A Space Efficient Algorithm for Sequence Alignment with Inversions

Yong Gao[1] *, Junfeng Wu[1] *, Robert Niewiadomski[1] *, Yang Wang[1] *, Zhi-Zhong Chen[2] **, and Guohui Lin[1] ***

[1] Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada.
[2] Mathematical Sciences, Tokyo Denki Univ., Hatoyama, Saitama 350-0394, Japan.

Abstract. A dynamic programming algorithm to find an optimal alignment for a pair of DNA sequences has been described by Schöniger and Waterman. The alignments use not only substitutions, insertions, and deletions of single nucleotides, but also inversions, which are the reversed complements, of substrings of the sequences. With the restriction that the inversions are pairwise non-intersecting, their proposed algorithm runs in $O(n^2m^2)$ time and consumes $O(n^2m^2)$ space, where n and m are the lengths of the input sequences respectively. We develop a space efficient algorithm to compute such an optimal alignment which consumes only $O(nm)$ space within the same amount of time. Our algorithm enables the computation for a pair of DNA sequences of length up to 10,000 to be carried out on an ordinary desktop computer. Simulation study is conducted to verify some biological facts about gene shuffling across species.

1 Introduction

Sequence alignment has been well studied in the 80's and 90's, where a bunch of algorithms have been designed for various purposes. The interested reader may refer to [2] and the references therein. Normally, the input sequence is considered as a linear series of symbols taken from a fixed alphabet, which consists of 4 nucleotides in the case of DNA/RNA sequences or 20 amino acids in the case of proteins. An alignment of two sequences is obtained by first inserting spaces into or at the ends of the sequences and then placing the two resulting sequences one above the other, so that every symbol or space in either sequence is opposite to a unique symbol or space in the other sequence. An alignment, which is associated with some objective function, can be naturally partitioned into columns and the column order is required to agree with the symbol orders in the input sequences.

* Email: {ygao,jeffwu,niewiado,ywang}@cs.ualberta.ca.

** Supported in part by the Grant-in-Aid for Scientific Research of the Ministry of Education, Science, Sports and Culture of Japan, under Grant No. 14580390.Email: chen@r.dendai.ac.jp. Part of the work done while visiting at University of Alberta.

*** Corresponding author. Supported in part by NSERC grants RGPIN249633 and A008599, and a REE Startup Grant from the University of Alberta.Email: ghlin@cs.ualberta.ca.

In other words, these alignments use substitutions, insertions, and deletions of symbols only and are called normal alignments. For our purpose, we limit the input sequences to DNA sequences. In the literature, a number of algorithms [2] have been designed to compute an optimal normal alignment between two DNA sequences, under various pre-determined score schemes. A score scheme basically specifies how to calculate the score associated with an alignment. In this paper, the score schemes are symbol-independent. The simplest such score scheme would be the notion of *Longest Common Subsequence*, where every match column gets a score of 1 and every other column gets a score of 0 and the objective is to maximize the sum of the column scores. In the more general linear gap penalty score scheme specified by a triple (w_m, w_s, w_i), every match column gets a score w_m, every substitution column gets a score w_s, and every insertion or deletion (an *indel*) gets a score w_i. The linear gap penalty score schemes assume every single nucleotide mutation is independent of the others, which appears to be not biologically significant. The most commonly used score scheme nowadays is the so-called *affine gap penalty score scheme*. Given an alignment, a maximal segment of consecutive spaces in one sequence is called a *gap*, whose length is measured as the number of spaces inside. An affine gap penalty score scheme is defined by a quadruple (w_m, w_s, w_o, w_e), where w_m is the score for a match, w_s is the score for a replacement/substitution, and $w_o + w_e \times \ell$ is the penalty for a gap of length ℓ. Intuitively, affine gap penalty score schemes assume single nucleotide mutations might depend on its neighboring nucleotide mutations. As an example, under the affine gap penalty score scheme $(10, -11, -15, -5)$, the following alignment has a score of 4:

```
123456789012345678901234 5
-CCAATCTAC----TACTGCTTGCA
 ||| ||| |    |||||  ||
GCCACTCT-CGCTGTACTG--TG--
```

In fact, one can easily verify that the above alignment is optimal for DNA sequences `CCAATCTACTACTGCTTGCA` and `GCCACTCTCGCTGTACTGTG` under the specific score scheme.

Alignments are used to reveal the information content of genes in DNA sequences and to make inferences about the relatedness of genes or more general inferences about the relatedness of long-range-segments of DNA sequences. With many genomic projects been carried out, emphasis has been put on the study on the genetic linkage among species and/or organisms. The sequences thus under consideration could come from different species and inferences can be made about the relatedness of species. In this aspect, the normal sequence alignment has the limitation on using evolutionary transformations to substitutions and indels. It has been widely accepted that duplications and inversions are common events in molecular evolution [3,7]. Some pioneer work on studying sequence alignment/comparison with inversions is done by Wagner [6] and Schöniger and Waterman [5]. In particular, in [5], a dynamic programming algorithm is developed to compute an optimal (local) alignment with inversions. The algorithm

runs in $O(n^2m^2)$ time and consumes $O(n^2m^2)$ spaces, where n and m are the lengths of two input sequences respectively.

The high order of running time seems computationally impractical, nonetheless, it is the huge space requirement that actually makes the computation infeasible. For instance, suppose the algorithm needs $\frac{1}{4}n^2m^2$ byte memory, then an optimal alignment between two sequences of length 250 won't be carried out in a normal desktop of 1Gb memory. Schöniger and Waterman [5] designed an algorithm to compute a *suboptimal* alignment with inversions restricted to a constant number of *highest scoring inversions*. This latter algorithm runs in $O(nm + \sum_{i=1}^{k} \ell_i)$ time and requires an order of $O(nm + \sum_{i=1}^{k} \ell_i)$ space, where k is the number of restricted highest scoring inversions and ℓ_i's are their lengths respectively. In this paper, we develop a space efficient algorithm to compute an *optimal* alignment with non-restricted inversions, in terms of both the number and their scores, within $O(m^2n^2)$ time. The algorithm is non-trivial in the sense that deep computation relationships are carefully characterized.

Problem Description. An inversion of a DNA substring is defined to be the reverse complement of the substring. In this paper, we won't put a limit on the number of inversions in the alignments, but we do want to put a lower bound on the lengths of inversions, for the reason that we believe inversions correspond to some conserved coding regions and a conserved region must have a least number of nucleotides inside. We let L denote this lower bound. In the extreme case where $L = 0$, it becomes the problem considered in [5]. The inversions are restricted to be on one of the two input sequences (by symmetry) and they are not allowed to intersect one another. Moreover, the substrings from the other sequence against which those inversions are aligned are not allowed to intersect one another either.

The score schemes used in this paper are all affine gap penalty ones. For the sake of comparison, the parameters in the schemes may be different in different simulations. We associate each inversion with a (constant) penalty of C [5], to compensate for the fact that the inversion is an evolutionary event.

Now it is ready to formulate the computational problem we will be considering in the paper. The input is a pair of DNA sequences S_1 and S_2 over the nucleotide alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, together with an affine gap penalty score scheme (w_m, w_s, w_o, w_e), inversion lower bound L, and inversion penalty C. The goal is to compute an optimal alignment between S_1 and S_2 with inversions. For simplicity, we associate *standard similarity* with a normal optimal alignment, which takes substitutions and indels only; and we use *inversion similarity* to associate with an optimal alignment taking inversions in addition (called an *optimal inversion alignment*). Both similarities (also called sequence identities hereafter) are defined to be the ratio of the number of match columns in the alignment over the length of the shorter sequence.

Organization. The paper is organized as follows: In the next section we will detail our algorithm to compute an optimal alignment between a pair of DNA sequences S_1 and S_2. The algorithm runs in $O(m^2n^2)$ time and requires only

$O(mn)$ space, where m and n are the lengths of S_1 and S_2, respectively. In Section 3, we show the simulation results of our algorithm applied to the instances tested in [5]. We conclude the paper with some remarks in Section 4.

2 The Algorithms

Let $S_1[1..m]$ and $S_2[1..n]$ denote the two input sequences of length m and n, respectively. For simplicity of presentation, we simplify the problem a little by requiring that gaps in non-inversion regions and inversion regions are counted separately and independently. We let $\overline{S[i]}$ denote the complement nucleotide of $S[i]$ (the Watson-Crick rule); let $\overline{S_1[i_1..i_2]}$ denote the inversion of $S_1[i_1..i_2]$, that is, $\overline{S_1[i_2]}\ \overline{S_1[i_2-1]}\ldots\overline{S_1[i_1]}$.

2.1 A less space efficient algorithm

In this subsection, we introduce a less space efficient dynamic programming algorithm which is conceptually simple to understand. This algorithm is for linear gap penalty score scheme (that is, we won't use w_o and w_e here, but use w_i instead which is the score for an indel). The more complicated yet more space efficient algorithm in Section 2.2 for the affine gap penalty score scheme will be a refinement of this simple algorithm.

Inversion Table Computation. Suppose that $S_1[i_1, i_2]$ and $S_1[i_3, i_4]$ are two inversion substrings in sequence S_1, then by the length constraint and non-intersecting requirement we have: $i_2 - i_1 \geq L - 1$, $i_4 - i_3 \geq L - 1$, and either $i_2 < i_3$ or $i_4 < i_1$. For every quartet (i_1, i_2, j_1, j_2), where $0 \leq i_1 \leq i_2 \leq m$ and $0 \leq j_1 \leq j_2 \leq n$, let $I[i_1, i_2; j_1, j_2]$ denote the standard sequence similarity between two substrings $\overline{S_1[i_1..i_2]}$ and $S_2[j_1..j_2]$ (that is, without inversions). As an easy example, $S_1 = \texttt{ACGT}$ and $S_2 = \texttt{ACGA}$, then $S_1[2..4] = \texttt{CGT}$ and thus $\overline{S_1[2..4]} = \texttt{ACG}$, and $S_2[1..3] = \texttt{ACG}$. Therefore, $I[2, 4; 1, 3] = 3w_m$.

Let $w(a, b)$ denote the score of matching nucleotide a against nucleotide b, where a and b are both in the alphabet (which includes nucleotides `A`, `C`, `G`, `T`). Therefore,

$$w(a,b) = \begin{cases} w_m, \text{ if } a = b, \\ w_s, \text{ if } a \neq b \end{cases}$$

Let $S_1[0] = S_2[0] = -$. To assist the *inversion table* I computation, I is enlarged to include the following boundary entries:

- $I[i+1, i; j_1, j_2] = (j_2 - j_1 + 1)w_i$, where $0 \leq i \leq m$ and $0 \leq j_1 \leq j_2 \leq n$;
- $I[i_1, i_2; j+1, j] = (i_2 - i_1 + 1)w_i$, where $0 \leq i_1 \leq i_2 \leq m$ and $0 \leq j \leq n$.

The recurrence relation for computing a general entry $I[i_1, i_2; j_1, j_2]$ is as follows:

$$I[i_1,i_2;j_1,j_2] = \max \begin{cases} I[i_1+1, i_2; j_1, j_2-1] + w(\overline{S_1[i_1]}, S_2[j_2]), \\ I[i_1+1, i_2; j_1, j_2] + w_i, \\ I[i_1, i_2; j_1, j_2-1] + w_i, \end{cases}$$

where $1 \leq i_1 \leq i_2 \leq m$ and $1 \leq j_1 \leq j_2 \leq n$.

Notice that table I contains in total $\frac{1}{4}(m+1)^2(n+1)^2$ entries and filling each of them takes a constant amount of time.

Dynamic Programming Table Computation Let $DP[i, j]$ denote the score of an optimal inversion alignment between prefixes $S_1[1..i]$ and $S_2[1..j]$, where $0 \leq i \leq m$ and $0 \leq j \leq n$. The boundary entries of the *dynamic programming table* DP are:

- $DP[0, j] = j \times w_i$, where $0 \leq j \leq n$;
- $DP[i, 0] = i \times w_i$, where $0 \leq i \leq m$.

Recall that every inversion in S_1, as well as its aligned segment in S_2, must have length at least L. The recurrence relation for computing a general entry $DP[i, j]$ is as follows:

- When $i \geq L$ and $j \geq L$,

$$DP[i,j] = \max \begin{cases} DP[i-1, j-1] + w(S_1[i], S_2[j]), \\ DP[i-1, j] + w_i, \\ DP[i, j-1] + w_i, \\ \max\limits_{\substack{1 \leq i' \leq i-L+1 \\ 1 \leq j' \leq j-L+1}} \Big\{ DP[i'-1, j'-1] + I[i', i; j', j] \Big\} - C. \end{cases}$$

- In the other case,

$$DP[i,j] = \max \begin{cases} DP[i-1, j-1] + w(S_1[i], S_2[j]), \\ DP[i-1, j] + w_i, \\ DP[i, j-1] + w_i. \end{cases}$$

Since the dynamic programming table contains $(m + 1)(n + 1)$ entries and entry $DP[i, j]$ takes up to $O(ij)$ time to fill, the running time of the overall algorithm is $O(m^2n^2)$.

The correctness of the algorithm follows directly from the recurrences and therefore we have the following conclusion.

Theorem 1. *The inversion similarity between $S_1[1..m]$ and $S_2[1..n]$ can be computed in $O(m^2n^2)$ time using $O(m^2n^2)$ space.*

2.2 A space efficient algorithm for affine gap penalty score schemes

In the algorithm in Section 2.1 we divide the computation into two phases. In the first phase we prepare all the possible inversions together with their all possible aligned substrings from the other sequence and the associated scores. Direct extension can lead to an algorithm for affine gap penalty score schemes, running in the same amount of time and requiring the same amount of space. In the following, we present a single-phase computation. We fill the dynamic programming table row-wise. Furthermore, when filling the i^{th} row, we compute

all possible inversions ending at position i. We will prove that there are $O(mn)$ such possible inversions and each can be calculated in constant time based on the intermediate results of computation for the $(i-1)^{\text{th}}$ row. This single-phase computation still takes $O(m^2n^2)$ time, nonetheless requires $O(mn)$ space only.

Let $DP_M[i,j]$, $DP_I[i,j]$, and $DP_D[i,j]$ denote the scores of an optimal inversion alignment between prefixes $S_1[1..i]$ and $S_2[1..j]$, where $0 \le i \le m$ and $0 \le j \le n$, such that the last operation is either a match or a mismatch, an insertion, and a deletion, respectively. The boundary entries of these dynamic programming tables are:

- $DP_M[0,0] = DP_I[0,0] = DP_D[0,0] = 0$;
- $DP_I[0,j] = w_o + j \times w_e$, where $1 \le j \le n$;
- $DP_D[i,0] = w_o + i \times w_e$, where $1 \le i \le m$.

Let $I_M^{i,j}[i',j']$, $I_I^{i,j}[i',j']$, and $I_D^{i,j}[i',j']$ denote the sequence similarities between inversion $\overline{S_1[i'..i]}$ and $S_2[j'..j]$, where $1 \le i' \le i \le m$ and $1 \le j' \le j \le n$, such that the last operation is either a match or a mismatch, an insertion, and a deletion, respectively. Again, to assist the computation for these 3 inversion tables, they are enlarged to include the following boundary entries:

- $I_M^{i,j}[i+1,j+1] = I_I^{i,j}[i+1,j+1] = I_D^{i,j}[i+1,j+1] = 0$;
- $I_I^{i,j}[i+1,j'] = w_o + (j-j'+1)w_e$, where $1 \le j' \le j$;
- $I_D^{i,j}[i',j+1] = w_o + (i-i'+1)w_e$, where $1 \le i' \le i$.

The recurrence relations for dynamic programming tables computation are:

$$DP_M[i,j] = \max \begin{cases} w(S_1[i], S_2[j]) + \max \begin{cases} DP_M[i-1,j-1], \\ DP_I[i-1,j-1], \\ DP_D[i-1,j-1]; \end{cases} \\ \max\limits_{\substack{1 \le i' \le i-L+1 \\ 1 \le j' \le j-L+1}} \left\{ I_M^{i,j}[i',j'] + \max \begin{cases} DP_M[i'-1,j'-1], \\ DP_I[i'-1,j'-1], \\ DP_D[i'-1,j'-1] \end{cases} \right\} - C \end{cases}$$

$$DP_I[i,j] = \max \begin{cases} DP_M[i,j-1] + w_o + w_e, \\ DP_I[i,j-1] + w_e, \\ DP_D[i,j-1] + w_o + w_e, \\ \max\limits_{\substack{1 \le i' \le i-L+1 \\ 1 \le j' \le j-L+1}} \left\{ I_I^{i,j}[i',j'] + \max \begin{cases} DP_M[i'-1,j'-1], \\ DP_I[i'-1,j'-1], \\ DP_D[i'-1,j'-1] \end{cases} \right\} - C \end{cases}$$

$$DP_D[i,j] = \max \begin{cases} DP_M[i-1,j] + w_o + w_e, \\ DP_I[i-1,j] + w_o + w_e, \\ DP_D[i-1,j] + w_e, \\ \max\limits_{\substack{1 \le i' \le i-L+1 \\ 1 \le j' \le j-L+1}} \left\{ I_D^{i,j}[i',j'] + \max \begin{cases} DP_M[i'-1,j'-1], \\ DP_I[i'-1,j'-1], \\ DP_D[i'-1,j'-1] \end{cases} \right\} - C \end{cases}$$

Before computing these i^{th} row entries, $I_M^{i,j}$, $I_I^{i,j}$, and $I_D^{i,j}$ tables are pre-computed using the following recurrence relations.

$$I_M^{i,j}[i',j'] = w(\overline{S_1[i']}, S_2[j]) + \max \begin{cases} I_M^{i,j-1}[i'+1,j'], \\ I_I^{i,j-1}[i'+1,j'], \\ I_D^{i,j-1}[i'+1,j'] \end{cases}$$

$$I_I^{i,j}[i',j'] = \max \begin{cases} I_M^{i,j-1}[i',j'] + w_o + w_e, \\ I_I^{i,j-1}[i',j'] + w_e, \\ I_D^{i,j-1}[i',j'] + w_o + w_e \end{cases}$$

$$I_D^{i,j}[i',j'] = \max \begin{cases} I_M^{i,j}[i'+1,j'] + w_o + w_e, \\ I_I^{i,j}[i'+1,j'] + w_o + w_e, \\ I_D^{i,j}[i'+1,j'] + w_e \end{cases}$$

Notice that the computation of $I_M^{i,j}$, $I_I^{i,j}$, and $I_D^{i,j}$ tables needs the values in $I_M^{i,j-1}$, $I_I^{i,j-1}$, and $I_D^{i,j-1}$ tables only. It follows that we may just keep 3 inversion tables $I_M^{i,j-1}$, $I_I^{i,j-1}$, and $I_D^{i,j-1}$ after computing entries $DP_M[i,j-1]$, $DP_I[i,j-1]$, and $DP_D[i,j-1]$. These 3 tables are then used in computing entries $DP_M[i,j]$, $DP_I[i,j]$, and $DP_D[i,j]$, where we create 3 new inversion tables $I_M^{i,j}$, $I_I^{i,j}$, and $I_D^{i,j}$. After that, those 3 inversion tables $I_M^{i,j-1}$, $I_I^{i,j-1}$, and $I_D^{i,j-1}$ will no longer be used and thus can be deallocated.

In other words, we need only in total 9 2-dimensional tables during the overall computation, which consume $O(mn)$ space. The overall running time $O(m^2n^2)$ is obviously seen from the recurrences, where trying all possible combinations of i' and j' for pair (i,j) dominates the computation.

Theorem 2. *The inversion similarity between $S_1[1..m]$ and $S_2[1..n]$, using affine gap penalty score schemes, can be computed in $O(m^2n^2)$ time using $O(mn)$ space.*

3 Simulation Results

In the example instance in the introduction, $S_1 =$ `CCAATCTACTACTGCTTGCA` and $S_2 =$ `GCCACTCTCGCTGTACTGTG`. Under the affine gap penalty score scheme specified by $(10, -11, -15, -5)$, we showed there an optimal normal alignment. Using the lower bound $L = 5$ and inversion penalty $C = 2$, the following shows an optimal inversion alignment, associated with a score of 43:

```
1234567890 123456 789012345
-CCAATCTAC gcagta TTGCA
 ||| ||| | || |||  ||
GCCACTCT-C GCTGTA CTGTG
```

In which the lower case substring "`gcagta`" is an inversion from $S_1[10..15] =$ `TACTGC`.

In [5], it has been calculated under the same score scheme that $S_1[10..15]$ *vs.* $S_2[10..15]$ is the highest scoring inversion and $S_1[7..9]$ *vs.* $S_2[13..15]$ is the

second highest. And using these 2 highest scoring inversions, the output inversion alignment in their paper is exactly the same as ours as shown. Therefore, our work confirms that the inversion alignment computed using 2 highest scoring inversions for the above instance in [5] is in fact an optimal one.

A biological instance used for simulation in [5] consists of a DNA sequence from *D. yakuba* mitochondria genome using nucleotides 9,987–11,651 and a DNA sequence from mouse mitochondria genome using nucleotides 13,546–15,282. Under the affine gap penalty score scheme specified by $(10, -9, -15, -5)$ and inversion penalty $C = 20$, by pre-computing a list of 400 highest scoring inversions, the alignment output in [5] found an inversion substring in *D. yakuba* consisting of nucleotides 7–480 which aligns to nucleotides 58–542 from mouse. The putative organization of genes in the two DNA sequences is described in Table 1:

	D. yakuba	mouse
URF6	1–525	519–1 (inverted)
tRNA Glu		588–520 (inverted)
cytochrome b	529–1,665	594–1737

Table 1. Putative organization of genes in the two DNA sequences.

So the identified inversion to some extent detects the biologically correct inversion. In our simulation, we use the same score scheme and again add the lower bound on the inversion lengths $L = 5$.

Unfortunately, the algorithm didn't detect any *good* inversions. What it found are three short inversions $S_1[344..349]$ which aligns to $S_2[326..331]$, $S_1[387..391]$ which aligns to $S_2[357..361]$, and $S_1[456..463]$ which aligns to $S_2[417..424]$. With these three inversions, the detected inversion identity is 0.6853, contrast to the standard identity 0.6847. We did another experiment by cutting out the two URF6 genes from both sequences and calculating their inversion identity, namely, the first 525 nucleotides from *D. yakuba* and the first 519 nucleotides from mouse. It turned out that the standard identity between these two substrings is 0.5780 and the inversion identity remains the same as the standard one without any inversion detected.

Since the inversion algorithm didn't detect any meaningful inversions, we modified the algorithm to detect reversals, which only reverse a substring but not take the complement. We define the reversal identity similarly to be the ratio in an optimal reversal alignment. For the two URF6 genes, by setting the length lower bound $L = 5$, we found a lot of reversals which are listed in the Table 2. The reversal identity is 0.6301 as detected.

By setting $L = 300$, we found a reversal substring $S_1[128..513]$ which is aligned to $S_2[121..507]$. The alignment score is improved from 152 to 167 with a little bit identity sacrifice down to 0.5742. The standard identity between these two segments is 0.5622 with alignment score 55 (an optimal standard alignment is shown in the left side of Figure 1); The reversal identity between them is 0.5596 with alignment score 110 (an optimal reversal alignment is shown in the right side of Figure 1).

D. yakuba	mouse
15–23	16–25
37–41	37–44
44–73	47–70
78–82	75–79
100–114	105–119
130–165	134–172
180–192	187–199
209–227	216–238
244–249	255–260
254–261	265–272

D. yakuba	mouse
266–276	277–287
283–302	294–315
316–322	329–335
324–342	337–350
350–378	358–383
383–387	388–392
394–436	399–438
437–447	439–449
459–512	461–504
518–525	510–519

Table 2. Fragmental reversal segments and their aligned partner.

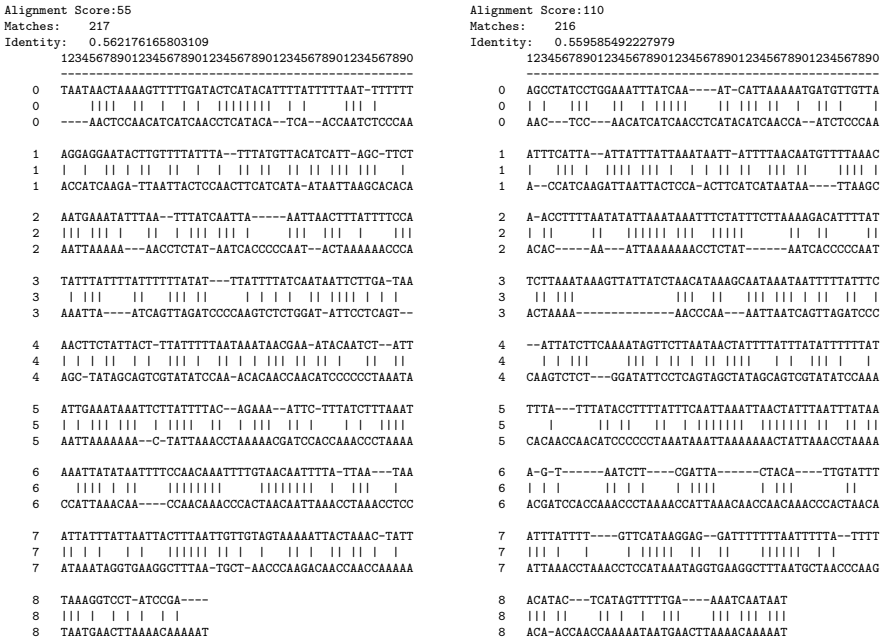

Fig. 1. An optimal standard alignment (left) and an optimal reversal alignment (right) between $S_1[128..513]$ and $S_2[121..507]$.

4 Conclusions

The space efficient algorithm developed in this paper enables the computation of an optimal inversion alignment between two DNA sequences of length up to 10,000bp on a normal desktop with 1Gb memory. Previous algorithms either fail on long sequences or only produce a suboptimal inversion alignment restricted to a number of pre-computed highest scoring inversions. The simulation con-

ducted shows a disagreement with previous simulation. Further investigation is necessary, typically on the selection of suitable score schemes.

The recurrences for computing DP_M, DP_I, and DP_D tables are written for the case where gaps inside inversion segments and gaps inside non-inversion segments are treated separately and independently. If two gaps from different categories are adjacent to each, then they might be counted as one gap. The recurrences can be slightly modified, where one copy of inversion penalty C should be merged to $DP_M[i'-1, j'-1]$, $DP_I[i'-1, j'-1]$, and $DP_D[i'-1, j'-1]$ during the computation, to take care of this case.

Our algorithm can be easily modified to compute an optimal reversal alignment between sequences. Some simulation has been run which shows something different from inversions. We have also done some preliminary simulation study on applying the inversion and reversal algorithms to detect similar secondary structure units for RNA sequences, which correspond to reversed substrings, in the RNase P Database (`http://www.mbio.ncsu.edu/RNaseP/home.html`) [1]. The result will be reported elsewhere.

Upon acceptance, we were informed of a recent work [4], where Muthukrishnan and Sahinalp consider the problem of minimizing the number of character replacements (no insertions and deletions) and reversals and propose an $O(n \log^2 n)$ time deterministic algorithm, where n is the length of either sequence.

Acknowledgments. We thank Dr. Patricia Evans (Computer Science, University of New Brunswick) and Bin Li (Biological Science, University of Alberta) for helpful discussions.

References

1. J. W. Brown. The Ribonuclease P Database. *Nucleic Acids Research*, 27:314, 1999.
2. D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge, 1997.
3. C. J. Howe, R. F. Barker, C. M. Bowman, and T. A. Dyer. Common features of three inversions in wheat chloroplast dna. *Current Genetics*, 13:343–349, 1988.
4. S. Muthukrishnan and S. C. Sahinalp. An improved algorithm for sequence comparison with block reversals. In *Proceedings of The 5th Latin American Theoretical Informatics Symposium (LATIN'02)*, LNCS 2286, pages 319–325, 2002.
5. M. Schöniger and M. S. Waterman. A local algorithm for DNA sequence alignment with inversions. *Bulletin of Mathematical Biology*, 54:521–536, 1992.
6. R. A. Wagner. On the complexity of the extended string-to-string correction problem. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, Strings Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, pages 215–235. Addison-Wesley, 1983.
7. D. X. Zhou, O. Massenet, F. Quigley, M. J. Marion, F. Monéger, P. Huber, and R. Mache. Characterization of a large inversion in the spinach chloroplast genome relative to *marchantia*: a possible transposon-mediated origin. *Current Genetics*, 13:433–439, 1988.