February 1, 2000
RT0343
Computer Science 14 pages

Research Report

New Approaches for Analyzing Recombinations of Biological Sequences

Tetsuo Shibuya

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

New Approaches for Analyzing Recombinations of Biological Sequences

Tetsuo SHIBUYA
IBM Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato-shi, Kanagawa 242-8502, Japan.
Tel: +81-462-73-5915   Fax: +81-462-73-7413
E-mail: tshibuya@trl.ibm.co.jp

Abstract

Recombination is one of the most important mechanisms of genomic mutation. Recent work has revealed that recombinations occur far more frequently than previously thought in sequences such as RNA sequences of human immunodeficiency viruses (HIVs). But existing techniques for detecting recombinations of sequences are inaccurate or impractical. For this problem, we propose two kinds of approaches. One is an approach by the alignment of sequences considering recombinations, and the other is a statistical approach that detects recombinations using aligned multiple sequences.

For the first approach by the alignment technique, we first formulate an alignment problem for sequences containing recombinations. We then give algorithms with practical bounds for various versions of the problem. In the statistical approach, we consider a simple model of the statistical behavior of point mutations, and propose a method for detecting recombinations by computing what we call z-values.

To demonstrate both techniques, we conducted experiments using actual RNA sequences of HIVs and artificially recombined sequence data.

Keywords: sequence analysis, recombination, sequence alignment, dynamic programming, algorithm, statistical analysis

1 Introduction

Recombination is one of the most important mechanisms of genomic mutation. It produces a new sequence by crossing two parent sequences: the new sequence is constructed by copying a substring of a parent sequence, then crossing over to somewhere in the other parent and copying a substring of that sequence. This mechanism leads to many other complex events, such as block insertions and deletions, tandem repeats, and substring shuffling. Recent work has revealed that recombinations occur far more frequently than previously thought in sequences such as RNA sequences of human immunodeficiency viruses (HIVs) [3, 5, 6, 16, 18, 19, 21]. We must therefore take account of recombinations in sequence analysis. However, a lot of computing time is often needed to deal with these higher-order phylogenetic events [1, 14, 15].

Most work related to recombination has focused on statistical tests or analyses for population genetics studies [2, 8, 12, 13, 24]. In sequence analysis, there are two frequently-used methods for detecting such recombinations, both of which use multiple sequence alignment of the sequences. One method is to plot local similarity [27]. In this method, we first align the candidates for parent sequences with the child sequence, and then we plot lines along the sequences which describe the similarities of fixed-length substrings (called windows) of the child sequence and each of the candidates for parents. Crossovers are found by locating crossings of the plotted lines of two parent candidates. The other method constructs local phylogenetic trees [22, 23], by using substrings of at the same position in the alignment. If different trees appear in different positions of the sequences, we conclude that a crossover may have occurred. Both methods are very useful for showing the "atmosphere" of recombination among sequences, but neither gives evidence of occurrences of recombinations from a statistical or mathematical viewpoint.

Sequence alignment is one of the most famous and useful techniques for analyzing sequences in computational biology. It is used in various fields of molecular biology, such as the finding of motifs in genome sequences, the prediction of protein structures, and the inference of phylogenetic trees. But the ordinary alignment technique ignores many factors such as recombination and tandem repeats. The first work related to the alignment technique that considers recombination, by Hein [10, 11], deals only with equal-length crossovers of appropriately aligned sequences. Kececioglu and Gusfield [14] considered aligning sequences to take account of recombination, but their algorithms have large bounds on the computing time. We will introduce their work briefly in section 2.1.

In this paper, we propose two different approaches for detecting recombinations. One is an extension of the ordinary sequence alignment techniques, and the other is a statistical approach that uses ordinary alignment techniques. In the first approach, we begin by formulating the problem, and then present an algorithm that can be computed in $O(kn^2)$ time, where n is the maximum length of the sequences and k is the number of sequences. In the statistical approach, we consider a simple model of the statistical behavior of point mutations, and propose a method for detecting recombinations by computing what we call z-values. Furthermore, we perform experiments using actual RNA sequences of HIVs and artificially recombined sequence data to demonstrate our methods.

2 Alignment of Sequences with Recombinations

In this section, we propose a new alignment technique that can detect recombinations among sequences.

2.1 Previous Works

The pairwise sequence alignment problem is one of the most fundamental problems in computational molecular biology. Given a finite alphabet set Σ including a gap denoted by -, and a score function between alphabets $\psi : \Sigma \times \Sigma \to R$, the most commonly used definition of this problem is as follows.

Each member of Σ except for the gap is called a character, and a finite string of characters is called a sequence. On the other hand, a finite string of members in Σ is called a padded sequence. The set of ith elements of two padded sequences is called the ith column of the set of the sequences. A set of two padded sequences (S'_1, S'_2) is called an alignment of S_1 and S_2 if and only if all of the padded sequences have the same length l, the ith column contains at least one character for any i $(1 \le i \le l)$, and S'_1 and S'_2 are identical with S_1 and S_2, respectively, if we ignore gaps. The

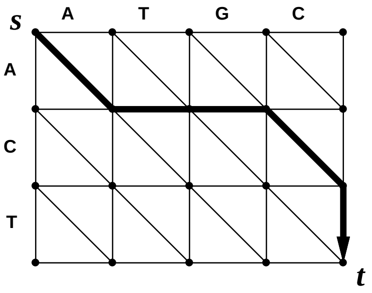


Figure 1: The graph for the alignment of two sequences `ATGC` and `ACT`. The *s*-*t* path in the bold line represents the alignment of `ATGC-` and `A--CT`

problem is to find the alignment of S_1 and S_2 with the largest score, *i.e.*, $\sum_{1\le i\le l}\psi(S'_1(i), S'_2(i))$, where $S(i)$ denotes the ith element of S.

In this definition of the problem, the gap penalty is linear to the number of gap characters. This kind of a gap penalty is called the linear gap penalty. We often impose some starting gap penalty on the starting of a gap sequence. In this case, the score to maximize is $g\cdot\gamma+\sum_{1\le i\le l}\psi(S'_1(i), S'_2(i))$ where g is the number of the sets of consequent gaps and γ is the starting gap penalty. This strategy of imposing a gap penalty is called the affine gap penalty.

The most famous and fundamental exact algorithm for the alignment problem is the dynamic programming (DP) [7, 9, 17, 25, 26]. Let p_{ij} be the best score of the alignment of sequences $S_1^{(i)}$ and $S_2^{(j)}$, where $S^{(i)}$ denotes the prefix substring of S whose length is i. p_{ij} can be obtained by $p_{i-1,j}$, $p_{i-1,j-1}$, and $p_{i,j-1}$ as follows. Let ξ be the gap penalty, and $p_{i,j}$ ($i<0$ or $j<0$) be $-\infty$.

$$p_{ij} = \max\{p_{i-1,j}+\xi, p_{i-1,j-1}+\psi(S_1(i), S_2(j)), p_{i,j-1}+\xi\} \tag{1}$$

It is obvious that this value can be obtained by dynamic programming. Note that this problem can be represented by the longest path problem on a grid-like graph as in Figure 1. Note also that there are also many heuristic algorithms for this problem, and many of these heuristic algorithms are based on dynamic programming. This simple dynamic programming can be easily extended to an algorithm that can deal with the affine gap penalty [7].

In the above formulation of the problem, we consider only deletion, insertion, and base substitution as evolutionary events. But there are many higher-order events such as recombination. As for the recombination, Kececioglu and Gusfield [14] formulated the *Recombination Distance Problem* in several ways and proposed algorithms for them. They considered two types of recombination: pure recombination and recombination with a point mutation. A sequence S_0 is formed from S_1 and S_2 by pure recombination if S_1 and S_2 can be cut at h locations into $h+1$ substrings ($S_1 = v_1v_2\cdots v_{h+1}$, $S_2 = w_1w_2\cdots w_{h+1}$) such that S_0 can be written as the concatenation of $h+1$ of these substrings, alternating between S_1 and S_2 (without loss of generality, $S_0 = v_1w_2v_3w_4\cdots$). We call these cuttings recombinations or crossovers. We call S_1 and S_2 parent sequences of S_0. In recombination with point mutation, errors are allowed in S_0: $S_0 = \tilde{v_1}\tilde{w_2}\tilde{v_3}\tilde{w_4}\cdots$, where $\tilde{s}$ differs from s by insertion, deletion, and base substitution. Let $d(v,v')$ be some appropriately defined score between substrings v and v'. The problem is to maximize the following value:

$$d(v_1,\tilde{v_1}) + d(w_2,\tilde{w_2}) + \cdots + h\cdot\chi, \tag{2}$$

where h is the number of recombinations and χ is the penalty for the recombination, which is preferably a negative value. Note that the most appropriate score for $d(v,v')$ is the ordinary alignment score.

The above formulation of recombination has several drawbacks:

- The case in which a child is formed from more than two sequences is not considered.
- Shuffling of the order of substrings is not allowed.
- The repetition of substrings is not allowed.

Because recombinations can lead to these three phenomena, these limitations are not desirable for analyzing recombinations. The formulation also has the following drawback:

- If we use the alignment score as $d(v, v')$, its computation time is $O(n^3)$, where n is the maximum length of the three sequences. It is hard to call this a practical bound, considering that the length of the sequences to be analyzed is often larger than 10,000. Note that we can solve this problem faster if we do not consider any point mutations, but it is unlikely from a biological viewpoint.

A new formulation of recombination is thus required.

2.2 New Problem Definition

We consider the following problem, which we call the *Recombination Alignment Problem*:

Definition 1 *The Recombination Alignment Problem is as follows. Let $d(v, v')$ be an appropriate score function for two strings v and v', χ be an appropriate penalty for a crossover. Given a sequence S_0 to be analyzed and a set of sequences $T = \{S_1, S_2, \ldots, S_k\}$ that are candidates for parents, find the subdivision of sequence $S_0 = v_1 v_2 \ldots v_{h+1}$ such that the following value is maximized:*

$$d(v_1, v_1') + d(v_2, v_2') + \cdots + d(v_{h+1}, v_{h+1}') + h \cdot \chi, \tag{3}$$

where v_i' is a substring of one of the sequences in T.

By this definition, we can deal with not only recombination but also shuffling of the order of substrings, tandem repeats, block insertion/deletion, and so on. We will consider algorithms for several versions of the definitions of $d(v, v')$.

2.3 Alignment Algorithm for Recombined Sequences

In this section, we will provide algorithms for three versions of the above *Recombination Alignment Problem.* At first, we do not permit any point mutations (deletions, insertions, and substitutions). This case is not a realistic model, but we can achieve a linear time algorithm in this case. Next we consider an algorithm for the version that permits point mutations. We then extend this algorithm to deal with affine gap penalties.

2.3.1 Recombination without a point mutation

In the recombination alignment problem without any point mutation, we let $d(v, v') = 0$ if $v = v'$, and $d(v, v') = +\infty$ otherwise. This version of the problem does not allow any point mutations such as deletions, insertions, and substitutions, which is not desirable from a biological viewpoint, but it can be solved in a time linear to the input size.

Matching statistics [4, 9] is a very useful notion for this problem:

Definition 2 *Consider two sequences S and T. Let $ms(i)$ be the length of the longest substring of S starting at position i that matches a substring somewhere in T. The set of matching statistics for S against T is the array of $ms(i)$, therefore its length is the same as the length of the sequence S.*

By using the suffix tree of T, it is known that the matching statistics can be obtained in $O(|S| + |T|)$ time if the alphabet size is constant [4].

Let `@` denote a character that does not appear anywhere in S_0. First, consider the sequence $S = S_1$+'`@`'+S_2+'`@`'+$\cdots$+'`@`'+S_k, where + means that a sequences is appended. Then compute the matching statistics for S_0 against S. This can be done in $O(N)$ time, where N is the sum of the lengths of all the input sequences. Let $rc(i)$ be the value defined by the following induction:

$$\begin{cases} rc(1) = 1 \\ rc(i+1) = rc(i) + ms(rc(i)) \quad (rc(i+1) \leq |S_0|) \end{cases} \tag{4}$$

Let h^+ be the largest i such that $rc(i)$ is defined in the induction above. Let $S[i \ldots j]$ denote the substring of S that starts at the ith base and ends at the jth base. Then consider the following subdivision of S_0: $S_0 = S_0[1 \ldots (rc(2) - 1)] + S_0[rc(2) \ldots (rc(3) - 1)] + \cdots + S_0[rc(h^+) \ldots S_0(|S_0|)$. Here, each substring has a same substring somewhere in at least one of the candidates for parents, and this subdivision achieves the largest score for this problem, that is, $(h^+ - 1) \cdot \chi$. Note that if the corresponding positions of these substrings in the candidates for parents are required, we can easily find them in $O(N)$ time by traversing the suffix tree of S or by using any other standard string matching algorithms [9].

2.3.2 Recombination with point mutations

In this section, we propose an algorithm for the *Recombination Alignment Problem* with point mutations. We consider the standard alignment score using the linear gap penalty as $d(v, v')$ in definition 3. Let $n_i = |S_i|$. In the following discussion, we use notations like $recomb()$ and $prev()$ which describe set of values of indices. For example, if we let $recomb(i)$ be $\{k, l\}$, $p_{j,recomb(i)}$ means $p_{j,k,l}$. For another example, "we set $\{i, recomb(i)\}$ to $prev(i, j, k)$" means that we let the first value of $prev(i, j, k)$ be i and the second and third values be the first and second values of $recomb(i)$ respectively. The algorithm is as follows:

Algorithm 1

1. *Let* $p_{0,j,l} = 0$ *for any* j *and* $l \geq 0$*, and* $p_{i,j,-1} = -\infty$ *for any* i *and* j*.*
2. *For* $i = 1, \dots, n_0$ *do the following:*
   (a) *For* $j = 1, \dots, k$ *do the following:*
      - *For* $l = 0, \dots, n_j$*, set the following value to* $p_{i,j,l}$*:*

$$p_{i,j,l} = \max\{p_{i-1,j,l-1} + \psi(S_0(i), S_j(l)), p_{i-1,j,l} + \xi, p_{i,j,l-1} + \xi\} \tag{5}$$

        Note that this expression is very similar to expression (1), which is used for the normal alignment problem. Let $prev(i, j, l)$ *be the values of* $\{i', j', l'\}$ *such that* $p_{i',j',l'}$ *determines the value of* $p_{i,j,l}$ *in the above expression.*

   (b) *Let* $recomb(i)$ *be the values of* $\{j, k\}$ *that give the largest value of* $p_{i,j,l}$ $(1 \leq j \leq k, 0 \leq l \leq n_j)$*.*
   (c) *For* $j = 1, \dots, k$ *do the following:*
      - *For* $l = 0, \dots, n_j$*, if* $p_{i,j,l} < p_{i,recomb(i)} + \chi$*, set* $p_{i,recomb(i)} + \chi$ *to* $p_{i,j,l}$*, and* $\{i, recomb(i)\}$ *to* $prev(i, j, l)$*.*

At the end of the algorithm, $p_{k,recomb(k)}$ is the maximized score for this problem. If we need only the score and do not need the alignment itself, the computing time is $O(n_0 \cdot N^-)$ and the space is $O(N^-)$, where $N^- = \sum_{1 \leq j \leq k} n_j$ which is $O(kn)$ letting n be the maximum length. This is because we do not have to remember old p, $prev$, and $recomb$ values. If we need the alignment, we can obtain it by tracing back $prev()$ as follows:

Algorithm 2

1. *Let* $i = k$ *and* $\{j, l\} = recomb(k)$*.*
2. *Let* $\{i', j', l'\} = prev(i, j, l)$*. If* $i' = i - 1$*,* $j' = j$ *and* $l' = l - 1$*, align* $S_0(i)$ *with* $S_j(l)$*. If* $i' = i - 1$*,* $j' = j$ *and* $l' = l$*, align* $S_0(i)$ *with a gap inserted between* $S_j(l)$ *and* $S_j(l + 1)$*. If* $i' = i$*,* $j' = j$ *and* $l' = l - 1$*, align a gap inserted between* $S_0(i)$ *and* $S_0(i + 1)$ *with* $S_j(l)$*. Otherwise, we assume that a recombination occurred at the position between* $S_0(i - 1)$ *and* $S_0(i)$*.*
3. *Let* $\{i, j, k\} = prev(i, j, k)$*. If* $i = 0$*, stop. Otherwise go to step 2.*

Tracing back requires only a time linear to the output alignment size. In this case, the total computing time is the same as above $(O(n_0 \cdot N^-))$, and the space is $O(n_0 \cdot N^-)$, because we have to remember $prev$ values. But if we do not require alignment but only the score, the required space is only $O(N^-)$. Note that this is a practical bound and the same as that for obtaining the optimal alignment of two sequences by dynamic programming.

2.3.3 Recombination alignment problem with affine gap penalty

The affine gap penalty is a strategy for imposing a penalty on the starting of consequent gaps. In the alignment problem, the algorithm for the linear gap penalty can be extended very easily [7]. The same technique also applies to our algorithm for the *Recombination Alignment Problem.*

The algorithm in detail is given in Appendix, which requires about three times as much computation time and space as the previous one for the linear gap penalty. The order of the computation time does not change: the time is $O(n_0 \cdot N^-)$, and the space is $O(N^-)$ if only the score is required, or $O(n_0 \cdot N^-)$ if the alignment is required.

Table 1: RNA sequences of HIVs and their pairwise similarity

Sequences			Pairwise Similarity (%)					
Name	Type	Length	*ss.D*	*ss.E*	*nr.20*	*nr.A*	*ns.1*	*ns.12*
sr.10	SI	5643	98.7	97.3	95.5	95.2	95.2	94.8
ss.D	SI	5642		98.1	95.1	95.2	95.5	95.2
ss.E	SI	5633			95.4	95.5	95.8	95.5
nr.20	NSI	5603				98.5	98.4	95.8
nr.A	NSI	5624					99.0	95.9
ns.1	NSI	5624						96.2
ns.12	NSI	5617						

2.3.4 Linear space divide-and-conquer algorithm

For the problem with point mutations, we have mentioned that we can compute it using $O(N^-)$ space if we require only the score. In this section, we consider how to obtain the alignment in $O(n_0 \cdot N^-)$ time and $O(N^-)$ space. This algorithm is a typical divide-and-conquer algorithm, and requires twice time as that of the above algorithms.

Notice that we can also compute the alignment by starting from the last character of the child sequence. Thus, by doing dynamic programming from both ends, we can obtain the node $p_{\lceil n_0/2 \rceil,j,l}$ that must be encountered in the tracing back procedure. It can be done in $O(n_0 \cdot N^-)$ time and $O(N^-)$ space because it does not require any tracing back procedure. Recursively, we can search for $p_{\lceil n_0/4 \rceil,j,l}$ and $p_{\lceil 3 \cdot n_0/4 \rceil,j,l}$ by doing the same way. We continue the recursive procedure until we obtain all the p's that will be encountered if we trace back. The computing time will be twice as the original algorithm. Note that it can be applied also to the case of affine gap penalty.

2.3.5 Considering reverse and/or complementary sequences

The reverse sequence is a sequence in which the order of the bases in the original sequence is reversed. DNA has four kinds of bases: `A`, `T`, `C`, and `G`. The complementary bases of `A`, `T`, `C`, and `G` are `T`, `A`, `G`, and `C`, respectively. The base and its complementary base sometimes combine with each other. A complementary sequence is one whose bases are replaced by their complementary bases.

It is known that a DNA sequence often contains reverse and/or complementary substrings of itself or other sequences. Thus it is also important to consider such substrings in aligning sequences. This is very easy to do: add the reverse and/or complementary sequences of the candidates for parents to the set of original candidates, and align them in the same way as in the previous sections.

2.3.6 The Bottleneck Recombination History Problem

Kececioglu and Gusfield [14] proposed the *Bottleneck Recombination History Problem*, which is a generalization of evolutionary tree construction problems. This problem includes the *Recombination Distance Problem* as a subproblem, and our models of recombination can also be applied to it. Their algorithm for the version of multiple crossovers without point mutation required $O(khN^2)$ preprocessing time, but our model can reduce this to $O(k^2N)$ time, where k is the number of the input sequences, h is the number of maximum crossovers, and N is the total length of all the input sequences. Furthermore, their algorithm for the version of multiple crossovers with point mutation required $O(N^3)$ preprocessing time, but our model can also reduce it to $O(kN^2)$ time.

2.4 Computational Experiments

It is known that HIV RNA sequences mutate very fast and recombinate with others very often. We carried out an experiment using the actual RNA sequences of HIVs, all of which were taken from the same patient. Table 1 shows the sequences used in the experiments and their pairwise similarity. According to the table, the most closely related sequence to *sr.10* seems to be *ss.D*, because the similarity between them is 98.7%, which is highest in the *sr.10* row.

```
sr.10      1  CCCTCAAATCACTCTTTGGCAACGACCCCT  ···  GGGAGTTGGAGGTTTTATCAAAGTAAGACA   174
nr.A       1  CCCTCAAATCACTCTTTGGCAACGACCCCT  ···  GGGAATTGGAGGTTTTATCAAAGTAAGACA   174

sr.10    175  GTATGATCAGATACCCATAGAAATCTGTGG  ···  CACTAATGATGTGAAACAATTAACAGAGGT  1410
nr.20    155  GTATGATCAGATACCCATAGAAATCTGTGG  ···  CACTAATGATGTAAAACAATTAACAGAGGT  1390

sr.10   1411  AGTGCAAAAAATAACCACAGAAAGCATAGT  ···  TTCTTGGGAGCAGCAGGAAGCACTATGGGC  5643
ss.D    1411  AGTGCAAAAAATAACCACAGAAAGCATAGT  ···  TTCTTGGGAGCAGCAGGAAGCACTATGGG-  5642
```

Figure 2: Optimal alignment taking account of recombinations

Figure 2 shows the optimal alignment of *sr.10* with the other sequences by our method. In the experiment, we used linear gap costs, and let the matching score be 1, the mismatching score be 0, the gap score be -1, and the recombination score be -10. According to the figure, the first 174 bases are aligned with the bases in *nr.A*, the next 1236 bases are aligned with the bases in *nr.20*, and the remaining bases are aligned with the bases in *ss.D*. This result claims that *sr.10* might be a recombined sequence of *nr.20* and *ss.D*. Note that we cannot know whether or not *sr.10* is in part a recombined sequence of *nr.A* and *nr.20*, because *nr.20* lacks the data of the first 20 bases.

3 Statistical Method for Detecting Recombinations

In the previous section, we presented a new alignment technique that can detect recombinations. But how confidently can we say that it is a recombined sequence? In this section, we propose a new statistical method for detecting recombination among sequences, which is totally different from the approach in the previous section and answers this question.

3.1 New statistical techniques

To detect recombination, there must be at least three sequences to analyze: we cannot detect recombination from only two sequences. From now on, we consider how to determine the possibility that there is a recombination site in given three sequences.

Let S_0 be a candidate for a child sequence, and S_1 and S_2 be candidates for parent sequences. Let some appropriate alignment of S_0 and S_1, and that of S_0 and S_2 be given. Let $l(x)$ denote the length of sequence x, $l(x,y)$ denote the length of the alignment of sequences x and y, and $d(x,y)$ denote the number of mutations found in the alignment of sequences x and y. From now on, consider x and y to be related to each other by only point mutations (*i.e.*, not by recombinations, repeats, or any other higher-order events). Let $t(x,y)$ be the evolutionary time between x and y, that is, $t(x,z)+t(z,y)$, where z is the most recent common ancestor of x and y. Consider the probability p that one fixed base in x and y mutates in a unit of evolutionary time. Then let $p(x,y)$ be the mutation probability of one base between x,y. It is computed as follows:

$$p(x,y) = 1-(1-p)^{t(x,y)} - q(x,y), \tag{6}$$

where $q(x,y) = p(x,y)^2/((1-p(x,y))\cdot(\sigma+p(x,y)-1))$ (σ: the size of alphabet) is the probability that a base mutates more than once and becomes the same as the original base. If $p(x,y)$ and p are small enough, we can let $p(x,y) = p\cdot t(x,y)$. Let $\hat{p}(x,y) = d(x,y)/l(x,y)$ be the estimate for this probability.

Let u and v be arbitrary, non-overlapping substrings of S_0. Let u' and v' be the substrings of S_1 corresponding to u and v (*i.e.*, located in the same region of the alignment of S_0 and S_1). Let u'' and v'' be the substrings of S_2 that correspond to u and v.

If there are no evolutionary events other than point mutations between S_0 and S_1, and also between S_0 and S_2, there must be some τ such that the following equations hold:

$$\begin{aligned} p(u,u') &= \tau\cdot p(u,u''). && (7)\\ p(v,v') &= \tau\cdot p(v,v''). && (8) \end{aligned}$$

We can test for the existence of recombinations by testing for the non-existence of τ in these equations. Note that even if τ exists, it does not imply the non-existence of recombinations. For example, recombinations between same sequences cannot be detected (by any method, of course).

The number of point mutations in a sequence of length n, whose point mutation probability is p, follows the binomial distribution $Bi(n,p)$. The probability function of binomial distribution often requires a large computation time, hence we often approximate this distribution to the normal distribution $N(p, p(1-p))$ by the central limit theorem in cases where np is large enough ($np > 5$). Let d be the number of point mutations that occurred in the sequence of length n, and $\hat{p} = d/n$ be the estimate for the point mutation probability. Then the following equations hold:

$$P(p \geq \hat{p} + Z_\alpha\sqrt{\hat{p}(1-\hat{p})/n} + 1/(2n)) \quad \approx \quad \alpha, \tag{9}$$

$$P(p \leq \hat{p} - Z_\alpha\sqrt{\hat{p}(1-\hat{p})/n} - 1/(2n)) \quad \approx \quad \alpha, \tag{10}$$

where $P()$ denotes the probability of the inequality in the equation, and Z_α denotes the upper $(1-\alpha)$ confidence limit of the standard normal distribution $N(0,1)$: $\int_{-\infty}^{Z_\alpha} \exp(-x^2/2)dx/\sqrt{2\pi} = 1-\alpha$. Note that $+1/(2n)$ and $-1/(2n)$ in these expressions are called 0.5 corrections for the binomial distribution.

Without loss of generality, we can let $\hat{p}(u,u')/\hat{p}(u,u'') > \hat{p}(v,v')/\hat{p}(v,v'')$. Note that if $\hat{p}(u,u')/\hat{p}(u,u'') = \hat{p}(v,v')/\hat{p}(v,v'')$, τ can exist and we cannot tell whether crossovers exist or not. Let $r(x,y) = \sqrt{\hat{p}(x,y)(1-\hat{p}(x,y))/l(x,y)}$. Then the probability of the existence of τ is less than 4α if the following inequality holds:

$$\begin{aligned}&\{\hat{p}(u,u') - Z_\alpha \cdot r(u,u') - 1/(2l(u,u'))\}/\{\hat{p}(u,u'') + Z_\alpha \cdot r(u,u'') + 1/(2l(u,u''))\} \\ &\quad \geq \{\hat{p}(v,v') + Z_\alpha \cdot r(v,v') + 1/(2l(v,v'))\}/\{\hat{p}(v,v'') - Z_\alpha \cdot r(v,v'') - 1/(2l(v,v'))\} > 0.\end{aligned} \tag{11}$$

If this inequality holds, at least one of the four values $p(u,u')$, $p(u,v')$, $p(u,u'')$, and $p(v,v'')$ must lie outside the $1-\alpha$ confidence region, which means that the probability of the existence of τ is less than α. Note that if the number of point mutations is small, we can test for the existence of τ by means of the binomial distribution itself. It will take far more computation time, but the result will be more accurate.

The above inequality can test the existence of the τ, but we cannot know any probability. Consider z that satisfies the following expressions:

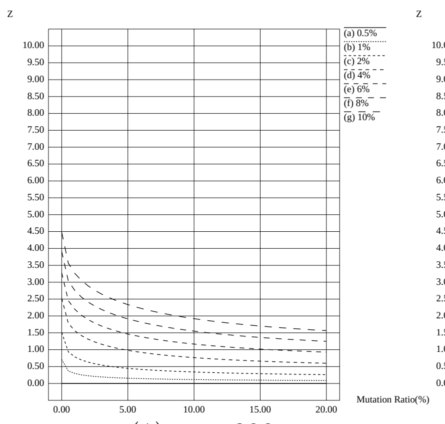
$$\begin{aligned}&\{\hat{p}(u,u') - z \cdot r(u,u') - 1/(2n)\}/\{\hat{p}(u,u'') + z \cdot r(u,u'') + 1/(2n)\} \\ &\quad = \{\hat{p}(v,v') + z \cdot r(v,v') + 1/(2n)\}/\{\hat{p}(v,v'') - z \cdot r(v,v'') - 1/(2n)\} \\ &\qquad (\hat{p}(u,u') > z \cdot r(u,u'), \quad \hat{p}(u,u'') > z \cdot r(u,u''), \quad z > 0)\end{aligned} \tag{12}$$

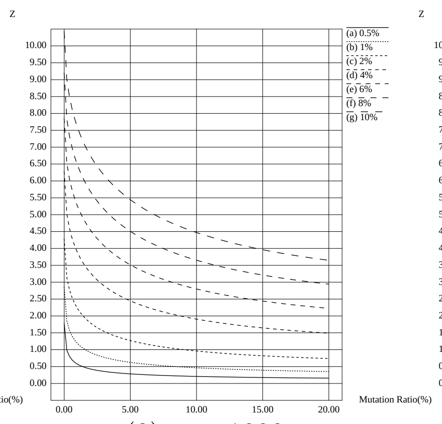
If there is no such value, let z be 0. We call this a z-value. If $z > Z_\alpha$, the probability of the existence of τ is less than 4α. For example, if $z = 3.0$, the probability of the existence of τ is only 0.54% at most, and we can conclude that there is a high possibility of recombination. This probability becomes larger if z becomes smaller. For example, if $z = 2.0$, the probability is about 9.1%.

Since we do not know where a crossover occurs, we must test several selected sets of u and v from S_0. z or α can be computed in constant time, and therefore the speed of this method depends on the number of these selected sets. The simplest way is to divide S_0 into two parts at every position, which requires only $O(l(x,y))$ time, because $d()$'s can be computed in a total time of $O(l(x,y))$. Another simple way is to test every consequent two substrings of fixed length, which also requires only $O(l(x,y))$ time. These are very reasonable and practical bounds. We call the latter technique the sliding window method. It may be useful if the sequence to be tested is a recombination of more than 2 sequences. Note also that the sites of crossovers can be predicted by plotting z-values along the alignment: if there is a site with a large z-value, there is a high probability that recombination occurred around that site.

3.2 Computational Experiments

First, we examined properties of z-values using artificial sequences. Consider two similar and evolutionarily related sequences of length $2n$, and let them be A and B. Let A_1 and A_2 be A's substrings of length n such that $A = A_1 + A_2$. In the same way, let B_1 and B_2 be B's substrings of length n such that $B = B_1 + B_2$. To simplify the problem, we let both the mutation ratio between


(1) $n = 200$


(2) $n = 1000$

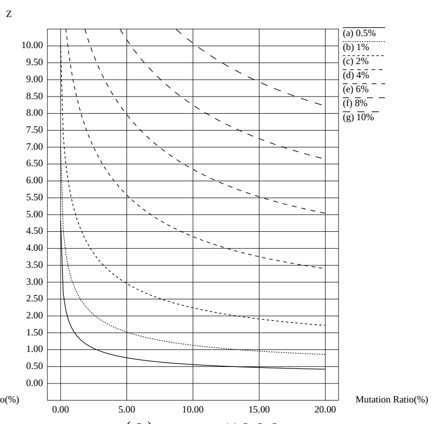

(3) $n = 5000$

Figure 3: Relationships between the z value and the time elapsed since the recombination

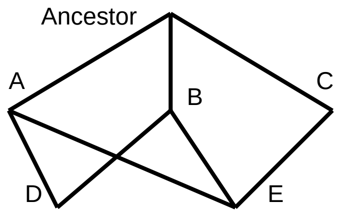


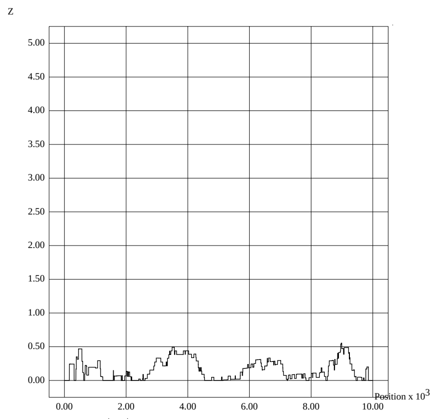
Figure 4: Phylogenetic diagram of artificial sequences to be tested.

A_1 and B_1, and that between A_2 and B_2 be p (*i.e.*, A_i (B_i) has $p \cdot n$ mutations from B_i (A_i)). In other words, the similarities between them are both $1 - p$. Then consider a recombination of A and B that produces a new sequence $C = A_1 + B_2$. Ordinarily, observed recombinated sequences have mutated after the event of recombination in most cases. Hence we consider $A' = A'_1 + A'_2$, $B' = B'_1 + B'_2$, and $C' = C'_1 + C'_2$ be the mutated sequences respectively. To simplify the problem, we assume that the number of the mutations in each substring is the same, and let it be $n \cdot q/2$. We also assume that all the mutations (including those between A and B) occur at different positions. In this situation, the mutation ratio between A'_1 and C'_1, and that between B'_2 and C'_2 are both q, while the mutation ratio between B'_1 and C'_1, and that between A'_2 and C'_2 are both $p + q$.

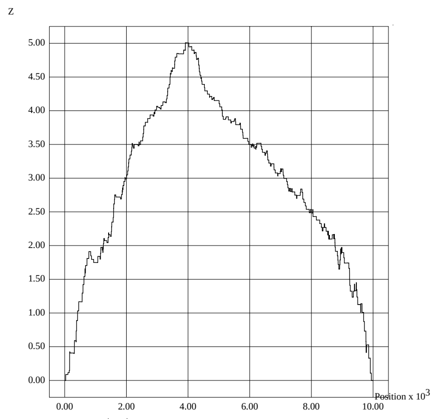
Figure 3 shows the relationships between z-values and q (the mutation ratio, which represents the time elapsed since the recombination) for various values of n ((1) 200, (2) 1000, (3) 5000) and p ((a) 0.5%, (b) 1%, (c) 2%, (d) 4%, (e) 6%, (f) 8%, (g) 10%). As we mentioned in the previous section, if the z-value is larger than 3.0, the possibility of the existence of τ is smaller than 0.54%. Let us use this value for detecting recombinations in these cases. In case (1), in which the sequences are short ($n = 200$), we cannot detect the recombinations of A and B if $p = 0.02$ (*i.e.*, if the similarity between them is at least 98%). Even if $p = 0.1$, we cannot detect recombinations if q is larger than 2%. It seems that $n = 200$ is not large enough for detecting recombinations. In case (2), we can always detect recombinations in the experiments if $p \geq 0.08$. In case (3), we can always detect recombinations if $p \geq 0.04$. (Note that we do not consider the cases where $q > 0.2$.) These observations reveal the following facts:

- Detection becomes easier as n becomes larger. This is a very important fact. We cannot determine the existence of recombinations if the given sequences are too short.
- Detection becomes easier as p becomes larger. This means that recombinations between similar sequences are difficult to detect.
- Detection becomes more difficult as q becomes larger. This means that old recombinations are difficult to detect.

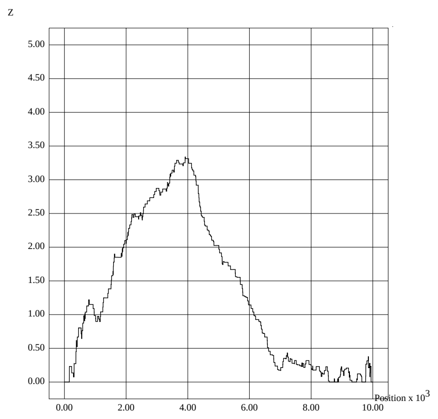
Note that in some actual cases, the sequences may be influenced by some unknown higher order phylogenetic events other than recombination if n becomes too large.



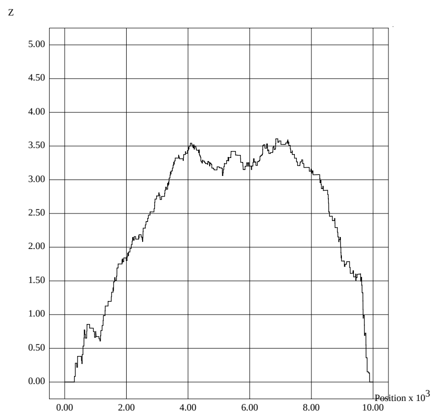
(1) C *vs.* A and B



(2) D *vs.* A and B



(3) E *vs.* A and B



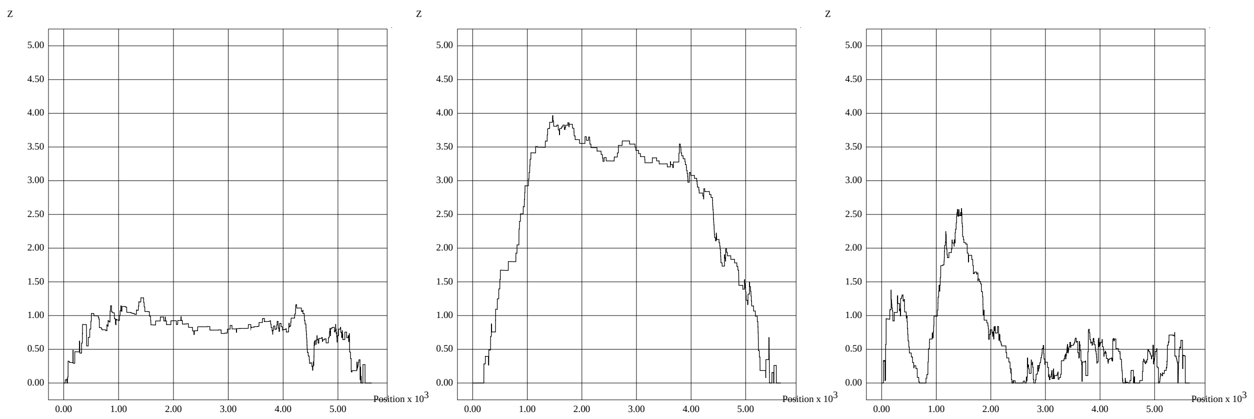
(4) E *vs.* A and C

Figure 5: z-values over artificial sequences

We also conducted experiments to detect the recombination site. For these experiments, we constructed 6 sequences. First, we constructed an arbitrary sequence of length 10000 (which is the ancestor of all the other sequences). Then, we constructed three sequences A, B, and C by mutating randomly chosen 1% of the bases of the ancestor. Let $S[i \dots j]$ denote the substring of S that starts at the ith base and ends at the jth base. We then constructed a sequence D by mutating 1% of the bases of the recombined sequence $A[1 \dots 4000] + B[4001 \dots 10000]$. We also constructed sequence E by mutating 1% of the bases of the recombined sequence $A[1 \dots 4000] + B[4001 \dots 7000] + C[7001 \dots 10000]$. Figure 4 shows the phylogenetic diagram of these constructed sequences.

In the experiments, we plotted z-values along the aligned sequences by dividing the alignment to two aligned substrings at each position (see the previous subsection). Figure 5 shows the results of the experiments. Case (1) is to test whether C is a recombined sequence of A and B. In this case, z-values are smaller than 0.5 at most positions. Actually, C is not a recombined sequence of A and B, so it is a reasonable result. Case (2) is to test whether D is a child of A and B. The z-value becomes largest (about 5.0) around the position 4000, which means that our method succeeded in correctly detecting not only the existence of a recombination but also the site of it. Cases (3) and (4) are to test whether E is a child of A and B, or a child of A and C. In both cases, our method correctly detected the existence recombination. In case (3), our method detects the recombination site of A and B. But in case (4), it is difficult to determine the recombination site from the result (the peaks at the positions 4000 and 7000 are not clear). This is because the part of the substring of A and that of C are widely separated. The experiment revealed that our method can detect recombinations of more than two sequences to some extent.

We also conducted an experiment using actual RNA sequences of HIVs. In the experiment, we used 4 sequences (*ns.12, sr.10, ss.D,* and *nr.20*) of the HIV RNA sequences used in the last section.



(1) *ns.12 vs. ss.D and nr.20* (2) *sr.10 vs. ss.D and nr.20* (3) by the sliding window technique

Figure 6: z-values over HIV sequences

Before the experiment, we deleted the columns with gaps from the alignment, because gaps are often produced by higher-order events other than point mutations. In the experiment, we tested whether *ns.12* and *sr.10* are recombinated sequences of *ss.D* and *nr.20*. Figure 6 shows the result. Experiment (1) showed that *ns.12* cannot be said to be a child of *ss.D* and *nr.20*. Experiment (2) confidently insists that *sr.10* is a recombinated sequence of *ss.D* and *nr.20*. But it is difficult to know where the crossover occurred from this experiment. Experiment (3) uses the sliding window technique with a window size of 2000; that is, we did not count the mutations farther than $1000 = 2000/2$ from the dividing position in computing the z-value. The result indicated that the site may be around the 1450th base. This experiment revealed that the sliding window technique is a good choice for detecting the site of the recombination. But note that the largest z-value in this experiment is smaller than that in experiment (2). In this way, we can justify the results of the experiments in section 2.4.

4 Concluding Remarks

In this paper, we have proposed two different approaches for detecting recombinations. One approach is an alignment method that takes account of recombinations. It has a practical bound of $O(kn^2)$, where k is the number of sequences and n is the maximum length of the sequences. The other is a statistical method that detects recombinations by computing values called z-values. For both techniques, we conducted experiments using actual RNA sequences of HIVs and artificially recombined sequence data. In future work, we need to determine an appropriate recombination penalty (see section 2). It would also be interesting to consider how to detect statistically a recombined sequence formed from many parent sequences.

Acknowledgements

The author wishes to thank Dr. Shinichi Oka, Dr. Setsuko Ida, and Ms. Saori Aizawa at the International Medical Center of Japan, for introducing him to the problem of recombination in HIV, and for providing him with several useful data for computational experiments.

References

[1] G. Benson, "Sequence Alignment with Tandem Duplication," *J. Comput. Biol., Vol. 4, No. 3,* 1997, pp. 351-367.

[2] E. Bertran, J. Rozas, A. Navarro, and A. Barbadilla, "The Estimation of the Number of the Length Distribution of Gene Conversion Tracts from Population DNA Sequence Data," *Genetics, Vol. 146,* 1997, pp. 89-99.

[3] J. K. Carr et al., "Full-Length Sequence and Mosaic Structure of a Human Immunodeficiency Virus Type 1 Isolate from Thailand," *J. Virology,* September 1996, pp. 5935-5943.

[4] W. I. Chang and E. L. Lawler, "Sublinear Expected Time Approximate String Matching and Biological Applications," *Algorithmica, Vol. 12,* 1994, pp. 327-344.

[5] K. D. Chenault and U. Melcher, "Phylogenetic Relationships Reveal Recombination among Isolates of Cauliflower Mosaic Virus," *J. Mol. Evol., Vol. 39,* 1994, pp. 496-505.

[6] D. R. Forsdyke, "A Stem-Loop "Kissing" Model for the Initiation of Recombination and the Origin of Introns," *Mol. Biol. Evol., Vol. 12, No. 5,* 1995, pp. 949-958.

[7] O. Gotoh, "An Improved Algorithm for Matching Biological Sequences," *J. Mol. Biol.* Vol. 162, 1982, pp. 705–708.

[8] R. C. Griffiths and P. Marjoram, "Ancestral Inference from Samples of DNA Sequences with Recombination," *J. Comput. Biol., Vol. 3, No. 4,* 1996, pp. 479-502.

[9] D. Gusfield, "Algorithms on Strings, Trees, and Sequences: computer science and computational biology," *Cambridge University Press,* 1997.

[10] J. Hein, "Reconstructing the Evolution of Sequences Subject to Recombination Using Parsimony," *Mathematical Biosciences 98,* 1990, pp. 185-200.

[11] J. Hein, "A Heuristic Method to Reconstruct the History of Sequences Subject to Recombination," *J. Molecular Evolution, Vol. 36,* 1993, pp. 396-405.

[12] R. R. Hudson, "Estimating the Recombination Parameter of a Finite Population Model without Selection," *Genet. Res. Comb., Vol. 50,* 1987, pp. 245-250.

[13] R. R. Hudson and N. L. Kaplan, "Statistical Properties of the Number of Recombination Events in the History of a Sample of DNA Sequences," *Genetics, Vol. 111,* 1985, pp. 147-164.

[14] J. Kececioglu and D. Gusfield, "Reconstructing a History of Recombinations from a Set of Sequences," *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms,* 1994, pp. 471-480.

[15] B. Ma, L. Wang. and M. Li, "Fixed Topology Alignment with Recombination," *Proc. 9th Annual Symposium on Combinatorial Pattern Matching (CPM98), Springer-Verlag LNCS 1448,* 1998, pp. 174-188.

[16] L. Moutouh, J. Corbeil, and D. D. Richman, "Recombination Leads to the Rapid Emergence of HIV-1 Dually Resistant Mutants under Selective Drug Pressure," *Proc. Natl. Acad. Sci. USA, Vol. 93,* June 1996, pp. 6106-6111.

[17] S. B. Needleman and C. D. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins," *J. Mol. Biol., Vol.48,* 1970, pp. 443-453.

[18] D. L. Robertson, B. H. Hahn, and P. M. Sharp, "Recombination in AIDS Viruses," *J. Mol. Evol., Vol. 40,* 1995, pp. 249-259.

[19] D. L. Robertson, P. M. Sharp, F. E. McCutchan, and B. H. Hahn, "Recombination in HIV-1," *Nature, vol. 374,* 9 March, 1995, pp. 124-126.

[20] J. Rozas and R. Rozas, "DnaSP Version 2.0: A Novel Software Package for Extensive Molecular Population Genetics Analysis," *CABIOS, Vol. 13, No. 3,* 1997, pp. 307-311.

[21] K. Sherefa, B. Johansson, M. Salminen, and A. Sonnerborg, "Full-Length Sequence of Human Immunodeficiency Virus Type 1 Subtype A, Recombined with Subtype C in the *env* V3 Domain," *AIDS Research and Human Retroviruses, Vol. 14, No. 3,* 1998, pp. 289-292.

[22] A. C. Siepel, A. L. Halpern, C. Machen, and B. T. M. Korber, "A Computer Program Designed to Screen Rapidly for HIV Type 1 Intersubtype Recombinant Sequences," *AIDS Research and Human Retroviruses, Vol. 11, No. 11* 1995, pp. 1413-1416.

[23] A. C. Siepel and B. T. Korber, "Scanning the Database for Recombinant HIV-1 Genomes," *The Human Retroviruses and AIDS, 1995 Compendium,* Part III, 1995, pp. 35-60.

[24] K. L. Simonsen and G. A. Churchill, "A Markov Chain Model of Coalescence with Recombination," *Theor. Popul. Biol., Vol. 52,* 1997, pp. 43-59.

[25] T. F. Smith and M. S. Waterman, "Identification of Common Molecular Subsequences," *J. Mol. Biol., Vol. 147,* 1981, pp. 195-197.

[26] M. S. Waterman, "Introduction to Computational Biology: Maps, Sequences and Genomes," Chapman & Hall, 1995.

[27] G. F. Weiller, "Phylogenetic Profiles: A Graphical Method for Detecting Genetic Recombinations in Homologous Sequences," *Mol. Biol. Evol., Vol. 15, No. 3,* 1998, pp. 326-335.

Appendix: The Algorithm for Recombination Alignment Problem with Affine Gap Penalty

Let κ be the penalty imposed on the starting of gaps. Note that *recomb*() and *prev*() in the following descriptions are different from those in the subsection—2.3.2. Then the algorithm is as follows:

Algorithm 3

1. *Let* $p_{0,j,l,x} = 0$ *for any* j, $l \geq 0$, *and* x. *Let* $p_{i,j,-1,x} = -\infty$ *for any* i, j, *and* x.
2. *For* $i = 1, \ldots, n_0$ *do the following:*

   (a) *For* $j = 1, \ldots, k$ *do the following:*
   - *For* $l = 0, \ldots, n_j$, *set the value of* $p_{i,j,l,x}$ *as follows:*

$$p_{i,j,l,1} = \psi(S_0(i), S_j(l)) + \max\{p_{i-1,j,l-1,1}, p_{i-1,j,l-1,2}, p_{i-1,j,l-1,3}\} \quad (13)$$
$$p_{i,j,l,2} = \xi + \max\{p_{i-1,j,l,1} + \kappa, p_{i-1,j,l,2}, p_{i-1,j,l,3} + \kappa\} \quad (14)$$
$$p_{i,j,l,3} = \xi + \max\{p_{i,j,l-1,1} + \kappa, p_{i,j,l-1,2} + \kappa, p_{i,j,l-1,3}\} \quad (15)$$

   Let $prev(i,j,l,x)$ *be the values of* $\{i', j', l', x'\}$ *such that* $p_{i',j',l',x'}$ *determines the value of* $p_{i,j,l,x}$ *in the above expression.*

   (b) *Let* $recomb(i)$ *be the values of* $\{j, k, x\}$ *that give the largest value of* $p_{i,j,l,x}$ $(1 \leq j \leq k, 0 \leq l \leq n_j, 1 \leq x \leq 3)$.

   (c) *For* $j = 1, \ldots, k$ *do the following:*
   - *For* $l = 0, \ldots, n_j$ *and* $x = 1, 2, 3$, *if* $p_{i,j,l,x} < p_{i,recomb(i)} + \chi$, *set* $p_{i,recomb(i)} + \chi$ *to* $p_{i,j,l,x}$, *and* $\{i, recomb(i)\}$ *to* $prev(i,j,l,x)$.

The tracing algorithm for obtaining the alignment is as follows:

Algorithm 4

1. *Let* $i = k$ *and* $\{j, l, x\} = recomb(k)$.
2. *Let* $\{i', j', l', x'\} = prev(i,j,l,x)$. *If* $i' = i - 1$, $j' = j$ *and* $l' = l - 1$, *align* $S_0(i)$ *with* $S_j(l)$. *If* $i' = i - 1$, $j' = j$ *and* $l' = l$, *align* $S_0(i)$ *with a gap inserted between* $S_j(l)$ *and* $S_j(l+1)$. *If* $i' = i$, $j' = j$ *and* $l' = l - 1$, *align a gap inserted between* $S_0(i)$ *and* $S_0(i+1)$ *with* $S_j(l)$. *Otherwise, we assume that a recombination occurred at the position between* $S_0(i-1)$ *and* $S_0(i)$.
3. *Let* $\{i, j, k, x\} = prev(i,j,k,x)$. *If* $i = 0$, *stop. Otherwise, go to step 2.*

This algorithm requires about three times as much computation time and space as the previous one for the linear gap penalty. The order of the computation time does not change: the time is $O(n_0 \cdot N^-)$, and the space is $O(N^-)$ if only the score is required, or $O(n_0 \cdot N^-)$ if the alignment is required.