem can be described in terms of the local alignment problem.

```
Algorithm Dinkelbach
Pick an arbitrary alignment, and let $\lambda^*$ be the adjusted length-normalized score
of this alignment
Repeat
  $\lambda \leftarrow \lambda^*$
  Solve $LA(\lambda)$ and let $\lambda^*$ be the adjusted length-normalized score of
  an optimal alignment
Until $\lambda^* = \lambda$
Return($\lambda^*$)
```

Figure 4: `Dinkelbach` Algorithm for *ANLA*

Proposition 1 *(Arslan et al. 2001) For $\lambda < \frac{1}{2}$, the optimum value $LA^*(\lambda)$ of the parametric LA problem can be formulated in terms of the optimum value LA^* of an LA problem.*

Proof Under the basic scoring scheme the optimum value of the parametric problem, when $\lambda < \frac{1}{2}$, is

$$LA^*_{\delta,\mu}(\lambda) = (1-2\lambda)LA^*_{\delta',\mu'} - \lambda L \text{ where } \delta' = \frac{\delta + 2\lambda}{1-2\lambda},\ \mu' = \frac{\mu+\lambda}{1-2\lambda}\ . \tag{8}$$

We can easily verify that a similar relation exists in the case of arbitrary scoring matrices, and affine gap penalties. Thus, computing $LA^*(\lambda)$ involves solving the local alignment problem LA, and performing some simple arithmetic afterward.

We assume without loss of generality that for any alignment the score does not exceed the number of matches. Therefore for any alignment, its normalized score $\lambda \leq \frac{1}{2}$. We consider $\lambda = \frac{1}{2}$ as a special case since it can only happen when the alignment is composed of matches only and $L = 0$.

An optimal solution to a ratio-optimization problem *ANLA* can be achieved via a series of optimal solutions of the parametric problem with different parameters $LA(\lambda)$. In fact $\lambda = ANLA^*$ iff $LA^*(\lambda) = 0$. That is, an alignment vector $v \in AVt$ has the optimum normalized score λ iff v is an optimal alignment vector for the parametric problem $LAt(\lambda)$ with optimum value zero. (See Arslan et al. 2001 for details; also see Craven 1988 and Sniedovich 1992 for many interesting properties of fractional programming.) The *Dinkelbach* algorithm for the *ANLA* problem is shown in Figure 4. Solutions of the parametric problems through the iterations yield improved (higher) values to λ except for the last iteration. When the algorithm terminates, the final alignment is optimal with respect to both the ordinary

```
Algorithm RationalANLA
Let $\sigma$ be the smallest gap between two adjusted length normalized scores
Initialize $[e, f] \leftarrow [0, \frac{1}{2}\sigma^{-1}]$
While $(e + 1 < f)$ do
   $k \leftarrow \lfloor (e + f)/2 \rfloor$
   If $LA^*(k\sigma) > 0$ then $e \leftarrow k$ else $f \leftarrow k$
End {while}
Return($e\sigma$)
```

Figure 5: *ANLA* Algorithm RationalANLA for Rational Scores

scores used at that iteration, and the length-normalized scoring with the original scores. This mimics the manual operation of changing the scores until the result is satisfactory.

As reported by Arslan et al. (2001), experiments suggest that the number of iterations is a small constant: three to five on average. However, a theoretical bound is yet to be established. If we assume that the sequences involved in alignments are fixed (for example, consider the normalized global alignment), and the simple scoring scheme is used, then the number of iterations is bounded by the size of the convex hull of lattice points whose diameter is bounded by the length of the strings. In this case, each parametric problem is optimized at one of the extreme points of the convex hull, and each extreme point is visited at most once during the iterations. It is known that the size of a convex hull of diameter N is $O(N^{2/3})$ (See for example Arslan and Eğecioğlu 2001). Even this rough estimate shows that the algorithm in the worst case is better than the straightforward dynamic-programming extension.

In practice the scores are rational, and in the case of rational scores there is a provably better result (Arslan et al. 2001), which is achieved by Algorithm *RationalANLA* given in Figure 5. The algorithm uses Megiddo's technique (Megiddo 1979) to perform a binary search for optimum normalized score over an interval of integers. The search is based on the sign of the optimum value of the parametric problem. In this case, if $LA^*(\lambda) = 0$, then $\lambda = ANLA^*$, and an optimal alignment vector of $LA(\lambda)$ is also an optimal solution of $ANLA$. On the other hand, if $LA^*(\lambda) > 0$ then a larger λ should be tested, and if $LA^*(\lambda) < 0$ a smaller λ should be tested (i.e., Problem $LA(\lambda)$ should be solved with a different value of λ). When the scores are rational numbers the effective search space includes $O(n^2)$ integers because the gap between any two distinct length normalized scores is $\Omega(1/n^2)$. The algorithm solves $O(\log n)$ parametric problems. Therefore, the resulting time complexity is $O(nm \log n)$, and the space complexity is $O(m)$.

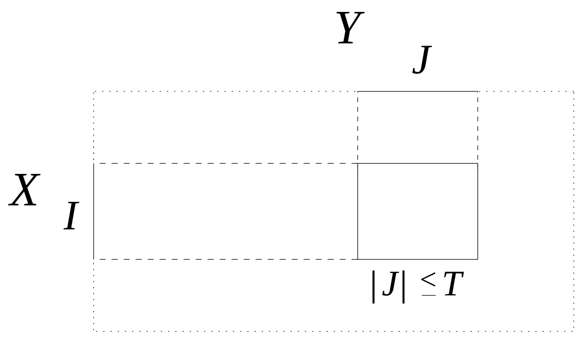


Figure 6: Candidates for I and J in the Computation of $LRLA^*(X,Y,T)$

5. Length-Restricted Local Alignment

In the *length-restricted local alignment* ($LRLA$) problem, the objective is to find 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. The objective is similar to that of the normalized local alignment in that it aims to circumvent the undesirable mosaic and the shadow effects. Indirectly, an optimal alignment is forced to have a high normalized score. The length of subsequence J in an optimal alignment is controlled by the bound T. Detecting a number of important local alignments of different horizontal lengths may require solving a series of $LRLA$ problems with different values of T.

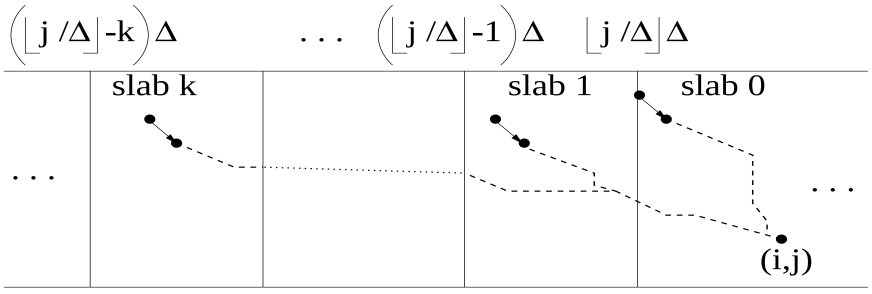
Formally, given a limit T, the $LRLA$ problem (Arslan and Eğecioğlu 2002) between X and Y is defined as follows:

$$LRLA^*(X,Y,T) = \max\{s(I,J) \mid I \subseteq X, J \subseteq Y, \text{ and } |J| \leq T\} \ . \tag{9}$$

Figure 6 illustrates the length constraint schematically. In the $LRLA$ problem, the horizontal lengths of the resulting alignments are controlled by the upper limit T on the length of one of the substrings, which in practice will be determined experimentally, or by other considerations.

$LRLA$ can be solved by extending the dynamic-programming formulation of the local alignment problem as before. However, the resulting time complexity is $O(Tnm)$, which is impractical for large values of n, m, and T.

There are two approximation algorithms (Arslan and Eğecioğlu 2002) for the $LRLA$ problem. The first is Algorithm *HALF*, which returns a score whose difference from the optimum is guaranteed to be within half of the optimum. The algorithm's complexity is the same as that of the ordinary local-alignment problem. The second algorithm, *APX-LRLA*,



$(\lfloor j/\Delta \rfloor - k)\Delta$
...
$(\lfloor j/\Delta \rfloor - 1)\Delta$
$\lfloor j/\Delta \rfloor \Delta$
slab k
slab 1
slab 0
...
...
(i,j)

Figure 7: Slabs with Respect to Column j, and Alignments Ending at Node (i, j) Starting at Different Slabs

returns a score guaranteed to be within 2Δ of the optimum for a given $\Delta \geq 1$. The time complexity of this algorithm is $O(nmT/\Delta)$, with $O(mT/\Delta)$ space.

In some cases we can control the approximation ratio of Algorithm *APX-LRLA* with the help of Algorithm *HALF*. Suppose that there exists a constant c such that for the scores of alignments of interest we can set a lower limit cT. Then first running *HALF*, and then running *APX-LRLA* with $\Delta = HALF^*/(2r)$ for any positive r we choose, we can obtain an alignment with score $\geq (1 - \frac{1}{r})LRLA^*$ in time $O(nmr)$ and space $O(mr)$. That is, the approximation ratio, and complexity of Algorithm *APX-LRLA*, can be controlled through the parameter r.

In Algorithm *HALF*, the alignment graph $G_{X,Y}$ is imagined as grouped into vertical slabs of horizontal length T each. Consider a horizontal window of size $2T$ at a time, and consider all such windows separated from each other by horizontal distance T. The algorithm computes optimal alignments for each window. The alignment with maximal score over these alignments has horizontal length not exceeding $2T$, and when split into two horizontally, one of its halves has a score within half of the optimum.

Similarly, Algorithm *APX-LRLA* assumes that the columns of the graph $G_{X,Y}$ are grouped into vertical slabs of $\Delta + 1$ columns each, starting with the leftmost column (i.e. $j = 0$). Two consecutive slabs share a column that 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. A slab does not contain the vertical edges among the vertices on the left boundary. Figure 7 includes sample slabs with respect to column j, and alignments ending at some node (i, j).

Algorithm *APX-LRLA* is shown in Figure 8. The algorithm extends the dynamic-programming formulation in (2) by considering at each node a list of scores of optimal

```
Algorithm $APX\text{-}LRLA(\delta, \mu)$
1. Run a modified Smith-Waterman algorithm.  If the maximum score is
   achieved within horizontal length $\leq T$ then return this score and exit
2. Initialization:
   set $LRLA^* = 0$
   set $\mathcal{S}_{0,j,k} = 0$ for all $j, k$,  $0 \leq j \leq m$, and $0 \leq k \leq \lfloor T/\Delta \rfloor - 1$
3. Main computations :
   for $i = 1$ to $n$ do {
    set $\mathcal{S}_{i,0,k} = 0$ for all $k$, $0 \leq k \leq \lfloor T/\Delta \rfloor - 1$
    for $j = 1$ to $m$ do {
     if $(j \bmod \Delta = 1)$ then
      {
       set $\mathcal{S}_{i,j,0} = \max\{0, s(x_i, y_j), \mathcal{S}_{i-1,j,0} - \mu\}$
       set $LRLA^* = \max\{LRLA^*, \mathcal{S}_{i,j,0}\}$
       for $k = 1$ to $\lfloor T/\Delta \rfloor - 1$ do {
         set $\mathcal{S}_{i,j,k} = \max\{0, \mathcal{S}_{i-1,j,k} - \mu, \mathcal{S}_{i-1,j-1,k-1} \oplus s(x_i, y_j), \mathcal{S}_{i,j-1,k-1} - \mu\}$
         set $LRLA^* = \max\{LRLA^*, \mathcal{S}_{i,j,k}\}$
       }
      }
      else
      {
         for $k = 0$ to $\lfloor T/\Delta \rfloor - 1$ do {
           set $\mathcal{S}_{i,j,k} = \max\{0, \mathcal{S}_{i-1,j,k} - \mu, \mathcal{S}_{i-1,j-1,k} \oplus s(x_i, y_j), \mathcal{S}_{i,j-1,k} - \mu\}$
           set $LRLA^* = \max\{LRLA^*, \mathcal{S}_{i,j,k}\}$
         }
      }
    }
   }
3. Return $LRLA^*$
```

Figure 8: Algorithm *APX-LRLA*, which Approximates $LRLA^*$ Within 2Δ

alignments, each starting in a different slab. At the heart of the algorithm is a step that considers two cases at each node (i, j):

- 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, for $0 \leq k \leq \lfloor T/\Delta \rfloor - 1$, $\mathcal{S}_{i,j,k}$ is calculated in an obvious way by using $\mathcal{S}_{i-1,j,k}$, $\mathcal{S}_{i-1,j-1,k}$ and $\mathcal{S}_{i,j-1,k}$ as

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

  where $\mathcal{S}_{i-1,j-1,k} \oplus s(x_i, y_j) = \mathcal{S}_{i-1,j-1,k} + s(x_i, y_j)$ if $\mathcal{S}_{i-1,j-1,k} > 0$ or $k = 0$, and 0 otherwise. This is because a local alignment necessarily has a positive score, and it is either a single match, or it is an extension of an alignment whose score is positive. Therefore, an alignment with no score is not extended unless the resulting alignment is a single match in the current slab.

- If the current node is on the first column following a boundary ($j \bmod \Delta = 1$), then the slabs for the nodes involved in the computations for node (i, j) differ. In this case, slab k for node (i, j) is slab $k - 1$ for the nodes at column $j - 1$. 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 $1 \leq k \leq \lfloor T/\Delta \rfloor - 1$ $\mathcal{S}_{i,j,k}$ is calculated by

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

During these computations, the running maximum score is also updated whenever a newly computed score $\mathcal{S}_{i,j,k}$ 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 for each optimal alignment its start and end positions, 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.

For the approximation result about the algorithm to hold, i.e., to prove that the algorithm approximates $LRLA^*$ within 2Δ, we first need to assume that the maximum positive score for any individual operation is at most one. In the scoring schemes we address in this paper, this can be satisfied by normalizing all the scores by dividing them by the maximum individual positive score, which does not affect the optimality of the alignments. Next, to establish that the algorithm returns an alignment whose score is within 2Δ of $LRLA^*$, we use induction on nodes (i, j), and analyze the different cases for the orientation of optimal alignments ending at each node (i, j). We omit these details, and refer the reader to Arslan and Eğecioğlu (2002).

There are variants of Algorithm *APX-LRLA* for the cases of arbitrary scoring matrices, and affine gap penalties (Arslan and Eğecioğlu 2002). Each algorithm extends the corresponding dynamic-programming formulation for ordinary local alignment. For example, the variant for affine gap penalties is based on the formulation in (4). These algorithms have the same approximation guarantee and complexity (Arslan and Eğecioğlu 2002). Although the algorithms use different formulations, the approximation and complexity results are shown similarly. For example, in the case of affine gap penalties, at each entry of matrices $\mathcal{E}$, $\mathcal{F}$, and $\mathcal{S}$, we maintain a list of scores of optimal alignments, each starting in a different slab.

5.1 Application to Cyclic Sequence Comparison

The *cyclic edit distance* (*CED*) (Maes 1990) between X and Y is the minimum edit distance between X and any cyclic shift of Y,

$$CED^*(X, Y) = \min\{ed(X, \sigma^k(Y)) \mid 0 \leq k < m\}, \tag{10}$$

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 $0 < k < m$, $\sigma^k(Y) = y_{k+1} \ldots y_m y_1 \ldots y_k$.

Cyclic edit distance appears in many applications. Bunke and Bühler (1993) presented a method that uses the cyclic edit distance for two-dimensional shape recognition. Uliel et al. (1999) suggested using it for detecting circular permutations in proteins. Figure 9 schematically describes the problem.

There are many algorithms for this problem. The most general algorithm was proposed by Maes (1990). There are other algorithms that are either output-size sensitive, or sub-optimal, or that assume some restriction on the weights. A list of references for these algorithms can be found in Arslan and Eğecioğlu (2002).

As a dual approach to the *CED* problem, we can define the *cyclic local alignment* (*CLA*)

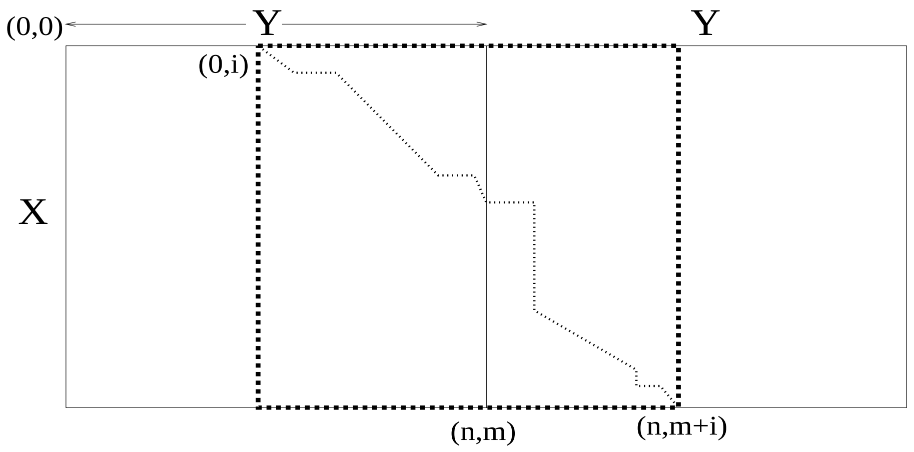


Figure 9: Definition of $CED^*(X, Y)$

problem (Arslan and Eğecioğlu 2002) by expressing its objective in the form

$$CLA^*(X, Y) = \max\{s(I, J) \mid I \subseteq X,\ J \subseteq \sigma^k(Y) \text{ for some } k,\ 0 \leq k < m\}\ . \tag{11}$$

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

$$CLA^*(X, Y) = LRLA^*(X, YY, |Y|)\ . \tag{12}$$

Maes' (1990) algorithm uses the "non-crossing" property of shortest paths. We note that this idea does not generalize to the case of *affine gap penalties*, whereas the approximation and complexity results of Algorithm $APX\text{-}LRLA$ readily holds for the case of affine gaps for the approximation of CLA^*, since CLA is a special case of $LRLA$.

6. Long Alignments with High Ordinary Score

For a given t, we define the *local-alignment-with-length-threshold* score between X and Y as

$$LAt^*(X, Y) = \max\{s(I, J) \mid I \subseteq X, J \subseteq Y, \text{ and } |I| + |J| \geq t\}\ . \tag{13}$$

Equivalently

$$LAt\ :\ maximize\ SCORE(a) \qquad \text{s.t. } a \in AVt\ .$$

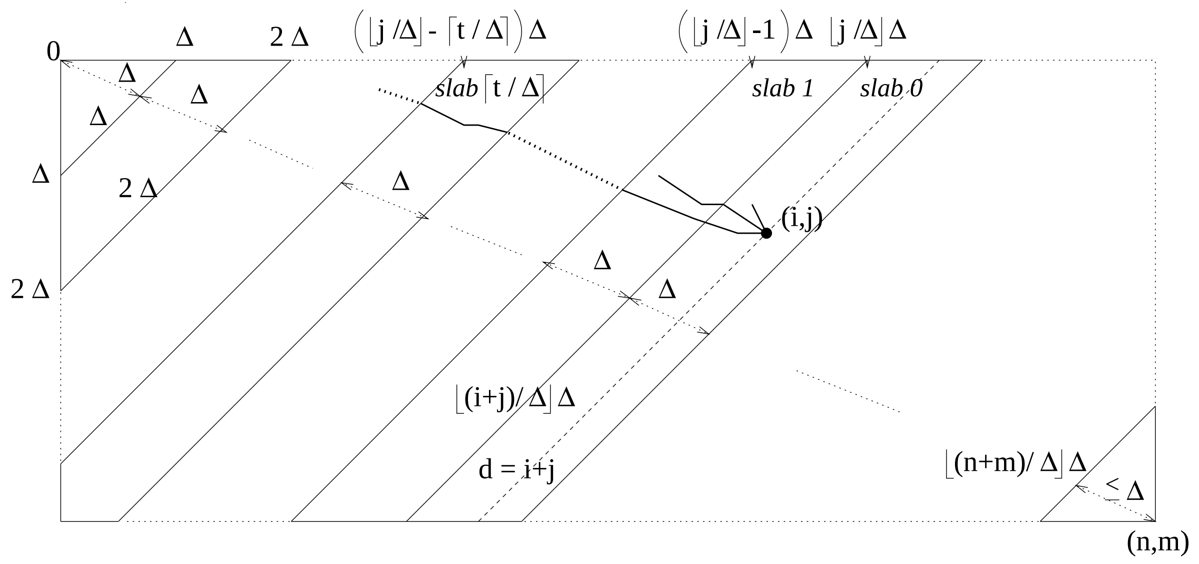
Although the problem itself is not very interesting, an algorithm for the problem can be used to find a long alignment with length-normalized score $> \lambda$ for a given positive λ, as we explain in Section 7. We also show that the algorithm for the local alignment with length threshold leads to improved approximation algorithms for the normalized local alignment problem, as we explain in Section 8.

To solve LAt we can extend the dynamic-programming formulation in (2) by adding another dimension. At each entry of the dynamic-programming matrix we store optimum scores for all possible lengths up to $m + n$, increasing the time and space complexity to $O(n^2m)$ and $O(nm)$, respectively, which are unacceptably high in practice.

We give an approximation algorithm *APX-LAt* that computes a local alignment whose score is at least LAt^*, and whose length is at least $(1 - \frac{1}{r})t$ provided that the LAt problem is feasible, i.e., the algorithm finds two sequences $\widehat{I}$ and $\widehat{J}$ such that $s(\widehat{I}, \widehat{J}) \geq LAt^*$ and $|\widehat{I}| + |\widehat{J}| \geq (1 - \frac{1}{r})t$. The algorithm runs in time $O(rnm)$ using $O(rm)$ space. For simplicity, we assume a basic scoring scheme. Our approximation idea is similar to that of Algorithm *APX-LRLA*. Instead of a single score, we maintain at each node (i, j) of $G_{X,Y}$ a list of alignments with the property that for positive s, where s is the optimum score achievable over the set of alignments with length $\geq t$ and ending at (i, j), at least one element of the list achieves score s and length $t - \Delta$, where Δ is a positive integral parameter. We show that the dynamic-programming formulation can be extended to preserve this property through the nodes. In particular, an alignment with score $\geq LAt^*$ and length $\geq t - \Delta$ will be observed in one of the nodes (i, j) during the computations.

We imagine the vertices of $G_{X,Y}$ as grouped into $\lfloor (n + m)/\Delta \rfloor$ diagonal slabs at distance Δ from each other, as shown in Figure 10.

Since we define the length of an alignment as the sum of the lengths of the substrings involved in the alignment, on a given alignment the contribution of each diagonal arc to the alignment length is two (each match, or mismatch, involves two symbols, one from each sequence), while that of each horizontal, or vertical arc is one (each indel involves one symbol from one of the sequences). Equivalently, we say that the length of a diagonal arc is two, and the length of each horizontal, or vertical arc is one. The length of an alignment a is the total length of the arcs on a. Each slab consists of $\lfloor \Delta/2 \rfloor + 1$ diagonals. Two consecutive slabs share a diagonal that we call a *boundary*. The *left* and the *right boundaries* of slab b are, respectively, the boundaries shared by the left and right neighboring slabs of b. As a subgraph, a slab contains all the edges in $G_{X,Y}$ incident to the vertices in the slab except for the horizontal and vertical edges incident to the vertices on the left boundary (which belong to the preceding slab), and the diagonal edges incident to the vertices on the first diagonal following the left boundary.



0
Δ
2 Δ
$(\lfloor j/\Delta\rfloor - \lceil t/\Delta\rceil)\Delta$
$(\lfloor j/\Delta\rfloor - 1)\Delta$
$\lfloor j/\Delta\rfloor\Delta$
slab ⌈t / Δ⌉
slab 1
slab 0
(i,j)
⌊(i+j)/Δ⌋ Δ
d = i+j
⌊(n+m)/ Δ⌋ Δ
≤ Δ
(n,m)

Figure 10: Slabs with Respect to Diagonal d, and Alignments Ending at Node (i, j) Starting at Different Slabs

Now to a given diagonal d in $G_{X,Y}$, we associate a number of slabs as follows. Let *slab 0 with respect to diagonal* d be the slab that contains the diagonal d itself. The slabs to the left of *slab 0* are then ordered consecutively as *slab 1, slab 2,* ... with respect to d. In other words, *slab k* with respect to diagonal d is the subgraph of $G_{X,Y}$ composed of vertices placed inclusively between diagonals $\lfloor d/\Delta\rfloor$ and d if $k = 0$, and between diagonal $(\lfloor d/\Delta\rfloor - k)\Delta$ and $(\lfloor d/\Delta\rfloor - k + 1)\Delta$ otherwise. Figure 10 includes sample slabs with respect to diagonal d, and alignments ending at some node (i, j) on this diagonal.

Let $\mathcal{S}_{i,j,k}$ represent the optimum score achievable at (i, j) by any alignment starting at slab k with respect to diagonal $i + j$ for $0 \le k < \lceil t/\Delta\rceil$. For $k = \lceil t/\Delta\rceil$, $\mathcal{S}_{i,j,k}$ is slightly different: it is the maximum of all achievable scores by an alignment starting in or before slab k. Also, let $\mathcal{L}_{i,j,k}$ be the length of an optimal alignment starting at slab k, and achieving score $\mathcal{S}_{i,j,k}$. A single slab can contribute at most Δ to the length of any alignment. At each node (i, j) we store $\lceil t/\Delta\rceil + 1$ score-length pairs $(\mathcal{S}_{i,j,k}, \mathcal{L}_{i,j,k})$ for $0 \le k \le \lceil t/\Delta\rceil$ corresponding to $\lceil t/\Delta\rceil + 1$ optimal alignments that end (i, j). Figure 11 shows the steps of our approximation algorithm *APX-LAt*. The processing is done row by row starting with the top row $(i = 0)$ of $G_{X,Y}$.

Step 1 of the algorithm performs the initialization of the lists of the nodes in the top row $(i = 0)$. Step 2 implements computation of scores as dictated by the dynamic-programming formulation in (2). Let maxp of a list of score-length pairs be a pair with the maximum

```
Algorithm $APX\text{-}LAt(\delta,\mu)$
1. Initialization:
   set $\widehat{LAt} = 0$
   set $(\mathcal{S}_{0,j,k}, \mathcal{L}_{0,j,k}) = (0,0)$ for all $j,k$,  $0 \le j \le m$, and $0 \le k \le \lceil t/\Delta \rceil$
2. Main computations:
   for $i = 1$ to $n$ do
   {
    set $(\mathcal{S}_{i,0,k}, \mathcal{L}_{i,0,k}) = (0,0)$ for all $k$, $0 \le k \le \lceil t/\Delta \rceil$
    for $j = 1$ to $m$ do
    {
      if $(i + j$ mod $\Delta = 1)$ then
       {
        set $(\mathcal{S}_{i,j,0}, \mathcal{L}_{i,j,0}) = (0,0)$
        for $k = 1$ to $\lceil t/\Delta \rceil - 1$ do
2.a.1       set $(\mathcal{S}_{i,j,k}, \mathcal{L}_{i,j,k}) = \text{maxp}\{\ (0,0),\ (\mathcal{S}_{i-1,j,k-1}, \mathcal{L}_{i-1,j,k-1}) + (-\mu, 1),$
                                       $(\mathcal{S}_{i-1,j-1,k-1}, \mathcal{L}_{i-1,j-1,k-1}) \oplus (s(x_i,y_j), 2),$
                                       $(\mathcal{S}_{i,j-1,k-1}, \mathcal{L}_{i,j-1,k-1}) + (-\mu, 1)\ \}$
        for $k = \lceil t/\Delta \rceil$
2.a.2       set $(\mathcal{S}_{i,j,k}, \mathcal{L}_{i,j,k}) = \text{maxp}\{\ (0,0),\ (\mathcal{S}_{i-1,j,k-1}, \mathcal{L}_{i-1,j,k-1}) + (-\mu, 1),$
                                       $(\mathcal{S}_{i-1,j-1,k-1}, \mathcal{L}_{i-1,j-1,k-1}) \oplus (s(x_i,y_j), 2),$
                                       $(\mathcal{S}_{i,j-1,k-1}, \mathcal{L}_{i,j-1,k-1}) + (-\mu, 1),$
                                       $(\mathcal{S}_{i-1,j,k}, \mathcal{L}_{i-1,j,k}) + (-\mu, 1),$
                                       $(\mathcal{S}_{i-1,j-1,k}, \mathcal{L}_{i-1,j-1,k}) \oplus (s(x_i,y_j), 2),$
                                       $(\mathcal{S}_{i,j-1,k}, \mathcal{L}_{i,j-1,k}) + (-\mu, 1)\ \}$
       } else
         {
           for $k = 0$ to $\lceil t/\Delta \rceil$ do
2.b            set $(\mathcal{S}_{i,j,k}, \mathcal{L}_{i,j,k}) = \text{maxp}\{\ (0,0),\ (\mathcal{S}_{i-1,j,k}, \mathcal{L}_{i-1,j,k}) + (-\mu, 1),$
                                          $(\mathcal{S}_{i-1,j-1,k}, \mathcal{L}_{i-1,j-1,k}) \oplus (s(x_i,y_j), 2),$
                                          $(\mathcal{S}_{i,j-1,k}, \mathcal{L}_{i,j-1,k}) + (-\mu, 1)\ \}$
         }
       for $k = \lceil t/\Delta \rceil - 1$ if $\mathcal{L}_{i,j,k} \ge t - \Delta$ then set $\widehat{LAt} = \max\{\widehat{LAt}, \mathcal{S}_{i,j,k}\}$
       for $k = \lceil t/\Delta \rceil$ set $\widehat{LAt} = \max\{\widehat{LAt}, \mathcal{S}_{i,j,k}\}$
    }
   }
3. Return $\widehat{LAt}$
```

Figure 11: Algorithm *APX-LAt*

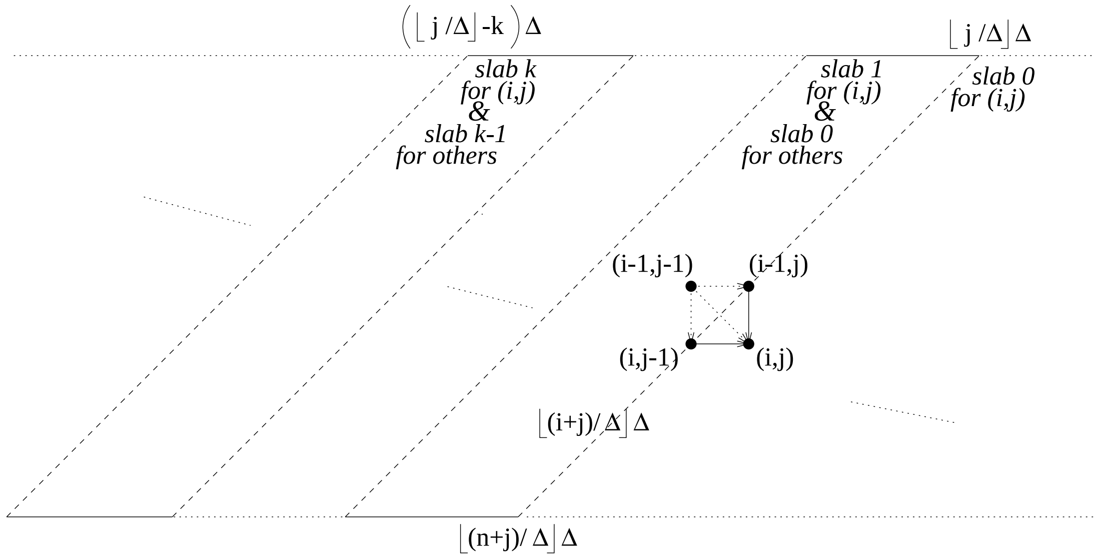


Figure 12: Relative Numbering of the Slabs with Respect to (i, j), $(i-1, j)$, $(i-1, j-1)$ and $(i, j-1)$ when Node (i, j) is on the First Diagonal Following Boundary $\lfloor (i+j)/\Delta \rfloor$

score in the list. We obtain an optimal alignment with score $\mathcal{S}_{i,j,k}$ by extending an optimal alignment from one of the nodes $(i-1, j)$, $(i-1, j-1)$, or $(i, j-1)$. We note that extending an alignment at (i, j) from node $(i-1, j-1)$ increases the length by two and the score by $s(x_i, y_j)$, whereas from nodes $(i-1, j)$ or $(i, j-1)$ adds one to the length and $-\mu$ to the score of the resulting alignment. There are two cases:

Case 1. If the current node (i, j) is not on the first diagonal 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 $(\mathcal{S}_{i,j,k}, \mathcal{L}_{i,j,k})$ is calculated by using $(\mathcal{S}_{i-1,j,k}, \mathcal{L}_{i-1,j,k})$, $(\mathcal{S}_{i-1,j-1,k}, \mathcal{L}_{i-1,j-1,k})$, and $(\mathcal{S}_{i,j-1,k}, \mathcal{L}_{i,j-1,k})$ as shown in Step 2.b where

$(\mathcal{S}_{i-1,j-1,k}, \mathcal{L}_{i-1,j-1,k}) \oplus (s(x_i, y_j), 2) = (\mathcal{S}_{i-1,j-1,k} + s(x_i, y_j), \mathcal{L}_{i-1,j-1,k} + 2)$ if $\mathcal{S}_{i-1,j-1,k} > 0$ or $k = 0$, and $(0, 0)$ otherwise. This is because, by definition, every local alignment has a positive score, and it is either a single match, or it is an extension of an alignment whose score is positive. Therefore we do not let an alignment with no score be extended unless the resulting alignment is a single match in the current slab.

Case 2. If the current node is on the first diagonal following a boundary (i.e., $i + j \bmod \Delta = 1$), then the slabs for the nodes involved in the computations for node (i, j) differ as shown in Figure 12. In this case slab k for node (i, j) is slab $k-1$ for nodes $(i-1, j)$, $(i-1, j-1)$, and $(i, j-1)$. Moreover, any alignment ending at (i, j) starting at slab 0 for (i, j) can include only one of the edges $((i-1, j), (i, j))$ or $((i-1, j-1), (i, j))$, both of which have negative weight $-\mu$. Therefore, $(\mathcal{S}_{i,j,0}, \mathcal{L}_{i,j,0})$ is set to $(0,0)$. Steps $2.a.1$ and $2.a.2$ show the calculation of $(\mathcal{S}_{i,j,k}, \mathcal{L}_{i,j,k})$ respectively for $0 < k < \lceil t/\Delta \rceil$ and for $k = \lceil t/\Delta \rceil$.

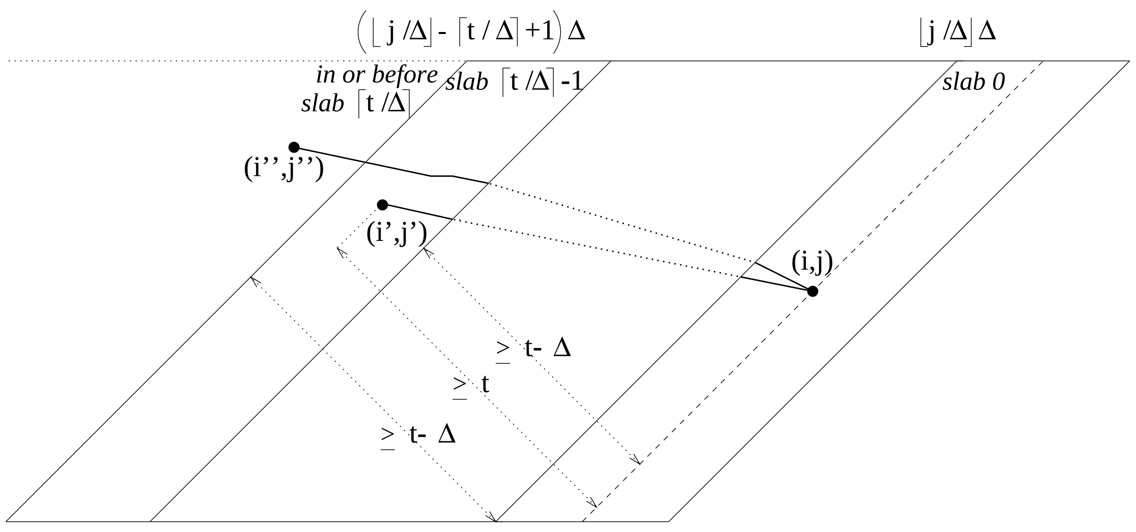
The running maximum score $\widehat{LAt}$ is updated whenever a newly computed score for an alignment with length $\geq t - \Delta$ is larger than the current maximum, which can only happen with alignments starting in or before slab $\lceil t/\Delta \rceil - 1$. The final value $\widehat{LAt}$ is returned in Step 3. The alignment position achieving this score may also be desired. This can be done by maintaining for each optimal alignment information on its start and end position in addition to its score and length. In this case, in addition to the running maximum score, the start and end positions of a maximal alignment should be stored and updated.

We first show that $\mathcal{S}_{i,j,k}$ calculated by the algorithm is the optimum score achievable, and $\mathcal{L}_{i,j,k}$ is the length of an alignment achieving this score over the set of all alignments ending at node (i,j) and starting with respect to diagonal $i+j$: (1) at slab k for $0 \leq k < \lceil t/\Delta \rceil$, (2) in or before slab k for $k = \lceil t/\Delta \rceil$. This claim can be 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 2 of the algorithm that the claim holds for node (i,j) and its slabs.

Let optimum score LAt^* for the alignments of length $\geq t$ be achieved at node (i,j). Consider the calculations of the algorithm at (i,j) at which an optimal alignment ends. There are two possible orientations of an optimal alignment, as shown in Figure 13: (1) It starts at some node (i', j') of slab $k = \lceil t/\Delta \rceil - 1$. By our previous claim, an alignment starting at slab k with score $\mathcal{S}_{i,j,k} \geq LAt^*$ is captured in Step 2. The length of this alignment $\mathcal{L}_{i,j,k}$ is at least $t - \Delta$ since the length of the optimal alignment is $\geq t$, and both start at the same slab and end at (i,j). (2) It starts at some node (i'', j'') in or before slab $k = \lceil t/\Delta \rceil$. Again, by the previous claim, an alignment starting in or before slab k with score $\mathcal{S}_{i,j,k} \geq LAt^*$ is captured in Step 2. The length of this alignment $\mathcal{L}_{i,j,k}$ is at least $t - \Delta$ since slab k is at distance $\geq t - \Delta$ from (i,j). Therefore the final value $\widehat{LAt}$ returned in Step 3 is $\geq LAt^*$ and it is achieved by an alignment whose length is $\geq t - \Delta$. We summarize these results in the following theorem.

Theorem 1 *For a feasible LAt problem, Algorithm APX-LAt returns an alignment* $(\widehat{I}, \widehat{J})$ *such that* $s(\widehat{I}, \widehat{J}) \geq LAt^*$ *and* $|\widehat{I}| + |\widehat{J}| \geq (1 - \frac{1}{r})t$ *for any* $r > 1$*. The algorithm's complexity is* $O(rnm)$ *time and* $O(rm)$ *space.*

Proof Algorithm *APX-LAt* is similar to the Smith-Waterman algorithm except that at



$\left(\lfloor j/\Delta\rfloor - \lceil t/\Delta\rceil + 1\right)\Delta$
$\lfloor j/\Delta\rfloor\Delta$
in or before slab $\lceil t/\Delta\rceil$
slab $\lceil t/\Delta\rceil - 1$
slab 0
(i'',j'')
(i',j')
(i,j)
$\geq$ t- Δ
$\geq$ t
$\geq$ t- Δ

Figure 13: Two Possible Orientations of an Optimal Alignment of Length $\geq t$ Ending at (i, j): it Starts Either at Some (i', j') at Slab $\lceil t/\Delta \rceil - 1$, or (i'', j'') in or Before Slab $\lceil t/\Delta \rceil$

each node, instead of a single score, $\lceil t/\Delta \rceil + 1$ entries for score-length pairs are stored and manipulated. Therefore, the resulting complexity exceeds that of the Smith-Waterman algorithm by a factor of $\lceil t/\Delta \rceil + 1$. That is, the time complexity of *APX-LAt* is $O(nmt/\Delta)$. The algorithm requires $O(mt/\Delta)$ space since the computations proceed row by row, and we need the entries in the previous and the current row to calculate the entries in the current row. When the *LAt* problem is feasible, it is guaranteed that Algorithm *APX-LAt* returns an alignment $(\widehat{I}, \widehat{J})$ such that $s(\widehat{I}, \widehat{J}) \geq LAt^* > 0$ and $|\widehat{I}| + |\widehat{J}| \geq t - \Delta$ for any positive Δ. Therefore, setting $\Delta = \max\{2, \lfloor t/r \rfloor\}$ for a choice of r, $1 < r \leq t$, and using Algorithm *APX-LAt* we can achieve the approximation and complexity results expressed in the theorem. We also note that for $\Delta = 2$ the algorithm becomes a dynamic-programming algorithm extending the dimension by storing all possible alignment lengths.

A variant of *APX-LAt* for arbitrary scoring matrices can be obtained by simple modifications. Figure 14 shows Algorithm *APX-LAt-AFFINE*, which is a variation of our algorithm *APX-LAt* for affine gap penalties. The algorithm is essentially quite similar to Algorithm *APX-LAt*. It uses the same idea, that at each entry of a dynamic-programming matrix, instead of a single score, a number of scores (and lengths) are maintained and manipulated as dictated by the dynamic-programming formulation in (4). Algorithm *APX-LAt* is based on the formulation in (2), which only involves matrix $\mathcal{S}$. The formulation (4) involves two

```
Algorithm $APX\text{-}LAt\text{-}AFFINE(\delta,\alpha,\beta)$
1. Initialization:
   set $\widehat{LAt}=0$
   set $(\mathcal{E}_{0,j,k},\mathcal{L}^{\mathcal{E}}_{0,j,k})=(\mathcal{F}_{0,j,k},\mathcal{L}^{\mathcal{F}}_{0,j,k})=(\mathcal{S}_{0,j,k},\mathcal{L}^{\mathcal{S}}_{0,j,k})=(0,0)$
       for all $j,k$,  $0\le j\le m$,  $0\le k\le\lceil t/\Delta\rceil$
2. Main computations :
   for $i=1$ to $n$ do {
    set $(\mathcal{E}_{i,0,k},\mathcal{L}^{\mathcal{E}}_{i,0,k})=(\mathcal{F}_{i,0,k},\mathcal{L}^{\mathcal{F}}_{i,0,k})=(\mathcal{S}_{i,0,k},\mathcal{L}^{\mathcal{S}}_{i,0,k})=(0,0)$
       for all $k$,  $0\le k\le\lceil t/\Delta\rceil$
    for $j=1$ to $m$ do {
      if $(i+j$ mod $\Delta=1)$ then {
        set $(\mathcal{E}_{i,j,0},\mathcal{L}^{\mathcal{E}}_{i,j,0})=(\mathcal{F}_{i,j,0},\mathcal{L}^{\mathcal{F}}_{i,j,0})=(\mathcal{S}_{i,j,0},\mathcal{L}^{\mathcal{S}}_{i,j,0})=(0,0)$
        for $k=1$ to $\lceil t/\Delta\rceil-1$ do {
          set $(\mathcal{E}_{i,j,k},\mathcal{L}^{\mathcal{E}}_{i,j,k})=\max\{$ $(\mathcal{S}_{i,j-1,k-1},\mathcal{L}^{\mathcal{S}}_{i,j-1,k-1})+(-\alpha,1)$,
                                  $(\mathcal{E}_{i,j-1,k-1},\mathcal{L}^{\mathcal{E}}_{i,j-1,k-1})+(-\beta,1)$ $\}$
          set $(\mathcal{F}_{i,j,k},\mathcal{L}^{\mathcal{F}}_{i,j,k})=\max\{$ $(\mathcal{S}_{i-1,j,k-1},\mathcal{L}^{\mathcal{S}}_{i-1,j,k-1})+(-\alpha,1)$,
                                  $(\mathcal{F}_{i-1,j,k-1},\mathcal{L}^{\mathcal{F}}_{i-1,j,k-1})+(-\beta,1)$ $\}$
          set $(\mathcal{S}_{i,j,k},\mathcal{L}^{\mathcal{S}}_{i,j,k})=\max\{$ $(0,0)$,
                                  $(\mathcal{S}_{i-1,j-1,k-1},\mathcal{L}^{\mathcal{S}}_{i-1,j-1,k-1})\oplus(s(x_i,y_j),2)$,
                                  $(\mathcal{E}_{i,j,k},\mathcal{L}^{\mathcal{E}}_{i,j,k})$, $(\mathcal{F}_{i,j,k},\mathcal{L}^{\mathcal{F}}_{i,j,k})$ $\}$
        }
        for $k=\lceil t/\Delta\rceil$ do {
          set $(\mathcal{E}_{i,j,k},\mathcal{L}^{\mathcal{E}}_{i,j,k})=\max\{$ $(\mathcal{S}_{i,j-1,k-1},\mathcal{L}^{\mathcal{S}}_{i,j-1,k-1})+(-\alpha,1)$,
                                  $(\mathcal{E}_{i,j-1,k-1},\mathcal{L}^{\mathcal{E}}_{i,j-1,k-1})+(-\beta,1)$,
                                  $(\mathcal{S}_{i,j-1,k},\mathcal{L}^{\mathcal{S}}_{i-1,j,k})+(-\alpha,1)$,
                                  $(\mathcal{E}_{i,j-1,k},\mathcal{L}^{\mathcal{E}}_{i,j-1,k})+(-\beta,1)$ $\}$
          set $(\mathcal{F}_{i,j,k},\mathcal{L}^{\mathcal{F}}_{i,j,k})=\max\{$ $(\mathcal{S}_{i-1,j,k-1},\mathcal{L}^{\mathcal{S}}_{i-1,j,k-1})+(-\alpha,1)$,
                                  $(\mathcal{F}_{i-1,j,k-1},\mathcal{L}^{\mathcal{F}}_{i-1,j,k-1})+(-\beta,1)$,
                                  $(\mathcal{S}_{i-1,j,k},\mathcal{L}^{\mathcal{S}}_{i-1,j,k})+(-\alpha,1)$,
                                  $(\mathcal{F}_{i-1,j,k},\mathcal{L}^{\mathcal{F}}_{i-1,j,k})+(-\beta,1)$ $\}$
          set $(\mathcal{S}_{i,j,k},\mathcal{L}^{\mathcal{S}}_{i,j,k})=\max\{$ $(0,0)$, $(\mathcal{S}_{i-1,j-1,k-1},\mathcal{L}^{\mathcal{S}}_{i-1,j-1,k-1})\oplus(s(x_i,y_j),2)$,
                                  $(\mathcal{S}_{i-1,j-1,k},\mathcal{L}^{\mathcal{S}}_{i-1,j-1,k})\oplus(s(x_i,y_j),2)$,
                                  $(\mathcal{E}_{i,j,k},\mathcal{L}^{\mathcal{E}}_{i,j,k})$, $(\mathcal{F}_{i,j,k},\mathcal{L}^{\mathcal{F}}_{i,j,k})$ $\}$
        }
      }
      else {
                for $k=0$ to $\lceil t/\Delta\rceil$ do {
                  set $(\mathcal{E}_{i,j,k},\mathcal{L}^{\mathcal{E}}_{i,j,k})=\max\{$ $(\mathcal{S}_{i,j-1,k},\mathcal{L}^{\mathcal{S}}_{i,j-1,k})+(-\alpha,1)$,
                                         $(\mathcal{E}_{i,j-1,k},\mathcal{L}^{\mathcal{E}}_{i,j-1,k})+(-\beta,1)$ $\}$
                  set $(\mathcal{F}_{i,j,k},\mathcal{L}^{\mathcal{F}}_{i,j,k})=\max\{$ $(\mathcal{S}_{i-1,j,k},\mathcal{L}^{\mathcal{S}}_{i-1,j,k})+(-\alpha,1)$,
                                         $(\mathcal{F}_{i-1,j,k},\mathcal{L}^{\mathcal{F}}_{i-1,j,k})+(-\beta,1)$ $\}$
                  set $(\mathcal{S}_{i,j,k},\mathcal{L}^{\mathcal{S}}_{i,j,k})=\max\{$ $(0,0)$, $(\mathcal{S}_{i-1,j-1,k},\mathcal{L}^{\mathcal{S}}_{i-1,j-1,k})\oplus(s(x_i,y_j),2)$,
                                         $(\mathcal{E}_{i,j,k},\mathcal{L}^{\mathcal{E}}_{i,j,k})$, $(\mathcal{F}_{i,j,k},\mathcal{L}^{\mathcal{F}}_{i,j,k})$ $\}$
                }
      }
      for $k=\lceil t/\Delta\rceil-1$ if $\mathcal{L}^{\mathcal{E}}_{i,j,k}\ge t-\Delta$ then set $\widehat{LAt}=\max\{\widehat{LAt},\mathcal{S}_{i,j,k}\}$
      for $k=\lceil t/\Delta\rceil$ set $\widehat{LAt}=\max\{\widehat{LAt},\mathcal{S}_{i,j,k}\}$
    }
   }
3. Return $LAt^*$
```

Figure 14: Algorithm *APX-LAt-AFFINE*

additional matrices $\mathcal{E}$ and $\mathcal{F}$, in addition to the main matrix $\mathcal{S}$. Matrices $\mathcal{E}$ and $\mathcal{F}$ keep track of optimal scores belonging to alignments ending with, respectively, at least one or more insertions, and at least one or more deletions. The overall optimum values are collected in matrix $\mathcal{S}$. In Algorithm *APX-LAt*, the dynamic-programming formulation is translated into list operations on matrices $\mathcal{E}, \mathcal{F}$, and $\mathcal{S}$.

We can prove that Algorithm *APX-LAt-AFFINE* returns an alignment with score $\geq LAt^*$ and length $\geq (1 - \frac{1}{r})t$ by using arguments very similar to those in the proof of approximation results for Algorithm *APX-LAt*. The claim for every node (i, j) about optimal scores, and alignments achieving these scores, are made separately on each of the pairs of $\mathcal{E}_{i,j,k}$ and $Ł^{\mathcal{E}}_{i,j,k}$, $\mathcal{F}_{i,j,k}$ and $Ł^{\mathcal{F}}_{i,j,k}$, and $\mathcal{S}_{i,j,k}$ and $Ł^{\mathcal{S}}_{i,j,k}$. These can be proved by induction on all nodes (i, j) by assuming the truth of the claims at neighboring nodes, $(i-1, j), (i, j-1)$, and $(i-1, j-1)$ in the induction step. This way we can establish that for some node (i, j), $\mathcal{S}_{i,j,k} \geq LAt^*$ (i.e., the algorithm returns an alignment whose score is $\geq LAt^*$), and $Ł^{\mathcal{S}}_{i,j,k}$ is the length of the alignment achieving this score. Next we can show that the alignment returned by the algorithm has length $\geq (1 - \frac{1}{r})t$. This essentially involves the same alignment-orientation analysis we did in the case of the approximation proof for Algorithm *APX-LAt*. Therefore, the same approximation and complexity results of Theorem 1 hold in this case as well.

7. Long Alignments Satisfying Normalized Score Threshold

We consider the problem *Qt* of finding two subsequences with normalized score higher than λ, and total length at least t. More formally,

$$Qt: \text{ find } (I, J) \text{ such that } I \subseteq X, J \subseteq Y, \ \frac{s(I, J)}{|I| + |J|} > \lambda \text{ and } |I| + |J| \geq t \ . \tag{14}$$

The following simple query explains the motivation for the problem: "Do two sequences share a (sufficiently long) fragment with more than 70% similarity?"

We present an approximation algorithm that (provided that *Qt* is feasible) finds two subsequences $\widehat{I} \subseteq X$ and $\widehat{J} \subseteq Y$ with normalized score higher than λ, and $|\widehat{I}|+|\widehat{J}| \geq (1 - \frac{1}{r})t$.

The problem is feasible for given thresholds t and $\lambda > 0$, if the answer to this query is not empty, i.e., there exists a pair of subsequences I and J with total length $|I| + |J| \geq t$, and normalized score $s(I, J)/(|I| + |J|) > \lambda$. We note that *Qt* is feasible iff $NLAt^* > \lambda$. We present an algorithm that returns, for a feasible problem, two subsequences $\widehat{I} \subseteq X$ and

$\widehat{J} \subseteq Y$ with normalized score higher than λ, and total length $|\widehat{I}| + |\widehat{J}| \geq (1 - \frac{1}{r})t$. The approximation ratio is controlled by the parameter r. The computations take $O(rnm)$ time and $O(rm)$ space.

For a given λ, we define the *parametric-local-alignment-with-length-threshold problem* $LAt(\lambda)$ as:

$$LAt(\lambda) \;:\; \textit{maximize } SCORE(a) \;-\; \lambda \, LENGTH(a) \qquad \text{s.t. } a \in AVt \, .$$

A parametric-local-alignment-with-length-threshold problem can be described in terms of a local-alignment-with-length-threshold problem.

Proposition 2 *For $\lambda < \frac{1}{2}$, the optimum value $LAt^*(\lambda)$ of the parametric LAt problem can be formulated in terms of the optimum value LAt^* of an LAt problem.*

Proof The proof is very similar to that of Proposition 1. Under the basic scoring scheme, the optimum value of the parametric problem, when $\lambda < \frac{1}{2}$, is

$$LAt^*_{\delta,\mu}(\lambda) = (1 - 2\lambda) LAt^*_{\delta',\mu'} \text{ where } \delta' = \frac{\delta + 2\lambda}{1 - 2\lambda},\; \mu' = \frac{\mu + \lambda}{1 - 2\lambda} \; . \tag{15}$$

We can easily see that a similar relation exists in the case of arbitrary scoring matrices and affine gap penalties. Computing $LAt^*(\lambda)$ involves solving the local alignment with length threshold problem LAt and performing some simple arithmetic afterward.

Under the scoring schemes we study, we assume without loss of generality that for any alignment, its normalized score is $\leq \frac{1}{2}$. We consider $\lambda = \frac{1}{2}$ as a special case that can only happen when the alignment is composed of matches only.

Proposition 3 *When solving $LAt(\lambda)$, the underlying algorithm for LAt returns an alignment $(\widehat{I}, \widehat{J})$ with normalized score higher than λ, and $|\widehat{I}| + |\widehat{J}| \geq (1 - \frac{1}{r})t$ if Problem Qt is feasible.*

Proof Assume that Problem Qt is feasible. Then $LAt^*(\lambda) > 0$, which implies that the algorithm that solves the corresponding LAt problem (of Proposition 3) returns an alignment $(\widehat{I}, \widehat{J})$ such that its score is positive (i.e., $s(\widehat{I}, \widehat{J}) - \lambda(|\widehat{I}| + |\widehat{J}|) > 0$) and $|\widehat{I}| + |\widehat{J}| \geq (1 - \frac{1}{r})t$ by the approximation results of Algorithm *APX-LAt* .

Thus, solving Qt requires a single application of Algorithm *APX-LAt*.

8. Approximation Algorithms for Normalized Local Alignment

The approximation algorithm *APX-LAt* can be applied to solving the parametric problems that arise in computing $NLAt^*$.

We present algorithms to obtain an alignment that has a normalized score no smaller than the optimum score of the original normalized-local-alignment problem with total length at least $(1-\frac{1}{r})t$ for a given r, provided that the original problem is feasible (Theorem 2). The algorithms are similar to those developed for adjusted normalized-local-alignment problems in structure, but instead of ordinary local-alignment problems they solve local alignment with length threshold problems using Algorithm *APX-LAt* presented in Section 6.

In both of the resulting algorithms, the space complexity is $O(rm)$. The number of subproblems that need to be solved is the same as in the adjusted normalized-local-alignment *ANLA* problem defined in Section 4: while one algorithm establishes that $O(\log n)$ invocations of our approximation algorithm is sufficient, experiments suggest that the other algorithm performs only three to five iterations on average, resulting in observed $O(rnm)$ time complexity.

We reiterate the definitions of the local-alignment-with-length-threshold *LAt*, normalized-local-alignment *NLAt*, and the parametric-local-alignment $LAt(\lambda)$ problems as the following optimization problems defined in terms of *SCORE* and *LENGTH* functions that are linear over *AVt* under the scoring schemes we study:

$$\begin{array}{lll} LAt & : \ maximize\ SCORE(a) & \text{s.t. } a \in AVt. \\ NLAt & : \ maximize\ \frac{SCORE(a)}{LENGTH(a)} & \text{s.t. } a \in AVt. \\ LAt(\lambda) & : \ maximize\ SCORE(a) - \lambda\ LENGTH(a) & \text{s.t. } a \in AVt. \end{array}$$

If we apply fractional programming to the normalized local alignment computation, then we can obtain an optimal solution to *NLAt* via a series of optimal solutions of the parametric problem with different parameters $LAt(\lambda)$ such that $\lambda = NLAt^*$ iff $LAt^*(\lambda) = 0$.

Theorem 2 *If $NLAt^* > 0$ then an alignment with normalized score at least $NLAt^*$, and total length at least $(1-\frac{1}{r})t$, can be computed for any $r > 1$ in time $O(rnm \log n)$ and space $O(rm)$.*

Proof Algorithm *RationalNLAt* given in Figure 15 accomplishes this. The algorithm is

```
Algorithm $APX\text{-}RationalNLAt$
If there is an exact match of size $(1-\frac{1}{r})t$ then return($\frac{1}{2}$) and exit
Let $\sigma$ be the smallest gap between two length-normalized scores
$[e,f] \leftarrow [0, \frac{1}{2}\sigma^{-1}]$
$\lambda^* \leftarrow 0$
While ($e+1 < f$) do
  $k \leftarrow \lceil (e+f)/2 \rceil$
  $APX\text{-}LAt^*(k\sigma) > 0$ then {
    $e \leftarrow k$
    $\lambda^* \leftarrow$ the normalized score of an optimal alignment obtained
  } else $f \leftarrow k$
End {while}
Return($\lambda^*$)
```

Figure 15: $APX - RationalNLAt$ Algorithm for Rational Scores

based on a binary search for optimum normalized score over an interval of integers. This takes $O(\log n)$ parametric problems to solve. The algorithm is similar to the *RationalANLA* algorithm in Figure 5, and the results are derived similarly. It first determines if there is an exact match of size $(1-\frac{1}{r})t$, which can easily be done by using the Smith-Waterman algorithm. If the answer is yes, then the algorithm returns the maximum possible normalized score and exits. The skeleton of the rest of the algorithm is the same as Algorithm *RationalNLAt* in Figure 5, based on Megiddo's search technique (Megiddo 1979). The difference is that the parametric alignment problems now have a length constraint. The algorithm computes the smallest possible gap σ between any two distinct possible normalized scores, which is $\Omega(1/(n+m)^2)$ (Arslan et al. 2001). It maintains an interval $[e,f]$, on which a binary search is done to find the largest λ for which $LAt^*(\lambda)$ is positive where e and f are integer variables. Initially e is set to zero, and f is set to $\frac{1}{2}\sigma^{-1}$ since $NLAt^*$ is in $[0, \frac{1}{2}]$. A parametric LAt problem with parameter $k\sigma$ is iteratively solved, where k is the median of integers in $[e,f]$. At each iteration the interval is updated according to the sign of the value of the parametric problem. The effective search space is the integers in $[e,f]$, and each iteration reduces this space by half. The iterations end whenever there remains no integer between e and f. By Theorem 1 and Proposition 3 in Section 6, for every $k\sigma < NLAt^*$, Algorithm *APX-LAt* returns an alignment with a positive score, and length at least $(1-\frac{1}{r})t$ as a solution to the parametric problem. After the search ends, $\lambda^* \geq NLAt^*$, and λ^* is achieved by an alignment whose length is at least $(1-\frac{1}{r})t$ for *NLAt* feasible. Note that if $NLAt^* = 0$ then the algorithm returns 0.

```
Algorithm Dinkelbach
If $APX\text{-}LAt^*(0) > 0$ then
  set $\lambda^*$ to the length-normalized score of an optimal alignment
else exit
Repeat
  $\lambda \leftarrow \lambda^*$
  If $APX\text{-}LAt^*(\lambda) > 0$ then set $\lambda^*$ to the length normalized score
  of an optimal alignment
Until $\lambda^* \leq \lambda$
Return($\lambda^*$)
```

Figure 16: Dinkelbach Algorithm for *NLAt*

The asymptotic space requirement is the same as that of Algorithm *APX-LAt*, and the loop iterates $O(\log n)$ times. Therefore the complexity results are as described in the theorem.

If $NLAt^* > 0$ then we can also achieve the same approximation guarantee by using a Dinkelbach algorithm given by Arslan et al. (2001) as the template. The details of the resulting algorithm appear in Figure 16. At each iteration, except for the last, Algorithm *APX-LAt* returns an alignment with a positive score, and length at least $(1 - \frac{1}{r})t$ as a solution to the parametric problem, by Theorem 1 and Proposition 2 in Chapter 6, since $\lambda < NLAt^*$. Solutions of the parametric problems through the iterations yield improved (higher) values of λ except for the last iteration. The resulting algorithm performs no more than three to five iterations on average (never more than nine in the worst case) in our tests. When the algorithm terminates, the optimal alignment whose length-normalized score is λ^* has total length of at least $(1 - \frac{1}{r})t$, and $\lambda^* \geq NLAt^*$.

We have implemented versions of Algorithm *APX-LAt-AFFINE* and the Dinkelbach algorithm and tested the Dinkelbach program on *bli-4* locus in *C. elegans* and *C.briggsae* for various values of parameters t and r. We have observed that the program performs three to five invocations of the *APX-LAt-AFFINE* implementation on average. Therefore, for a reasonable choice of r, its time requirement is $3r$ to $5r$ times that of a Smith-Waterman implementation, on average. In Figure 17 we include results for optimal alignments obtained as t runs from $1,000$ to $22,000$ in increments of $1,000$, and from $30,000$ to $90,000$ in increments of $10,000$, and for fixed $r = 5$. On a Beowulf class super-computer composed of a cluster of 42 linux-based 400-500 MHz workstations, it took about eight days to complete the tests. We note that if we used a fast heuristic algorithm to solve the parametric local

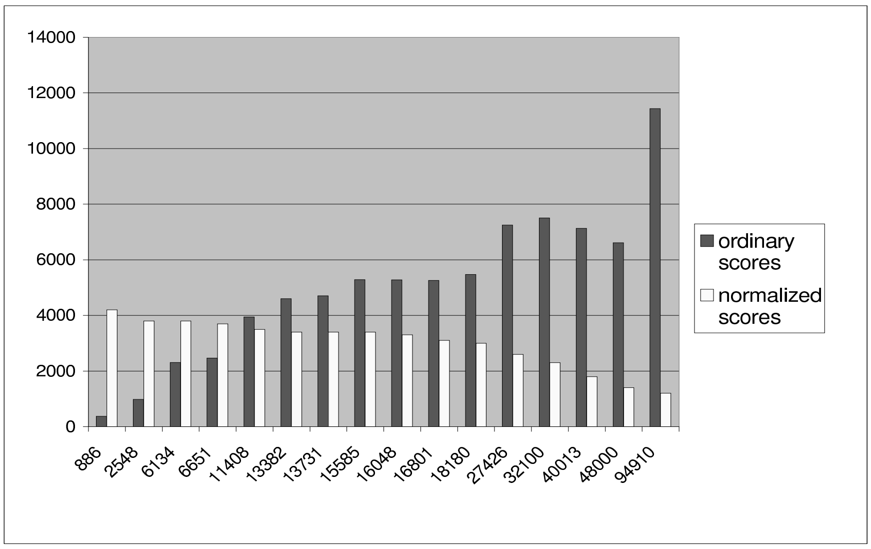


Figure 17: Ordinary vs. Normalized Scores for 16 Different Alignments. The Lengths of the Alignments are Shown on the x-Axis while the y-Axis Represents the Similarity Scores.

alignment problems, then we would have improved the running time by orders of magnitude, but the approximation guarantee of the results may no longer hold.

In our tests we have used a score of 1 for a match, -1 for a mismatch, and $-6.2-0.2(k-1)$ for a gap of length k. Figure 17 includes information about 16 different alignments, each of which is obtained for a different pair of t and r. For each alignment, we show its length on the x-axis, and both its ordinary and normalized scores on the y-axis. We have multiplied the normalized scores by $10,000$ to be able to display them on the same scale as the ordinary scores. As expected in general, normalized scores steadily decrease with increasing alignment lengths. The alignments whose lengths exceed $32,100$ include regions with very poor scores.

Test runs like this can generate important statistical information. For instance, in this case, we can infer from our approximation results and from the normalized score 0.33 of the alignment with length $16,048$ that 0.33 cannot be obtained by any alignment whose length exceeds $16,048/(1-1/5) \approx 20,000$.

As a final remark in this section, we point out the relation between the length-normalized local alignment, and a problem known as *parametric sequence alignment* (Fitch and Smith 1983) (which is different from our parametric local alignment problem) in the literature. The fractional-programming-based *NLA* algorithms iteratively and systematically change the four parameters (i.e., match score, mismatch, gap-open, and gap-extension penalties) until the resulting alignment is satisfactory (i.e., optimal with respect to ordinary scores at the last iteration, and with respect to length-normalized scores with the original scores). It is known that sequence alignment is sensitive to the choice of these parameters as they change the optimality of the alignments. Parametric sequence alignment studies the relation between the parameter settings and optimal alignments. The goal is to partition the parameter space into convex polygons such that the same alignment is optimal at every point in the same polygon. Clearly, a point in one of the polygons computed yields an optimal length-normalized alignment. The following results are summarized by Gusfield (1997): a polygonal decomposition requires $O(nm)$ time per polygon when scores are uniform (i.e., not dependent on individual symbols). When only two parameters are chosen to be variable, then the polygonal decomposition can contain at most $O(nm)$ polygons. When all four parameters are variables, then there is no known reasonable upper bound on the number of polygons. When the alignment is global and no scoring matrices are used, the number of polygons is bounded from above by $O(n^{2/3})$ (Gusfield et al. 1994).

9. Conclusion

For a given pair of sequences X and Y with lengths $n \geq m$, we have addressed a number of problems that are variants of the local-alignment problem, namely normalized local alignment (*NLAt*), length-restricted local alignment (*LRLA*), and cyclic local alignment (*CLA*). They all involve a length constraint. All of these problems have simple dynamic-programming formulations with resulting time complexities that are not practical.

The adjusted normalized local-alignment $ANLA$ problem is suggested to approximate the $NLAt$ problem by reformulating the objective function, and dropping the length constraint. For the $ANLA$ problem, the fractional-programming technique offers alternate solutions. One solution is a Dinkelbach algorithm, which has been experimentally verified to be fast. The other solution is based on binary search and it is provably fast. The time complexity of the fractional programming solution is open. We believe that in this case the existing approximation algorithms are efficient and effective.

For the *LRLA* problem there exist simple approximation algorithms that are obtained by extending the original dynamic-programming formulations by considering the alignment graph in groups of vertical or diagonal slabs, and maintaining information about a number of optimal alignments instead of a single one.

For the *LAt* problem we present an approximation algorithm that computes a local alignment whose score is at least LAt^*, and whose length is at least $(1 - \frac{1}{r})t$, provided that the LAt problem is feasible, The algorithm runs in time $O(rnm)$ using $O(rm)$ space.

Using this algorithm, we have proposed an algorithm that, given thresholds $\lambda > 0$ and t, finds an alignment with a normalized score higher than λ and with total length no smaller than $(1 - \frac{1}{r})t$, provided that the corresponding normalized local-alignment problem is feasible. The length of the result can be made arbitrarily close to t by increasing r. This is done at the expense of allocating more resources, since the time and space complexities depend on the parameter r as $O(rnm)$ and $O(rm)$, respectively.

Based on techniques previously proposed by Arslan et al. (2001) and using our approximation algorithm for the LAt problem, we have further developed ways to find an alignment with normalized score no smaller than the maximum normalized score achievable by alignments with length at least t. The alignment returned by the algorithm is guaranteed to have total length $\geq (1 - \frac{1}{r})t$. In our experiments, we have observed that the time requirement of the Dinkelbach implementation is $O(rnm)$, on average. This is better than the worst-case time complexity $O(n^2m)$ of the naive algorithm.

We believe that our approximation algorithms have made normalized scores a viable similarity measure in pairwise local alignment since they provide approximate control over the desired alignment lengths. Since the computed normalized score for a particular value of t is an upper bound for the actual normalized scores achievable by sequences of length at least t, these algorithms can also be used to collect statistics about scores of alignments versus length for a particular pair of input sequences. A number of interesting problems are open for further study:

- How many iterations do the Dinkelbach *ANLA* or *NLAt* algorithms perform in the worst case?
- Are there (provably) faster algorithms for the *NLA* problems based on other techniques such as cutting planes?
- Are there faster exact, or better approximation, algorithms for *LRLA*, *LAt*, or *Qt*?

Our algorithms for *NLA* computations use as subroutine algorithms for *LA* and *APX-LAt*, both of which are dynamic-programming-based. Clearly, one way to improve the complexity of *NLA* algorithms is to develop more efficient algorithms for *LA* and *LAt*. The ordinary local-alignment problem *LA* has been studied extensively in the literature. For this problem, there are several fast heuristic algorithms, such as FASTA (Lipman and Pearson 1985) and BLAST (Altschul et al. 1990, 1997; Altschul and Gish 1996). FASTA starts with locating exact short matches (subalignments), and combines them if they are close (in dot matrix, or the alignment graph). In this way, it aims to find the high-scoring ungapped alignments, and finally the gapped alignments, by joining the ungapped alignments. BLAST starts with a short stretch of identities and uses them as seeds (subalignments) for larger alignments. These subalignments are extended as long as the resulting score is positive, hoping that they yield optimal local alignments. Use of these or similar algorithms in our *NLA* algorithms can be empirically studied. It may be possible to devise heuristics directly for *NLA* computations, which may start with some set of subalignments, and assemble them progressively.

Acknowledgments

This work was supported in part by the second author's NSF Grant No. EIA–9818320.

References

Altschul, S., W. Gish. 1996. Local alignment statistics. *Methods in Enzymology* **266** 460–480.

Altschul, S., W. Gish, W. Miller, E. Myers, J. Lipman. 1990. Basic local alignment search tool. *Journal of Molecular Biology* **215** 413–410.

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

Arslan, A.N., Ö. Eğecioğlu. 2001. An improved upper bound on the size of planar convex-hulls. *Lecture Notes in Computer Science 2108 (COCOON 2001)* 111–120.

Arslan, A.N., Ö. Eğecioğlu. 2002. Approximation algorithms for local alignment with length constraints. *International Journal of Foundations of Computer Science* **13** 751–767.

Arslan, A.N., Ö. Eğecioğlu, P.A. Pevzner. 2001. A new approach to sequence comparison: normalized local alignment. *Bioinformatics* **17** 327–337.

Bunke, H., U. Bühler. 1993. Applications of approximate string matching to 2d shape recognition. *Pattern Recognition* **26** 1797–1812.

Cormen, T.H., C.E. Leiserson, R.L. Rivest, C. Stein, 2001. *Introduction to Algorithms*, 2nd ed. The MIT Press, Cambridge, MA.

Craven, B.D. 1988. *Fractional Programming.* Helderman Verlag, Berlin, Germany.

Fitch, W.M., T.F. Smith. 1983. Optimal sequence alignments. *Proc. Natl. Academy Science* **80** 1382–1386.

Goad, W.B., M.I. Kanehisa. 1982. Pattern recognition in nucleic acid sequences, I: a general method for finding local homologies and symmetries. *Nucleic Acid Research* **10** 247–163.

Gusfield, D. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* The Press Syndicate of The University of Cambridge, New York.

Gusfield, D., K. Balasubramanian, D. Naor. 1994. Parametric optimization of sequence alignment. *Algorithmica* **12** 312–326.

Huang, X., P.A. Pevzner, W. Miller. 1994. Parametric recomputing in alignment graph. *Proc. of the 5th Annual Symp. on Comb. Pat. Matching*, Asilomar, CA. 87–101.

Lin, Y.L., T. Jiang, K.M. Chao. 2002. Efficient algorithms for locating the length-constrained heaviest segments with applications to biomolecular sequence analysis. *Journal of Computer and System Sciences* **65** 570–586.

Lipman, D.J., W.R. Pearson. 1985. Rapid and sensitive protein searches. *Science* **227** 1435–1441.

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

Megiddo, N. 1979. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research* **4** 414–424.

Sellers, P.H. 1984. Pattern recognition in genetic sequences by mismatch density. *Bull. Math. Bio.* **46** 501–504.

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

Sniedovich, M. 1992. *Dynamic Programming.* Marcel Dekker, New York.

Uliel, S., A. Fliess, A. Amir, R. Unger. 1999. A simple algorithm for detecting circular permutations in proteins. *Bioinformatics* **15** 930–936.

Waterman, M.S. 1995. *Introduction to Computational Biology*. Chapman & Hall, London, U.K.

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

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