CONTEXT-FREE CODON ALIGNMENT

CONTEXT-FREE CODON ALIGNMENT

By

BIN WU, B.SC.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (1998)	MCMASTER UNIVERSITY
(Computer Science)	Hamilton, Ontario

TITLE:	Context-free Codon Alignment
AUTHOR:	Bin Wu B.Sc. (Northwestern University, China)
SUPERVISOR:	Dr. Tao Jiang
NUMBER OF PAGES:	x, 93

Abstract

We study an alignment model for coding DNA sequences recently proposed by J. Hein in [4] that takes into account both DNA and protein information, and attempts to minimize the total amount of evolution at both DNA and protein levels[4,5,6]. Although there are two quadratic algorithms (*i.e.* Hua-Jiang algorithm[8] and PLH algorithm[9]) for Hein's model if the gap penalty function is affine, both of them are impractical because of the large constant factor embedded in the quadratic time complexity function. We therefore consider a mild simplification named *Context-free Codon Alignment* and present a much more efficient algorithm for the simplified model. The algorithms have been implemented and tested on both real and simulated sequences, and it is found that they produce almost identical alignments in most cases. Furthermore, we extend our model and design a heuristic algorithm to handle frame-shift errors and overlapping frames.

Acknowledgments

First, I would like to thank Dr. Tao Jiang, my supervisor, for granting the opportunity, and for his invaluable supervision in guiding this work.

I would also like to thank the following people for their help:

- Dr. William F. Smyth and Dr. Sanzheng Qiao for their agreeing to read this thesis.

- Dr. Xian Zhang and Miss Yufang Hua for their helpful discussions.

- Mr. Dan Trottier for his practical advice and technical assistance in the laboratory.

Finally, I would like to thank my parents and family for their love and everlasting support.

Contents

List of Figures

List of Tables

Chapter 1

An Introduction to Codon Based Alignment

We first give an overview of the problem of comparing genomic sequences in Section 1.1. The formal definition of codon based alignment is presented in Section 1.2. Finally, we preview our main results in Section 1.3.

1.1 Overview

Genomic sequence alignment is a model of comparing DNA or protein sequences under the assumptions that (i) insertion, deletion, and mutation are the elementary evolutionary events and (ii) evolution usually takes the most economic course. Classical alignment algorithms either align DNA sequences based on DNA evolution or align protein sequences based on protein evolution. It is well known that protein evolves slower than its coding DNA, and alignments of protein are usually more reliable than that of the underlying DNA.

We are interested in the alignment of coding DNA sequences. It is clearly desirable that an alignment of coding DNA sequences incorporate the information from their protein sequences. A straightforward method is to align the protein sequences first and then back-translate the alignment into DNA. The method has several shortfalls including (i) it forces insertions and deletions (abbreviated as *indels*) to occur at *codon* [1] boundaries and (ii) it ignores homologies at the DNA level.

In 1994, Jotun Hein proposed a new model of DNA sequence alignment where evolutionary changes at both the DNA and protein levels are dealt with simultaneously[4]. The basic idea of Hein's model is that in computing an alignment, we consider each nucleotide mutation and indel, and penalize it appropriately taking into account any amino acid change it might induce. The model allows indels to occur within codons and assumes that each indel involves a multiple of three nucleotides so that the *reading frame* [2] never changes during the evolution. A *gap* (*i.e.* a block of consecutive spaces;

[1]A codon is a triple of nucleotides which encodes an amino acid (see Table 1.1).

[2]Roughly speaking, the reading frame in a coding DNA depicts where the codons begin.

Amino Acids	Codons
Ala	GCT GCC GCA GCG
Arg	CGT CGC CGA CGG AGA AGG
Asn	AAT AAC
Asp	GAT GAC
Cys	TGT TGC
Gln	CAA CAG
Glu	GAA GAG
Gly	GGT GGC GGA GGG
His	CAT CAC
Ile	ATT ATC ATA
Leu	TTA TTG CTT CTC CTA CTG
Lys	AAA AAG
Met	ATG
Phe	TTT TTC
Pro	CCT CCC CCA CCG
Ser	AGT AGC TCT TCC TCA TCG
Thr	ACT ACC ACA ACG
Trp	TGG
Tyr	TAT TAC
Val	GTT GTC GTA GTG

Table 1.1: Codons map to amino acids

representing an indel) of length i is penalized with a cost $g(i)$, where g is any positive function satisfying $g(i) + g(j) \geq g(i+j)$. A dynamic programming algorithm is demonstrated in [4] for computing optimal alignment in this model that runs in $O(m^2n^2)$ time, where m and n are the lengths of the two DNA sequences aligned. The algorithm is too slow to be useful in practice even for moderate m and n. It is left as an open question in [4] whether the time complexity can be improved to $O(mn)$ when the gap penalty function is affine, *i.e.* $g(i) = g_{open} + i * g_{ext}$ for some constants g_{open} and g_{ext} where g_{open} is the cost of opening an indel and g_{ext} is the cost of extending an indel. Affine functions are perhaps the most popular gap function among computational biologists. A fast heuristic algorithm for the problem, assuming affine gaps, is proposed in [5,6] which does not guarantee an optimal alignment.

1.2 Codon alignment and Hein's model of genomic sequence comparison

Let $A = a_1a_2a_3...a_{3m-2}a_{3m-1}a_{3m}$ and $B = b_1b_2b_3...b_{3n-2}b_{3n-1}b_{3n}$ be two coding DNA sequences consisting of m and n codons respectively. Each sequence has a fixed reading frame starting at the first base. An *alignment* of A and B is a correspondence between the bases in A and B, and postulates a possible evolution from A and B in terms of single nucleotide mutations

```
T – – – T G
T T G C T C
```

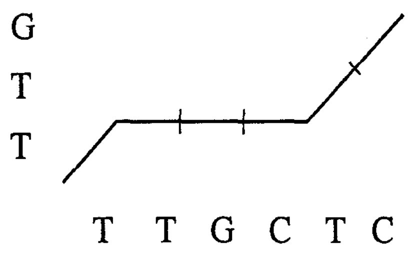

Figure 1.1: An alignment and its corresponding path representation.

and indels of blocks of nucleotides. An alignment can also be conveniently expressed as a path in a grid graph. Figure 1.1 demonstrates an alignment from TTGCTC to TTG and the corresponding path. It postulates that a mutation from C to G and a deletion of TGC have happened in the evolution from TTGCTC to TTG.

Since indels of length other than a multiple of three change the reading frame and hence the entire protein, for simplicity, Hein assumes that all indels have lengths divisible by three as in [4,5,6].

The cost of an alignment between A and B is decided by both the evolutionary events of the nucleotides postulated by the alignment and the evolutionary changes at the protein level. We will look at the three events mutation, insertion and deletion separately. For each pair of nucleotides a and b, let $c_d(a,b)$ denote the cost of substituting b for a, without worrying about the effect of this change at the protein level. For each pair of codons $e_1e_2e_3$ and $f_1f_2f_3$, let $c_p(e_1e_2e_3, f_1f_2f_3)$ denote the cost of substituting the amino acid coded by $f_1f_2f_3$ for the amino acid coded by $e_1e_2e_3$. For any integer i, functions $g_d(i)$ and $g_p(i)$ denote the costs of inserting (or deleting) a block of i nucleotides and a block of i amino acids, respectively. For convenience, let $g(i) = g_d(3i) + g_p(i)$.

- **Mutation**. The *combined cost* of a nucleotide mutation $e_1 \rightarrow f_1$ in codon $e_1e_2e_3$ is

  $$c_d(e_1, f_1) + c_p(e_1e_2e_3, f_1e_2e_3).$$

  The combined costs of mutations at the second or third positions of a codon are defined in a similar way.

- **Insertion**. Consider the event of inserting $3i$ nucleotides $f_1...f_{3i}$ in the codon $e_1e_2e_3$. If the insertion happens to the immediate left of e_1 or the immediate right of e_3, its combined cost is simply $g(i)$. Otherwise suppose that the string $f_1...f_{3i}$ is inserted between the nucleotides e_1 and e_2. Then the combined cost of the insertion is

  $$g(i) + min\{c_p(e_1e_2e_3, e_1f_1f_2), c_p(e_1e_2e_3, f_{3i}e_2e_3)\}.$$

  The case when the insertion happens between the nucleotides e_2 and e_3 is handled similarly.

- **Deletion**. This is symmetric to insertion. Consider the event of deleting $3i$ nucleotides from a sequence of $i+1$ codons $e_1e_2e_3...e_{3i+1}e_{3i+2}e_{3i+3}$.

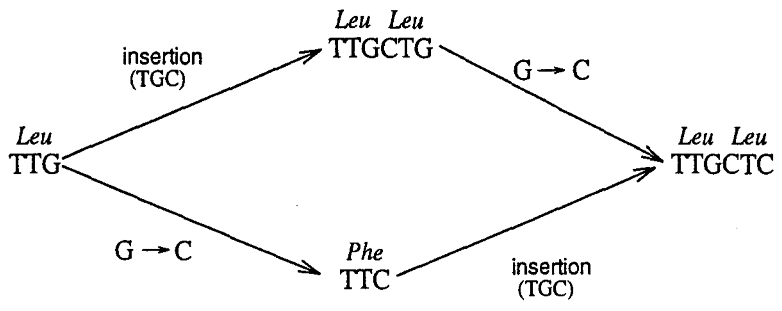

Figure 1.2: Different orders yield different costs.

If the deletion happens at e_1 or e_4, its combined cost is simply $g(i)$. Otherwise suppose that the string $e_2 ... e_{3i+1}$ is deleted. Then the combined cost of deletion is

$$g(i) + min\{c_p(e_1e_2e_3, e_1e_{3i+2}e_{3i+3}), c_p(e_{3i+1}e_{3i+2}e_{3i+3}, e_1e_{3i+2}e_{3i+3})\}.$$

The case when $e_3 ... e_{3i+2}$ is deleted can be handled similarly.

Although an alignment of A and B postulates a set of evolutionary events that transform A into B, it does not specify the order that the events should take place. In fact, all permutations of the events are possible. However, different permutations may yield different overall combined costs. For example, in Figure 1.2, the overall combined cost is

$$g(1) + min\{c_p(TTG, TTG), c_p(TTG, CTG)\} + c_d(G, C) + c_p(CTG, CTC)$$

if the insertion happens first or

$$c_d(G, C) + c_p(TTG, TTC) + g(1) + min\{c_p(TTC, TTG), c_p(TTC, CTC)\}$$

if the mutation happens first. In other words, the evolutionary events are no longer independent when it comes to computing the combined cost. An event may influence the cost of other events. Therefore, we define the cost of an alignment of A and B as the minimum overall combined cost among all possible permutations of the evolutionary events postulated by the alignment. An *optimal alignment* is one with the minimum cost.

Computing an optimal alignment of A and B is not an easy task due to the influence between events. The notion of a *codon alignment* introduced in [4] will help simplify the matter and is accepted by computational biologists. An alignment of A and B is called a *codon alignment* if

1. m = 0 or

2. n = 0 or

3. There do not exist i and j, $1 \leq i \leq 3m$ and $1 \leq j \leq 3n$, such that a_i is aligned with b_j, and (i) $i \bmod 3 = j \bmod 3 = 1$ and $i + j > 2$ or (ii) $i \bmod 3 = j \bmod 3 = 0$ and $i + j < 3m + 3n$.

In other words, except in the first and last columns, a codon alignment does not align a base at some codon boundary of A with a base at any codon boundary of B. For example, the alignment in Figure 1.1 is in fact a codon alignment. The cost of a codon alignment is defined the same way as for an

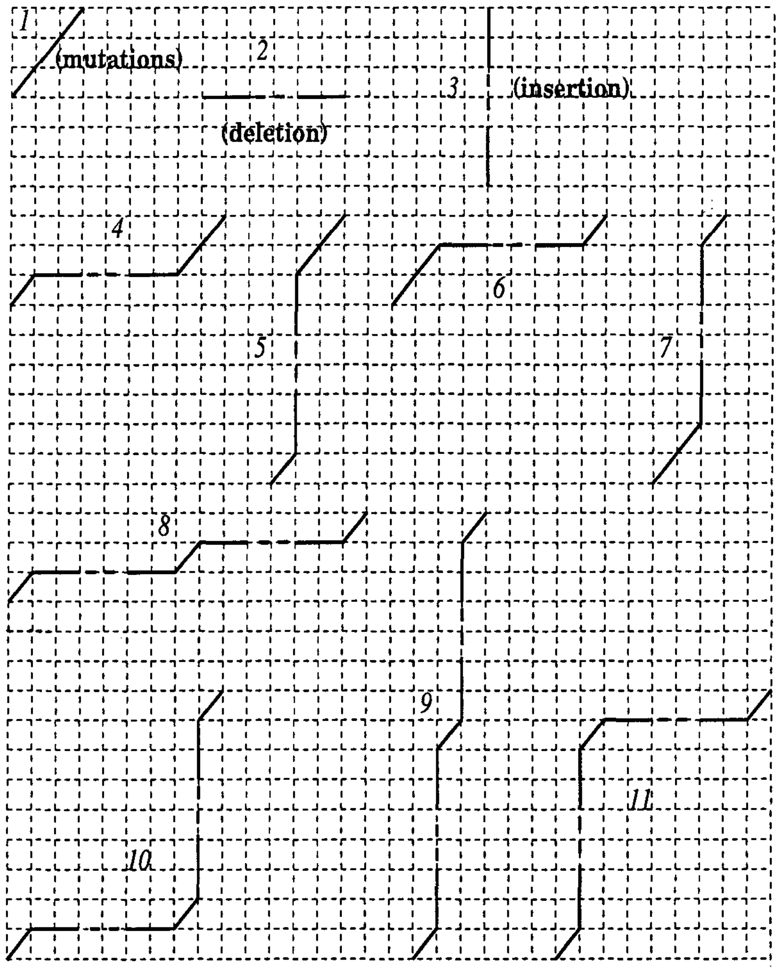

Figure 1.3: The 11 types of codon alignment.

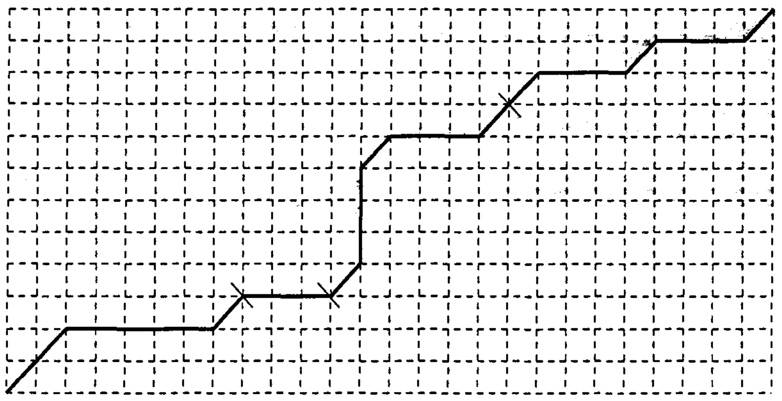

Figure 1.4: Decomposing an alignment into codon alignments.

alignment.

It is known [4] that there are 11 distinct types of codon alignment, as depicted in Figure 1.3. Type 1 has three mutations and no indel. Type 2 only has a deletion and type 3 only has an insertion. Types 4, 5, 6, and 7 have an indel and three mutations. Types 8, 9, 10, and 11 have two indels and three mutations. Observe that each codon alignment can involve at most 5 evolutionary events. Hence, the cost of a codon alignment, which is the minimum total combined cost over all possible permutations of the events postulated by the alignment, can be computed in linear time.

We can always decompose an alignment of A and B uniquely into a sequence of maximal codon alignments, as illustrated in Figure 1.4. Although the evolutionary events in a same codon alignment may influence each other's cost, events in different codon alignments are independent. This gives rise to a straightforward dynamic programming algorithm for computing an optimal alignment of A and B in $O(m^2n^2)$ time, as described in [4]. It is clear that the algorithm is too slow to be useful in practice even for moderate m and n. Recently, two quadratic (*i.e.* $O(mn)$) time algorithms have been developed in [8,9]. These algorithms are not practical because their quadratic time bounds all contain large constant factors. We discuss these two algorithms in detail in Chapter 2.

1.3 Our contribution

Since large constant factors seem to be inherent in all quadratic time algorithms for Hein's model, we simplify the model slightly. A much more efficient quadratic time algorithm is devised for the simplified model which needs only to compute $292mn$ table entries, again assuming affine gaps. Although the framework of the algorithm is still dynamic programming, the crux of this algorithm is a careful partition of the state space in order to minimize the total number of table entries that it has to compute. Furthermore, we extend our algorithm to handle frame-shift errors and overlapping frames using a heuristic approach.

The algorithm has been implemented and tested on both real and simulated sequences. The test results show that the algorithm for our simplified model and algorithm for Hein's model produce almost identical alignment in most cases. Also our program can correctly detect and locate frame-shift errors for reasonable indel and mutation rates.

This thesis is organized as follows. In the next chapter, we describe two existing quadratic time algorithms for Hein's model. Our simplified model and faster algorithm are presented in Chapter 3. We then extended our algorithm to handle frame-shift errors and overlapping frames in Chapter 4. Chapter 5 discusses some issues arising in the implementation of the algorithm and also gives some test results. Finally, we give conclusions and future work Chapter 6.

Chapter 2

Two Quadratic Algorithms for Hein's Model

In this chapter, first we describe the Hua-Jiang algorithm in Section 2.1, and then the PLH algorithm in Section 2.2.

2.1 Hua-Jiang algorithm

In 1997 Y. Hua and T. Jiang designed a dynamic programming algorithm in [8] that computes an optimal alignment for Hein's model in $O(mn)$ time, assuming affine gaps. However, the algorithm is impractical because of the large constant factor embedded in its time complexity function. The large constant factor comes from the fact that the algorithm has to compute $16644mn$ table entries. The following is a sketch of the construction of the

Hua-Jiang algorithm.

Again assume that the gap penalty function $g(i)$ is affine, *i.e.* $g(i) = g_{open} + i * g_{ext}$ for some fixed non-negative constants g_{open} and g_{ext}. Consider DNA sequences

$$A = a_1a_2a_3 \cdots a_{3m-2}a_{3m-1}a_{3m}$$

and

$$B = b_1b_2b_3 \cdots b_{3n-2}b_{3n-1}b_{3n}.$$

For any indices $i = 1, \ldots, m$ and $j = 1, \ldots, n$, let

$$A(i) = a_1a_2a_3 \cdots a_{3i-2}a_{3i-1}a_{3i},$$

$$B(j) = b_1b_2b_3 \cdots b_{3j-2}b_{3j-1}b_{3j},$$

and $c(i, j)$ denote the cost of an optimal alignment between the prefix $A(i)$ and prefix $B(j)$. In order to derive a recurrence equation for $c(i, j)$, we need the following notation.

Let's classify alignments into 11 classes according to the type of their terminating codon alignments (see Figure 1.3 for values of t). For $1 \leq t \leq 3$, let $c_t(i, j)$ denote the cost of an optimal alignment between $A(i)$ and $B(j)$ whose terminating codon alignment is type t.

For $t = 4$ or $t = 6$ and any nucleotides $x_1, x_2, x_3 \in \{A, C, G, T\}$, let $c_t(i, j, x_1x_2x_3)$ denote the cost of an optimal alignment between $A(i)$ and

$B(j)x_1x_2x_3$ ending with a codon alignment of type t. Also define

$$c_t(i,j) = c_t(i, j-1, b_{3j-2}b_{3j-1}b_{3j}).$$

For $t = 5$ or $t = 7$ and any nucleotides $x_1, x_2, x_3 \in \{A, C, G, T\}$, let $c_t(i, x_1x_2x_3, j)$ denote the cost of an optimal alignment between $A(i)x_1x_2x_3$ and $B(j)$ ending with a codon alignment of type t. Also define

$$c_t(i,j) = c_t(i-1, a_{3i-2}a_{3i-1}a_{3i}, j).$$

For $t = 8$ and any nucleotides $x_1, x_2, x_3, x_4, x_5, x_6 \in \{A, C, G, T\}$, let $c_8(i, j, x_1x_2x_3x_4x_5x_6)$ denote the cost of an optimal alignment between $A(i)$ and $B(j)x_1x_2x_3x_4x_5x_6$ ending with a codon alignment of type 8. Also define

$$c_8(i,j) = c_8(i, j-1, b_{3j-5}b_{3j-4}b_{3j-3}b_{3j-2}b_{3j-1}b_{3j}).$$

The expressions $c_9(i, x_1x_2x_3x_4x_5x_6, j)$ and $c_9(i,j)$ are defined analogously.

For $t = 10$ and any nucleotides $x_1, x_2, x_3, y_1, y_2, y_3 \in \{A, C, G, T\}$, let $c_{10}(i, x_1x_2x_3, j, y_1y_2y_3)$ denote the cost of an optimal alignment between sequences $A(i)x_1x_2x_3$ and $B(j)y_1y_2y_3$ ending with a codon alignment of type 10. Also define

$$c_{10}(i,j) = c_{10}(i-1, a_{3i-2}a_{3i-1}a_{3i}, j-1, b_{3j-2}b_{3j-1}b_{3j}).$$

The expressions $c_{11}(i, x_1x_2x_3, j, y_1y_2y_3)$ and $c_{11}(i,j)$ are defined analogously.

Note that, in the above, for types $t = 4, \ldots, 11$, we have to plant up to 6 imaginary trailing bases in order to complete the recurrence equations.

Clearly, for any $i = 0, \ldots, m$ and $j = 0, \ldots, n$,

$$c(i,j) = \min_{t=1}^{11} c_t(i,j)$$

Hence it suffices to give recurrence equations for $c_t(i,j)$, $t = 1, \ldots, 11$, using $c(i,j)$. First, we initialize the following items:

- $c(0,0) = 0$
- For $i = 1, ..., m$, $c(i,0) = g(i)$.
- For $j = 1, ..., n$, $c(0,j) = g(j)$.
- For $i = 1, ..., m$ and $j = 1, ..., n$, $c(i,j) = \infty$.
- For $i = 1, ..., m$, $j = 1, ..., n$, and $t = 1, ..., 11$, $c_t(i,j) = \infty$.

Below we only list recurrence equations for types $t = 1, 2, 4, 8, 10$. The other cases are highly symmetric to these types. In the following, when there is a *unique* codon alignment between sequences X and Y of type t, we use $ca_t(X,Y)$ to denote the optimal cost of that codon alignment for different event orders. For $1 \leq i \leq m$ and $1 \leq j \leq n$, the recurrence equations are as follows:

$$c_1(i,j) \quad = \quad c(i-1, j-1) + ca_1(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j})$$

$$c_2(i,j) = \min\{c_2(i,j-1) + g_{ext}, c(i,j-1) + g(1)\}$$

$$\begin{aligned} c_4(i,j,x_1x_2x_3) &= \min\{c_4(i,j-1,x_1x_2x_3) + g_{ext}, \\ &\quad c(i-1,j-1) + \\ &\quad ca_4(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}x_1x_2x_3)\} \end{aligned}$$

$$\begin{aligned} c_8(i,j,x_1x_2x_3x_4x_5x_6) &= \min\{c_8(i,j-1,b_{3j-2}b_{3j-1}b_{3j}x_4x_5x_6) + g_{ext}, \\ &\quad c_8(i,j-1,x_1x_2x_3x_4x_5x_6) + g_{ext}, \\ &\quad c(i-1,j-1) + \\ &\quad ca_8(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}x_1x_2x_3x_4x_5x_6)\} \end{aligned}$$

$$\begin{aligned} c_{10}(i,x_1x_2x_3,j,y_1y_2y_3) &= \min\{c_{10}(i-1,x_1x_2x_3,j,y_1y_2y_3) + g_{ext}, \\ &\quad c_{10}(i,x_1x_2x_3,j-1,y_1y_2y_3) + g_{ext}, \\ &\quad c(i-1,j-1) + \\ &\quad ca_{10}(a_{3i-2}a_{3i-1}a_{3i}x_1x_2x_3, b_{3j-2}b_{3j-1}b_{3j}y_1y_2y_3)\} \end{aligned}$$

The equations for $t = 1, 2, 4, 10$ are self-explanatory. The equation for $t = 8$ is elaborated in Figure 2.1. The first term in the minimization corresponds to the case when the second gap (from the left) is longer than one codon (case (a) in the figure), the second term represents the case when the second gap is one codon long and the first gap is longer than one codon (case (b)), and the third term corresponds to the case when both gaps are one codon long (case (c)).

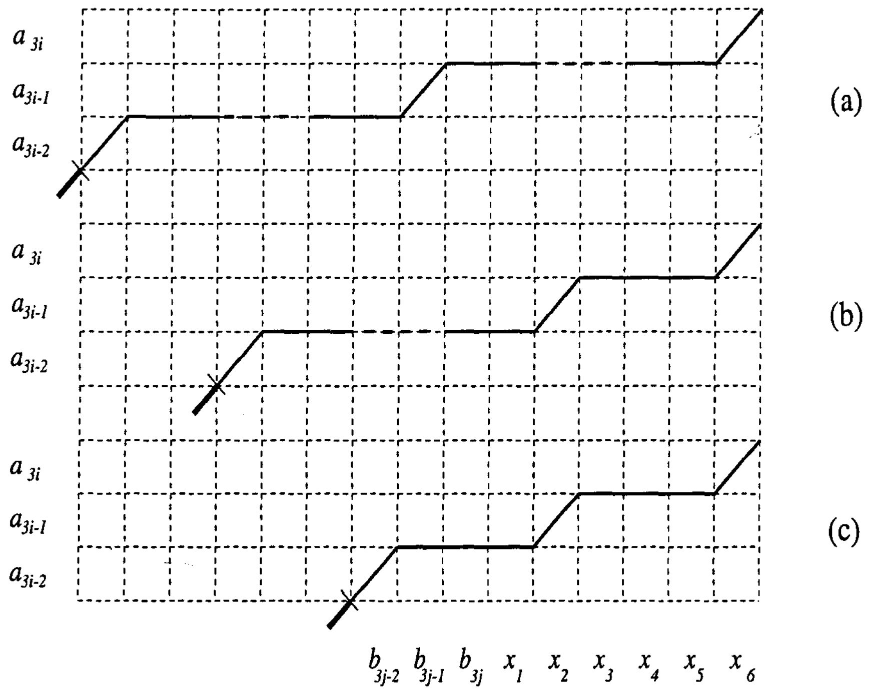

Figure 2.1: An illustration for the recurrence equation for type 8.

The base cases of the above recurrence equations can be easily formulated. Since the cost $ca_t(X, Y)$ can be computed in $O(1)$ time for any sequences X and Y of lengths at most 9 bases, the recurrence equations obviously imply a dynamic programming algorithm for computing $c(m, n)$ in $O(mn)$ time. This algorithm can be easily expanded to also produce an optimal alignment between A and B, using the standard back-tracking technique[2].

A version of the algorithm has been implemented in GNU C, called *Codon Alignment Tool* (CAT)[8]. To avoid computing the cost $ca_t(X, Y)$ repeatedly for the same short sequences X and Y, a table, indexed by X and Y, is used to store the value $ca_t(X, Y)$ once it is computed so that for each pair X and Y, the cost $ca_t(X, Y)$ is computed at most once. Although this technique greatly improves the time efficiency, the program is still quite slow due to the fact that it has to compute 12 tables for $c(i, j)$ and $c_t(i, j)$, where $t = 1, \ldots, 11$, with a total size of $4 + 4 * 64 + 4 * 4096 = 16644mn$ entries before obtaining the value $c(m, n)$. Clearly, codon alignments of types 8 through 11 are the main reason why such large tables are required. Because of the influence between evolutionary events within a same codon alignment and the fact that the events may happen in any of up to 5! different orders, the dynamic programming algorithm has to hypothesize 6 trailing bases for each of these four types, and carry out the computation for each of the 4096 hypotheses.

2.2 PLH algorithm

Independent of the work reported in this thesis, recently, C. Pedersen, R. Lyngsø, and J. Hein designed another quadratic algorithm for Hein's model.

We call this algorithm the PLH algorithm. The framework of PLH algorithm is still dynamic programming. Similar to Hua-Jiang algorithm, an alignment is also classified into 11 classes according to the type of its last codon alignment. A key idea behind the PLH algorithm is that it keeps track of the "internal status" of a mutation. In other words, it sets some indicators of some key mutations. The algorithm is valid under the assumption that the cost of mutations at the protein level is a metric. We describe more details of the PLH algorithm below.

The recurrence equations of the first three types are the same as that of the Hua-Jiang algorithm. For types 4, 5, 6, and 7, the PLH algorithm guesses the internal status of all relevant mutations just before the deletion or insertion. We give an example for type 6 as shown in Figure 2.2, where $x_1x_2x_3$ indicates the status of the three mutations (*i.e.* whether or not the mutations have taken place) just before the deletion of length k. Four key stages of the evolution changing $b_{3(j-k)-2}b_{3(j-k)-1}...b_{3j}$ to $a_{3i-2}a_{3i-1}a_{3i}$ are depicted in (a), (b), (c), and (d) in the figure respectively. The minimum cost of an alignment whose last codon alignment is type 6, denoted $c_6(i,j)$, can be calculated as

$$c_6(i,j) = cost(subs) + cost(del) + c(i-1, j-k-1),$$

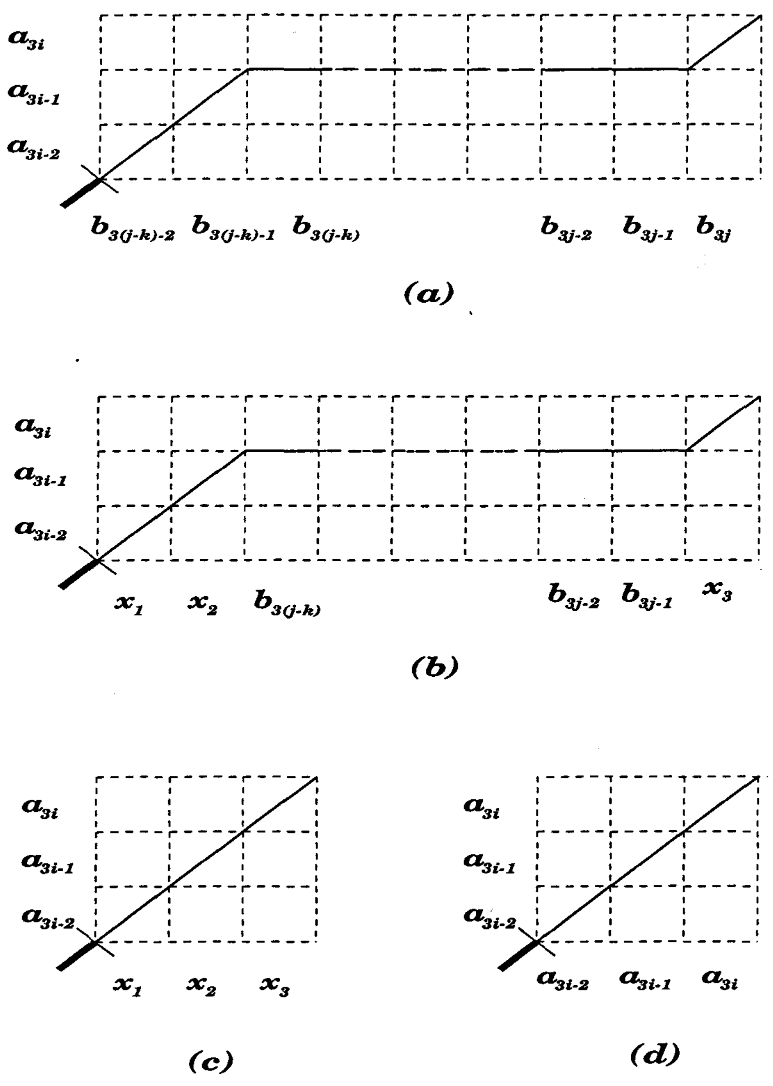
a_{3i}
a_{3i-1}
a_{3i-2}
$b_{3(j-k)-2}$ $b_{3(j-k)-1}$ $b_{3(j-k)}$ b_{3j-2} b_{3j-1} b_{3j}
(a)
a_{3i}
a_{3i-1}
a_{3i-2}
x_1 x_2 $b_{3(j-k)}$ b_{3j-2} b_{3j-1} x_3
(b)
a_{3i}
a_{3i-1}
a_{3i-2}
x_1 x_2 x_3
(c)
a_{3i}
a_{3i-1}
a_{3i-2}
a_{3i-2} a_{3i-1} a_{3i}
(d)

Figure 2.2: Four stages in the evolution of type 6.

where

$$\begin{aligned} cost(subs) &= c_p^*(b_{3(j-k)-2}b_{3(j-k)-1}b_{3(j-k)}, x_1x_2b_{3(j-k)}) + \\ &\quad c_p^*(b_{3j-2}b_{3j-1}b_{3j}, b_{3j-2}b_{3j-1}x_3) + \\ &\quad c_p^*(x_1x_2x_3, a_{3i-2}a_{3i-1}a_{3i}). \end{aligned}$$

The difference between the notation c_p and c_p^* is that c_p accounts for at most one mutation, but c_p^* may account for up to three mutations. The term $cost(del)$ represents the cost of the deletion, and can be computed using dynamic programming. For more details, the reader is referred to the paper [9].

The idea can be extended to types 8, 9, 10 and 11, but these types require two internal status indicators, one for the first indel and the other for the second indel.

An advantage of the PHL algorithm is that it "hides" the orders of events in internal status indicators. But this advantage comes with an assumption, namely, the cost of mutations at the protein level is a metric. Unfortunately, in practice, most of popular protein scores, *e.g.* PAM, are not metrics. In the case that the protein mutation cost is not a metric, the algorithm needs more table entries to record information. We estimate that it has to compute $4100mn$ table entries under the metric assumption and $15476mn$ table entries without that assumption. Since the algorithm has not been implemented, it is hard to compare its speed with that of Hua-Jiang's. However it is clear that the algorithm is too slow to be used in practice because of the large constant factor in the quadratic time bound.

In the next chapter, we will simplify Hein's model slightly and present a much faster quadratic time algorithm.

Chapter 3

Context-free Codon Alignment

This chapter is organized as follows. In Section 3.1 we describe the simplified model of genomic sequence alignment. Then we show our faster algorithm in Section 3.2. We compare the test results of CAT and Context-free CAT in Section 3.3.

3.1 A simplified model

Our model differs from Hein's model only in the definition of the cost of an indel. Recall that in Hein's model each indel of $3i$ nucleotides within a codon induces an amino acid indel and an amino acid substitution, and hence the combined cost of such an indel is defined as $g(i)$ plus the cost of the amino acid substitution, where $g(i) = g_{open} + i * g_{ext}$ for some constants g_{open} and g_{ext}. Our model will disregard the latter cost, and simply define the combined cost of an indel of $3i$ nucleotides as $g(i)$. [1]

Observe that in Hein's model the cost of an indel in general depends on the surrounding nucleotides, as shown in Figure 1.3, whereas indels in our model do not have such context sensitivity. For this reason we will refer to indels in our model as *context-free indels* and name our model *context-free codon alignment*. In the following, we take advantage of the context-freeness in indels and devise a more efficient algorithm than the algorithms reviewed in the last chapter. Note that, even though indels are now context-free, the influence between evolutionary events still exists because the combined cost of a substitution may depend on other substitutions and indels in the same codon alignment. Therefore, it does not seem possible for the algorithms presented in the last chapter (or simple extensions of them) to take advantage of context-free indels. We have to use a different technique.

3.2 A faster algorithm

The framework of our algorithm is still dynamic programming based on codon alignments. We again classify an alignment according to the type of its last codon alignment. The new idea is to refine the classes according to

[1]It is unclear such a simplification is biologically plausible, although one supporting argument may be that the amino acid substitution is a superficial event. Our tests on real and simulated data in Section 3.3 will show that optimal alignments for the two models are in fact very similar.

the order of some events in the last codon alignment so we could avoid having to hypothesize (or equivalently, remember) too many nucleotides. This will greatly reduce the total size of the tables required.

To demonstrate our idea, we need to introduce some notation first. Let $A = a_1a_2a_3 \cdots a_{3m-2}a_{3m-1}a_{3m}$, and $B = b_1b_2b_3 \cdots b_{3n-2}b_{3n-1}b_{3n}$.

- For any indices $i = 1, \ldots, m$ and $j = 1, \ldots, n$, let

  $$A(i) = a_1a_2a_3 \cdots a_{3i-2}a_{3i-1}a_{3i}$$

  and

  $$B(j) = b_1b_2b_3 \cdots b_{3j-2}b_{3j-1}b_{3j}.$$

  $A(0)$ and $B(0)$ are empty strings.

- For any indices $i = 0, ..., m$ and $j = 0, ..., n$, let $c(i, j)$ denote the cost of an optimal alignment between $A(i)$ and $B(j)$.

- For any indices $i = 0, ..., m$, $j = 0, ..., n$, and $t = 1, ..., 11$, let $c_t(i, j)$ denote the cost of an optimal alignment between $A(i)$ and $B(j)$ ending with a codon alignment of type t.

To derive the necessary recurrence equations, we will need to consider *partial* (*i.e.* incomplete) codon alignments consisting of a front portion of some codon alignments and *restricted* codon alignments whose events are required to occur only in some specific orders. In the following discussion, we assume that sequence B evolves to sequence A. Before we give the general recurrences type by type, we need to initialize the following items:

- $c(0,0) = 0$
- For $i = 1, ..., m$, $c(i,0) = g(i)$.
- For $j = 1, ..., n$, $c(0,j) = g(j)$.
- For $i = 1, ..., m$ and $j = 1, ..., n$, $c(i,j) = \infty$.
- For $i = 0, ..., m$, $j = 0, ..., n$, and $t = 1, ..., 11$, $c_t(i,j) = \infty$.

For $1 \leq i \leq m$ and $1 \leq j \leq n$, the recurrence equations are as follows. First of all, the main recurrence equation is

$$c(i,j) = \min_{t \in \{1,...,11\}} c_t(i,j).$$

The recurrence equations of the first three types are straightforward. They are

$$c_1(i,j) = c(i-1, j-1) + ca_1(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j})$$

$$c_2(i,j) = \min\{c_2(i, j-1) + g_{ext}, c(i, j-1) + g(1)\}$$

$$c_3(i,j) = \min\{c_3(i-1, j) + g_{ext}, c(i-1, j) + g(1)\}$$

where $ca_1(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j})$ is a function to compute the minimum cost of evolving $b_{3j-2}b_{3j-1}b_{3j}$ to $a_{3i-2}a_{3i-1}a_{3i}$ by trying 6 different orders.

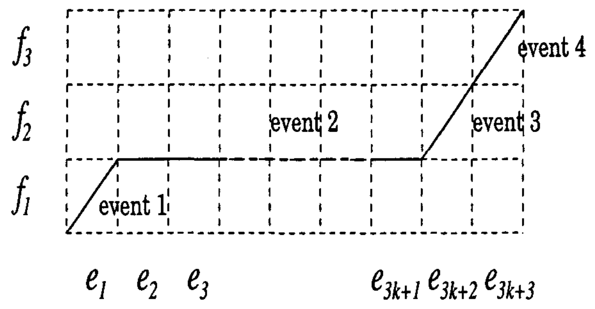

Figure 3.1: Four events of type 4 codon alignments.

A type 4 codon alignment involves 4 evolutionary events as shown in Figure 3.1. Event 1 is the first mutation (*i.e.* $e_1 \rightarrow f_1$). Event 2 is a deletion whose length is $3k$ nucleotides (*i.e.* delete $e_2...e_{3k+1}$). Event 3 evolves e_{3k+2} to f_2 and event 4 changes e_{3k+3} to f_3. We give an example of evolving $e_1...e_{3k+3}$ to $f_1f_2f_3$ in order 1234 as follows:

1. $e_1e_2e_3...e_{3k+1}e_{3k+2}e_{3k+3} \rightarrow f_1e_2e_3...e_{3k+1}e_{3k+2}e_{3k+3}$

2. $f_1e_2e_3...e_{3k+1}e_{3k+2}e_{3k+3} \rightarrow f_1e_{3k+2}e_{3k+3}$

3. $f_1e_{3k+2}e_{3k+3} \rightarrow f_1f_2e_{3k+3}$

4. $f_1f_2e_{3k+3} \rightarrow f_1f_2f_3$

We find that the information of e_2 and e_3 is only used in computing $c_p(e_1e_2e_3, f_1e_2e_3)$ when event 1 occurs before event 2. For example, the cost for order 1234, denoted $cost^4_{1234}$, is

$$\begin{aligned} cost^4_{1234} &= c_p(e_1e_2e_3, f_1e_2e_3) + c_d(e_1, f_1) + g(k) + \\ &\quad c_p(f_1e_{3k+2}e_{3k+3}, f_1f_2e_{3k+3}) + c_d(e_{3k+2}, f_2) + \\ &\quad c_p(f_1f_2e_{3k+3}, f_1f_2f_3) + c_d(e_{3k+3}, f_3). \end{aligned}$$

It uses the information of e_2 and e_3. But the cost for order 2134 shown below does not use the information.

$$\begin{aligned} cost^4_{2134} &= g(k) + c_p(e_1e_{3k+2}e_{3k+3}, f_1e_{3k+2}e_{3k+3}) + c_d(e_1, f_1) + \\ &\quad c_p(f_1e_{3k+2}e_{3k+3}, f_1f_2e_{3k+3}) + c_d(e_{3k+2}, f_2) + \\ &\quad c_p(f_1f_2e_{3k+3}, f_1f_2f_3) + c_d(e_{3k+3}, f_3). \end{aligned}$$

We consider alignments ending with type 4 codon alignments, and partition them into some classes depending on the relative order of events 1 and 2 and the nucleotide e_1. Since there are two possible relative orders of events 1 and 2, and e_1 might be A, or C, or G, or T, the total number of classes is 8. The reason we need 8 classes will be clear when we discuss how to compute $p_4(i, j, x, \sigma)$ which is defined later.

There are two stages for computing the cost of an optimal alignment between $A(i)$ and $B(j)$ ending with a codon alignment of type 4. In the first stage, we consider the cost of the deletion, and the cost of event 1 when event 1 occurs before event 2. In the second stage, we consider the costs of events

3 and 4, and the cost of event 1 when event 1 occurs after event 2. We will describe the details of the two stages in the following paragraphs.

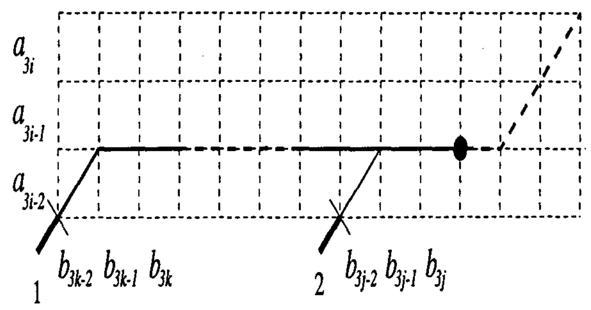
a_{3i}
a_{3i-1}
a_{3i-2}
1
b_{3k-2} b_{3k-1} b_{3k}
2
b_{3j-2} b_{3j-1} b_{3j}

Figure 3.2: Dealing with trailing codon alignments of type 4 (stage 1).

In Figure 3.2, the dot indicates that we are computing at point (i, j). The left part of the path (*i.e.* left to the dot) is computed in stage 1 and the right part of the path is computed in stage 2. In Figure 3.2, $k < j$ and $b_{3k-2} = b_{3j-2}$. The reason we need $b_{3k-2} = b_{3j-2}$ is that if $b_{3k-2} \neq b_{3j-2}$, two paths depicted in the figure are not in the same class, thus they don't have any relation. We will use a variable x to remember the value of b_{3j-2} (since $b_{3k-2} = b_{3j-2}$, x remember the value of b_{3k-2} also). The information about x will be used in the second stage.

In the first stage, there are two cases (see Figure 3.2). Case 1 extends the deletion by 3 nucleotides and case 2 starts a new partial codon alignment of type 4. For any nucleotide $x \in \{A, C, G, T\}$ and $\sigma \in \{0, 1\}$, let $p_4(i, j, x, \sigma)$ denote the cost of an optimal alignment between $A(i)$ and $B(j)$ ending with a partial codon alignment of type 4. The variable x remember the value of b_{3j-2}. The variable σ indicates the order of event 1 and event 2. If $\sigma = 0$, event 1 occurs after event 2; otherwise, *i.e.* $\sigma = 1$, event 1 occurs before event 2.

Now, it is time to define the recurrence equations for type 4 codon alignments. For $x \in \{A, C, G, T\}$ and $\sigma \in \{0, 1\}$, $p_4(i, j, x, \sigma)$ is computed as follows:

$$p_4(i, j, x, \sigma) = \min\{tmp, p_4(i, j-1, x, \sigma) + g_{ext}\},$$

where

$$tmp = c(i-1, j-1) + g(1) + \sigma \cdot (c_d(x, a_{3i-2}) + c_p(xb_{3j-1}b_{3j}, a_{3i-2}b_{3j-1}b_{3j}))$$

if $x = b_{3j-2}$; otherwise, $tmp = \infty$.

In the above equation, $p_4(i, j-1, x, \sigma) + g_{ext}$ is for case 1 and tmp corresponds to case 2 (see Figure 3.2). The first one is trivial. It just extends the deletion by 3 nucleotides. For computing tmp, first we add the cost of the previous codon alignments (*i.e.* $c(i-1, j-1)$), then add the cost of opening an indel whose length is one (*i.e.* $g(1)$). When $\sigma = 0$ (*i.e.* event 1 occurs after event 2), the item $\sigma \cdot (c_d(x, a_{3i-2}) + c_p(xb_{3j-1}b_{3j}, a_{3i-2}b_{3j-1}b_{3j}))$ is equal to zero. That means we do not consider the cost of event 1 in computing $p_4(i, j, x, 0)$. It will be added in the second stage. When $\sigma = 1$ (*i.e.* event 1 occurs before event 2), the value of $c_d(x, a_{3i-2}) + c_p(xb_{3j-1}b_{3j}, a_{3i-2}b_{3j-1}b_{3j})$

is added to *tmp*. In this case, we consider the cost of event 1 in computing $p_4(i,j,x,1)$ and it will be not added in the second stage.

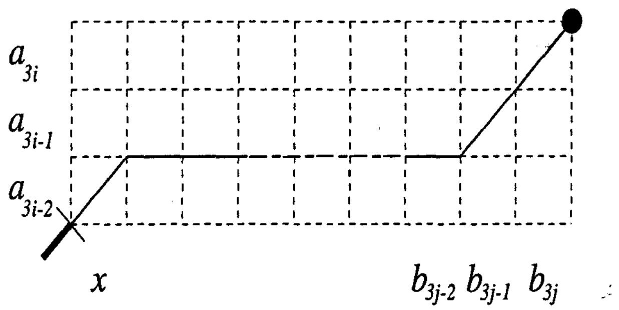

Figure 3.3: Dealing with trailing codon alignments of type 4 (stage 2).

In the second stage, we will complete the computation for type 4 using the information recorded in $p_4(i,j,x,\sigma)$. First, we have to consider the costs of events 3 and 4,*i.e.* $b_{3j-1} \to a_{3i-1}$ and $b_{3j} \to a_{3i}$ (see Figure 3.3). When $\sigma = 0$, we must add the cost of event 1 (*i.e.* $x \to a_{3i-2}$) since it is not considered in computing $p_4(i,j,x,0)$. The cost of an optimal alignment between $A(i)$ and $B(j)$ ending with a type 4 codon alignment, denoted $c_4(i,j)$, is computed as

$$c_4(i,j) = \min_{\substack{x\in\{A,C,G,T\}\\ \sigma\in\{0,1\}}} \{p_4(i,j-1,x,\sigma) + ca_4^{\sigma}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)\},$$

where $p_4(i,j-1,x,\sigma)$ is discussed above and $ca_4^{\sigma}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)$ for $\sigma \in \{0,1\}$ is explained below.

$ca_4^0(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)$ is a function to compute the minimum cost of the mutations in the last type 4 codon alignment that are not accounted for in $p_4(i,j-1,x,0)$ with the constraint that event 1 occurs after event 2. In this function, we need to consider all three mutation events (*i.e.* events 1, 3, and 4) for a total of 12 different orders. We give an example of computing the restricted cost with order 2134, denoted $Rcost^4_{2134}$, as follows:

$$\begin{aligned} Rcost^4_{2134} &= c_p(xb_{3j-1}b_{3j}, a_{3i-2}b_{3j-1}b_{3j}) + c_d(x, a_{3i-2}) + \\ & \quad c_p(a_{3i-2}b_{3j-1}b_{3j}, a_{3i-2}a_{3i-1}b_{3j}) + c_d(b_{3j-1}, a_{3i-1}) + \\ & \quad c_p(a_{3i-2}a_{3i-1}b_{3j}, a_{3i-2}a_{3i-1}a_{3i}) + c_d(b_{3j}, a_{3i}). \end{aligned}$$

Similarly, $ca^1_4(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)$ computes the minimum cost of the mutations in the last type 4 codon alignment that are not accounted for in $p_4(i, j-1, x, 1)$ with the constraint that event 1 occurs before event 2. But now we only need to consider the last two mutation events (*i.e.* events 3 and 4) since the cost of event 1 has been considered in computing $p_4(i, j, x, 1)$. Again, we give an example of computing the restricted cost with order 3124 below.

$$\begin{aligned} Rcost^4_{3124} &= c_p(b_{3j-2}b_{3j-1}b_{3j}, b_{3j-2}a_{3i-1}b_{3j}) + c_d(b_{3j-1}, a_{3i-1}) + \\ & \quad c_p(a_{3i-2}a_{3i-1}b_{3j}, a_{3i-2}a_{3i-1}a_{3i}) + c_d(b_{3j}, a_{3i}). \end{aligned}$$

Note that, the above recurrence equations for type 4 codon alignment only require a table of $8mn$ entries (for storing $p_4(i, j, x, \sigma)$) to compute $c_4(i, j)$ instead of a table of $64mn$ entries as required in the last chapter. The number $8mn$ comes from i, j, x, and σ, where $i = m$, $j = n$,

$x \in \{A, C, G, T\}$, and $\sigma \in \{0, 1\}$. Actually, the parameter x in the second function (*i.e.* $ca_4^1(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)$) can be disregarded. Although it appears that the table entries could be reduced to $5mn$, unfortunately, the answer is no. The reason is that when computing the recurrence equation of $p_4(i, j, x, \sigma)$, we must check if $x = b_{3j-2}$. In other words, we must make sure that the following case cannot take place:

$$p_4(i, j, x, 1) = p_4(i, j-1, x', 1) + g_{ext}$$

where $x \neq x'$.

The recurrence equations for type 5 are symmetric to those for type 4. We define $p_5(i, j, x, 0)$ and $p_5(i, j, x, 1)$ similarly. Again, for $x \in \{A, C, G, T\}$ and $\sigma \in \{0, 1\}$, $p_5(i, j, x, \sigma)$ is computed as

$$p_5(i, j, x, \sigma) = min\{tmp, p_5(i-1, j, x, \sigma) + g_{ext}\},$$

where

$$tmp = c(i-1, j-1) + g(1) + \sigma \cdot (c_d(b_{3j-2}, x) + c_p(b_{3j-2}a_{3i-1}a_{3i}, xa_{3i-1}a_{3i}))$$

if $x = a_{3i-2}$; otherwise, $tmp = \infty$.

Also,

$$c_5(i, j) = \min_{\substack{x \in \{A,C,G,T\} \\ \sigma \in \{0,1\}}} (p_5(i-1, j, x, \sigma) + ca_5^g(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)),$$

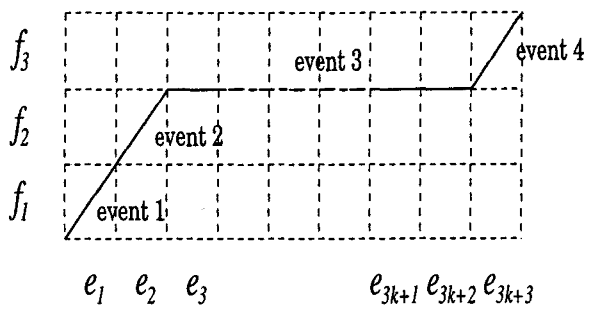

Figure 3.4: Four events of type 6 codon alignments.

where $ca_5^\sigma(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)$ is similar to the function for type 4 (*i.e.* $ca_4^\sigma(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)$).

A type 6 codon alignment involves 4 evolutionary events as shown in Figure 3.4. Event 1 is the first mutation (*i.e.* $e_1 \to f_1$). Event 2 is the second mutation (*i.e.* $e_2 \to f_2$). Event 3 is a deletion whose length is $3k$ nucleotides (*i.e.* $e_3 ... e_{3k+2}$). Event 4 evolves e_{3k+3} to f_3. We give an example to compute the cost for order 1234, denoted $cost_{1234}^6$, as follows:

$$\begin{aligned} cost_{1234}^6 &= c_p(e_1e_2e_3, f_1e_2e_3) + c_d(e_1, f_1) + \\ &\quad c_p(f_1e_2e_3, f_1f_2e_3) + c_d(e_2, f_2) + g(k) + \\ &\quad c_p(f_1f_2e_{3k+3}, f_1f_2f_3) + c_d(e_{3k+3}, f_3). \end{aligned}$$

Alignments ending with type 6 codon alignments can be treated in the same spirit as for type 4. Again, there are two stages for type 6 (*i.e.* the first stage is for computing $p_6(i, j, x, \sigma)$ and the second is for computing $c_6(i, j)$).

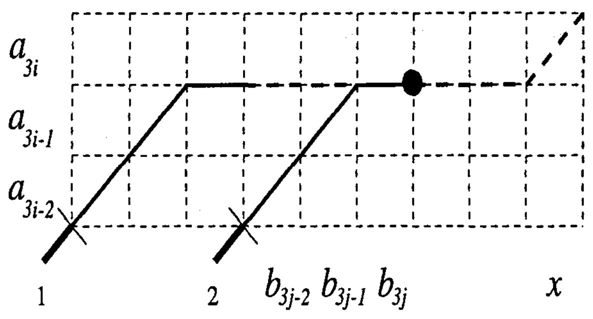

Figure 3.5: Dealing with trailing codon alignments of type 6 (stage 1).

However, instead of "cutting" the codon alignment at event 2 we should cut it at event 3 (to obtain the partial codon alignment), instead of considering the relative order of events 1 and 2 we consider the order of events 4 and 3, and instead of remembering the nucleotide x in type 4 we hypothesize the nucleotide x (see Figure 3.5). Thus, we define $p_6(i, j, x, 0)$ assuming that event 4 is after event 3 (i.e. the deletion) and event 4 starts from the nucleotide x, and define $p_6(i, j, x, 1)$ assuming the opposite order. The only tricky point is that $p_6(i, j, x, 0)$ should include the combined cost of event 4 while $p_6(i, j, x, 1)$ does not. Both of $p_6(i, j, x, 0)$ and $p_6(i, j, x, 1)$ compute the costs of events 1, 2, and 3. Let us summarize the recurrence equation for type 6 in stage 1 as follows.

For $x \in \{A, C, G, T\}$ and $\sigma \in \{0, 1\}$,

$$\begin{aligned} p_6(i, j, x, \sigma) &= min\{c(i-1, j-1) + g(1) + \\ &\quad ca_6^{\sigma}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x), \end{aligned}$$

$$p_6(i, j-1, x, \sigma) + g_{ext}\}.$$

In the above equation, $p_6(i, j-1, x, \sigma) + g_{ext}$ is for case 1 (see Figure 3.5). It just extends an indel by 3 nucleotides. $c(i-1, j-1) + g(1) + ca_6^{\sigma}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)$ is for case 2 in Figure 3.5. It starts a new partial codon alignment of type 6.

As that for type 4, $ca_6^0(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)$ is a function to compute the minimum cost of the mutations in the last type 6 codon alignment with the constraint that event 4 is after event 3. In this case we need consider three mutation events (*i.e.* events 1, 2, and 4). The following is an example of computing the partial cost of order 1234, denoted $Pcost_{1234}^6$.

$$\begin{aligned} Pcost_{1234}^6 &= c_p(b_{3j-2}b_{3j-1}b_{3j}, a_{3i-2}b_{3j-1}b_{3j}) + c_d(b_{3j-2}, a_{3i-2}) + \\ &\quad c_p(a_{3i-2}b_{3j-1}b_{3j}, a_{3i-2}a_{3i-1}b_{3j}) + c_d(b_{3j-1}, a_{3i-1}) + \\ &\quad c_p(a_{3i-2}a_{3i-1}x, a_{3i-2}a_{3i-1}a_{3i}) + c_d(x, a_{3i}). \end{aligned}$$

Another function for type 6, *i.e.* $ca_6^1(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)$, is for the case when event 4 is before event 3. In this function, we only need consider the first two mutation events (*i.e.* events 1 and 2). The last mutation event (*i.e.* event 4) is considered in stage 2 (*i.e.* computing $c_6(i, j)$). Again, we give an example of computing the partial cost of order 1243 below.

$$Pcost_{1243}^6 = c_p(b_{3j-2}b_{3j-1}b_{3j}, a_{3i-2}b_{3j-1}b_{3j}) + c_d(b_{3j-2}, a_{3i-2}) +$$

$$c_p(a_{3i-2}b_{3j-1}b_{3j}, a_{3i-2}a_{3i-1}b_{3j}) + c_d(b_{3j-1}, a_{3i-1}).$$

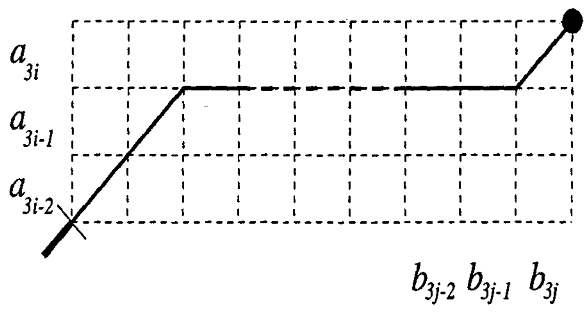

Figure 3.6: Dealing with trailing codon alignments of type 6 (stage 2).

In the second stage (see Figure 3.6), if $\sigma = 1$ (*i.e.* event 4 is before event 3), we need add the cost of event 4 to $c_6(i,j)$; Otherwise, $c_6(i,j)$ is equal to $(p_6(i,j-1,x,\sigma)$. The recurrence equation for computing $c_6(i,j)$ is

$$c_6(i,j) = \min_{\sigma\in\{0,1\}}(p_6(i,j-1,x,\sigma) + \sigma \cdot tmp),$$

where $x = b_{3j}$ and

$$tmp = c_d(x, a_{3i}) + c_p(b_{3j-2}b_{3j-1}x, b_{3j-2}b_{3j-1}a_{3i}).$$

Again, computing the costs $c_6(i,j)$ requires only a table of $8mn$ entries.

Analogously, for $x \in \{A,C,G,T\}$ and $\sigma \in \{0,1\}$,

$$\begin{aligned} p_7(i,j,x,\sigma) &= min\{c(i-1,j-1) + g(1) + \\ &\qquad ca_7^{\sigma}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x), \\ &\qquad p_7(i-1,j,x,\sigma) + g_{ext}\}. \end{aligned}$$

Also,

$$c_7(i,j) = \min_{\sigma\in\{0,1\}}(p_7(i-1,j,x,\sigma) + \sigma \cdot tmp),$$

where $x = a_{3i}$ and

$$tmp = c_d(b_{3j}, x) + c_p(a_{3i-2}a_{3i-1}b_{3j}, a_{3i-2}a_{3i-1}x).$$

The function, $ca_7^\sigma(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)$, is similar to that for type 6 (*i.e.* $ca_6^\sigma(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x)$).

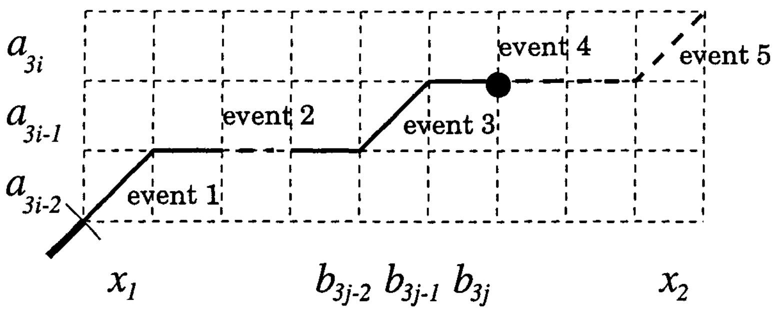

Figure 3.7: Dealing with trailing codon alignments of type 8.

The treatment of alignments ending with type 8 alignments combines the techniques for both type 4 and type 6 alignments, and builds on the information $p_4(i,j,x_1,0)$ and $p_4(i,j,x_1,1)$. Define $\sigma_1 = 0$ if event 1 is after event 2 (i.e. the first deletion) or $\sigma_1 = 1$ otherwise, and $\sigma_2 = 0$ if event 5 is after event 4 (i.e. the second deletion) or $\sigma_2 = 1$ otherwise. For any nucleotides $x_1, x_2 \in \{A, C, G, T\}$ and orders $\sigma_1, \sigma_2 \in \{0,1\}$, let $p_8(i,j,x_1,\sigma_1,x_2,\sigma_2)$ denote the cost of an optimal alignment between

$A(i)$ and $B(j)$ ending with a restricted partial codon alignment of type 8 consisting of events 1 through 5 such that (i) event 1 starts from the base x_1, (ii) the relative order between events 1 and 2 is as prescribed by σ_1, (iii) event 5 starts from the base x_2, and (iv) the relative order between events 5 and 4 is as prescribed by σ_2. (See Figure 3.7). Again, the value $p_8(i,j,x_1,\sigma_1,x_2,0)$ should include the combined cost of event 5 while $p_8(i,j,x_1,\sigma_1,x_2,1)$ does not. The cost $p_8(i,j,x_1,\sigma_1,x_2,\sigma_2)$ can be easily computed from the values $p_8(i,j-1,x_1,\sigma_1,x_2,\sigma_2)$ and $p_4(i,j-1,x_1,\sigma_1)$, and the nucleotides $x_1, x_2, a_{3i-2}, a_{3i-1}, a_{3i}, b_{3j-2}, b_{3j-1}, b_{3j}$. Hence, we can compute the cost $c_8(i,j)$ using a table of $64mn$ entries for storing $p_8(i,j,x_1,\sigma_1,x_2,\sigma_2)$. The recurrence equations for type 8 are as follows.

For $x_1, x_2 \in \{A,C,G,T\}$ and $\sigma_1, \sigma_2 \in \{0,1\}$,

$$\begin{aligned} p_8(i,j,x_1,\sigma_1,x_2,\sigma_2) &= min\{p_4(i,j-1,x_1,\sigma_1) + \\ &\quad ca_8^{\sigma_1,\sigma_2}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2), \\ &\quad p_8(i,j-1,x_1,\sigma_1,x_2,\sigma_2) + g_{ext}\}, \end{aligned}$$

where $ca_8^{\sigma_1,\sigma_2}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2)$ is a function to compute some partial cost for each of the four order groups for type 8.

$ca_8^{0,0}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2)$ computes the minimum cost of mutation events 1, 3, and 5 assuming that event 1 is after event 2 and event 5 is after event 4. We give an example for order 21345 below:

$$\begin{aligned}
Pcost^{8}_{21345} &= c_p(x_1 b_{3j-1} b_{3j}, a_{3i-2} b_{3j-1} b_{3j}) + c_d(x_1, a_{3i-2}) + \\
&\quad c_p(a_{3i-2} b_{3j-1} b_{3j}, a_{3i-2} a_{3i-1} b_{3j}) + c_d(b_{3j-1}, a_{3i-1}) + \\
&\quad c_p(a_{3i-2} a_{3i-1} x_2, a_{3i-2} a_{3i-1} a_{3i}) + c_d(x_2, a_{3i}).
\end{aligned}$$

$ca^{0,1}_{8}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2)$ finds the optimal cost of the mutations assuming that event 1 is after event 2 and event 5 is before event 4. In this function, we need only consider the first two mutation events (*i.e.* events 1 and 3). The last mutation event (*i.e.* event 5) is considered in computing $c_8(i, j)$. An example for order 21354 is

$$\begin{aligned}
Pcost^{8}_{21345} &= c_p(x_1 b_{3j-1} b_{3j}, a_{3i-2} b_{3j-1} b_{3j}) + c_d(x_1, a_{3i-2}) + \\
&\quad c_p(a_{3i-2} b_{3j-1} b_{3j}, a_{3i-2} a_{3i-1} b_{3j}) + c_d(b_{3j-1}, a_{3i-1}).
\end{aligned}$$

The third function, $ca^{1,0}_{8}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2)$, computes the optimal cost of the mutations assuming that event 1 is before event 2 and event 5 is after event 4. But in this function we need consider the last two mutation events (*i.e.* events 3 and 5). The first mutation event (*i.e.* event 1) has been included in $p_4(i, j, x_1, 1)$. We give an example for order 12345 as follows:

$$\begin{aligned}
Pcost^{8}_{12345} &= c_p(a_{3i-2} b_{3j-1} b_{3j}, a_{3i-2} a_{3i-1} b_{3j}) + c_d(b_{3j-1}, a_{3i-1}) + \\
&\quad c_p(a_{3i-2} a_{3i-1} x_2, a_{3i-2} a_{3i-1} a_{3i}) + c_d(x_2, a_{3i}).
\end{aligned}$$

The last function for type 8, $ca_8^{1,1}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2)$, is for the case when which event 1 is before event 2 and event 5 is before event 4. In this function, we need only consider one mutation event (*i.e.* event 3). The other mutation events (*i.e.* events 1 and 5) are computed in $p_4(i, j, x_1, 1)$ and $c_8(i, j)$ respectively. The following is an example for order 12354:

$$Pcost_{12354}^{8} = c_p(a_{3i-2}b_{3j-1}b_{3j}, a_{3i-2}a_{3i-1}b_{3j}) + c_d(b_{3j-1}, a_{3i-1}).$$

Finally, the cost of an optimal alignment between $A(i)$ and $B(j)$ ending with a type 8 codon alignment is computed as

$$c_8(i, j) = \min_{\substack{x_1 \in \{A,C,G,T\} \\ \sigma_1, \sigma_2 \in \{0,1\}}} (p_8(i, j-1, x_1, \sigma_1, x_2, \sigma_2) + \sigma_2 \cdot tmp),$$

where $x_2 = b_{3j}$ and

$$tmp = c_d(x_2, a_{3i}) + c_p(b_{3j-2}b_{3j-1}x_2, b_{3j-2}b_{3j-1}a_{3i}).$$

Similarly, for $x_1, x_2 \in \{A, C, G, T\}$ and $\sigma_1, \sigma_2 \in \{0, 1\}$,

$$\begin{aligned} p_9(i, j, x_1, \sigma_1, x_2, \sigma_2) \quad = \quad & min\{p_5(i-1, j, x_1, \sigma_1) + \\ & ca_9^{\sigma_1,\sigma_2}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2), \\ & p_9(i-1, j, x_1, \sigma_1, x_2, \sigma_2) + g_{ext}\}. \end{aligned}$$

Also,

$$c_9(i, j) = \min_{\substack{x_1 \in \{A,C,G,T\} \\ \sigma_1, \sigma_2 \in \{0,1\}}} (p_9(i-1, j, x_1, \sigma_1, x_2, \sigma_2) + \sigma_2 \cdot tmp),$$

where $x_2 = a_{3i}$ and

$$tmp = c_d(b_{3j}, x_2) + c_p(a_{3i-2}a_{3i-1}b_{3j}, a_{3i-2}a_{3i-1}x_2).$$

The function, $ca_9^{\sigma_1,\sigma_2}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2)$, is similar to that for type 8 (*i.e.* $ca_8^{\sigma_1,\sigma_2}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2)$).

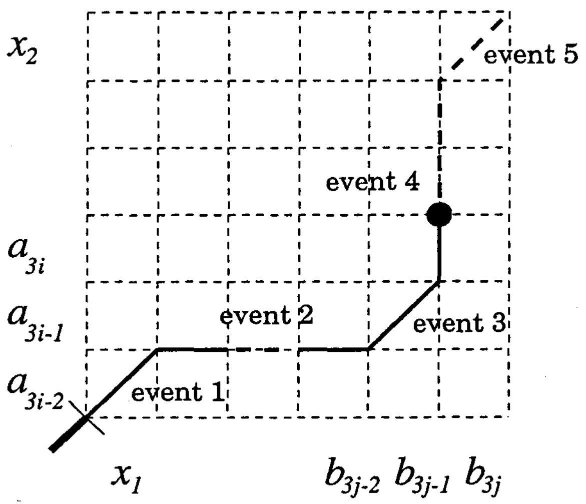

Figure 3.8: Dealing with trailing codon alignments of type 10.

We deal with alignments ending with type 10 codon alignments by combining the techniques for type 4 and type 7 codon alignments, making use of the information $p_4(i, j, x_1, 0)$ and $p_4(i, j, x_1, 1)$. We still cut the codon alignment at event 4 and consider the order of events 5 and 4; but we hypothesize the nucleotide x_2 instead of b_{3j} (see Figure 3.8). The cost $p_{10}(i, j, x_1, \sigma_1, x_2, \sigma_2)$ is defined in a straightforward way as follows, and requires a table of $64mn$ entries to store.

For $x_1, x_2 \in \{A, C, G, T\}$ and $\sigma_1, \sigma_2 \in \{0, 1\}$,

$$\begin{aligned} p_{10}(i, j, x_1, \sigma_1, x_2, \sigma_2) &= min\{p_4(i, j-1, x_1, \sigma_1) + \\ &\quad ca_{10}^{\sigma_1,\sigma_2}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2), \\ &\quad p_{10}(i-1, j, x_1, \sigma_1, x_2, \sigma_2) + g_{ext}\}. \end{aligned}$$

Also,

$$c_{10}(i, j) = \min_{\substack{x_1 \in \{A,C,G,T\} \\ \sigma_1, \sigma_2 \in \{0,1\}}} (p_{10}(i-1, j, x_1, \sigma_1, x_2, \sigma_2) + \sigma_2 \cdot tmp),$$

where $x_2 = a_{3i}$ and

$$tmp = c_d(b_{3j}, x_2) + c_p(a_{3i-2}a_{3i-1}b_{3j}, a_{3i-2}a_{3i-1}x_2).$$

Similarly, for $x_1, x_2 \in \{A, C, G, T\}$ and $\sigma_1, \sigma_2 \in \{0, 1\}$,

$$\begin{aligned} p_{11}(i, j, x_1, \sigma_1, x_2, \sigma_2) &= min\{p_5(i-1, j, x_1, \sigma_1) + \\ &\quad ca_{11}^{\sigma_1,\sigma_2}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2), \\ &\quad p_{11}(i, j-1, x_1, \sigma_1, b_{3j}, \sigma_2) + g_{ext}\} \end{aligned}$$

Also,

$$c_{11}(i, j) = \min_{\substack{x_1 \in \{A,C,G,T\} \\ \sigma_1, \sigma_2 \in \{0,1\}}} (p_{11}(i, j-1, x_1, \sigma_1, x_2, \sigma_2) + \sigma_2 \times tmp),$$

where $x_2 = b_{3j}$ and

$$tmp = c_d(x_2, a_{3i}) + c_p(b_{3j-2}b_{3j-1}x_2, b_{3j-2}b_{3j-1}a_{3i}).$$

In the above equations, $ca_{10}^{\sigma_1,\sigma_2}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2)$ and $ca_{11}^{\sigma_1,\sigma_2}(a_{3i-2}a_{3i-1}a_{3i}, b_{3j-2}b_{3j-1}b_{3j}, x_1, x_2)$ compute the costs of mutations for type 10 and 11 respectively, using the same idea as that for type 8.

This algorithm can be easily expanded to also produce an optimal alignment between A and B, using the standard back-tracking technique [2].

The above discussion yields a quadratic time dynamic programming algorithm which needs to compute 12 tables of a total size of only $(4 + 4 * 8 + 4 * 64)mn = 292mn$ entries. (The first four tables are for storing $c(i,j), c_1(i,j), c_2(i,j)$, and $c_3(i,j)$.) The algorithm has been implemented as *Context-free CAT* in GNU C, and we will show some test results in the next section.

3.3 The comparison of CAT and Context-free CAT

We have performed tests of the two programs CAT and Context-free CAT on 3 pairs of HIV1 and HIV2 sequences and 13 groups of simulated sequences of length 100 through 1500 bases. The three pairs of real data include (i) HIV1 *gag* gene (bases 790..2304) and HIV2 *gag* gene (bases 548..2113), (ii) HIV1 *vif* gene (bases 5053..5631) and HIV2 *vif* gene (bases 4868..5515), and (iii) HIV1 *nef* gene (bases 8784..9434) and HIV2 *nef* gene (bases 8562..9329).

Since we are not sure how to combine cost parameters for amino acids with those of nucleotides, two combinations were considered (a) Dayhoff PAM 40 Matrix for amino acids and DNA PAM 30 Matrix for nucleotides and (b) Dayhoff PAM 40 Matrix for amino acids and DNA PAM 47 Matrix for nucleotides. Overall, CAT and Context-free CAT produced very similar alignments in these tests. The following table summarizes the discrepancy between the alignments produced by the two programs.

Table 3.2: The discrepancy between alignments produced by the two programs.

	PAM 40 & DPAM 30		PAM 40 & DPAM 47	
	location	type	location	type
HIV1&2 *gag*	2/14	4/12	1/10	3/9
HIV1&2 *vif*	1/7	1/6	1/7	2/6
HIV1&2 *nef*	1/7	3/6	0/7	4/7

In the table, we first count the number of codon alignments involving indels (*i.e.* any codon alignment except those of type 1) that are placed at different locations by the two programs, and then the number of codon alignments that are at the same locations but have different types. For example, the entry 2/14 means that out of the 14 codon alignments involving indels, two are placed at different locations by the two programs, and the entry 4/12 means that out of the 12 remaining codon alignments four have different types. In all the cases where indels are placed at different locations, one program merges two adjacent indels produced by the other program. On the other hand, the discrepancy in the types of codon alignments is always because Context-free CAT would sometimes expand a type 2 or 3 codon alignment produced by CAT into a codon alignment of type $4, 5, 6$, or 7 by shifting the indel inside an adjacent codon alignment. It is interesting to note that CAT produces very few codon alignments of types higher than 3 while Context-free CAT produces types $4, 5, 6$, and 7 almost as frequently as types 2 and 3. Also observe that the above discrepancies between CAT and Context-free CAT do not change very much with the two pairs of cost parameters we used.

The 13 groups of simulated sequences were generated randomly on a naive stochastic model using some fixed mutation and indels rates. The amino acid mutation/indel rates are based on Dayhoff PAM 120 Matrix and the nucleotide mutation/indel rates are based on DNA PAM 30 Matrix. We ran CAT and Context-free CAT on these groups of data using cost parameters consistent with the above rates. It is observed that both programs again produced very similar alignments and, moreover, they were all able to identify most indels correctly.

Table 3.2 shows the average speeds of CAT and Context-free CAT on SPARC Ultra II Model 1300. The speed-up of Context-free CAT over CAT is

Table 3.3: The average speeds (in seconds) of CAT and Context-free CAT.

length	102	201	300	402	501	600
CAT	898.5	1872	2496	3032.5	3486	3463
C.f. CAT	1	2.5	5.5	9	13.5	17.5
length	702	801	900	1002	1200	1500
CAT	4166.5	4490	4820	5414.5	6177	8138
C.f. CAT	26.5	33.5	40	50.5	68	104

illustrated in Figure 3.9. The speed-up decreases with the length of sequences because the "atomic" codon alignments (*i.e.* the ones that cannot be further reduced), such as the codon alignments of types 4 through 11 for CAT, are more complicated and require more time to compute than the ones for Context-free CAT, and the percentage of time spent by each program on setting up the atomic codon alignment table decreases with the length. We expect the speed-up to approach $\frac{16644}{292} = 57$ (but never goes below 57) when the sequences get really long.

In the next chapter, we extend our context-free codon alignment algorithm to allow sequences with frame-shift errors and overlapping frames.

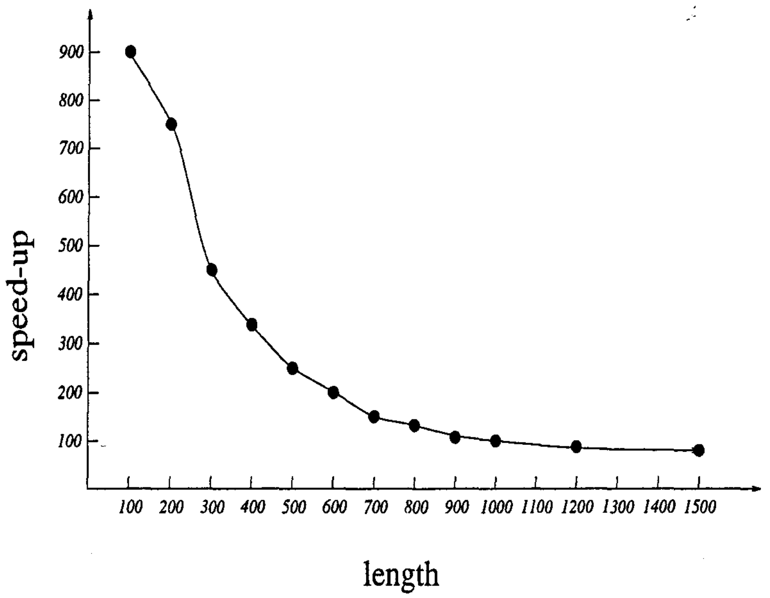

Figure 3.9: The speed-up of Context-free CAT over CAT.

Chapter 4

An Extended Model and Algorithm

Indels of lengths indivisible by three cause a coding frame to shift, and are often referred to as frame-shift errors. It is known that sometimes adjacent codons may overlap (*i.e.* share common nucleotides), thus creating overlapping frames. Frame-shift errors and overlapping frames are two complications in protein sequence alignment. Since Hein's model combines both DNA and protein alignment, it is clearly desirable that our context-free codon alignment algorithm can be extended to handle frame-shift errors and overlapping frames. In this chapter, we extend our context-free codon alignment model so that it does not require the length of an indel to be a multiple of three. From now on, we will use the term *type t atomic alignment* instead of *type t* *codon alignment* because of the existence of frame-shift errors.

The rest of this chapter is organized as follows. In Section 4.1 we introduce frame-shift errors and overlapping frames. We extend our algorithm to handle frame-shift errors in Section 4.2. In Section 4.3 we describe how to handle overlapping frames using a heuristic method. The pseudo code of our extended algorithm is listed in Section 4.4. Finally, we analyze time and space complexity of the algorithm in Section 4.5.

4.1 The frame-shift errors and overlapping frames problems

We know that a DNA sequence has six reading frames, three from $3'$ to $5'$ and three more from $5'$ to $3'$. Figure 4.1 depicts three reading frames from $5'$ to $3'$ in sequence ATGGGTTAA. The other three reading frames from $3'$ to $5'$ are similar.

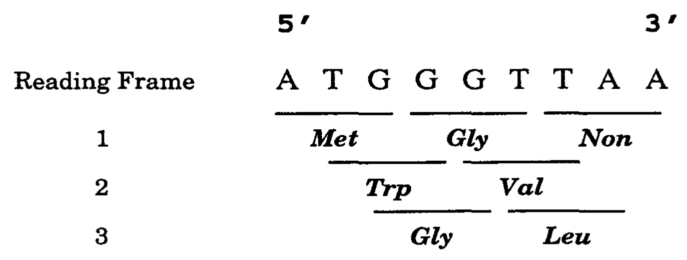

Figure 4.1: Three reading frames from $5'$ to $3'$.

Usually, there is only one reading frame for each gene. But, occasionally a *frame-shift* occurs when a gene changes its reading frame at a position of its coding region. The new reading frame will stop at a new stop amino acid or at the end of genome.

Sometimes, more than one gene is coded in the same region of DNA. We call this phenomenon *gene overlapping.* The overlapped genes might have different reading frames as shown in Figure 4.2. The overlapping frames problem could be very complicated. For example, ten genes may overlap each other in different regions, some of them from 3′ to 5′ and the others from 5′ to 3′. We will have to make some assumptions to simply the problem in Section 4.3.

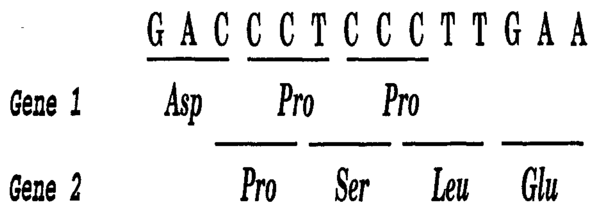

Figure 4.2: Two overlapped genes.

4.2 An extended algorithm to handle frame-shift errors

Since the reading frame may not be unique within a gene and a frame-shift could occur at any position of an alignment, our algorithm to handle frame-shift errors can't be simply based on codons. The following algorithm is based on nucleotides. It is clear that the size of the tables thus increases by at least a factor of nine. The framework of our algorithm is still dynamic programming, but we need to consider more cases than in the previous algorithm because of frame-shift errors.

Most of the notation we will use in this chapter is the same as that in the last chapter except the following:

- Let $A = a_1a_2a_3...a_{m-2}a_{m-1}a_m$, and $B = b_1b_2b_3...b_{n-2}b_{n-1}b_n$ be two DNA sequences. For any indices $i = 0, ..., m$ and $j = 0, ..., n$, let $A(i) = a_1a_2a_3...a_{i-2}a_{i-1}a_i$, and $B(i) = b_1b_2b_3...b_{j-2}b_{j-2}b_j$. Note that A, B, $A(i)$, and $B(j)$ are based on nucleotides now instead of codons.

- Instead of using $g(i)$ to denote an affine gap penalty, we use the following notations. g_{Dopen} is the cost of opening an indel at the DNA level, and g_{Popen} is the cost of opening an indel at the protein level; g_{Dext} is the cost of extending an indel by a nucleotide at the DNA level, and

g_{Pext} is the cost of extending an indel by an amino acid at the protein level; FS is a (big) constant denoting the cost of a frame-shift error.

In the following discussion, we again assume that the sequence B evolves to the sequence A. Since the frame-shift errors problem has been introduced, the two directions of evolution are no longer symmetric.

First, we initialize the following variables.

- $c(0,0) = 0$.
- For $i = 1, ..., m$, $c(i,0) = g_{Dopen} + g_{Popen} + g_{Dext} \cdot \lfloor i/3 \rfloor$.
- For $j = 1, ..., n$, $c(0,j) = g_{Dopen} + g_{Popen} + g_{Dext} \cdot \lfloor j/3 \rfloor$.
- For $i = 1, ..., m$ and $j = 1, ..., n$, $c(i,j) = \infty$.
- For $i = 0, ..., m$, $j = 0, ..., n$, and $t = 1, ..., 11$, $c_t(i,j) = \infty$.

We define 11 types of atomic alignments in the same spirit as codon alignments, and classify an alignment into 11 types according to the type of its trailing atomic alignment.

The main recurrence equation looks the same as that in the last chapter; but now i and j are the numbers of nucleotides instead of codons in the sequences A and B respectively:

$$c(i,j) = \min_{t \in \{1,...,11\}} c_t(i,j).$$

The recurrence equation for type 1 is the same as that in the last chapter except that i and j are the numbers of nucleotides instead of codons in the sequences A and B respectively:

$$c_1(i,j) = c(i-3, j-3) + ca_1(b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i),$$

where $ca_1(b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i)$ has been defined in the last chapter.

A type 2 atomic alignment only involves one evolutionary event (*i.e.* a deletion of any length). So $p_2(i,j)$ is computed as follows:

$$\begin{aligned} p_2(i,j) &= min\{p_2(i,j-3) + 3 \cdot g_{Dext} + g_{Pext}, \\ &\quad c(i,j-3) + g_{Dopen} + g_{Popen} + 3 \cdot g_{Dext} + g_{Pext}\}. \end{aligned}$$

Also,

$$\begin{aligned} c_2(i,j) &= min\{p_2(i,j), \\ &\quad p_2(i,j-1) + g_{Dext} + FS, \\ &\quad c(i,j-1) + g_{Dopen} + g_{Popen} + g_{Dext} + FS, \\ &\quad p_2(i,j-2) + 2 \cdot g_{Dext} + FS, \\ &\quad c(i,j-2) + g_{Dopen} + g_{Popen} + 2 \cdot g_{Dext} + FS\}. \end{aligned}$$

Analogously, $p_3(i,j)$ is computed as

$$\begin{aligned} p_3(i,j) &= min\{p_3(i-3,j) + 3 \cdot g_{Dext} + g_{Pext}, \\ &\quad c(i-3,j) + g_{Dopen} + g_{Popen} + 3 \cdot g_{Dext} + g_{Pext}\}. \end{aligned}$$

Also,

$$\begin{aligned}
c_3(i,j) &= min\{p_3(i,j),\\
&\quad p_3(i-3,j-1)+2\cdot g_{Dext}+FS+tmp_1,\\
&\quad c(i-3,j-1)+g_{Dopen}+g_{Popen}+2\cdot g_{Dext}+FS+tmp_1,\\
&\quad p_3(i-3,j-2)+g_{Dext}+FS+tmp_2,\\
&\quad c(i-3,j-2)+g_{Dopen}+g_{Popen}+g_{Dext}+FS+tmp_2\}.
\end{aligned}$$

where

$$\begin{aligned}
tmp_1 &= min\{c_p(a_{i-2}a_{i-1}b_j, a_{i-2}a_{i-1}a_i)+c_d(b_j,a_i),\\
&\quad c_p(b_jb_{j+1}b_{j+2}, a_ib_{j+1}b_{j+2})+c_d(b_j,a_i)\}.
\end{aligned}$$

and tmp_2 is described below.

Since there are an indel and two mutations that need to be considered when we compute tmp_2, we need to find the minimum cost by trying $3! = 6$ different orders. Let the indel be event 1, the first mutation (*i.e.* $b_{j-1} \rightarrow a_{i-1}$) be event 2, and the second mutation (*i.e.* $b_j \rightarrow a_i$) be event 3. We give an example of computing the partial cost for order 213 as follows:

$$\begin{aligned}
cost^3_{213} &= c_p(b_{j-1}b_jb_{j+1}, a_{i-1}b_jb_{j+1})+c_d(b_{j-1},a_{i-1})+\\
&\quad c_p(a_{i-2}a_{i-1}b_j, a_{i-2}a_{i-1}a_i)+c_d(b_j,a_i).
\end{aligned}$$

From the above equation, we can see that whenever a frame-shift error occurs, we use a new reading frame to compute the costs of amino acid mutations. In the following discussion, we will not explain details of computing the minimum cost for different orders since we have done so much about that in this chapter and the last chapter. But keep in mind, whenever a frame-shift error occurs, we need to use a new reading frame.

A type 4 atomic alignment involves 4 evolutionary events as shown in Figure 3.1. The basic idea of computing the cost $p_4(i, j, x, \sigma)$ of partial alignments is the same as that in Section 3.2 except that this algorithm is now based on nucleotides instead of codons. For $x \in \{A, C, G, T\}$ and $\sigma \in \{0, 1\}$, the recurrence equation is

$$p_4(i, j, x, \sigma) = min\{tmp, p_4(i, j-1, x, \sigma) + g_{ext}\},$$

where

$$\begin{aligned} tmp &= min\{c(i-3, j-3) + g_{Dopen} + g_{Popen} + 3 \cdot g_{Dext} + g_{Pext} + \\ &\quad \sigma \cdot (c_d(x, a_{i-2}) + c_p(xb_{j-1}b_j, a_{i-2}b_{j-1}b_j)), \end{aligned}$$

if $x = b_{j-2}$; otherwise, $tmp = \infty$.

Now $c_4(i, j)$ could be one of five possible paths as shown in Figure 4.3. It is computed as

$$c_4(i, j) = \min_{\substack{x \in \{A,C,G,T\} \\ \sigma \in \{0,1\}}} \{t(i, j, x, \sigma) + ca_4^{\sigma}(b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i, x),$$

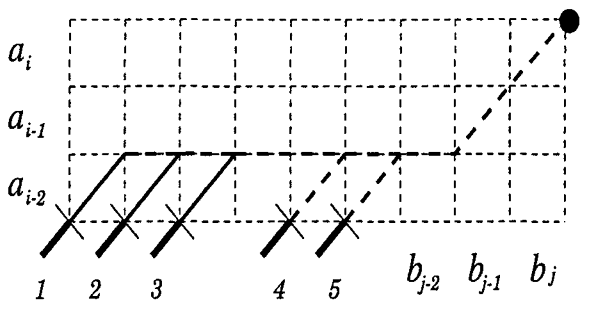

Figure 4.3: Five possible paths for type 4.

$$c(i-3, j-4) + ca_4(b_{j-3}b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i),$$

$$c(i-3, j-5) + ca_4(b_{j-4}b_{j-3}b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i)\}.$$

where

$$\begin{aligned} t(i,j,x,\sigma) &= min\{p_4(i,j-3,x,\sigma), \\ &\quad p_4(i,j-4,x,\sigma) + FS + g_{Dext}, \\ &\quad p_4(i,j-5,x,\sigma) + FS + 2 \cdot g_{Dext}\}. \end{aligned}$$

In the above equation, $ca_4^{\sigma}(b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i, x)$ is defined in the last chapter. $ca_4(b_{j-3}b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i)$ is a function to compute the minimum cost of evolving $b_{j-3}b_{j-2}b_{j-1}b_j$ to $a_{i-2}a_{i-1}a_i$ for type 4 alignment by trying $4! = 24$ different orders. It corresponds to path 5 in Figure 4.3. Again, $ca_4(b_{j-4}b_{j-3}b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i)$ computes the minimum cost of evolving $b_{j-4}b_{j-3}b_{j-2}b_{j-1}b_j$ to $a_{i-2}a_{i-1}a_i$ for type 4 by trying 24 different orders. Path 4 in Figure 4.3 depicts this case. Again, in these functions we need to use new reading frames after frame shift errors occur.

In Figure 4.3, path 1 corresponds to the first case in $t(i, j, x, \sigma)$ equation, path 2 is for the third case in the equation, and the second case is depicted by path 3.

$p_5(i, j, x, \sigma)$ is similar to that for type 4, and so we define $p_5(i, j, x, 0)$ and $p_5(i, j, x, 1)$ similarly. Again, for $x \in \{A, C, G, T\}$ and $\sigma \in \{0, 1\}$, $p_5(i, j, x, \sigma)$ is computed as

$$p_5(i, j, x, \sigma) = min\{tmp, p_5(i-1, j, x, \sigma) + g_{ext}\},$$

where

$$\begin{aligned} tmp &= c(i-3, j-3) + g_{Dopen} + g_{Popen} + 3 \cdot g_{Dext} + g_{Pext} + \\ &\quad \sigma \cdot (c_d(b_{j-2}, x) + c_p(b_{j-2}a_{i-1}a_i, xa_{i-1}a_i)), \end{aligned}$$

if $x = a_{i-2}$; otherwise, $tmp = \infty$.

But the idea to compute $c_5(i, j)$ is a little different from that for $c_4(i, j)$ since the frame-shift errors problem is introduced. The recurrence equation is

$$\begin{aligned} c_5(i, j) &= \min_{\substack{x \in \{A,C,G,T\} \\ \sigma \in \{0,1\}}} \{p_5(i-3, j, x, \sigma) + ca_5^{\sigma}(b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i, x), \\ &\quad p_5(i, j, x, \sigma) + FS - g_{Dext} - g_{Pext} + \\ &\quad \sigma \cdot (c_p(a_{i-2}a_{i-1}a_i, b_{j-2}a_{i-1}a_i + c_d(a_{i-2}, b_{j-2})), \\ &\quad p_5(i-3, j-1, x, = sigma) - 2 \cdot g_{Dext} + FS + tmp_2, \end{aligned}$$

$$c(i-3, j-1) + g_{Dopen} + g_{Popen} + g_{Dext} + FS + tmp_2\}.$$

where tmp_2 is similar to that in type 3. It computes the minimum cost of three events (*i.e.* $b_{j-2} \to x$, an insertion, and $b_j \to a_i$) by trying $3! = 6$ different orders. It uses the information from p_5 and surrounding nucleotides.

The basic idea for computing alignments ending with type 6 atomic alignments is similar to the idea used in Section 3.2. The recurrence equation for $p_6(i, j, x_1, \sigma)$ is almost the same except that it is based on the number of nucleotides instead of codons. Since frame-shift errors are introduced, we need to find the minimum cost corresponding to the 5 possible paths as shown in Figure 4.4. $p_6(i, j, x, \sigma)$ is computed as

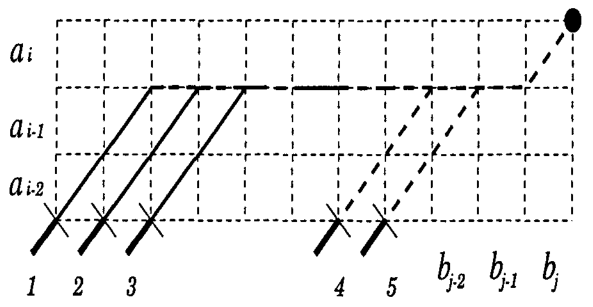

Figure 4.4: Five possible paths for type 6.

$$\begin{aligned} p_6(i, j, x, \sigma) \quad = \quad & min\{c(i-3, j-3) + g_{Dopen} + g_{Popen} + 3 \cdot g_{Dext} + g_{Pext} + \\ & ca_6^{\sigma}(b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i, x), \\ & p_6(i, j-3, x, \sigma) + 3 \cdot g_{Dext} + g_{Pext}\}, \end{aligned}$$

where $ca_6^\sigma(b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i, x)$ is defined in the last chapter.

The cost of an optimal alignment between $A(i)$ and $B(j)$ ending with a type 6 atomic alignment, denoted $c_6(i,j)$, is

$$\begin{aligned} c_6(i,j) \quad = \quad & \min_{\sigma\in\{0,1\}} \{t(i,j,x,\sigma) + \sigma \cdot (c_d(x,a_i) + c_p(b_{j-2}b_{j-1}x, b_{j-2}b_{j-1}a_i)), \\ & c(i-3,j-4) + ca_6(b_{j-3}b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i), \\ & c(i-3,j-5) + ca_6(b_{j-4}b_{j-3}b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i)\}. \end{aligned}$$

where $x = b_j$ and

$$\begin{aligned} t(i,j,x,\sigma) \quad = \quad & min\{p_6(i,j-3,x,\sigma), \\ & p_6(i,j-4,x,\sigma) + FS + g_{Dext}, \\ & p_6(i,j-5,x,\sigma) + FS + 2 \cdot g_{Dext}\}. \end{aligned}$$

In the above equation, $ca_6(b_{j-3}b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i)$ computes the minimum cost of evolving $b_{j-3}b_{j-2}b_{j-1}b_j$ to $a_{i-2}a_{i-1}a_i$ for type 6 atomic alignment by trying $4! = 24$ different orders. It corresponds to path 5 in Figure 4.4. $ca_6(b_{j-4}b_{j-3}b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i)$ is a function to find the minimum cost of evolving $b_{j-4}b_{j-3}b_{j-2}b_{j-1}b_j$ to $a_{i-2}a_{i-1}a_i$ for type 6 atomic alignment by trying 24 different orders. Path 4 in Figure 4.4 is for this case.

In Figure 4.4, path 1 corresponds to the third case in $t(i,j,x,\sigma)$ recurrence equation, path 2 is for the second case in the equation, and the first case is depicted by path 3.

Similarly, $p_7(i,j,x,\sigma)$ is computed as

$$\begin{aligned} p_7(i,j,x,\sigma) &= min\{c(i-3,j-3) + g_{Dopen} + g_{Popen} + 3\cdot g_{Dext} + g_{Pext} + \\ &\quad ca_7^{\sigma}(b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i, x), \\ &\quad p_7(i-3,j,x,\sigma) + 3\cdot g_{Dext} + g_{Pext}\}. \end{aligned}$$

where $ca_7^{\sigma}(b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i, x)$ is defined in the last chapter.

Using the same technique as that for type 5, we can compute $c_7(i,j)$ as follows.

$$\begin{aligned} c_7(i,j) &= \min_{\sigma\in\{0,1\}}\{p_7(i-3,j,x,\sigma) + \\ &\quad \sigma\cdot(c_d(b_j,a_i) + c_p(a_{i-2}a_{i-1}b_j, a_{i-2}a_{i-1}a_i)), \\ &\quad p_7(i,j,x,\sigma - 2\cdot g_{Dext} - g_{Pext} + FS, \\ &\quad p_7(i-3,j-2,x,\sigma) - g_{Dext} + FS + tmp_2, \\ &\quad c(i-6,j-4) + g_{Dopen} + g_{Popen} + 2\cdot g_{Dext} + tmp_3\}. \end{aligned}$$

where $x = a_i$ and tmp_2 and tmp_3 are functions of computing partial costs using the same technique as that for type 5.

Although our idea can be extended to handle atomic alignments involving two indels (*i.e.* types 8, 9, 10, and 11), it is clear that the speed of our algorithm would become too slow. For example, there are $5\cdot 5 = 25$ possible paths to consider in an atomic alignment of type 8. Since in this extended model we have already introduced a factor of nine, considering arbitrary indels in types 8, 9, 10, and 11 alignments would mean a slowdown factor of $25 \cdot 9 = 225$. Therefore we will assume the lengths of indels in the types 8, 9, 10, and 11 atomic alignments are multiples of three and use the recurrence equations for types 8 through 11 in the last chapter for our extended algorithm.

Although the table entries do not increase in this algorithm, computation is multiplied by a factor of 5 for types 2 through 7 since we need to consider 5 possible paths for each of them. Therefore, the time complexity of the above algorithm is $O(428mn)$ where $428 = 1 + 1 + 5 + 5 + 5 \cdot 8 \cdot 4 + 64 \cdot 4$.

The space complexity of the algorithm is $O(3mn)$ where 3 2-dimensional matrices are used for recording information about the terminating atomic alignments.

4.3 A heuristic method to handle overlapping frames

The general case of the overlapping frames problem is too complex. We need to make the following assumptions to simplify the problem.

Let $A = a_1a_2...a_m$ and $B = b_1b_2...b_n$ be two overlapped coding regions in two different DNA sequences. For $i = 1, ..., k$, where $k > 1$, $GeneA[i]$ is a gene in A and $GeneB[i]$ is a gene in B. $GeneA[i].start$ is the start position of the ith gene in A, $GeneA[i].end$ is the end position of the ith gene in A, and $GeneA[i].name$ is the name of $GeneA[i]$. $GeneB[i].start$, $GeneB[i].end$, and $GeneB[i].name$ are defined similarly. We assume

1. $GeneA[1].start = 1$, $GeneA[k].end = m$, $GeneB[1].start = 1$, and $GeneB[k].end = n$.

2. For $i = 2, ..., k$,

$$GeneA[i-1].start \leq GeneA[i].start,$$

$$GeneA[i-1].end > GeneA[i].start,$$

$$GeneB[i-1].start \leq GeneB[i].start,$$

$$GeneB[i-1].end > GeneB[i].start.$$

3. For $i = 1, ..., k$,

$$GeneA[i].name = GeneB[i].name.$$

Before describing our heuristic method, we need to introduce a new matrix named GN. The matrix GN whose size is mn saves the number of overlapped genes at point (i,j) for $i = 1, ..., m$ and $j = 1, ..., n$. GN can be easily computed using standard information about overlapped genes.

Since all 11 type atomic alignments use the same idea, we only list recurrence equations for types 1 and 4 below. The basic idea is that we multiply costs of indels and mutations by the number of overlapped genes.

For type 1, the recurrence equation is

$$c_1(i,j) = c(i-3, j-3) + GN(i,j) \cdot ca_1(b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i).$$

For type 4,

$$p_4(i,j,x,\sigma) = min\{tmp, p_4(i,j-1,x,\sigma) + GN(i,j) \cdot g_{ext}\},$$

where

$$\begin{aligned} tmp &= min\{c(i-3,j-3) + GN(i,j) \cdot (g_{Dopen} + g_{Popen} + \\ &\quad 3 \cdot g_{Dext} + g_{Pext} + \sigma \cdot (c_d(x, a_{i-2}) + c_p(xb_{j-1}b_j, a_{i-2}b_{j-1}b_j))), \end{aligned}$$

if $x = b_{j-2}$; otherwise, $tmp = \infty$. Also,

$$\begin{aligned} c_4(i,j) &= \min_{\substack{x \in \{A,C,G,T\} \\ \sigma \in \{0,1\}}} \{t(i,j,x,\sigma) + GN(i,j) \cdot ca_4^\sigma(b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i, x), \\ &\quad c(i-3, j-4) + GN(i,j) \cdot ca_4(b_{j-3}b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i), \\ &\quad c(i-3, j-5) + GN(i,j) \cdot ca_4(b_{j-4}b_{j-3}b_{j-2}b_{j-1}b_j, a_{i-2}a_{i-1}a_i)\}. \end{aligned}$$

where

$$\begin{aligned} t(i,j,x,\sigma) &= min\{p_4(i, j-3, x, \sigma), \\ &\quad p_4(i, j-4, x, \sigma) + GN(i,j) \cdot (FS + g_{Dext}), \\ &\quad p_4(i, j-5, x, \sigma) + GN(i,j) \cdot (FS + 2 \cdot g_{Dext})\}. \end{aligned}$$

One more thing we need to do is that for $i = 1, ..., m$ and $j = 1, ..., n$, if $GN(i,j) > GN(i-1,j)$ and $GN(i,j) > 1$, we add the cost of an indel, whose length is $i - GeneB[t].start$ where $GeneB[t]$ is the new overlapped gene, to $c(i,j)$. The case in which $GN(i,j) > GN(i,j-1)$ and $GN(i,j) > 1$ is similar.

4.4 The pseudo code

Under the assumptions discussed in the last section, we list the pseudo code of our extended algorithm to handle overlapping frames as well as coding and non-coding regions as follows.

```
Algorithm DNA_Protein_Alignment

1. Get the user input data.
2. Split coding and non-coding regions.
3. For each non-coding region, simply do a DNA alignment.
4. For each coding region, check if there are overlapped frames in it. If
   yes, go to 6; otherwise, do the following step.
5. Compute the minimum cost of this coding region using the algorithm
   discussed in Section 4.2 and generate the alignment for it. Then go to
   7.
```

```
6. Call the Overlapping_Region function.
7. Concatenate all the alignments of coding and non-coding regions and
   add all the costs of them.
8. Output the result.
```

```
Function Overlapping_Region
1. Compute the matrix GN.
2. For $i = 1, ..., m$ and $j = 1, ..., n$,
    (a) If $(GN(i, j) = 0)$, move to the next position.
    (b) If $(GN(i, j) = 1)$, use the algorithm discussed in Section 4.2.
    (c) If $(GN(i, j) > 1)$, use the algorithm discussed in Section 4.3.
3. Generate the alignment according to knowledge of the terminating atomic
   alignments.
```

4.5 Time and space complexity analysis

Since we use different methods to handle non-coding regions, coding regions without overlapped genes, and coding regions with overlapped genes, we discuss time and space complexity of them separately.

Let k_1 be the number of non-coding regions in two DNA sequences A and B. For $i = 1, ..., k_1$, let m_i be the number of nucleotides in the ith non-coding region of sequence A and n_i be the number of nucleotides in the ith non-coding region of sequence B. The time complexity for non-coding regions is

$$O(\sum_{i=1}^{k_1} 3m_i n_i),$$

and the space complexity is

$$O(\max_{i\in\{1,...,k_1\}} 3m_i n_i).$$

Let k_2 be the number of coding regions without overlapped genes. For $j = 1, ..., k_2$, let m_j be the number of nucleotides in the jth coding region without overlapped genes of sequence A and n_j be the number of nucleotides in the jth coding region without overlapped genes of sequence B. The time complexity for coding regions without overlapped genes is

$$O(\sum_{j=1}^{k_2} 428 m_j n_j),$$

and the space complexity is

$$O(\max_{j\in\{1,...,k_2\}} 3m_j n_j).$$

Let k_3 be the number of coding regions with overlapped genes. For $l = 1, ..., k_3$, let m_l be the number of nucleotides in the lth coding region with overlapped genes of sequence A and n_l be the number of nucleotides in the lth coding region with overlapped genes of sequence B. Let t_l be the number of points (i,j) with $GN(i,j) > 0$, where $1 \leq i \leq m_l$ and $1 \leq j \leq n_l$. The time complexity for coding regions with overlapped genes is

$$O(\sum_{l=1}^{k_3} 428t_l),$$

and the space complexity is

$$O(\max_{l\in\{1,\ldots,k_3\}} 4m_l n_l),$$

where the additional two-dimensional matrix is for GN.

Finally, the time complexity of our extended algorithm is

$$O(\sum_{i=1}^{k_1} 3m_i n_i + \sum_{j=1}^{k_2} 428m_j n_j + \sum_{l=1}^{k_3} 428t_l),$$

and the space complexity is

$$O(\max\{\max_{i\in\{1,\ldots,k_1\}} 3m_i n_i, \max_{j\in\{1,\ldots,k_2\}} 3m_j n_j, \max_{l\in\{1,\ldots,k_3\}} 4m_l n_l\}.$$

In the next chapter, we will implement our extended algorithm and report some test results.

Chapter 5

Implementation and Test Results

DPA, which is short for DNA and Protein Alignment, is the name of a software developed by us to implement the algorithm discussed in Chapter 4. Unlike Context-free CAT, DPA can handle frame-shift errors and overlapping frames.

In Section 5.1 we show the environment and programming language used in developing DPA. Then in Section 5.2 we describe the key modules of DPA. We analyze time and space used by DPA in Section 5.3. We give some test results concerning frame-shift errors and overlapping frames in Sections 5.4 and 5.5 respectively.

5.1 The environment and programming language used in developing DPA

We developed DPA on a Sun Sparc Ultra II Model 1300 using GNU C. The reason we did not use Java or C++ is that most biologists are not familiar with them. Another reason is that Java is slower than C and speed is a key consideration in our implementation.

DPA uses the algorithm discussed in the last chapter and makes the same assumptions listed in Section 4.3 for overlapping frames. Some ideas to speed up our program will be discussed in the next section. We do our best to make DPA as fast as possible.

5.2 Key modules of DPA

DPA consists of 5 modules, named *Input, Split, Cost, Align, and Output.* We will describe each module in the following subsections.

5.2.1 *Input* module

Input module is responsible for getting the user input data. DPA reads data from two files, named *DPA_Job* and *DPA_Setting*. *DPA_Job* is a job description file. All parameters used by DPA are in the *DPA_Setting* file.

The *Input* module reads the data from the two files and saves them in some variables used in the other modules.

5.2.2 *Split* module

In this module, DPA splits the coding and non-coding regions and finds pairs of genes to align. For non-coding regions, it only does DNA alignment. For coding regions, if two genes in different DNA sequences have the same name and they don't overlap with other genes, it is straightforward to align. But if there is overlapping in their coding regions, sometimes we can align all the genes in this region, sometimes we cannot. For example, suppose that in sequence A, *gene*1 is before *gene*2 and they overlap each other; but in sequence B, *gene*2 is before *gene*1 and they also overlap each other. Therefore, we must make a decision on which pair of genes should be aligned. DPA chooses the first gene in the first DNA sequence and its counterpart. If a user wants to align the second gene in the first sequence to its counterpart in the second sequence, the user can swap the two genes in the first DNA sequence in the DPA job file, i.e. *gene*2 in the first sequence should be moved to the first position in this coding region.

Another issue that should be mentioned here is that DPA changes all the characters in coding regions to upper case before passing them to the *Cost* module. This speeds up our program significantly since it saves almost half of the comparisons.

After splitting coding and non-coding regions and considering the overlapping frames problem, *Split* module passes the coding regions to the *Cost* module and concatenates the results from the *Align* module to generate the whole alignment.

5.2.3 *Cost* module

The *Cost* module is the heart of *DPA*. It computes the minimum cost and remembers all the path information of an optimal alignment in the table LC whose size is $m \cdot n$. $LC(i, j).type$ is the type of the last atomic alignment of $A(i)$ and $B(j)$. $LC(i, j).indel1$ and $LC(i, j).indel2$ are the lengths of the first indel and the second indel of the last atomic alignment of $A(i)$ and $B(j)$ respectively.

As discussed in the last chapter, if we consider frame-shift errors in atomic alignments of types 8, 9, 10, and 11, then the speed of our algorithm would be too slow in practice. Thus, DPA only considers frame-shift errors in the first seven types, and requires that an indel in the last four types must be a multiple of three nucleotides.

We find that DPA spends a lot of time computing the base cases for types

1, and 4,...,11. A base case of type 1 needs to consider $3! = 6$ different orders. For types 4, 5, 6, and 7, each base case requires the computation of $4! = 24$ different orders. For types 8, 9, 10, and 11, $5! = 120$ different orders must be tried in the base case. We use some base case tables to avoid duplicate computations for the same base case.

Another interesting issue is that we observe that the costs of some orders in an atomic alignment may be identical, hence we only need to compute one of them when this occurs. We give two examples below.

For type 4, the costs of orders 1342, 3142, and 3412 are always equal (see Figure 3.1). The reason is simple. Whenever an indel separates an alignment into two parts, the events in the two different parts have no influence on each other. The same phenomenon can be found in type 8. For example, the cost of the following orders are same (see Figure 3.3): 13524, 15324, 31524, 35124, 51324, 53124, 13542, 15342, 31542, 35142, 51342, and 53142.

One more trick to speed up our program is to use the base case table of type 1 in computing the base cases of types 4,...,11 for some special orders which have three continuous mutations after deletions or before insertions.

When *Cost* module finishes its job, the minimum cost of two input sequences has been found and information about the path has been remembered in the matrix *LC*.

5.2.4 *Align* module

The *Align* module uses the matrix LC to generate alignments using the standard back-tracking technique. It starts at entry $LC(m, n)$. Since the type and indel lengths of the last atomic alignment have been computed and recorded in the *Cost* module, it is trivial to generate the last atomic alignment. Then we move to the end position of the second to last atomic alignment, and so on. For example, if $LC(i, j).type = 8$, $LC(i, j).indel1 = 3$, and $LC(i, j).indel2 = 9$, the next position to consider would be $LC(i-3, j-3-3-9)$ (*i.e.* $LC(i-3, j-15)$). The *Align* module concatenates all atomic alignments together and terminates at entry $LC(0, 0)$.

5.2.5 *Output* module

DPA translates codons to amino acids for each gene in this module, and outputs the result according to the format used in *GenAl*[4].

5.3 Time and space complexity analysis of DPA

Our work station has 512MB physical memory and a Sun Sparc Ultra II 300MHz cpu. We have tested 16 groups of sequences with lengths ranging from 100 to 5000 nucleotides. We summarize the results in Table 5.1 and

Table 5.1: Time and space of DPA

length(nuc)	100	200	300	400	500	600	700	800
space(MB)	5	6	8	9	12	14	17	20
time(sec)	5	19	41	72	114	165	222	292
length(nuc)	900	1000	1200	1500	2000	3000	4000	5000
space(MB)	23	27	34	48	76	154	258	390
time(sec)	374	459	679	1085	1942	4421	8394	12392

illustrate the relation between speed and length in Figure 5.1.

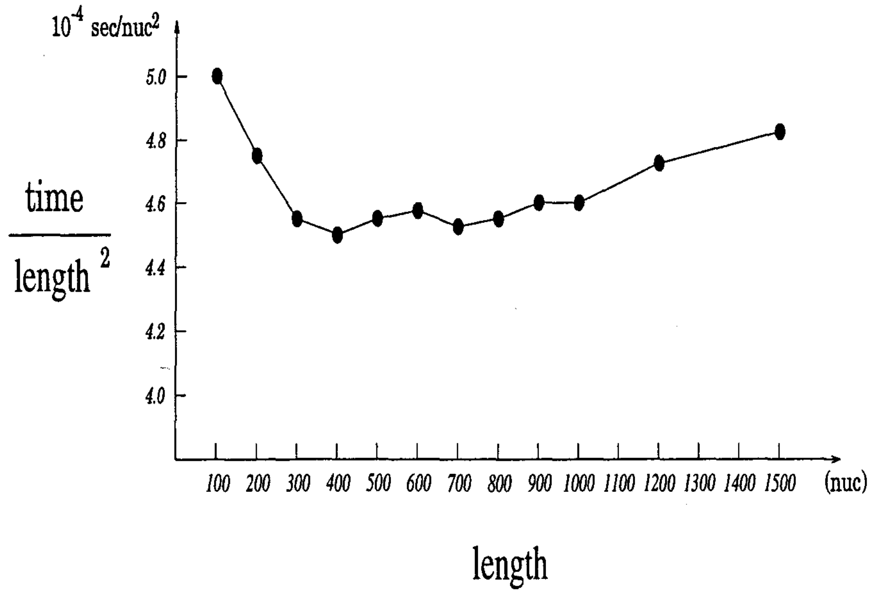

Figure 5.1: The time complexity analysis of DPA.

From Figure 5.1, we can see that DPA spends a significant amount of time in computing the costs of atomic alignments, even though a table is used to avoid duplicate computations for the same atomic alignment. This is especially obvious in the first region (*i.e.* 100...300 nuc) of the figure, where the *time/length*^{2} ratio deceases from 5 to 4.56. When the same set of cost parameters are used again and again, it is possible to speed up the program by recycling the atomic codon alignment cost table. In the second region (*i.e.* 300...800 nuc) of the figure, the ratio is stable. Its value fluctuates between 4.5 and 4.58. In the third region (*i.e.* 900...1500 nuc), it increases slowly since the memory begins to become a factor.

In general, the speed of DPA is acceptable in practice. It is much faster than CAT, but slower than Context-free CAT due to the computation based on the number of nucleotides instead of the number of codons in CAT. But DPA can handle frame-shift errors and overlapping frames while CAT does not.

The space used by the base case tables is fixed and not too great. Since we use the standard back-tracking technique instead of Hirschberg's divide and conquer algorithm, we are able to save some time although we use more space to remember the path information. This is a trade-off between time and space, and we think that time is more important than space in our program.

Table 5.2: Test results concerning frame-shift errors (1)

mutation rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
detection	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
localization	0/0	0/0	0/0	3/0	3/0	9/0	9/12	9/12

5.4 Tests concerning frame-shift errors

When there are no frame-shift errors and overlapping frames in an alignment, the test results of DPA and Context-free CAT are the same except that DPA is slower than Context-free CAT. In this section, we will describe the test results involving frame-shift errors.

We have performed two groups of tests on frame-shift errors using Dayhoff PAM 120 Matrix and DNA PAM 47 Matrix. In the first group, we fix the rate of frame shifting indel, and vary the mutation rate. Both rates of insertion and deletion we used are 0.01. The test results with mutation rate changes from 0.1 to 0.8 are listed in Table 5.2. In the detection row of the table, it gives the number of indels detected by DPA and the number of original indels respectively. The distance between the original indel and the detected indel is listed in the localization row. For example, 3/0 means that the first indel is shifted by 3 nucleotides and the second indel is exactly at the same position as the original one.

From the table we can see that when the mutation rate is less than 0.4, DPA can correctly identify the indels and frame shifts. When the mutation rate is between 0.4 and 0.6, DPA can still detect all the indels, but may put some of them at slightly wrong locations. When the mutation rate is greater than 0.6, although DPA can detect the indels, the locations of them are totally off. We list one of the results when the mutation rate is 0.2 below.

The simulated data is

```
1          11         21         31         41         51
agacccgctg aggcggcaac agatgcggtg agacaaactc agcgagcacc agtgggggtg
agacgagctg ag----caac agatacgttg agacaaactc agccaccagc agtggcggtg

61         71         81         91         101        111
ggagcagcat ---cagacct gcaccgacat ggagcaatca caactaggaa tacggcagct
ggagcggcat caccagccct ccaaagacat ggctcaatca caactctcaa tacagcagct
```

The output produced by DPA is

```
This is DPA, Version 0.90 Beta1.
Written by Bin Wu <binwu@church.dcss.McMaster.CA>.
Copyright (c) 1998 by Tao Jiang & Bin Wu. All rights reserved!

One optimal codon alignment of two input sequences is :


    Arg   Pro   Ala   Glu   Ala   Ala   Thr   Asp   Ala   Val   Arg   Gln
  1 a g a c c c g c t g a g g c g g c a a c a g a t g c g g t g a g a c a a
  1 a g a c g a g c t g a g - - - - c a a c a g a t a c g t t g a g a c a a
    Arg   Arg   Ala   Glu         Gln   Gln   Ile   Arg   Non   Asp   Lys

    Thr   Gln   Arg   Ala   Pro   Val   Gly   Val   Gly   Ala   Ala
 37 a c t c a g c g a g c a c c a g t g g g g g t g g g a g c a g c a t - -
 33 a c t c a g c c a c c a g c a g t g g c g g t g g g a g c g g c a t c a
      Leu   Ser   His   Gln   Gln   Trp   Arg   Trp   Glu   Arg   His   His
```

Table 5.3: Test results concerning frame-shift errors (2)

mutation rate	0.1	0.2	0.3
detection	4/4	4/4	4/4
localization	0/1/0/1	10/1/0/0	10/1/0/0
mutation rate	0.4	0.5	0.6
detection	3/4	4/4	2/4
localization	10/21/3/∞	9/3/3/0	24/∞/60/∞

```
      Ser   Asp   Leu   His   Arg   His   Gly   Ala   Ile   Thr   Thr   Arg
 71 - c a g a c c t g c a c c g a c a t g g a g c a a t c a c a a c t a g g
 69 c c a g c c c t c c a a a g a c a t g g c t c a a t c a c a a c t c t c
        Gln   Pro   Ser   Lys   Asp   Met   Ala   Gln   Ser   Gln   Leu   Ser

      Asn   Thr   Ala   Ala
 106 a a t a c g g c a g c t
 105 a a t a c a g c a g c t
        Ile   Gln   Gln

The minimum cost is 1901

Thank you for using DPA!
See you next time!
```

In the second group we fix the frame shifting insertion and deletion rates at 0.02. Again, we vary the mutation rate. The test results are summarized in Table 5.3. From the table, we can see that the performance of DPA worsens when more indels are introduced.

5.5 Tests concerning overlapping frames

With regard to overlapping frames, we have tested several groups of simulated sequences and 3 pairs of real sequences. The 3 pairs of real data include (i) HIVMN coding region for *gag* and *pol* genes, HIVNDK coding region for *gag* and *pol* genes. (ii) HIVMN coding region for *vif* and *vpr* genes, HIVNDK coding region for *vif* and *vpr* genes. (iii) HIVMN coding region for *tat*1 and *rev*1 genes, HIVNDK coding region for *tat*1 and *rev*1 genes. Again, we use Dayhoff PAM 120 Matrix for amino acids and DNA PAM 47 for nucleotides.

The following scripts are the outputs of DPA for 3 pairs of real data. Since the output for gene *gag* and *pol* is too long (14 pages), we only list a part of that.

A part of the output for *gag* and *pol* genes is

```
      Cys   Arg   Ala   Pro   Arg   Lys   Arg   Gly   Cys   Trp   Lys   Cys
1222 t g c a g g g c c c c t a g g a a a a g g g g c t g t t g g a a a t g t
1207 t g c a g g g c c c c t a g a a a a a a g g g c t g t t g g a a a t g c
      Cys   Arg   Ala   Pro   Arg   Lys   Lys   Gly   Cys   Trp   Lys   Cys

      Gly   Lys   Glu   Gly   His   Gln   Met   Lys   Asp   Cys   Thr   Glu
1258 g g a a a g g a a g g a c a c c a a a t g a a a g a t t g t a c t g a g
1243 g g a a g g g a a g g a c a c c a a a t g a a a g a t t g c a c t g a a
      Gly   Arg   Glu   Gly   His   Gln   Met   Lys   Asp   Cys   Thr   Glu

                            Phe   Phe   Arg   Glu   Asp   Leu   Ala   Phe
      Arg   Gln   Ala   Asn   Phe   Leu   Gly   Lys   Ile   Trp   Pro   Ser
1294 a g a c a g g c t a a t t t t t t a g g g a a g a t c t g g c c t t c c
1279 a g a c a g g c t a a t t t t t t a g g g a a g a t t t g g c c t t c c
      Arg   Gln   Ala   Asn   Phe   Leu   Gly   Lys   Ile   Trp   Pro   Ser
                            Phe   Phe   Arg   Glu   Asp   Leu   Ala   Phe
```

```
    Leu   Gln   Gly   Lys   Ala         Glu   Phe   Ser   Ser   Glu   Gln
      Cys   Lys   Gly   Arg         Arg   Asn   Phe   Pro   Gln   Ser   Arg
1330 t g c a a g g g a a g g c - - - g g a a t t t t c c t c a g a g c a g a
1315 c a c a a g g g a a g g c c g g g g a a t t t t c t t c a g a g c a g a
      His   Lys   Gly   Arg   Pro   Gly   Asn   Phe   Leu   Gln   Ser   Arg
    Pro   Gln   Gly   Lys   Ala   Gly   Glu   Phe   Ser   Ser   Glu   Gln

    Asn   Arg   Ala   Asn   Ser   Pro   Thr   Arg   Arg   Glu   Leu   Gln
      Thr   Glu   Pro   Thr   Ala   Pro   Pro   Glu   Glu   Ser   Phe   Arg
1363 a c a g a g c c a a c a g c c c c a c c a g a a g a g a g c t t c a g g
1351 c c a g a g c c a a c a g c c c c a c c a g c a g a g a g c t t c g g g
      Pro   Glu   Pro   Thr   Ala   Pro   Pro   Ala   Glu   Ser   Phe   Gly
    Thr   Arg   Ala   Asn   Ser   Pro   Thr   Ser   Arg   Glu   Leu   Arg

    Val   Trp   Gly   Arg   Asp   Asn   Asn   Ser   Leu   Ser   Glu   Ala
      Phe   Gly   Glu   Glu   Thr   Thr   Thr   Pro   Tyr   Gln   Lys   Gln
1399 t t t g g g g a a g a g a c a a c a a c t c c c t a t c a g a a g c a g
1387 t t t g g g g a g g a g a t a a c c c c c t c t - - - - - - - - - - - -
      Phe   Gly   Glu   Glu   Ile   Thr   Pro   Ser
    Val   Trp   Gly   Gly   Asp   Asn   Pro   Leu

    Gly   Glu   Glu   Ala   Gly   Asp   Asp   Arg   Gln   Gly   Pro   Val
      Glu   Lys   Lys   Gln   Glu   Thr   Ile   Asp   Lys   Asp   Leu   Tyr
1435 g a g a a g a a g c a g g a g a c g a t a g a c a a g g a c c t g t a t
1411 - - - c a g a a a c a g g a g c a g a a a g a c a a g g a a c t g t a t
            Gln   Lys   Gln   Glu   Gln   Lys   Asp   Lys   Glu   Leu   Tyr
          Ser   Glu   Thr   Gly   Ala   Glu   Arg   Gln   Gly   Thr   Val

    Ser   Phe   Ser   Phe   Pro   Gln   Ile   Thr   Leu   Trp   Gln   Arg
      Pro   Leu   Ala   Ser   Leu   Lys   Ser   Leu   Phe   Gly   Asn   Asp
1471 c c t t t a g c t t c c c t c a a a t c a c t c t t t g g c a a c g a c
1444 c c t t t a g c t t c c c t c a a a t c a c t c t t t g g c a a c g a c
      Pro   Leu   Ala   Ser   Leu   Lys   Ser   Leu   Phe   Gly   Asn   Asp
    Ser   Phe   Ser   Phe   Pro   Gln   Ile   Thr   Leu   Trp   Gln   Arg
```

The output for *vif* and *vpr* genes is

```
This is DPA, Version 0.90 Beta1.

Written by Bin Wu <binwu@church.dcss.McMaster.CA>.

Copyright (c) 1998 by Tao Jiang & Bin Wu. All rights reserved!


One optimal codon alignment of two input sequences is :


     Met   Glu   Asn   Arg   Arg   Gln   Val   Met   Ile   Val   Trp   Gln
   1 a t g g a a a a c a g a c g g c a g g t g a t g a t t g t g t g g c a a
   1 a t g g a a a a c a g a t g g c a g g t g a t g a t t g t g t g g c a a
     Met   Glu   Asn   Arg   Trp   Gln   Val   Met   Ile   Val   Trp   Gln

     Ala   Asp   Arg   Met   Arg   Ile   Arg   Thr   Trp   Lys   Ser   Leu
  37 g c a g a c a g g a t g a g g a t t a g a a c a t g g a a a a g t t t a
  37 g t a g a c a g g a t g a g g a t t a a c a c a t g g a a a a g t t t a
     Val   Asp   Arg   Met   Arg   Ile   Asn   Thr   Trp   Lys   Ser   Leu

     Val   Lys   His   His   Met   Tyr   Ile   Ser   Lys   Lys   Ala   Lys
  73 g t a a a a c a c c a t a t g t a t a t t t c a a a g a a a g c t a a a
  73 g t a a a a t a c c a t a t g t a t g t t t c a a a g a a a g c t a a c
     Val   Lys   Tyr   His   Met   Tyr   Val   Ser   Lys   Lys   Ala   Asn

     Gly   Arg   Phe   Tyr   Arg   His   His   Tyr   Glu   Ser   Thr   His
 109 g g a c g g t t t t a t a g a c a t c a c t a t g a a a g c a c t c a t
 109 a g a t g g t t t t a t a g a c a t c a c t a t g a c a g c c a c c a c
     Arg   Trp   Phe   Tyr   Arg   His   His   Tyr   Asp   Ser   His   His

     Pro   Arg   Ile   Ser   Ser   Glu   Val   His   Ile   Pro   Leu   Gly
 145 c c a a g a a t a a g t t c a g a a g t a c a c a t c c c a c t a g g g
 145 c c a a a a a t a a g t t c a g a a g t a c a c a t c c c a c t a g g a
     Pro   Lys   Ile   Ser   Ser   Glu   Val   His   Ile   Pro   Leu   Gly

     Asp   Ala   Arg   Leu   Val   Ile   Thr   Thr   Tyr   Trp   Gly   Leu
 181 g a t g c t a g a t t g g t a a t a a c a a c a t a t t g g g g t c t g
 181 g a a g c t a g a c t g g t a g t a a c a a c a t a t t g g g g t c t g
     Glu   Ala   Arg   Leu   Val   Val   Thr   Thr   Tyr   Trp   Gly   Leu
     His   Thr   Gly   Glu   Arg   Asp   Trp   His   Leu   Gly   Gln   Gly
217 c a t a c a g g a g a a a g a g a c t g g c a t t t a g g t c a g g g a
217 c a t a c a g g a g a a a a a g a a t g g c a t c t g g g t c a g g g a
     His   Thr   Gly   Glu   Lys   Glu   Trp   His   Leu   Gly   Gln   Gly

     Val   Ser   Ile   Glu   Trp   Arg   Lys   Lys   Arg   Tyr   Ser   Thr
253 g t c t c c a t a g a a t g g a g g a a a a a g a g a t a t a g c a c a
253 g t c t c c a t a g a a t g g a g g a a a a g g a g a t a t a g c a c a
     Val   Ser   Ile   Glu   Trp   Arg   Lys   Arg   Arg   Tyr   Ser   Thr

     Gln   Val   Asp   Pro   Asp   Leu   Ala   Asp   His   Leu   Ile   His
289 c a a g t a g a c c c t g a c c t a g c a g a c c a c c t a a t t c a t
289 c a a g t a g a c c c t g g c c t g g c a g a c c a a c t a a t t c a t
     Gln   Val   Asp   Pro   Gly   Leu   Ala   Asp   Gln   Leu   Ile   His

     Leu   His   Tyr   Phe   Asp   Cys   Phe   Ser   Asp   Ser   Ala   Ile
325 c t g c a t t a c t t t g a t t g t t t t t c a g a c t c t g c c a t a
325 a t g t a t t a t t t t g a t t g t t t t g c a g a a t c t g c t a t a
     Met   Tyr   Tyr   Phe   Asp   Cys   Phe   Ala   Glu   Ser   Ala   Ile

     Arg   Lys   Ala   Ile   Leu   Gly   His   Arg   Val   Ser   Pro   Ile
361 a g a a a g g c c a t a t t a g g a c a t a g a g t t a g t c c t a t t
361 a g a a a a g c c a t a t t a g g a c a t a t a g t t a g t c c t a g t
     Arg   Lys   Ala   Ile   Leu   Gly   His   Ile   Val   Ser   Pro   Ser

     Cys   Glu   Phe   Gln   Ala   Gly   His   Asn   Lys   Val   Gly   Pro
397 t g t g a a t t t c a a g c a g g a c a t a a c a a g g t a g g a c c t
397 t g t g a g t a t c a a g c a g g a c a t a a c a a g g t a g g a t c c
     Cys   Glu   Tyr   Gln   Ala   Gly   His   Asn   Lys   Val   Gly   Ser

     Leu   Gln   Tyr   Leu   Ala   Leu   Thr   Ala   Leu   Ile   Thr   Pro
433 c t a c a g t a c t t g g c a c t a a c a g c a t t a a t a a c a c c a
433 t t a c a g t a t t t g g c a c t a g c a g c a t t a a t a g c a c c a
     Leu   Gln   Tyr   Leu   Ala   Leu   Ala   Ala   Leu   Ile   Ala   Pro
```

```
      Lys   Lys   Ile   Lys   Pro   Pro   Leu   Pro   Ser   Val   Lys   Lys
  469 a a a a a g a t a a a g c c a c c t t t g c c t a g t g t t a a g a a a
  469 a a a a a g a t a a a g c c a c c t t t g c c t a g t g t t a g g a a g
      Lys   Lys   Ile   Lys   Pro   Pro   Leu   Pro   Ser   Val   Arg   Lys

                                   Met   Glu   Gln   Ala   Pro   Glu   Asp
      Leu   Thr   Glu   Asp   Arg   Trp   Asn   Lys   Pro   Gln   Lys   Thr
  505 c t g a c a g a g g a t a g a t g g a a c a a g c c c c a g a a g a c c
  505 c t a a c a g a a g a t a g a t g g a a c a a g c c c c a g a a g a c c
      Leu   Thr   Glu   Asp   Arg   Trp   Asn   Lys   Pro   Gln   Lys   Thr
                                   Met   Glu   Gln   Ala   Pro   Glu   Asp

    Gln   Gly   Pro   Gln   Arg   Glu   Pro   Tyr   Asn   Gln   Trp   Ala
      Lys   Gly   His   Arg   Gly   Ser   His   Thr   Ile   Asn   Gly   His
  541 a a g g g c c a c a g a g g g a g c c a t a c a a t c a a t g g g c a c
  541 a a g g g c c g c a g a g g g a g c c a t a c a a t g a a t g g a c a t
      Lys   Gly   Arg   Arg   Gly   Ser   His   Thr   Met   Asn   Gly   His
    Gln   Gly   Pro   Gln   Arg   Glu   Pro   Tyr   Asn   Glu   Trp   Thr

    Leu   Glu   Leu   Leu   Glu   Glu   Leu   Lys   Asn   Glu   Ala   Val
      Non
  577 t a g a g c t t t t a g a g g a g c t t a a g a a t g a a g c t g t t a
  577 t a g a g c t t t t a g a g g a g c t t a a g a g t g a a g c t g t c a
      Non
    Leu   Glu   Leu   Leu   Glu   Glu   Leu   Lys   Ser   Glu   Ala   Val

    Arg   His   Phe   Pro   Arg   Ile   Trp   Leu   His   Gly   Leu   Gly

  613 g a c a t t t t c c t a g g a t a t g g c t c c a t g g c t t a g g g c
  613 g a c a t t t t c c t a g g a t a t g g c t c c a t a g c t t a g g a c

    Arg   His   Phe   Pro   Arg   Ile   Trp   Leu   His   Ser   Leu   Gly

    Gln   His   Ile   Tyr   Glu   Thr   Tyr   Gly   Asp   Thr   Trp   Ala

  649 a a c a t a t c t a t g a a a c t t a t g g g a t a c t t g g g c a g
```

```
 649 a a c a t a t c t a t g a a a c t t a t g g g g a t a c c t g g g c a g

   Gln   His   Ile   Tyr   Glu   Thr   Tyr   Gly   Asp   Thr   Trp   Ala

   Gly   Val   Glu   Ala   Ile   Ile   Arg   Ile   Leu   Gln   Gln   Leu

 685 g a g t g g a a g c c a t a a t a a g a a t t c t a c a a c a a c t g c
 685 g t g t t g a a g c t a t a a t a a g a a t t c t g c a a c a a c t a c

   Gly   Val   Glu   Ala   Ile   Ile   Arg   Ile   Leu   Gln   Gln   Leu

   Leu   Phe   Ile   His   Phe   Arg   Ile   Gly   Cys   Arg   His   Ser

 721 t g t t t a t t c a t t t c a g a a t t g g g t g t c g a c a t a g c a
 721 t g t t t a t t c a t t t c a g a a t t g g g t g t c a a c a t a g c a

   Leu   Phe   Ile   His   Phe   Arg   Ile   Gly   Cys   Gln   His   Ser

   Arg   Ile   Gly   Ile   Ile   Arg   Gln   Arg   Arg   Ala   Arg   Asn

 757 g a a t a g g c a t t a t t c g a c a g a g g a g a g c a a g a a a t g
 757 g a a t a a g t a t t a c t c g a c a g a g a a g a g c a a g a a a t g

   Arg   Ile   Ser   Ile   Thr   Arg   Gln   Arg   Arg   Ala   Arg   Asn

   Gly   Ala   Ser   Arg   Ser   Non

 793 g a g c c a g t a g a t c c t a g
 793 g a t c c a g t a g a t c c t a a

   Gly   Ser   Ser   Arg   Ser   Non

The minimum cost is 7962

Thank you for using DPA!
See you next time!
```

The output for *tat*1 and *rev*1 genes is

```
This is DPA, Version 0.90 Beta1.

Written by Bin Wu <binwu@church.dcss.McMaster.CA>.

Copyright (c) 1998 by Tao Jiang & Bin Wu. All rights reserved!


 One optimal codon alignment of two input sequences is :


     Met   Glu   Pro   Val   Asp   Pro   Arg   Leu   Glu   Pro   Trp   Lys

   1 a t g g a g c c a g t a g a t c c t a g a c t a g a g c c c t g g a a g

   1 a t g g a t c c a g t a g a t c c t a a t c t a g a g t c c t g g a a c

     Met   Asp   Pro   Val   Asp   Pro   Asn   Leu   Glu   Ser   Trp   Asn


     His   Pro   Gly   Ser   Gln   Pro   Lys   Thr   Ala   Cys   Thr   Thr

  37 c a t c c a g g a a g t c a g c c t a a g a c t g c t t g t a c c a c t

  37 c a t c c a g g a a g t c a g c c t a g g a c t g c t t g t a a t a a g

     His   Pro   Gly   Ser   Gln   Pro   Arg   Thr   Ala   Cys   Asn   Lys


     Cys   Tyr   Cys   Lys   Lys   Cys   Cys   Phe   His   Cys   Gln   Val

  73 t g c t a t t g t a a a a a g t g t t g c t t t c a t t g c c a a g t t

  73 t g t c a t t g t a a a a a g t g t t g c t a t c a t t g c c a a g t t

     Cys   His   Cys   Lys   Lys   Cys   Cys   Tyr   His   Cys   Gln   Val


                                                                Met   Ala

     Cys   Phe   Thr   Lys   Lys   Ala   Leu   Gly   Ile   Ser   Tyr   Gly

 109 t g t t t c a c a a a a a a a g c c t t a g g c a t c t c c t a t g g c

 109 t g c t t c a t a a c g a a a g g c t t a g g c a t c t c c t a t g g c

     Cys   Phe   Ile   Thr   Lys   Gly   Leu   Gly   Ile   Ser   Tyr   Gly

                                                                Met   Ala


       Gly   Arg   Ser   Gly   Asp   Ser   Asp   Glu   Glu   Leu   Leu   Lys

     Arg   Lys   Lys   Arg   Arg   Gln   Arg   Arg   Arg   Ala   Pro   Glu

 145 a g g a a g a a g c g g a g a c a g c g a c g a a g a g c t c c t g a a

 145 a g g a a g a a g c g g a g a c a g c g a c g a a a a c c t c c t c a a
```

```
      Arg   Lys   Lys   Arg   Arg   Gln   Arg   Arg   Lys   Pro   Pro   Gln
        Gly   Arg   Ser   Gly   Asp   Ser   Asp   Glu   Asn   Leu   Leu   Lys

        Thr   Val   Arg   Leu   Ile   Lys   Phe   Leu   Tyr   Gln   Ser
      Asp   Ser   Gln   Thr   His   Gln   Val   Ser   Leu   Pro   Lys
181 g a c a g t c a g a c t c a t c a a g t t t c t c t a c c a a a g c a
181 g g c g a t c a g g c t c a t c a a g t t c c t a t a c c a g a g c a
      Gly   Asp   Gln   Ala   His   Gln   Val   Pro   Ile   Pro   Glu
        Ala   Ile   Arg   Leu   Ile   Lys   Phe   Leu   Tyr   Gln   Ser

The minimum cost is 3852

Thank you for using DPA!
See you next time!
```

From the test results, we can see that (i) The indel rate for short genes is lower than that for long genes. There is no indel in the alignments for the last two pairs of real sequences. (ii) The indel rate in overlapping regions is almost the same as that in non-overlapping regions. (iii) The mutation rate in the real data that we tested is usually lower than 0.3.

In the next chapter, we give conclusions and future work for our project.

Chapter 6

Conclusions and Future Work

In this thesis, we have studied an alignment model recently proposed by J. Hein and related algorithms for comparing coding DNA sequences which takes into account both DNA and protein information. Basing on Hein's model, we have proposed a mildly simplified model, *i.e.* the *context-free codon alignment* model, and presented a much more efficient algorithm for this simpler model. Furthermore, we have extended our algorithm to handle frame-shift errors and overlapping frames using a heuristic approach.

All of the algorithms have been implemented and tested on both real and simulated sequences. The test results show that the algorithm for our simplified model and the algorithm for Hein's model produce almost identical alignment in most cases. Also, our program can correctly detect and locate frame-shift errors for reasonable indel and mutation rates.

A disadvantage of our program is that it can't detect two frame-shift errors which are close to each other. To make up for this, we can use a local optimization method, *i.e.* we do not penalize two "complementary" frame-shift errors which are close to each other and realign that region taking this into account.

Future research may be concerned with (i) exact algorithms for the overlapping frames problem, (ii) speeding up our frame-shift algorithm so that it can handle atomic alignment involving two indels, and (iii) biologically plausible combinations of cost parameters from protein and DNA levels.

Finally, we hope our model will be accepted by biologists and our program will be widely used in practice.

Bibliography

[1] Michael S. Waterman, Introduction to computational biology, *Chapman & Hall Press*, 1995.

[2] Dan Gusfield, Algorithms on strings, trees, and sequences, *Cambridge University Press*, 1997.

[3] João Setubal and João Meidanis, Introduction to computational molecular biology, *PWS Publishing Company*, 1997.

[4] J. Hein, An algorithm combining DNA and protein alignment, *J. Theo. Biol.* 167 pp. 169-174, 1994.

[5] J. Hein and J. Støvlbæk, Genomic alignment, *J. Mol. Evol.* 38, pp. 310-316, 1994.

[6] J. Hein and J. Støvlbæk, Combined DNA and protein alignment, *Methods in Enzymology* 266, pp. 402-418, 1996.

[7] Y. Hua, T. Jiang, and B. Wu, Aligning DNA sequences to minimize the change in protein, to appear in Annual Conference on *Combinatorial Pattern Matching*, 1998.

[8] Y. Hua, An improved algorithm for combining DNA and protein alignment, *M. Eng. Thesis, McMaster University*, 1997.

[9] C. Pedersen, R. Lyngsø, and J. Hein, Comparison of coding DNA, to appear in *BRICS technical report*, RS-98-03, 1998.

[10] L. Arvestad, Aligning coding DNA in the presence of frame-shift errors, in Annual Conference on *Combinatorial Pattern Matching*, vol. 1264 of LNCS, pp. 180-190, 1997.

[11] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcott, A model of evolutionary change in proteins, *Atlas of Protein Sequence and Structure*, 5 suppl. 3, pp. 345-352, 1978.

[12] O. Gotoh, An improved algorithm for matching biological sequences, *J. Mol. Biol.* 162, pp. 705-708, 1981.

[13] T.F. Smith and M. Waterman, Comparison of biosequences, *Adv. Appl. Math.*, vol. 2, pp. 428-489, 1981.

[14] S. Needlemann and C. Wunsch, A general method applicable to the search for similarities in the amino acid sequences of two proteins, *J. Mol. Biol.* 48, pp. 443-453, 1970.

[15] D. Sankoff, Matching sequences under deletion/insertion constraints, *Proc. Nat. Acad. Sci.* 69(1), pp. 4-6, 1972.

[16] P. Sellers, On the theory and computation of evolutionary distances, *SIAM J. Appl. Math.* 26, pp. 787-793, 1974.

[17] D. Hirschberg, A linear space algorithm for computing maximal common subsequences, *Comm. ACM*, vol. 18, pp. 341-343, 1975.

[18] X. Guan and E.C. Uberbacher, Alignments of DNA and protein sequences containing frame-shift errors, *CABIOS*, vol. 12, no. 1, pp. 31-40, 1996.

[19] D. Sankoff, R. Cedergren and G. Lapalme, Frequency of insertion-deletion, transversion, and transition in the evolution of 5S ribosomal RNA, *J. Mol. Evol.* 7, pp.133-149, 1976.

[20] Y. Xu, R.J. Mural, and E.C. Uberbacher, Correcting sequencing errors in DNA coding regions using a dynamic programming approach, *CABIOS*, vol. 11, pp. 117-124, 1995.

[21] H. Peltola, H. Sõderlund, and E. Ukkonen, Algorithms for the search of amino acid patterns in nucleic acid sequences, *Nucleic acids research*, vol. 14, no. 1, pp. 99-107, 1986.

[22] Z. Zhang, W.R. Pearson, and W. Miller, Aligning a DNA sequence with a protein sequence, *RECOMB 97*, 1997.