Genomics & Informatics Vol. 1(1) 25-31, September 2003

A Heuristic Algorithm to Find All Normalized Local Alignments Above Threshold

Sangtae Kim[1], Jeong Seop Sim[2], Heejin Park[3], Kunsoo Park[2,3]*, Hyunseok Park[4,5] and Jeong-Sun Seo[4,6]

[1]Department of Computer Science, Korea Military Academy, Seoul, Korea
[2]Electronics and Telecommunications Research Institute, Daejeon, Korea
[3]School of Computer Science and Engineering, Seoul National University, Seoul, Korea
[4]Institute of Bioinformatics, Macrogen, Inc., Seoul, Korea
[5]Department of Computer Science, Ewha Womans University, Seoul, Korea
[6]Ilcheon Molecular Medicine Institute, Seoul National University, Seoul, Korea

Abstract

Local alignment is an important task in molecular biology to see if two sequences contain regions that are similar. The most popular approach to local alignment is the use of dynamic programming due to Smith and Waterman, but the alignment reported by the Smith-Waterman algorithm has some undesirable properties. The recent approach to fix these problems is to use the notion of normalized scores for local alignments by Arslan, Egecioglu and Pevzner. In this paper we consider the problem of finding all local alignments whose normalized scores are above a given threshold, and present a fast heuristic algorithm. Our algorithm is 180-330 times faster than Arslan *et al.*'s for sequences of length about 120 kbp and about 40-50 times faster for sequences of length about 30 kbp.

Keywords: local alignment, dynamic programming, normalized score, fractional programming

Introduction

Sequence alignment is a fundamental task in molecular biology to see if two sequences are related. The local alignment problem for two sequences is to find a pair of similar regions, one from each sequence. In many biological applications local alignment is more useful than global alignment because two sequences may not be similar as a whole, but they can contain small regions that are very similar.

A most popular approach to local alignment is the use of dynamic programming due to Smith and Waterman (Smith and Waterman, 1981; Waterman *et al.,* 1995). The Smith-Waterman algorithm finds a pair of regions whose alignment score is the highest (which is called the optimal local alignment). However, the alignment reported by the Smith-Waterman algorithm has some undesirable properties. The alignment may include regions whose similarity is very poor (Arslan *et al.,* 2001; Zhang *et al.,* 1999). Also, the Smith-Waterman algorithm does not consider the lengths of local alignments in computing scores, and therefore a biologically important but short alignment may not be detected(Arslan and Pevzner *et al.,* 2001).

There have been several approaches to fix the problems of the Smith-Waterman algorithm (Goad and Kanehisa, 1982; Sellers, 1984; Zhang *et al.,* 1998; Zhang *et al.,* 1999; Arslan and Pevzner *et al.,* 2001). For example, Zhang *et al.* (Zhang *et al.,* 1998; Zhang and Miller *et al.,* 1999) proposed an approach based on the notion of X-drop, a region that scores below $-X$ for some fixed X>0. They consider only alignments that contain no X-drop. A more recent approach to fix the problems is the use of normalized scores for local alignments (Marzal and Vidal 1993; Arslan and Ö. Eğecioğlu (1999); Arslan *et al.,* 2001). Arslan *et al.* (2001) used fractional programming to obtain the optimal normalized local alignment of two sequences. They also extended their solution to find all local alignments whose normalized scores are larger than a given threshold by running their solution repeatedly after masking those alignments that are already found.

In this paper we consider the problem of finding *all* local alignments whose normalized scores are above a given threshold. A local alignment whose normalized score is above a threshold will be called a *TNL alignment.* Since small regions are reported in the local alignment problem, finding *all* of TNL alignments can be more appropriate than just finding a single (optimal) local alignment in applications such as gene prediction (Gusfield, 1997). We present a fast algorithm for the problem of finding all the TNL

* Corresponding author: E-mail kpark@theory.snu.ac.kr

Accepted 26 March 2003

alignments. Our algorithm is based on fractional programming and the banded Smith-Waterman algorithm. The fractional programming approach uses the Smith-Waterman algorithm repeatedly to obtain one local alignment. Arslan *et al.* (Arslan *et al.,* 2001) use this solution repeatedly to find all the TNL alignments. Hence, Arslan *et al.*' s algorithm makes a double repetition of the Smith-Waterman algorithm, which makes it very slow in practice. Our algorithm first makes a careful selection of bands based on several parameters so that the selected bands can only contain TNL alignments with high probability. Then it runs the Smith-Waterman algorithm on the selected bands in such a way that the number of applications of the Smith-Waterman algorithm on each band is much smaller than that of Arslan *et al.*' s.

Experiments show that our algorithm is about 180-330 times faster than Arslan *et al.*' s for the data set of human chromosome X cosmids Qc8D3 (GenBank Acc. No. AF030876 (113 kbp)) and mouse chromosome X clone BAC B22804 (GenBank Acc. No. AF121351 (123 kbp)) and it is about 40-50 times faster for the data set of human (GenBank Acc. No. L10347 (31 kbp)) and mouse (GenBank Acc. No. M65161 (32.4 kbp)) pro-alpha1 type the benefit of II collagen, while finding all TNL alignments. In addition to extremely fast speed, our algorithm can change the parameters for selecting bands to adjust the time and accuracy of our algorithm.

Preliminaries

A *string* or a *sequence* is concatenations of zero or more characters from an alphabet Σ. A space is denoted by $\Delta \notin \Sigma$; we regard Δ as a character for convenience. The length of a string a is denoted by $|a|$. Let a_i denote *i*th character of a string a and $a(i, j)$ denote a substring $a_i a_i+1 \cdots a_j$ of a. When a sequence α is a substring of a sequence a, we denote it by $\alpha \langle a$. Given two strings $a=a_1a_2\cdots a_m$ and $b=b_1b_2\cdots b_n$, an *alignment* of a and b is $a^*=a_1^*a_2^*\cdots a_l^*$ and $b^*=b_1^*b_2^*\cdots b_l^*$ constructed by inserting zero or more Δ's into a and b so that each a_i^* maps to b_i^* for $1 \leq i \leq I$. There are three kinds of mappings in a^* *and* b^* according to the characters of and a_i^* *and* b_i^*.

· *match* : $a_i^*=b_i^* \neq \Delta$,
· *mismatch* : $(a_i^* \neq b_i^*)$ and $(a_i^*, b_i^* \neq \Delta)$,
· *insertion* or *deletion* (*indel* for short) : either a_i^* or b_i^* is Δ.

Note that we do not allow the case of $a_i^*=b_i^*=\Delta$.

If there exists an alignment of a and b whose number of matches, mismatches and indels are x y and z, respectively, then we call the triplet (x, y, z) an *alignment vector* of a and b. We define the *alignment score* of an alignment a^* and b^* with alignment vector (x, y, z) by score$(x, y, z) = x - \delta y - \mu z$ that is, we assume that a match score is 1, a mismatch penalty is δ and an indel penalty is μ. We define the *similarity* of two sequences a and b, denoted by $S(a, b)$, to be the highest alignment score of all possible alignments of a and b.

Local Alignments

Given two sequences a and b, a *local alignment* of a and b is an alignment of two strings α and β such $\alpha \langle a$ and $\beta \langle b$, and the *optimal local alignment* of a and b is the local alignment of a and b that has the highest similarity among all local alignments of a and b. We denote the similarity of an optimal local alignment by $SL(a, b)$.

A most well-known algorithm to find an optimal local alignment was given by Smith and Waterman, which is known as the *Smith-Waterman algorithm* (SW algorithm for short) (Smith and Waterman, 1981; Waterman, 1995). Given two sequences $a(|a|=m)$ and $b(|b|=n)$, the SW algorithm computes $SL(a, b)$ using a dynamic programming table (called the *H table*) of size $(m+1)(n+1)$. Let $H_{i,j}$ for $0 \leq i \leq m$ and $0 \leq j \leq n$ denote $SL(a(1, i), b(1, j))$. Then, $H_{i,j}$ can be computed by the following recurrence:

$$H_{i,0} = H_{0,j} = 0 \quad (0 \leq i \leq m, 0 \leq j \leq n)$$

$$H_{0,j} = \max\{0, H_{i-1,j-1} + s(a_i, b_j), H_{i,j-1} - \mu, H_{i-1,j} - \mu\}$$

$$s(a_i, b_j) = \begin{cases} 1 & \text{if } a_i = b_j \\ -\delta & \text{if } a_i \neq b_j \end{cases}$$

Among all $H_{i,j}$ for $0 \leq i \leq m$ and $0 \leq j \leq n$, the highest value is $SL(a, b)$, and the SW algorithm takes $O(mn)$ time to compute it.

Normalized Local Alignments

Since the SW algorithm does not consider the lengths of local alignments, the solution of the SW algorithm may include some regions with very low similarity. Moreover, a biologically important short alignment may not be detected (Arslan and Pevzner *et al.,* 2001; Alexandrov and Solovyev, 1998; Zhang and Miller *et al.,* 1999). To handle these problems, the notion of normalization for similarity has been introduced (Arslan Ö. Eğecioğlu (1999), Arslan *et al.,* 2001; Eğecioğlu and Ibel, 1996; Marzal and Vidal, 1993; Alexandrov and Solovyev, 1998).

Given two sequences a and b, we define the *normalized similarity* of a and b by, $S(a, b)/(|a|+|b|+L)$, where L is a constant. From this definition we can define a normalized alignment score for local alignment: the *normalized alignment score* of an alignment a^* and b^* with alignment vector (x, y, z) is defined by nscore $(x, y, z) = \frac{\text{score}(x, y, z)}{2x+2y+z+L}$ (Arslan and Pevzner *et al.,* 2001). The *optimal normalized local alignment* of a and b is the local alignment of a and b that has the highest normalized alignment score.

Here we consider two problems related to normalized local alignment: one is to find the optimal normalized local alignment of a and b; the other is to find all normalized local alignments of a and b above some threshold T_s.

Optimal Normalized Local Alignment

Given two sequences a and b, let $AV(a, b)$ be the set of all possible alignment vectors of $\alpha\langle a$ that $\beta\langle b$. The problem of finding an optimal local alignment (*LA problem for short*) and that of finding an optimal normalized local alignment (*NLA problem* for short) are defined as follows:

LA problem : Find $(x, y, z) \in AV(a, b)$ such that score(x, y, z) is greatest in $AV(a, b)$.

NLA problem : Find $(x, y, z) \in AV(a, b)$ such that nscore(x, y, z) is greatest in $AV(a, b)$.

Note that we can find the score of an optimal local alignment using the SW algorithm but with a simple modification of storing the starting position of each score, we can also find the location of the optimal local alignment. For a given λ, we define the *parametric local alignment problem* as follows:

LA(λ) problem : For all $(x, y, z)\in AV(a, b)$, find the maximal value of $x-\delta y - \mu z -\lambda (2x+2y+z+L)$

A parametric local alignment problem can be converted into a local alignment problem because the optimal solution of LA(λ) involves solving a local alignment problem and performing some simple computations (Arslan and Pevzner *et al.*, 2001).

We can use Dinkelbach' s algorithm to solve the NLA problem. Dinkelbach developed a general algorithm that uses *fractional programming* (Dinkelbach, 1967). The optimal solution to NLA can be obtained via a series of optimal solutions of LA(λ) for different λ' s using the following result:

λ is the optimal solution for NLA if and only if LA(λ)=0.

Dinkelbach's algorithm (Dinkelbach, 1967) (See Algorithm 1) starts with an initial value for λ and repeatedly solves LA(λ). Since the LA(λ) problem can be converted into an LA problem, it solves the LA problem and obtains an optimal alignment vector (x, y, z). Next, it sets λ= nscore(x, y, z) and repeats this process until λ is not changed any more. The convergence to the optimal solution of the NLA problem is guaranteed (Arslan *et al.*, 2001).

All Normalized Local Alignments Above a Threshold

In some biological applications, we need to find all local alignments whose normalized alignment scores are above a given threshold. Formally, we define this problem as follows:

TNLA problem : For given a threshold value T_s, find all $(x, y, z) \in AV(a, b)$ such that nscore(x, y, z) $\geq T_s$.

Algorithm 1 Dinkelbachs algorithm

```
1. Choose an arbitrary alignment vector $(x, y, z) \in AV(a, b)$
2. Set $\lambda^* = \frac{x-\delta y-\mu z}{2x+2y+z+L}$
3. Repeat
4. $\lambda = \lambda^*$
5. Convert the problem into the LA($\lambda$) problem and obtain $(x, y, z)$
6. $\lambda^* = \frac{x-\delta y-\mu z}{2x+2y+z+L}$
7. Until $\lambda = \lambda^*$
8. Return $\lambda^*$
```

A local alignment whose normalized alignment score is above a given threshold will be called a *TNL alignment*. Therefore, the TNLA problem is to find all the TNL alignments.

For the TNLA problem, Arslan *et al.* (2001) first solve the NLA problem and then mask the solution to repeatedly find the next optimal solution of the NLA problem.

Banded Smith-Waterman Algorithm

In general, it is highly likely that an optimal local alignment has a long part of exact matches in it. The *banded Smith-Waterman algorithm* uses this phenomenon and finds a solution very quickly for the LA problem with high probability (Setubal and Meidanis, 1997). Many biological systems such as Phrap (Green), STROLL (Chen and Skiena, 1997) and FASTA (Lipman and Pearson, 1998) are based on this algorithm.

The banded SW algorithm consists of the following two steps:

1. **Determine bands**
   Find all the bands that can have local alignments with high similarity. Usually, a band is defined by the information of exact matches and if two or more bands overlap, they are merged into one band.
2. **Run the SW algorithm**
   Run the SW algorithm on the defined bands. The entries of $H_{i,j}$ outside the bands are assumed 0 and they are not computed.

Methods

We first present an algorithm for finding an optimally normalized local alignment (the NLA problem) and then present algorithms for finding all local alignments whose normalized alignment scores are above a given threshold T_s (the TNLA problem). Our algorithm for the NLA problem is used as a subroutine in our algorithms for the TNLA problem.

Optimal Normalized Local Alignment

We present an algorithm for the NLA problem of two strings $a(|a|=m)$ and $b(|b|=n)$. Our algorithm is based on Dinkelbach' s algorithm which is a general algorithm to solve the NLA problem. We use the banded SW algorithm to solve the LA problem in Dinkelbach' s algorithm. Since we already described Dinkelbach' s algorithm and the banded SW algorithm, we concentrate on describing how to determine bands.

We first introduce some definitions before we describe how to determine the bands. Consider the H table generated by the SW algorithm in computing $SL(a, b)$. The ith-diagonal (or diagonal i), $0 \leq i \leq m$(resp. $-n \leq i \leq -1$), represents a diagonal that passes through (0, i) (resp. ($-i$, 0)) element of the H table. *A band* is a set of diagonals and the *width* of a band is the number of diagonals in the band. *A band i of width k* is a set of diagonals from i through $i+k-1$. We say that an exact *match* (i, j) occurs in diagonal k if $a(i, j)= b(i+k,j+k)$. Exact match (i, j) is *maximal* if $a_{i-1} \neq b_{i+k-1}$ and $a_{i+1} \neq b_{j+k+1}$. The *length* of exact match (i, j) is $j-i+1$.

We show how to determine the bands with given parameters T_l, w, and T_b. The parameter T_l is a threshold on the length of an exact match, w is the width of a band, and T_b is a threshold on the weight of a band. Determining the bands is composed of the following four steps:

1. Find all maximal exact matches longer than T_l in each diagonal. We first generate all suffixes of a and b and sort them in lexicographic order. Once all the suffixes of a and b are sorted, it is easy to find all maximal exact matches longer than T_l.
2. Compute the weight of each diagonal, where the weight of a diagonal is the sum of the lengths of all maximal exact matches longer than T_l in the diagonal.
3. Select every band b of width w whose weight is above T_b, where the weight of a band is the sum of the weights of all diagonals in the band.
4. Merge two bands $B_1=\{d_i, \cdots, d_j\}$ and $B_2=\{d_k, \cdots, d_l\}$ into a single band $B=\{min(d_i, d_k), \cdots, max(d_j, d_l)\}$, if they overlap (*i.e.*, $i \leq k \leq j$ *or* $i \leq l \leq j$). Repeat merging until no two bands overlap. (Fig. 1)

Parameters T_l, w, and T_b should be chosen appropriately so that an optimal normalized local alignment does not lie out of the bands. In Section 4, we suggest appropriate values for T_l, w, and T_b through experiments.

Our algorithm for the NLA problem first determines the bands as explained above and then runs the SW algorithm on the selected bands. The speedup achieved by our algorithm over Arslan *et al.*' s is due to the following two improvements.

· We uses the banded SW algorithm to solve the LA problem of Dinkelbach' s algorithm while Arslan *et al.* used the SW algorithm. Thus, the speedup achieved by the banded SW algorithm over the SW algorithm is reflected in our algorithm.

We perform the step of determining bands in the banded SW algorithm only once while we perform the banded SW algorithm several times in Dinkelbach' s algorithm. Since the bands are the sets of diagonals where long exact matches occur, the bands in all repetitions of the banded SW algorithm are the same. Hence, we need to determine the bands only once.

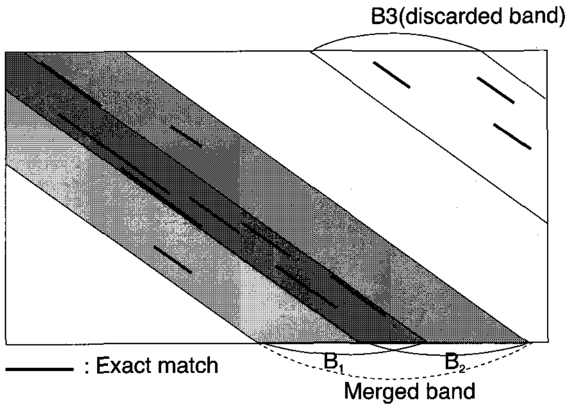

Fig. 1. Merging and discarding bands.

All Normalized Local Alignments Above a Threshold

We now present algorithms for the TNLA problem of two strings a and b. We first give a simple algorithm that finds all normalized local alignments whose alignment scores are above a given threshold T_s (the TNL alignments) in such a way that a TNL alignment of larger alignment score is found earlier than a TNL alignment of smaller alignment score. Arslan *et al.* (Arslan *et al.*, 2001) also found the TNL alignments in the same decreasing order. Then, we improve the simple algorithm by finding the TNL alignments in a somewhat different order.

The simple algorithm consists of two steps. Step 1 is performed once and step 2 is repeated until all TNL alignments are found.

Step 1. Find all maximal exact matches of a and b longer than T_l and determine the bands as we did in Section 3.1.

Step 2. Find an optimal normalized local alignment of a and b using the algorithm presented in Section 3.1. If the alignment score of the found alignment is above T_s, mask the found alignment and repeat step 2. Otherwise, terminate.

The simple algorithm finds the normalized local alignments in decreasing order of alignment scores. Hence, it is guaranteed that every TNL alignment has been found when the simple algorithm terminates.

We consider the speedup achieved by the simple algorithm over Arslan *et al.*' s algorithm. Let p denote the number of the TNL alignments. Step 2 iterates just p times. Since the time for masking an alignment is negligible and our algorithm performs step 1 only once, our simple algorithm for the TNLA problem takes about p times our algorithm for the NLA problem. Also, Arslan *et al.*' s algorithm for the TNLA problem takes p times their algorithm for the NLA problem. Therefore, the speedup achieved by our simple algorithm over Arslan *et al.*' s algorithm for the TNLA problem is the speedup achieved by our algorithm over Arslan *et al.*'s algorithm for the NLA problem.

As long as all the TNL alignments are guaranteed to be found, the order of the alignments that are found may not be important. If we find the TNL alignments in a different order, we can find them more efficiently. The improved algorithm is as follows.

Step 1. Find all maximal exact matches of a and b longer than T_l and determine the bands as in Section 3.1. Let $B_1, B_2, \cdots, B_k$ denote the bands.

Step 2. For each band B_i for $1 \leq i \leq k$, perform the following. Find an optimal normalized local alignment within B_i by running the SW algorithm only in B_i. If the normalized alignment score of the found alignment is above T_s, mask the found alignment and repeat. Otherwise, we are done with B_i.

It is easy to see that all the TNL alignments have been found when the improved algorithm terminates.

We consider the speedup achieved by the improved algorithm over the simple algorithm. Let C_i, $1 \leq i \leq k$, denote the number of the TNL alignments included in band B_i and W_i the width of B_i. In the improved algorithm, band B_i is accessed when we find each of C_i TNL alignments while in the simple algorithm, all bands $B_1, B_2, \cdots, B_k$ are accessed when we find each of $C_1+C_2+\cdots+C_k$ TNL alignments. Note that the improved algorithm accesses B_i at least once even if there is no TNL alignment in it. Hence, the speedup achieved by the improved algorithm over the simple algorithm is

$$\frac{(W_1+W_2+\cdots+W_k)\,(C_1+C_2+\cdots C_K)}{W_1\times\max(C_1,1)+W_2\times\max(C_2,1)+\cdots+W_k\times\max(C_k,1)} \qquad (1)$$

Remark : We can consider *affine gaps* in both problems of NLA and TNLA. The affine gap penalty for a gap is defined as $\gamma+\mu k$, where γ is a gap open penalty, μ is an indel penalty and k is the length of a gap. We can include the affine gap penalty in our algorithms for NLA and TNLA by a simple modification of the recurrence of the SW algorithm (Gotoh 1982; Waterman 1995).

Results and Discussion

We implemented and compared two algorithms for the NLA problem: Arslan *et al.*' s and ours, and three algorithms for the TNLA problem: Arslan *et al'*.s algorithm, our simple algorithm and our improved algorithm. In all our implementations, we considered the affine gaps.

We have chosen two data sets to test the efficiency of the algorithms: (i) human chromosome X cosmids Qc8D3 (GenBank Acc. No. AF030876 (113 kbp)) and mouse chromosome X clone BAC B22804 (GenBank Acc. No. AF121351 (123 kbp)) and (ii) human (GenBank Acc. No. L10347 (31kbp)) and mouse (GenBank Acc. No. M65161 (32.4 kbp)) pro-alpha1 type II collagen. Since the *repeats* which are biologically uninteresting regions may have high normalized scores, we masked the repeats by RepeatMasker (http:// repeatmasker. genome.washington. edu/) before running all the algorithms.

We implemented all the algorithms in C++ language. The programs were run on Pentium III 866MHz$\times$2 workstation with 768MB main memory. The parameter values used in our algorithms are as follows: L=2000 (a constant in defining normalized scores), δ=1 (mismatch penalty), γ =6 (gap open penalty), μ=0.2 (indel penalty), and T_s=0.035 (threshold of normalized score).

NLA Problem

Our algorithm finds the same optimal normalized local alignment as Arslan *et al.*'s algorithm does even when w=100 for both data sets. As shown in Table 1, our algorithm is about 20 times faster than Arslan *et al.*' s for both data sets when $w \geq 150$ and $T_l \leq 150$.

We have experimented with various values of the three parameters T_l, T_b and w, and here we show nine cases for three values of T_l and three values of w and we have chosen an appropriate value of T_b for each T_l. Note that we decrease T_b as T_l increases. Otherwise, good bands may be discarded since longer exact matches are fewer than shorter exact matches.

TNLA Problem

Table 2 and Table 3 show execution times of the three algorithms for the TNLA problem. We first analyze the ratios of execution times of the three algorithms. Assume that the widths of all bands are the same and the TNL alignments are equally distributed in the bands which include TNL alignments. Let k be the number of the bands

Table 1. Execution time (in seconds) for the NLA problem (first data set / second data set).

Arslan et al.' s algorithm		Our algorithm		
		T_l=12, T_b=80	T_l=15, T_b=50	T_l=20, T_b=30
59585 / 4340	w=100	1365 / 207	468 / 187	398 / 170
	w=150	2108 / 230	615 / 207	513 / 198
	w=200	2827 / 283	746 / 225	621 / 218

Table 2. Execution time (in seconds) for the TNLA problem for the first data set.

Arslan et al.' s algorithm		Simple algorithm			Improved algorithm		
		T_l=12, T_b=80	T_l=15, T_b=50	T_l=20, T_b=30	T_l=12, T_b=80	T_l=15, T_b=50	T_l=20, T_b=30
631603	w=100	14345	4876	3915	1923	1351	1259
	w=150	25249	7473	5332	2705	1895	1581
	w=200	35172	8879	6921	3492	2209	1996

Table 3. Execution time (in seconds) for the TNLA problem for the second data set.

Arslan et al.' s algorithm		Simple algorithm			Improved algorithm		
		T_l=12, T_b=80	T_l=15, T_b=50	T_l=20, T_b=30	T_l=12, T_b=80	T_l=15, T_b=50	T_l=20, T_b=30
67558	w=100	2390	2224	1964	970	638	491
	w=150	2643	2396	2384	1441	1333	1114
	w=200	3263	2592	2508	1578	1443	1391

that our algorithms determine, and k' be the number of the bands which include TNL alignments. Also, let t be the average number of TNL alignments in a band. Then, the equation (1) can be rewritten as follows:

$$\frac{kk't}{k't+k-k'}$$

Hence, our improved algorithm is faster than Arslan *et al.*'s by the following factor:

$$\frac{kk't}{k't+k-k'.}\times S$$

where S is the speedup of our simple algorithm over Arslan *et al.*' s.

Consider the experimental results for the first data set in Table 2. When w=150, T_l=15 and T_b=50, we have $S\cong 80$ and k=5, k'=5, t=1/3 (k, k' are not shown in Table 2 since the number of TNL alignments is 15 as shown in Table 4). By the above formula, our improved algorithm should be approximately 400 times faster than Arslan *et al.*'s and the result (631603/1895$\cong$330) is similar to our expectation. (Note that the time of the improved algorithm can vary depending on the value of T_b in each case.) For another example, when w=200, T_l=12 and T_b=80, we get $S\cong 18$, k=24, k'=5 and t=1/3. Thus, our improved algorithm should be about 200 times faster than Arslan *et al.*' s and the result (631603/3492$\cong$180) shows an approximately same ratio.

For the second data set, our algorithm is about 40 times faster than Arslan *et al.*' s when w=150, T_l=15 and T_b=50, and about 50 times faster when w=200, T_l=12 and T_b=80.

Table 4 shows the accuracy of our algorithms. For example, when w=100, T_l=20 and T_b=30 in Table 4, our algorithms find 12 out of 15 TNL alignments. But as w increases and T_l decreases, our algorithms are getting more accurate. Note that our algorithms find all TNL alignments when $w\geq 150$ and $T_l\leq 15$.

In addition to extremely fast speeds, another advantage of our algorithms is that we can change the parameters w, T_l and T_b to adjust the time and accuracy of our algorithms.

Acknowledgments

This work was supported by the Brain Korea 21 Project, the IMT 2000 Project AB02, the MOST grant M6-0203-00-0039, and the RIACT at Seoul National University.

References

Alexandrov, N.N., and Solovyev, V.V., (1998). Statistical significance of ungapped alignments, *Pacific Symposium on Biocomputing' 98*, 463-472.

Arslan, A.N., and Eğecioğlu, Ö.(1999), An efficient uniform-cost normalized edit distance algorithm, *Symposium on String Processing and Information Retrieval' 99*, IEEE Computer

Table 4. The number of TNL alignments that are found (first data set / second data set).

Arslan et al.' s algorithm		Simple/Improved algorithm		
		T_l=12, T_b=80	T_l=15, T_b=50	T_l=20, T_b=30
	w=100	13 / 12	13 / 12	12 / 11
15 / 12	w=150	15 / 12	15 / 12	13 / 12
	w=200	15 / 12	15 / 12	14 / 12

Society, 8-15.

Arslan, A.N., and Eğecioğlu, Ö.(2003), Efficient algorithms for normalized edit distances, *Journal of Discrete Algorithms*, Hermes Science Publications, in press.

Arslan, A.N., Eğecioğlu, Ö., and Pevzner, P. (2001). A new approach to sequence comparison: normalized sequence alignment, *Bioinformatics* 17, 327-337.

Chen, T., and Skiena, S.S., (1997). Trie-based data structures for sequence assembly, *Combinatorial Pattern Matching' 97*, 206-223.

Dinkelbach, W., (1967). On nonlinear fractional programming, *Management Science* 13, 492-498.

Eğecioğlu, Ö., and Ibel, M. (1996). Parallel algorithms for fast computation of normalized edit distances, *IEEE Symposium on Parallel and Distributed Processing' 96*, 496-503.

Gotoh, O., (1982). improved algorithm for matching biological sequences, *Journal of Molecular Biology* 162, 705-708.

Goad, W.B., and Kanehisa, M.I. (1982). Pattern recognition in nucleic acid sequences. i. a general method for finding local homologies and symmetries, *Nucleic Acids Research* 10, 247-263.

Green, P., *Documentation for phrap*, Genome Center, University of Washington, *http://www.phrap.org/ phrap.docs/phrap.html.*

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press.

Lipman, D., and Pearson, W. (1988) Improved tools for biological sequence comparison, *Proceedings of National Academy of Science* 85, 2444-2448.

Marzal, A., and Vidal, E. (1993) Computation of normalized edit distances and applications, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15, 926-932.

Sellers, P.H. (1984). Pattern recognition in genetic sequences by mismatch density, *Bulletin of Mathematical Biology* 46, 501-504.

Setubal, J., and Meidanis, J., (1997). Introduction to computational molecular biology, PWS Publishing Company.

Smith, T.F., and Waterman, M.S. (1981). Identification of common molecular subsequences, *Journal of Molecular Biology* 147, 195-197.

Waterman, M.S., (1995). Introduction to Computational Biology, Chapman & Hall, London

Zhang, Z., Berman, P., and Miller, W. (1998). Alignments without low scoring regions, *Journal of Computational Biology* 5, 197-200.

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