Dynamic Programming Based Approximation Algorithms for Sequence Alignment with Constraints

Abdullah N. Arslan
Department of Computer Science
University of Vermont
Burlington, VT 05401, USA
aarslan@cs.uwm.edu
and
Ömer Egecioglu*
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106, USA
omer@cs.ucsb.edu

Abstract

Pairwise local alignment is one of the most important problems in sequence analysis in computational biology. Classical dynamic programming solution to this problem searches for two segments with maximum similarity score by discarding poorly conserved initial and terminal fragments. As a consequence, an alignment returned as optimal may contain a mosaic of well-conserved fragments artificially connected by poorly-conserved or even unrelated fragments. In an attempt to solve the problems associated with the classical definition of local similarity, several approaches have redefined the objective of local alignment to incorporate segment lengths. These approaches give rise to problems with varying objective functions and constraints. When direct dynamic programming solutions exist for the constrained versions, the time complexity of the resulting exact algorithm is prohibitive. We present a survey of approximation algorithms and complexity results for each of these problems.

Keywords: Local alignment, cyclic sequence comparison, normalized local alignment, length restricted local alignment, approximation algorithm, dynamic programming, ratio maximization, fractional programming.

*Supported in part by NSF Grant No. EIA-9818320.
1 Introduction

The Smith-Waterman algorithm finds an optimal local alignment by locating segments with maximum similarity score using dynamic programming. It essentially discards poorly conserved initial and terminal fragments. Thus an alignment returned may contain a mosaic of well-conserved fragments artificially connected by poorly conserved or even unrelated fragments. 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.

The attempts to fix the problem of mosaic effect undertaken by Goad and Kanehisa, 1982 [7] (who introduced alignment with minimal mismatch density) and Sellers, 1984 [11] 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 [8, 14]. Zhang et al. [14] 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 [9]. However, the post-processing approach may miss the alignments with the best degree of similarity if the Smith-Waterman algorithm missed them. As a result, highly similar fragments may be ignored if they are not parts of larger alignments dominating other local similarities.

Another approach to fixing the problems with the Smith-Waterman algorithm is based on the notion of X-drop, a region within an alignment that scores below X. The alignments that contain no X-drops are called X-alignments. Although X-alignments are expensive to compute in practice, Altschul et al. [1] and Zhang et al. [15] used some heuristics for searching databases with this approach.

In both of 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.

In this paper we consider such constrained alignment problems and describe corresponding algorithms developed for them. In most cases, there are simple dynamic programming formulations for the exact version of a given alignment problem with length constraints. These are standard extensions of the existing dynamic programming algorithms which simply incorporate an additional dimension for the length parameter. However the resulting algorithms are of unacceptably high complexity for practical purposes, necessitating alternate approaches and approximate solutions. We describe approximation algorithms for the constrained alignment problems with guaranteed performance bounds, which provide a tradeoff between algorithm complexity and the quality of the solution obtained. The approximation can be 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, optimality of the alignment returned is assured, but the length constraints are satisfied to within a prescribed tolerance from the required values.

The outline of the paper is as follows. In section 2 we give the basic background for sequence comparison. In section 3 we give the definition of normalized local alignment. The
motivation for this problem has been to find sufficiently long local alignments with high
degree of similarity. In section 4 we give the definition of adjusted normalized local alignment
problem which is a modified normalized local alignment problem. We include in this section
two algorithms for the problem. In section 5 we define and give an approximation algorithm
for the length restricted local alignment problem. The resulting approximation algorithm has
applications in cyclic sequence comparison. A similar idea is used in an algorithm for finding
a long alignment with high ordinary score. In section 7 we describe how this algorithm can
be used to find an alignment with a given length normalized score, and whose length is
guaranteed to be only a prescribed fraction smaller than a given threshold. This algorithm
also yields new improved approximation algorithms for the normalized local alignment problem
as explained in section 8.

2 Framework for Pairwise Sequence Comparison

Given two strings $X = x_1 x_2 \ldots x_n$ and $Y = y_1 y_2 \ldots y_m$ 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 $0 \leq v \leq m$ (See for example, [10, 3]). An alignment path for substrings $x_i \ldots x_k$ and
$y_j \ldots y_l$ 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 which are either matching (if the corresponding symbols are the
same), or mismatching (otherwise). If we trace the arcs of an alignment path for substrings $I$
and $J$ and perform the indicated edit operations in the given order on $I$, we obtain $J$. Blocks
of insertions and deletions are also referred to as gaps.

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 $\delta$ (mismatch penalty) and $\mu$ (indel or gap penalty). We
assume that $s(x_i, y_j)$ is the similarity score between the symbols $x_i$, and $y_j$ which is normally
1 for a match $(x_i = y_j)$ and $-\delta$ for a mismatch $(x_i \neq y_j)$.

Given two strings $X$ and $Y$ the local alignment ($LA$) problem seeks for 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

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

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

The following is the classical dynamic programming formulation [10] to compute the
maximum local alignment score $S_{i,j}$ achieved by an optimal local alignment ending at each
vertex $(i, j)$:

$$S_{i,j} = \max\{0, S_{i-1,j} - \mu, S_{i-1,j-1} + s(x_i, y_j), S_{i,j-1} - \mu\}$$  \hspace{1cm} (2)

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

Then
\[ LA^*(X, Y) = \max_{i,j} S_{i,j} \]

\( LA^* \) can be computed using the Smith-Waterman algorithm [12] in time \( O(mn) \). The space complexity is \( O(m) \) because only \( O(m) \) entries of the dynamic programming matrix need to be stored at any given time. In what follows, for any optimization problem \( P \), we denote by \( P^* \) its optimum value, and sometimes drop the parameters from the notation when they are obvious from the context. An optimization problem \( P \) is called feasible if it has a solution with the given parameters.

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 \( \alpha \) is the gap open penalty, and \( \mu \) is called the gap extension penalty. The dynamic programming formulation for this case can be described by using three matrices [10]. This does not increase the complexity of the local alignment problem.

3 Normalized Local Alignment

Using length-normalized scores in the local alignment is suggested [6] to cope with the mosaic, and shadow effects. The degree of similarity are noted in statistics. 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 (NLA) problem [6] is
\[ \text{NLA}^*(X, Y) = \max \{ s(I, J)/(|I| + |J|) \mid I \subseteq X, J \subseteq Y, |I| + |J| \geq t \} \]

In general optimal alignments for \( LA \) and \( \text{NLA} \) are different (See [6] for a detailed example). The length constraint in the problem definition is necessary, as otherwise an optimal alignment returned may be too short to be meaningful.

To solve the \( \text{NLA} \) 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 unacceptably high \( O(n^2m) \) and \( O(nm) \), respectively, 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 adjusted normalized local alignment problem, we can modify the maximization ratio in such a way that we drop the length constraint, yet achieve a similar objective: 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 \textit{adjusted normalized local alignment (ANLA)} problem [6] 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:

\[
ANL^* (X, Y) = \max \{ s(I, J) / (|I| + |J| + L) \mid I \subseteq X, J \subseteq Y, L \geq 0 \}
\]  

(5)

The objective is still to obtain sufficiently long alignments with high length normalized scores. Parameter \( L \) provides some control over the resulting alignment lengths. Faster algorithms are possible because the length constraint is no longer there. One of these algorithms uses the \textit{parametric method} of \textit{fractional programming} and is the Dinkelbach ANLA Algorithm [6] shown in Figure 1. The algorithm iteratively solves a so-called \textit{parametric problem} \( LA_\lambda \) which is the following optimization problem: For a given \( \lambda \)

\[
LA^*_\lambda (X, Y) = \max \{ s(I, J) - \lambda (|I| + |J| + L) \mid I \subseteq X, J \subseteq Y \}
\]  

(6)

\( LA^*_\lambda \) can be expressed in terms of \( LA^* \) with different set of scores (See proposition 1 in [6] for the case of simple scoring scheme). Solving each \( LA^*_\lambda \) involves solving an \( LA^* \) problem and then performing some simple arithmetic afterwards. When the algorithm terminates, the final alignment is optimal with respect to both the ordinary 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.

\begin{algorithm}
Pick an arbitrary alignment, and let \( \lambda^* \) be the adjusted length normalized score of this alignment
Repeat
\( \lambda \leftarrow \lambda^* \)
Solve \( LA_\lambda \) and and let \( \lambda^* \) be the adjusted length normalized score of an optimal alignment
Until \( \lambda^* = \lambda \)
Return(\( \lambda^* \))

Figure 1: Dinkelbach algorithm for ANLA.

As reported in [6], experiments suggest that the number of iterations is a small constant: 3-5 on the 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 diamater \( N \) is \( O(N^{2/3}) \) (See for example [5]). Even this
rough estimate shows that the algorithm in the worst case is better than the straightforward dynamic programming extension.

For the case of rational scores there is a provably better result [6] which is achieved by Algorithm RationalANLA given in Figure 2. The algorithm uses Megiddo’s technique to perform a binary search for optimum normalized score over an interval of integers. When the scores are rational numbers the effective search space includes $O(n^2)$ integers. The algorithm solves $O(\log n)$ parametric problems. Therefore the resulting time complexity is $O(nm \log n)$, and the space complexity is $O(m)$.

**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

$k \leftarrow [(e + f)/2]$

If $LA^*(k \sigma) > 0$ then $e \leftarrow k$ else $f \leftarrow k$

End {while}

Return($e \sigma$)

Figure 2: ANLA algorithm RationalANLA for rational scores.

## 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 have 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 [4] between $X$ and $Y$ is defined as follows

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

(7)

Figure 3 illustrates the length constraint schematically. In LRLA problem, the horizontal lengths of the resulting alignments are controlled by upper limit $T$ on one of the length 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 local alignment problem as before. However the resulting time complexity is $O(T nm)$, which is impractical for large values of $n$, $m$, and $T$.

There are two approximation algorithms [4] for the LRLA problem. The first one is a simple $\frac{1}{2}$-approximation algorithm, Algorithm HALF, and its complexity is the same as that
of the ordinary local alignment problem. The second algorithm, Algorithm $APX-LRLA$, returns a score guaranteed to be within difference $2\Delta$ of the optimum for a given $\Delta \geq 1$. The time complexity of this algorithm is $O(mnT/\Delta)$, with $O(mT/\Delta)$ space.

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 yields a $\frac{1}{2}$-approximation result.

Similarly Algorithm $APX-LRLA$ assumes that the columns of the graph $G_{X,Y}$ are grouped into vertical slabs of $\Delta + 1$ columns each. Figure 4 includes sample slabs with respect to column $j$, and alignments ending at some node $(i,j)$.

![Figure 4: Slabs with respect to column $j$, and alignments ending at node $(i,j)$ starting at different slabs.](image)

The algorithm extends the dynamic programming formulation in (2) by considering at each node a list of scores of optimal alignments each starting in a different slab. For the approximation result to hold we need to assume that the maximum positive score for any individual operation is at most 1. 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. There are variants of Algorithm $APX-LRLA$ for the cases of arbitrary scoring matrices, and affine gap penalties. They have
the same approximation guarantee, and complexity [4].

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 $\text{HALF}$, and then running $\text{APX-LRLA}$ with $\Delta = \text{HALF}^* / (2r)$ we can obtain an alignment with score $\geq (1 - \frac{1}{r})LRLA^*$ in time $O(nmr)$ and space $O(mr)$ for any positive $r$.

5.0.1 Application to Cyclic Sequence Comparison

The cyclic local alignment (CLA) problem [4] is defined as follows:

$$CLA^*(X,Y) = \max\{s(I,J) \mid I \subseteq X, J \subseteq \sigma^k(Y) \text{ for some } k, 0 \leq k < m\} \quad (8)$$

where $\sigma^k(Y)$ is the cyclic shift of $Y$ by $k$ which is defined as follows: $\sigma^0(Y) = Y$, and for $0 < k < m$, $\sigma^k(Y) = y_ky_{k+1} \cdots y_my_1 \cdots y_k$.

Cyclic sequence comparison has applications in detecting circular permutations in proteins [13]. Note that $\text{CLA}$ is a special case of $\text{LRLA}$. More specifically

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

The approximation and complexity results of Algorithm $\text{APX-LRLA}$ holds for the approximation of $CLA^*$.

6 Long Alignments with High Ordinary Score

For a given $t$, the local alignment with length threshold score [3] between $X$ and $Y$ is defined as

$$LAT^*(X,Y) = \max\{s(I,J) \mid I \subseteq X, J \subseteq Y, \text{ and } |I| + |J| \geq t\} \quad (9)$$

To solve $\text{LAT}$ we can extend the dynamic programming formulation in (2) by adding another dimension for the length as is the case in general. This naive approach increases the time and space complexity to $O(n^2m)$ and $O(nm)$, respectively, which are again unacceptably high in practice. An approximation algorithm $\text{APX-LAT}$ is described in [3]. $\text{APX-LAT}$ computes a local alignment whose score is at least $LAT^*$, and whose total length is at least $(1 - \frac{1}{r})t$ provided that the $LAT$ problem is feasible.

The approximation idea is similar to the one in Algorithm $\text{APX-LRLA}$ [4]. Instead of a single score, the algorithm maintains at each node $(i,j)$ of $G_{X,Y}$, a list of alignments. These lists have the following property. Suppose $\Delta > 0$ is a positive integral parameter and $s > 0$ is the optimum score achievable over the set of alignments with length $\geq t$ and ending at $(i,j)$. Then at least one element of the list kept at $(i,j)$ achieves score $s$ and length $t - \Delta$. 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 computation.

The vertices of the alignment graph $G_{X,Y}$ are imagined to be grouped into $\lfloor (n + m)/\Delta \rfloor$ diagonal slabs at distance $\Delta$ from each other as shown in Figure 5. The length of a diagonal
Figure 5: Slabs with respect to diagonal \( d \), and alignments ending at node \((i, j)\) starting at different slabs.

arc is 2 while the length of each horizontal, or vertical arc is 1. Each slab consists of \([\Delta/2]+1\) diagonals.

Let \( \max \) of a list of score-length pairs be a pair with the maximum score in the list. An optimal alignment with score \( S_{i,j,k} \) is obtained by extending an optimal alignment from one of the nodes \((i-1, j)\), \((i-1, j-1)\), or \((i, j-1)\). Extending an alignment at \((i, j)\) from node \((i-1, j-1)\) increases the length by 2 and the score by \( s(x_i, y_j) \), whereas from nodes \((i-1, j)\) or \((i, j-1)\) adds 1 to the length and \(-\mu\) to the score of the resulting alignment. The algorithm basically extends the dynamic programming formulation in (2). At the heart of the algorithm is a step which considers two cases:

- 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 for \(0 \leq k \leq \lfloor t/\Delta \rfloor\)

\[
(S_{i,j,k}, L_{i,j,k}) = \max \{ (0,0), (S_{i-1,j,k}, L_{i-1,j,k}) + (-\mu, 1),
(S_{i-1,j-1,k}, L_{i-1,j-1,k}) \oplus (s(x_i, y_j), 2),
(S_{i,j-1,k}, L_{i,j-1,k}) + (-\mu, 1) \}
\]

where \( (S_{i-1,j-1,k}, L_{i-1,j-1,k}) \oplus (s(x_i, y_j), 2) = (S_{i-1,j-1,k} + s(x_i, y_j), L_{i-1,j-1,k} + 2) \) if \( S_{i-1,j-1,k} > 0 \) or \( k = 0 \); and \((0,0)\) otherwise.

- If the current node is on the first diagonal following a boundary (i.e. \(i + j \mod \Delta = 1\)) then the slabs for the nodes involved in the computations for node \((i, j)\) differ. In this case \((S_{i,j,0}, L_{i,j,0})\) is set to \((0,0)\), and for \(1 \leq k \leq \lfloor t/\Delta \rfloor - 1\), \((S_{i,j,k}, L_{i,j,k})\) is calculated by

\[
(S_{i,j,k}, L_{i,j,k}) = \max \{ (0,0), (S_{i-1,j,k-1}, L_{i-1,j,k-1}) + (-\mu, 1),
(S_{i-1,j-1,k-1}, L_{i-1,j-1,k-1}) \oplus (s(x_i, y_j), 2),
(S_{i,j-1,k-1}, L_{i,j-1,k-1}) + (-\mu, 1) \}
\]

\(9\)
The optimum score for \( k = \lfloor t/\Delta \rfloor \) is special because the length of an optimal alignment is allowed to grow arbitrarily:

\[
(S_{i,j,k}, L_{i,j,k}) = \max \left\{ (0, 0), (S_{i-1,j,k-1}, L_{i-1,j,k-1}) + (-\mu, 1), \\
(S_{i,j-1,k-1}, L_{i,j-1,k-1}) \oplus (s(x_i, y_j), 2), \\
(S_{i,j,k-1}, L_{i,j,k-1}) + (-\mu, 1), \\
(S_{i,j,k}, L_{i,j,k}) + (-\mu, 1), \\
(S_{i,j-1,k}, L_{i,j-1,k}) \oplus (s(x_i, y_j), 2), \\
(S_{i,j-1,k}, L_{i,j-1,k}) + (-\mu, 1) \right\}
\]

The results about the algorithm are summarized in the following theorem:

**Theorem 1** [3] For a feasible \( \text{LAT} \) problem, algorithm \( \text{APX-LAT} \) returns an alignment \((\hat{I}, \hat{J})\) such that \( s(\hat{I}, \hat{J}) \geq \text{LAT}^* \) and \( |\hat{I}| + |\hat{J}| \geq (1 - \frac{1}{r})t \) for any \( r > 1 \). The algorithm’s complexity is \( O(rnm) \) time and \( O(rm) \) space.

We can easily verify that the approximation and complexity results expressed in the theorem follow from the results established for Algorithm \( \text{APX-LAT} \) if we set \( \Delta = \max \{2, \lfloor t/r \rfloor \} \) for a choice of \( r \), \( 1 < r \leq t \). For \( \Delta = 2 \) the algorithm becomes a dynamic programming algorithm extending the dimension by storing all possible alignment lengths.

A variant of \( \text{APX-LAT} \) for arbitrary scoring matrices can be obtained by simple modifications. There is also a variant of the algorithm for affine gap penalties [2].

### 7 Long Alignments Satisfying Normalized Score Threshold

Another problem we consider is called \( Q_t \) which asks to find two subsequences with normalized score higher than \( \lambda > 0 \), and total length at least \( t \) as in [3]. That is

\[
Q_t : \text{find } (I, J) \text{ such that } I \subseteq X, J \subseteq Y, \frac{s(I, J)}{|I| + |J|} > \lambda \text{ and } |I| + |J| \geq t 
\]

(10)

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

Provided that \( Q_t \) is feasible we can find two subsequences \( \hat{I} \subseteq X \), and \( \hat{J} \subseteq Y \) with normalized score higher than \( \lambda \), and \( |\hat{I}| + |\hat{J}| \geq (1 - \frac{1}{r})t \).

Let \( \text{LAT}_\lambda \) for a given \( \lambda \) denote the parametric local alignment with length threshold problem which is the maximization of \( s(I, J) - \lambda(|I| + |J|) \) over all subsequences \( I \), and \( J \) of \( X \) and \( Y \) respectively where \( |I| + |J| \geq t \). \( \text{LAT}_\lambda^* \) can be described in terms of \( \text{LAT}^* \) (Proposition 1 [3]).

**Proposition 1** [3] When solving \( \text{LAT}_\lambda \), algorithm \( \text{APX-LAT} \) returns an alignment \((\hat{I}, \hat{J})\) with normalized score higher than \( \lambda > 0 \), and \( |\hat{I}| + |\hat{J}| \geq (1 - \frac{1}{r})t \) if problem \( Q_t \) is feasible.

Solving \( Q_t \) requires a single application of Algorithm \( \text{APX-LAT} \) to solve an \( \text{LAT} \) problem.
8 Approximation Algorithm for Normalized Local Alignment

The approximation algorithm $APX-LAt$ can be applied to solving the parametric problems that arise in computing $NLAt^*$. If we use the approximation algorithm in the binary search algorithm which uses the Megiddo’s technique then we obtain a new algorithm for $NLAt$ problem about which the following theorem can be established (Algorithm $Rational NLAt$ in [3]):

**Theorem 2** [3] 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 $O(rm)$. 

Similarly we can use $APX-LAt$ to solve the parametric problems in Dinkelbach algorithm, and obtain a result with the same approximation guarantees. This Dinkelbach algorithm is implemented for affine gap penalties, and it is tested on $bb-4$ locus in C. elegans and C. briggsae for various values of parameters $t$ and $r$ [3]. The results suggest that the number of iterations is about 3–5 on the average, and show that some long alignments with high total scores include regions with poor scores. Such alignments may overshadow other interesting alignments.

9 Conclusion

We have addressed a number of problems which are variants of local alignment problem. They all involve a length constraint. All of these problems have simple dynamic programming formulations with resulting time complexities that are not practical. For the case of normalized local alignment problem techniques used in ratio optimization offer alternate solutions. One solution uses the fractional programming technique 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 still open. We believe that in this case the existing approximation algorithms are effective. For the other problems there exist simple approximation algorithms which 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.

A number of interesting problems remain to be studied. How many iterations does the Dinkelbach $ANLA$, or $NLAt$ algorithms perform in the worst case? Are there (provable) faster algorithms for these problems based on other techniques such as cutting planes? Would it be effective to solve parametric problems in these algorithms using fast heuristic algorithms such as BLAST, or FASTA? Are there faster exact, or better approximation algorithms for $LRLA$, $LAt$, or $Qi$?
References


