76

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

Abdullah N. Arslan
University of Vermont

Ömer Eğecioğlu
University of California, Santa Barbara

76.1 Introduction

There are interesting algorithmic issues that arise when length constraints are taken into account in the formulation of a variety of problems on string similarity, particularly in the problems related to local alignment. These types of problems have their roots and most striking applications in computational biology. In fact, because of the applications in biological sequence analysis, detection of local similarities in two given strings has become an increasingly important computational problem. When there are additional constraints that need to be satisfied as a part of the search criteria, it is natural to consider approximation algorithms for the resulting computational problems for large parameters.

Given two strings X and Y, the classical dynamic programming solution to the local alignment problem searches for two substrings $I \subseteq X$ and $J \subseteq Y$ with maximum similarity score under a given scoring scheme, where $\subseteq$ indicates the substring relation. This classical definition of similarity has certain anomalies mainly because the lengths of the segments I and J are not taken into account. To cope with the possible anomalies of mosaic and shadow effects, many variations of the local alignment problem have been suggested. Mosaic effect is observed when an unrelated segment is sandwiched between two very similar segments. Shadow effect is observed when a biologically important short alignment is not detected because it overlaps with a longer yet biologically inadequate alignment with only a slightly higher score.

The variations suggested either define new objective functions, or include a length constraint on the substrings I and J for optimal alignments sought. This constraint can be driven by practical considerations for various objective functions (e.g., the maximization of *length-normalized* scores) and can be explicitly given such as requiring $|I| + |J| \geq t$ or $|J| \leq T$ for given parameters t and T. In addition, in some local alignment problems the constraint may also be implicit, as it happens in the case of cyclic sequence

comparison. In Table 76.1 we give a list of local alignment problems, their objectives, and computational results for them. The function $s(I, J)$ denotes the similarity score between I and J. The optimizations are over all possible substrings I of X, and J of Y. In the table, we use "nor. score" as a shorthand for length-normalized score. For any optimization problem $\mathcal{P}$, we denote by $\mathcal{P}^*$ its optimum value, and sometimes drop the parameters from the notation when they are obvious from the context. An optimization problem $\mathcal{P}$ is called *feasible* if it has a solution with the given parameters.

In most cases under consideration, there are simple dynamic programming formulations for the solution of the exact version of a given alignment problem with a length constraint. However, the resulting algorithms require cubic or higher time complexity, which is unacceptably high for practical purposes since the sequence lengths can be on the order of millions. To cope with such high complexity, approximations are considered both in definitions of similarity, and in the resulting computations.

There have been approximation algorithms proposed for various alignment problems with constraints, involving applications of techniques from fractional programming, and dynamic programming. In this chapter, we present a survey of the most interesting approximation algorithms for variations of local alignment problems. Our focus is on fractional programming algorithms, and algorithms returning results that meet the length constraint only partially but guaranteed to be within a given tolerance. These algorithms can be organized into three main categories:

1. *Fractional programming algorithms.* Application of fractional programming on *adjusted normalized local alignment* (the *ANLA* problem in Table 76.1) is of interest. The local alignment is normally defined as a graph problem. The fractional programming technique offers an iterative solution such that at each iteration an ordinary local alignment problem with modified weights is solved. This mimics the action of manually changing the weights until the results are found satisfactory. Fundamental theorems of fractional programming guarantee an optimal solution at the conclusion of these iterations. The termination properties of the iterative scheme are not obvious at all without referring to the results established for fractional programming.
2. *Approximation algorithms for partial constraint satisfaction.* Another noteworthy feature of some constrained local alignment approximation algorithms is their unusual performance measure. Ordinarily, performance of an approximation algorithm is measured by comparing the returned results against optimum value with respect to the objective function. In some approximation results regarding the length-constrained local alignment problems, such as the problem of finding a sufficiently long alignment with high score (the *LAt* problem in Table 76.1), the alignment returned is assured to have at least the score obtainable with respect to the given constraint, but the length constraint is satisfied to only within a prescribed tolerance from the required length value.
3. *Fractional programming approximation algorithms.* There are fractional programming approximation algorithms for the *normalized local alignment* problem with length constraint (the *NLAt* problem in Table 76.1). These algorithms iteratively invoke an approximation algorithm to solve a length-constrained local alignment problem (*LAt*) such that the length constrained is guaranteed to be satisfied within a given tolerance. This length guarantee carries over for the final result for the normalized local alignment problem. That is, the fractional programming algorithm returns an approximate result for which the guarantee on the satisfaction of the length constraint within some tolerance is due to the approximation algorithm used at each iteration, and the criteria is preserved over the iterations.

In this chapter, we start with the basic framework for local alignment in Section 76.2. We present the details of the topics enumerated above in three sections. In Section 76.3 we describe the fractional programming algorithms for the adjusted normalized local alignment problem (*ANLA*). In Section 76.4 we describe an approximation algorithm that uses decomposition of the alignment graph into slabs in order to find a sufficiently long alignment with high score (*LAt*). This algorithm is used to obtain a fractional programming approximation algorithm for the normalized local alignment problem (*NLAt*). This is done in such a way that the length constraint is met within a given tolerance, as described in detail

in Section 76.5. The main results and the algorithm descriptions given in the subsequent sections of this chapter are a compilation and a reorganization of the results that appear in Refs. [1–4].

76.2 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 $0 \leq v \leq m$. Figure 76.1 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 in Figure 76.1. 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. We sometimes use *indel* to refer to an insert or a delete operation. 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 alignment in Figure 76.1 includes two gaps with sizes 1 and 3. 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 given *scoring scheme*. In the *simple scoring scheme*, the arcs of $G_{X,Y}$ are assigned weights determined by nonnegative reals δ (*mismatch penalty*) and μ (*indel* or *gap penalty*). We assume that $s(x_i,\ y_j)$ is the similarity score between the symbols x_i and y_j which is normally 1 for a match $(x_i = y_j)$ and $-\delta$ for a mismatch $(x_i \neq y_j)$.

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. The optimum value $LA^*(X,\ Y)$ for this problem is given by

$$LA^*(X,\ Y) = \max\{s(I,\ J) \mid I \subseteq X,\ J \subseteq Y\} \tag{76.1}$$

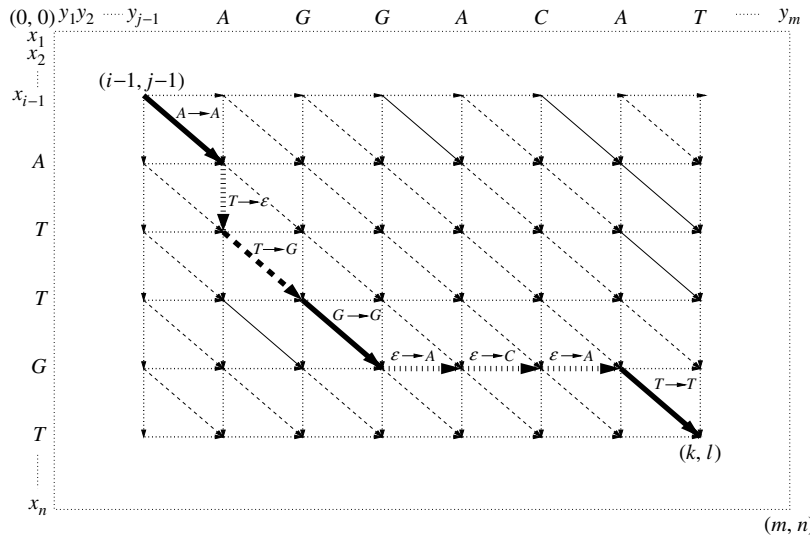

FIGURE 76.1 Alignment graph $G_{X,Y}$ where $x_i \cdots x_k = ATTGT$ and $y_j \cdots y_l = AGGACAT$.

where $s(I, J)$ is the best alignment score between I and J. Alignments have positive scores, or otherwise, they do not exist, that is, $s(I, J) = 0$ iff there is no alignment between I and J.

The following is the classical dynamic programming formulation [5] to compute the maximum local alignment score $\mathcal{S}_{i,j}$ achieved by an optimal local alignment ending at each vertex (i, j):

$$\mathcal{S}_{i,j} = \max\{0,\ \mathcal{S}_{i-1,j} - \mu,\ \mathcal{S}_{i-1,j-1} + s(x_i, y_j),\ \mathcal{S}_{i,j-1} - \mu\} \tag{76.2}$$

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

$$LA^*(X, Y) = \max_{i,j} \mathcal{S}_{i,j} \tag{76.3}$$

LA^* can be computed using the Smith–Waterman algorithm [6] 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 Eq. (76.2).

Affine gap penalties is another common scoring scheme in which the total penalty for a gap of size k, that is, 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 (see Ref. [5]): Let $\mathcal{E}_{i,j} = \mathcal{F}_{i,j} = \mathcal{S}_{i,j} = 0$ when i or j is 0, and define

$$\begin{aligned} \mathcal{E}_{i,j} &= \max\{\mathcal{S}_{i,j-1} - \alpha,\ \mathcal{E}_{i,j-1} - \mu\} \\ \mathcal{F}_{i,j} &= \max\{\mathcal{S}_{i-1,j} - \alpha,\ \mathcal{F}_{i-1,j} - \mu\} \\ \mathcal{S}_{i,j} &= \max\{0,\ \mathcal{S}_{i-1,j-1} + s(x_i, y_j),\ \mathcal{E}_{i,j},\ \mathcal{F}_{i,j}\} \end{aligned} \tag{76.4}$$

By virtue of this formulation, consideration of affine gap penalties does not increase the asymptotic complexity of the local alignment problem.

We can also express the alignment problems as optimization problems that involve linear functions. In the following sections we will describe fractional programming algorithms based on these expressions. 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 substrings $I \subseteq X$ and $J \subseteq Y$ with x matches, y mismatches, and z indels. In Figure 76.1, (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: the basic scoring scheme, arbitrary scoring matrices, and affine gap penalties. For example when simple scoring is used

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

where x,y,z of alignment vector a represent the number of matches, mismatches, and indels, respectively. We can easily verify that also for affine gap penalties and arbitrary scoring matrices, $SCORE$ can be expressed as a linear function.

The local alignment problem LA can be rewritten as follows:

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

76.3 Fractional Programming *ANLA* Algorithms

Using length-normalized scores in local alignment is suggested by Arslan et al. [2] to cope with the mosaic and shadow effects. The objective of the $NLAt$ problem [2] is

$$NLAt^*(X, Y) = \max\{s(I, J)/(|I| + |J|) \mid I \subseteq X,\ J \subseteq Y,\ |I| + |J| \ge t\} \tag{76.5}$$

To solve the *NLAt* problem we can extend the dynamic programming formulation for the scoring schemes that we address in this chapter 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 complexities to $\Theta(n^2 m)$ and $\Theta(nm)$, respectively. These are unacceptably high because in practice the values of both n and m may be on the order of millions.

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

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

where x, y, z represent the number of matches, mismatches, and indels, respectively. We can easily see that for affine gap penalties and arbitrary scoring matrices *LENGTH* can be expressed as a linear function. We assume that only the matches have nonnegative scores; therefore on any alignment the score cannot exceed the length.

The objective of *NLAt* may be achieved by a reformulation. In *adjusted normalized local alignment* (*ANLA*) problem, we can modify the maximization ratio function in such a way that we drop the length constraint, yet achieve a similar objective: to obtain sufficiently long alignments with a high degree of similarity. The adjusted length-normalized score of an alignment is computed by adding some parameter $L \geq 0$ to the denominator in the calculation of the quotient of ordinary scores by the length. Thus the *ANLA* problem [2] is a variant of the 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.

$$ANLA^*(X, Y) = \max\{s(I, J)/(|I| + |J| + L) \mid I \subseteq X, J \subseteq Y\} \tag{76.6}$$

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

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

For *ANLA* faster algorithms are possible using *fractional programming* technique. The provable time complexity of the *ANLA* problem for rational weights is $O(nm \log n)$, as we discuss later. Test results of a fractional programming-based approach suggest that the time complexity is $O(nm)$, although this result is empirical. Compared to $\Theta(n^2 m)$ time complexity of a naive dynamic programming algorithm for the *NLAt* problem, the *ANLA* problem can be solved much faster.

Fractional programming *ANLA* algorithms [2] use the *parametric method*. They iteratively solve a so-called *parametric problem* LA_λ which is the following optimization problem: for a given λ

$$LA^*_\lambda(X, Y) = \max\{s(I, J) - \lambda(|I| + |J| + L) \mid I \subseteq X, J \subseteq Y\} \tag{76.7}$$

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

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

Proposition 76.1 **(Arslan et al. [2])**

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

$$LA^*_{\delta,\mu}(\lambda) = (1 - 2\lambda) LA^*_{\delta',\mu'} - \lambda L, \quad \text{where} \quad \delta' = \frac{\delta + 2\lambda}{1 - 2\lambda},\ \mu' = \frac{\mu + \lambda}{1 - 2\lambda} \tag{76.8}$$

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. □

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

FIGURE 76.2 `Dinkelbach` algorithm for *ANLA* [2].

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 $L = 0$.

The thesis of the parametric method of fractional programming is that the optimum solution to the original problem that involves a ratio of two functions can be obtained via optimal solutions of the parametric problem. In this case, an optimal solution to a ratio optimization problem *ANLA* can be achieved via a series of optimal solutions of the parametric problem $LA(\lambda)$ with different parameters λ. In fact $\lambda = ANLA^*$ iff $LA^*(\lambda) = 0$. That is, an alignment vector $v \in AV$ has the optimum adjusted normalized score λ iff v is an optimal alignment vector for the parametric problem $LA(\lambda)$ with optimum value zero. (See Ref. [2] for more details, also see Refs. [7,8] for many interesting properties of fractional programming). The *Dinkelbach* algorithm for the *ANLA* problem is shown in Figure 76.2. Solutions of the parametric problems through the iterations yield improved (higher) values to λ except for the last iteration in which λ remains the same, and becomes the optimum value. In fractional programming algorithms convergence to an optimal result is guaranteed: In infinite sets the convergence to optimum is super-linear. In finite sets the termination is guaranteed. In the case of *ANLA* $Dinkelbach$ algorithm, when the algorithm terminates, the final alignment is optimal with respect to both the ordinary scores used at that iteration, and the adjusted length-normalized scoring with the original scores. This mimics manually changing the scores until the result is satisfactory.

As reported by Arslan et al. [2], experiments suggest that the number of iterations in the algorithm is a small constant: 3–5 on average. However, a theoretical bound is yet to be established. If we assume that the sequences involved in alignments are fixed (e.g., 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, e.g., Ref. [1]). Even this rough estimate shows that the algorithm in the worst case is better than the straightforward dynamic programming extension for *ANLA*.

In practice the scores are rational, and in the case of rational scores there is a provably better result [2] which is achieved by Algorithm *RationalANLA* given in Figure 76.3. The algorithm uses Megiddo's technique [9] to perform a binary search for optimum adjusted normalized score over an interval of

```
Algorithm RationalANLA
Let σ be the smallest gap between two adjusted length normalized scores
Initialize [e, f] ← [0, ½σ^−1]
While (e + 1 < f) do
   k ← ⌊(e + f)/2⌋
   If LA*(kσ) > 0 then e ← k else f ← k
End {while}
Return(eσ)
```

FIGURE 76.3 *ANLA* algorithm `RationalANLA` for rational scores [2].

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*. In contrast, if $LA^*(\lambda) > 0$, then a larger λ, and if $LA^*(\lambda) < 0$, then 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 score 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 $O(m)$.

76.4 Approximation Algorithms for Partial Constraint Satisfaction

In Table 76.1 we list several local alignment problems with length constraint. For these problems there are approximation algorithms that guarantee the satisfaction of the constraints partially, that is, they return alignments whose lengths are within a given tolerance of the required length.

These algorithms decompose the alignment graph into slabs. The length-restricted local alignment *LRLA* problem [3] is suggested to find alignments with optimal score over the alignments that involve substrings of up to a given length. The length limit is only on the substrings of one of the strings. The approximation algorithms for this problem imagine that the alignment graph is partitioned into vertical slabs. The results for this problem are summarized in Table 76.1. The cyclic sequence alignment *CLA* [3] is a special case of the *LRLA* problem. In the *CLA* problem the length constraint is implicit as shown in the table. The *LRLA* algorithms and results are applicable to the *CLA* problem, too.

We omit the details of *LRLA* approximation algorithms. Instead we describe another algorithm which is also based on the decomposition of the alignment graph into slabs. This algorithm is for the length-constrained local alignment problem *LAt* [4] (see also Table 76.1).

TABLE 76.1 Variations of Local Alignment Problems [4]

Alignment problem	Objective	Algorithm	Time	Space	Returned alignment satisfies
LA	maximize $s(I, J)$	Smith–Waterman	$O(nm)$	$O(m)$	Score $= LA^*$
ANLA	maximize $\frac{s(I,J)}{\|I\|+\|J\|+L}$ for parameter $L \geq 0$	Dinkelbach	$O(nm)$ (experimental)	$O(m)$	Score $= ANLA^*$
		Rational*ANLA*	$O(nm \log n)$	$O(m)$	Score $= ANLA^*$
LRLA	maximize $s(I, J)$ such	*HALF*	$O(nm)$	$O(m)$	Score $\geq \frac{1}{2} LRLA^*$
	that $\|J\| \leq T$	*APX-LRLA*	$O(nmT/\Delta)$	$O(mT/\Delta)$	Score $\geq LRLA^* - 2\Delta$
CLA	*LRLA* with parameters X, YY, and $T = \|Y\|$	The same *LRLA* algorithms, complexity, and results			
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 $\geq (1 - \frac{1}{r})t$

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

$$LAt^*(X, Y) = \max\{s(I, J) \mid I \subseteq X, J \subseteq Y, \text{ and } |I| + |J| \geq t\} \tag{76.9}$$

Equivalently

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

where $AVt \subseteq AV$ is the set of alignment vectors corresponding to alignments with length $\geq t$.

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, which is a practical query problem Qt included in Table 76.1. We also show that the algorithm for the local alignment with length threshold leads to improved approximation algorithms for the normalized local alignment problem (see Section 76.5).

To solve LAt we can extend the dynamic programming formulation in Eq. (76.2) 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 complexities to $\Theta(n^2 m)$ and $\Theta(nm)$, respectively.

We describe an approximation algorithm *APX-LAt* [4] which 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, that is the algorithm finds two substrings $\widehat{I} \subseteq X$, and $\widehat{J} \subseteq Y$ 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 the simple scoring scheme. 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 76.4.

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 2 (each match, or mismatch involves two symbols, one from each sequence), while that of each horizontal or vertical arc is 1 (each indel involves one symbol from one of the sequences). Equivalently we say that the length of a diagonal arc is 2, and the length of each horizontal, or vertical arc is 1. 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 which 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

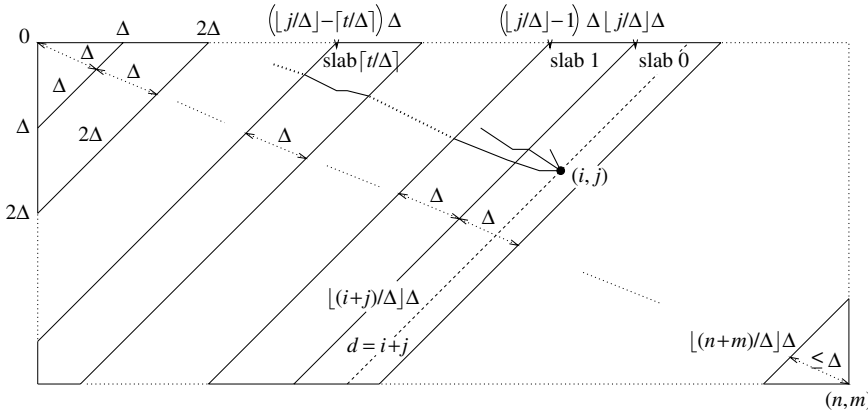

FIGURE 76.4 Slabs with respect to diagonal d and alignments ending at node (i, j) starting at different slabs.

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.

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 76.4 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 $0 \leq k < \lceil t/\Delta \rceil$. 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. We store at each node $(i,\ j)$, $\lceil t/\Delta \rceil + 1$ score–length pairs $(\mathcal{S}_{i,j,k}, \mathcal{L}_{i,j,k})$ for $0 \leq k \leq \lceil t/\Delta \rceil$ corresponding to $\lceil t/\Delta \rceil + 1$ optimal alignments that end at $(i,\ j)$. Figure 76.5 shows the steps of the 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 Eq. (76.2). Let maxp of a list of score–length pairs be a pair with the maximum 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

```
Algorithm APX-LAt(δ, μ)
1. Initialization:  set LAt^ = 0; and (S_{0,j,k}, L_{0,j,k}) = (0, 0) for all j, k,  0 ≤ j ≤ m, and 0 ≤ k ≤ ⌈t/Δ⌉
2. Main computations:
  for i = 1 to n do {
    set (S_{i,0,k}, L_{i,0,k}) = (0, 0) for all k, 0 ≤ k ≤ ⌈t/Δ⌉
    for j = 1 to m do {
      if (i + j mod Δ = 1) then {
        set (S_{i,j,0}, L_{i,j,0}) = (0, 0)
        for k = 1 to ⌈t/Δ⌉ − 1 do
2.a.1     set (S_{i,j,k}, L_{i,j,k}) = maxp{ (0, 0),  (S_{i−1,j,k−1}, L_{i−1,j,k−1}) + (−μ, 1),
                                             (S_{i−1,j−1,k−1}, L_{i−1,j−1,k−1}) ⊕ (s(x_i, y_j), 2),
                                             (S_{i,j−1,k−1}, L_{i,j−1,k−1}) + (−μ, 1) }
        for k = ⌈t/Δ⌉
2.a.2     set (S_{i,j,k}, L_{i,j,k}) = maxp{ (0, 0),  (S_{i−1,j,k−1}, L_{i−1,j,k−1}) + (−μ, 1),
                                             (S_{i−1,j−1,k−1}, L_{i−1,j−1,k−1}) ⊕ (s(x_i, y_j), 2),
                                             (S_{i,j−1,k−1}, L_{i,j−1,k−1}) + (−μ, 1), (S_{i−1,j,k}, L_{i−1,j,k}) + (−μ, 1),
                                             (S_{i−1,j−1,k}, L_{i−1,j−1,k}) ⊕ (s(x_i, y_j), 2), (S_{i,j−1,k}, L_{i,j−1,k}) + (−μ, 1) }
      } else {
          for k = 0 to ⌈t/Δ⌉ do
2.b           set (S_{i,j,k}, L_{i,j,k}) = maxp{ (0, 0),  (S_{i−1,j,k}, L_{i−1,j,k}) + (−μ, 1),
                                                 (S_{i−1,j−1,k}, L_{i−1,j−1,k}) ⊕ (s(x_i, y_j), 2), (S_{i,j−1,k}, L_{i,j−1,k}) + (−μ, 1) }
        }
        for k = ⌈t/Δ⌉ − 1 if L_{i,j,k} ≥ t − Δ then set LAt^ = max{LAt^, S_{i,j,k}}
        for k = ⌈t/Δ⌉ set LAt^ = max{LAt^, S_{i,j,k}}
    } }
3. Return LAt^
```

FIGURE 76.5 Algorithm *APX-LAt* [4].

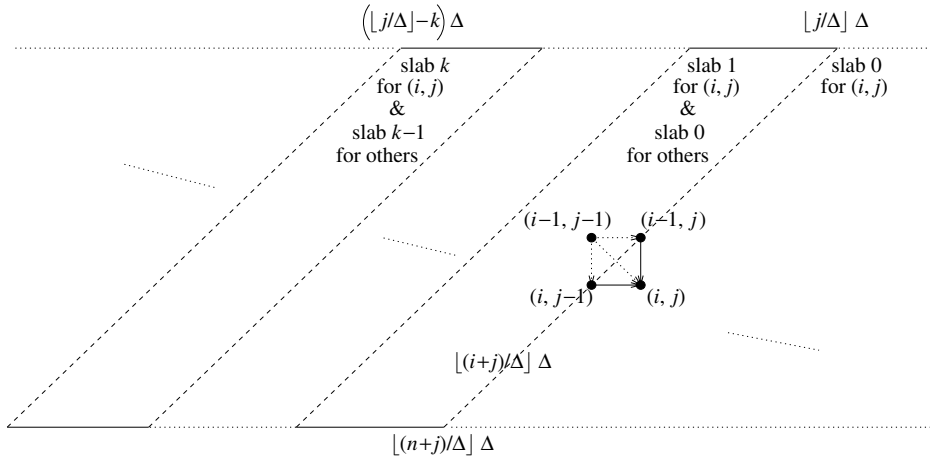

FIGURE 76.6 Relative numbering of the slabs with respect to $(i,\ j)$, $(i-1,\ j)$, $(i-1,\ j-1)$, and $(i,\ j-1)$ when node $(i,\ j)$ is on the first diagonal following boundary $\lfloor (i+j)/\Delta \rfloor$.

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. 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, a local alignment must have a positive score to exist, 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 zero score be extended. A new alignment starts with a single match in the current slap.

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 76.6. 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 only include 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 a start and end position information besides 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 $0 \leq k < \lceil t/\Delta \rceil$ and (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

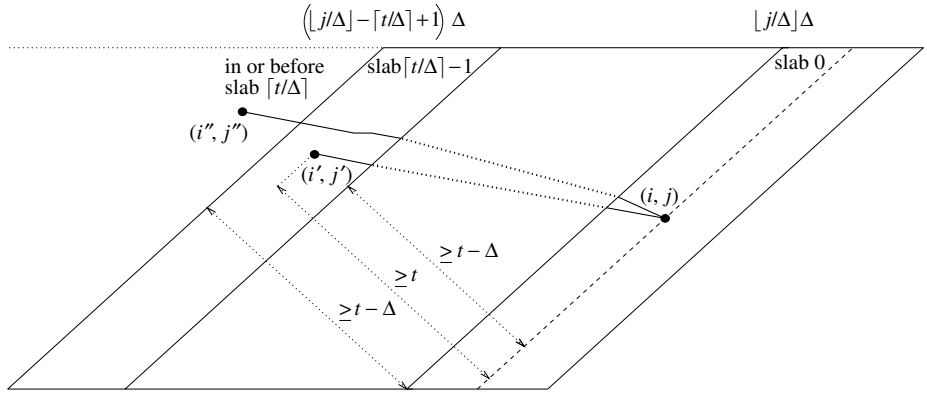

FIGURE 76.7 Two possible orientations of an optimal alignment of length $\geq t$ ending at (i, j): It starts either at some (i', j') at slab $\lceil t/\Delta \rceil - 1$, or (i'', j'') in or before slab $\lceil t/\Delta \rceil$.

of an optimal alignment as shown in Figure 76.7: (1) It starts at some node (i', j') of slab $k = \lceil t/\Delta \rceil - 1$. By a 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 76.1 (Arslan and Eğecioğlu [4])

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, $1 < r \leq t/2$. *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 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 $\Delta = \lfloor t/r \rfloor$ for a choice of r, $1 < r \leq t/2$, 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: At each entry of the dynamic programming matrix, instead of a single score a number of scores (and lengths) are maintained and manipulated as dictated by the underlying dynamic programming formulation (e.g., Eq. [76.4]).

An application of the *LAt* problem is on problem *Qt* which is defined as the problem of finding two subsequences with normalized score higher than λ, and total length at least t. More formally

$$Qt: \text{ find } (I, J) \text{ such that } I \subseteq X,\ J \subseteq Y,\ \frac{s(I, J)}{|I| + |J|} > \lambda \quad \text{and} \quad |I| + |J| \geq t \tag{76.10}$$

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

The problem is feasible for given thresholds t, and $\lambda > 0$, if the answer to this query is not empty, that is, there exists a pair of subsequences I and J with total length $|I| + |J| \geq t$, and normalized score $s(I, J)/(|I| + |J|) > \lambda$. Note that Qt is feasible iff $NLAt^* > \lambda$. We describe an algorithm which 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 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 follows:

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

Proposition 76.2 (Arslan and Eğecioğlu [4])

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

$$LAt^*_{\delta,\mu}(\lambda) = (1 - 2\lambda) LAt^*_{\delta',\mu'}, \text{ where } \delta' = \frac{\delta + 2\lambda}{1 - 2\lambda}, \; \mu' = \frac{\mu + \lambda}{1 - 2\lambda} \tag{76.11}$$

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 which can only happen when the alignment is composed of matches only.

Proposition 76.3 (Arslan and Eğecioğlu [4])

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 which solves the corresponding LAt problem (of Proposition 76.2) 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*.

76.5 Normalized Local Alignment

Need for a length constraint is clear when length-normalized scores are used because shorter alignments may have high normalized scores but they may not be biologically significant. The definition of the *NLAt* problem contains a length constraint as described in Section 76.3.

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

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

```
Algorithm APX-RationalNLAt
If there is an exact match of size (1−1/r)t then return(1/2) and exit
Let σ be the smallest gap between two length-normalized scores
[e, f] ← [0, 1/2 σ^−1]
λ* ← 0
While (e + 1 < f) do
  k ← ⌈(e + f)/2⌉
  APX-LAt*(kσ) > 0 then {
    e ← k
    λ* ← the normalized score of an optimal alignment obtained
  } else f ← k
End {while}
Return(λ*)
```

FIGURE 76.8 Algorithm *APX-RationalNLAt* for rational scores [4].

We present next approximation algorithms for the *NLAt* problem that apply fractional programming in which we use Algorithm *APX-LAt* as a subroutine. The approximation is in the sense that the length constraint is partially satisfied. These algorithms are the *Dinkelbach* algorithm for *NLAt*, and Algorithm *RationalNLAt*. Both algorithms obtain an alignment whose score is no smaller than the optimum score $NLAt^*$ 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 (Theorem 76.2). The Dinkelbach algorithm for *NLAt* (Figure 76.9) and Rational*NLAt* (Figure 76.8) are similar to the corresponding *ANLA* algorithms except that they iteratively solve *LAt* problems presented in Section 76.4 instead of *LA* problems. The approximation algorithm *APX-LAt* can be applied to solving the parametric problems that arise in computing $NLAt^*$.

In both resulting algorithms the space complexity is $O(rm)$. The observed time complexity of the *Dinkebach* algorithm for *NLAt* is $O(rnm)$ (in tests [4], it performs always smaller than 10, and on average 3–5 invocations to Algorithm *APX-LAt*). Algorithm *RationalNLAt* has proven time complexity $O(rnm \log n)$ since in this algorithm $O(\log n)$ invocations of *APX-LAt* is sufficient to solve the *NLAt* problem.

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:

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

If we apply the 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 76.2 (Arslan and Eğecioğlu [4])

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, $1 < r \leq t/2$ in time $O(rnm \log n)$ and space $O(rm)$.

Proof

Algorithm *Rational NLAt* given in Figure 76.8 accomplishes this. The algorithm is 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 76.3, 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 76.3, based on Megiddo's search technique [9]. The difference is that the parametric alignment

```
Algorithm Dinkelbach
If APX-LAt*(0) > 0 then set λ* to the length-normalized score of an optimal alignment else exit
Repeat
   λ ← λ*
   If APX-LAt*(λ) > 0 then set λ* to the length-normalized score of an optimal alignment
Until λ* ≤ λ
Return(λ*)
```

FIGURE 76.9 Dinkelbach algorithm for *NLAt* [4].

problems now have a length constraint. The algorithm computes the smallest possible gap σ between any two distinct possible normalized scores, which is $\Omega(1/(n+m)^2)$ [2]. 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 0, 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 76.1 and Proposition 76.3 in Section 76.4 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.

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 the Dinkelbach algorithm given by Arslan et al. [2] as the template. The details of the resulting algorithm appear in Figure 76.9. 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 76.1 and Proposition 76.3 in Section 76.4 since $\lambda < NLAt^*$. Solutions of the parametric problems through the iterations yield improved (higher) values to λ except for the last iteration. The resulting algorithm performs no more than 3–5 iterations on average, and never more than 9 in the worst case in tests [4]. When the algorithm terminates, the optimal alignment whose length-normalized score is λ^* has the total length at least $(1 - \frac{1}{r})t$ and $\lambda^* \geq NLAt^*$.

76.6 Discussion

We would like to point out the relation between the normalized local alignment and a problem known as *parametric sequence alignment* [10] (which is different from the parametric local alignment problem we discuss in this chapter) in the literature. The fractional programming-based *ANLA* and *NLAt* 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 both with respect to ordinary scores at the last iteration and with respect to length-normalized scores with the original scores). It has been 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 [11]: 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 the four parameters are variables

then there is no known reasonable upper bound on the number of polygons. When the alignment is global, and no character-specific scoring matrices are used the number of polygons is bounded from above by $O(n^{2/3})$ [12].

We also remark that to find long regions with high degree of similarity we may also 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 [13] seems a natural candidate. This solution requires adding extra edges to cause cycles of minimum certain 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 the naive dynamic programming solution.

We conclude by stating a few open problems for further study:

How many iterations do the Dinkelbach ANLA or NLAt algorithms perform in the worst case?

Are there (provably) faster exact or better approximation algorithms for the NLA, LRLA, LAt, or Qt problems?

References

[1] Arslan, A. N. and Eğecioğlu, Ö., An improved upper bound on the size of planar convex-hulls, *Proc. of COCOON*, Lecture Notes in Computer Science, 2108, 2001, p. 111.

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

[3] Arslan, A. N. and Eğecioğlu, Ö., Approximation algorithms for local alignment with length constraints, *Int. J. Found. Comput. Sci.*, 13(5), 751, 2002.

[4] Arslan, A. N. and Eğecioğlu, Ö., Dynamic programming based approximation algorithms for local alignment with length constraints, *INFORMS J. Comput., Special Issue on Comput. Mol. Biol./Bioinf.*, 16(4), 441, 2004.

[5] Waterman, M. S., *Introduction to Computational Biology*, Chapman & Hall/CRC, New York, 1995.

[6] Smith, T. F. and Waterman, M. S., The identification of common molecular subsequences, *J. Mol. Biol.*, 147(1), 195, 1981.

[7] Craven, B. D., *Fractional Programming*, Helderman Verlag, Berlin, 1988.

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

[9] Megiddo, N., Combinatorial optimization with rational objective functions, *Math. Oper. Res.*, 4, 414, 1979.

[10] Fitch, W. M. and Smith, T. F., Optimal sequence alignments, in *Proc. Natl. Acad. Sci.*, 80, 1983, 1382.

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

[12] Gusfield, D., Balasubramanian, K. and Naor, D., Parametric optimization of sequence alignment, *Algorithmica*, 12, 312, 1994.

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