Comparison of Coding DNA

Christian N. S. Pedersen
Rune B. Lyngsø
Jotun Hein

BRICS Report Series RS-98-3

ISSN 0909-0878 January 1998

Comparison of coding DNA

Christian N. S. Pedersen[1*] Rune Lyngsø[1*] Jotun Hein[2*]

[1] BRICS[†], Department of Computer Science, University of Aarhus
[2] Institute of Biological Sciences, University of Aarhus

Abstract

We discuss a model for the evolutionary distance between two coding DNA sequences which specializes to the DNA/protein model proposed in Hein [3]. We discuss the DNA/protein model in details and present a quadratic time algorithm that computes an optimal alignment of two coding DNA sequences in the model under the assumption of affine gap cost. The algorithm solves a conjecture in [3] and we believe that the constant factor of the running time is sufficiently small to make the algorithm feasible in practice.

1 Introduction

A straightforward model of the evolutionary distance between two coding DNA sequences is to ignore the encoded proteins and compute the distance in some evolutionary model of DNA. We say that such a model is a DNA level model. The evolutionary distance between two sequences in a DNA level model can most often be formulated as a classical alignment problem and be efficiently computed using a dynamic programming approach [7, 9, 10, 11].

It is well known that proteins evolve slower than its coding DNA, so it is usually more reliable to describe the evolutionary distance based on a comparison of the encoded proteins rather than on a comparison of the coding DNA itself. Hence, most often the evolutionary distance between two coding DNA sequences is modeled in terms of amino acid events, such as substitution of a single amino acid and insertion-deletion of consecutive amino acids, necessary to transform the one encoded protein into the other encoded protein. We say that such a model is a protein level model. The evolutionary distance between two coding DNA sequences in a protein level model can most often be formulated as a classical alignment problem of the two encoded proteins. Even though a protein level model is usually more

[*]E-mail: `cstorm@brics.dk`, `rlyngsoe@brics.dk` and `jotun@pop.bio.aau.dk`

[†]Basic Research in Computer Science,
Centre of the Danish National Research Foundation.

reliable than a DNA level model, it falls short because it postulates that all insertions and deletions on the underlying DNA occur at codon boundaries and because it ignores similarities on the DNA level.

In this paper we present a model of the evolutionary distance between two coding DNA sequences in which a nucleotide event is penalized by the change it induces on the DNA as well as on the encoded protein. The model is a natural combination of a DNA level model and a protein level model. The DNA/protein model introduced in Hein [3, 5] is a biological reasonable instance of the general model in which the evolution of coding DNA is idealized to involve only substitution of a single nucleotide and insertion-deletion of a multiple of three nucleotides. Hein [3, 5] presents an $\mathrm{O}(n^2m^2)$ time algorithm for computing the evolutionary distance in the DNA/protein model between two sequences of length n and m. This algorithm assumes certain properties of the cost function. We discuss these properties and present an $\mathrm{O}(nm)$ time algorithm that solves the same problem under the assumption of affine gap cost. The practicality of an algorithm not only depends on the asymptotic running time but also on the constant factor hidden by the use of O-notation. To determine the distance between two sequences of length n and m our algorithm computes $400nm$ table entries. Each computation involves a few additions, table lookups and comparisons. We believe the constant factor is sufficiently small to make the algorithm feasible in practice.

The problem of comparing coding DNA is also discussed by Arvestad [1] and Hua, Jiang and Wu [6]. The models discussed in these papers are inspired by the DNA/protein model in Hein [3, 5] but differs in the interpretation of gap cost. A heuristic algorithm for solving the alignment problem in the DNA/protein model is described In Hein [4]. A related problem of how to compare a coding DNA sequence with a protein has been discussed in [8, 12].

The rest of this paper is organized as follows: In section 2 we introduce and discuss the DNA/protein model. In section 3 we present a simple alignment algorithm. In section 4 we present quadratic time alignment algorithm. In section 5 we discuss improvements and future work.

2 The DNA/protein model

Let $a = a_1a_2a_3 \ldots a_{3n-2}a_{3n-1}a_{3n}$ be a coding sequence of DNA of length $3n$ with a reading frame starting at a_1. We introduce the notation $a_1^ia_2^ia_3^i$ to denote the ith codon $a_{3i-2}a_{3i-1}a_{3i}$ and the notation A_i to describe the amino acid coded by the ith codon. The amino acid sequence $A = A_1A_2 \ldots A_n$ describes the protein coded by a. An evolutionary event e on the DNA that transforms a to a' will also change the encoded protein from A to A'. Since amino acids are coded by several codons, the proteins A and A' might be identical. The cost of e should reflect the changes on the DNA as well as the changes on the encoded protein.

$$cost(a \xrightarrow{e} a') = cost_d(a \xrightarrow{e} a') + cost_p(A \xrightarrow{e} A') \tag{1}$$

We say that $cost_d(a \xrightarrow{e} a')$ is the cost of e on the DNA level and that $cost_p(A \xrightarrow{e} A')$ is the cost of e on the protein level. In this paper we assume that the DNA level cost and the protein level cost are combined by addition but other combination functions $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ could of course also be considered. The cost of a sequence E of evolutionary events $e_1, e_2, \ldots, e_k$ transforming $a^{(0)}$ to $a^{(k)}$ as $a^{(0)} \xrightarrow{e_1} a^{(1)} \xrightarrow{e_2} a^{(2)} \xrightarrow{e_3} \cdots \xrightarrow{e_k} a^{(k)}$ is defined as some function of the costs of each event. In the rest of this paper we will assume that this function is the sum of the costs of each event.

$$cost(a^{(0)} \xrightarrow{E} a^{(k)}) = \sum_{i=1}^{k} cost(a^{(i-1)} \xrightarrow{e_i} a^{(i)}) \tag{2}$$

We define the distance between two coding sequences of DNA a and b according to the parsimony principle as the minimum cost of a sequence of evolutionary events which transforms a to b.

$$dist(a, b) = \min\{cost(a \xrightarrow{E} b) \mid E \text{ is a sequence of events}\} \tag{3}$$

In order to compute $dist(a, b)$ we have to specify the set of allowed evolutionary events and define the cost of each event on the DNA level as well as on the protein level. The choice of evolutionary events and cost function influences both the biological relevance of the distance measure and the computational complexity of computing the distance.

The DNA/protein model introduced in [3] can be described as an instance of the above model where we idealize the evolution of a coding sequence of DNA to involve only substitution of a single nucleotide and insertion or deletion of a multiple of three consecutive nucleotides. The reason why gap lengths are restricted to a multiple of three is because an insertion or deletion of length not divisible by three changes the reading frame. This is called a frame shift and it may change the entire remaining amino acid sequence as illustrated in figure 1. Frame shifts are believed to be rare biological events, so it is not unreasonable to leave them out of the model.

The *DNA level cost* in the DNA/protein model is defined in the classical way by specifying a substitution cost and a gap cost. The cost of substituting a nucleotide σ with σ' is $c_d(\sigma, \sigma')$ for some metric c_d on nucleotides and the cost of inserting or deleting $3k$ consecutive nucleotides is $g_d(3k)$ for some convex[1] function $g_d : \mathbb{N} \to \mathbb{R}^+$. We note that $cost_d(a \xrightarrow{e} a')$ in the

[1]A convex function fulfills that $f(i + j) \leq f(i) + f(j)$. A convex gap cost function implies that an insertion-deletion of a consecutive block of nucleotides is best explained as a single event.

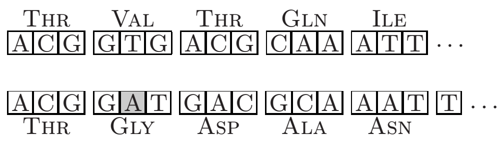


Figure 1: An insertion or deletion of length not divisible by three changes the reading frame.

DNA/protein model only depends on e (and not a and a'), so the order of events is irrelevant on the DNA level. The *protein level cost* of a nucleotide event which changes the encoded protein from A to A' should somehow reflect the difference between protein A and protein A'. Hence, $cost_p(A \xrightarrow{e} A')$ is defined as the minimum cost of a distance alignment of A and A' where we allow substitution of a single amino acid and insertion-deletion of consecutive amino acids. The substitution cost is given by a metric c_p on amino acids and the gap cost is given by a convex function $g_p : \mathbb{N} \to \mathbb{R}^+$ fulfilling that $g_p(0) = 0$. We use $dist_p(A, A')$ to denote the minimum cost of such an alignment of A and A' and note that $dist_p$ is commutative.

Except for the restriction on the length of an insertion or a deletion the DNA/protein model allows the traditional set of symbol based nucleotide events. An *alignment* of two sequences describes a set of substitution or insertion-deletion events necessary to transform the one sequence into the other sequence. The set of events is usually described by a matrix or a path in a graph as illustrated in figure 2.

$$\begin{bmatrix} T & T & G & C & T & - & - & - & C \\ T & - & - & - & C & A & T & G & C \end{bmatrix}$$

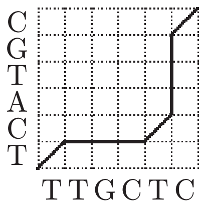


Figure 2: An alignment can be described by a matrix or a path in the alignment graph. The above alignment describes three matches and two gaps of combined length six.

The cost of an alignment is the optimal cost of any sequence of the events described by the alignment. Hence, the evolutionary distance in the DNA/protein model between two coding DNA sequences is the cost of an optimal alignment in the model. If the cost of any sequence of events is independent of the order but only depends on the set of events, then an optimal alignment can be computed efficiently using dynamic programming [7, 9, 10, 11]. The protein level cost in the DNA/protein model however depends on the order of events, so we cannot use a classical alignment algorithm to compute an optimal alignment in the DNA/protein model. In the rest of this section we will examine the protein level cost further in order to be able to formulate restrictions that allow an efficient alignment algorithm in the DNA/protein model.

A nucleotide event affects nucleotides in one or more consecutive codons. Since a nucleotide event in the DNA/protein model is assumed not to change the reading frame, then only the amino acids encoded by the affected codons are affected by the nucleotide event. A nucleotide event thus changes protein $A = UXV$ to protein $A' = UX'V$ where X and X' are the amino acids affected by nucleotide event. Hein [3] implicitly assumes that $dist_p(A, A')$ is the cost of an alignment of X and X' describing the minimum number of insertions or deletions. This property is essential to the formulation of alignment algorithms but it inflicts some restrictions on the substitution cost c_p and the gap cost g_p as summarized in lemma 1.

$$\left[\begin{array}{l|c|ll} A_1\,A_2\cdots A_{i-1} & A_i & A_{i+1}\cdots & A_n \\ A_1\,A_2\cdots A_{i-1} & A'_i & A_{i+1}\cdots & A_n \end{array}\right]$$

(a) A substitution in the ith codon.

$$\left[\begin{array}{l|c|l} A_1\,A_2\cdots A_{i-1}\,A_i & A_{i+1}\cdots A_{i+k} & A_{i+k+1}\cdots A_n \\ A_1\,A_2\cdots A_{i-1}\,A_i & - \ \cdots \ - & A_{i+k+1}\cdots A_n \end{array}\right]$$

(b) An insertion-deletion of $3k$ nucleotides affecting exactly k codons.

$$\left[\begin{array}{l|c|l} A_1\,A_2\cdots A_{i-1} & A_i\cdots A_{j-1}\,A_j\,A_{j+1}\cdots A_{i+k} & A_{i+k+1}\cdots A_n \\ A_1\,A_2\cdots A_{i-1} & -\cdots \ - \quad v \quad - \ \cdots \ - & A_{i+k+1}\cdots A_n \end{array}\right]$$

(c) An insertion-deletion of $3k$ nucleotides affecting $k + 1$ codons. The remaining amino acid v is matched with one of the amino acids affected by the deletion.

Figure 3: The protein level cost of a nucleotide event can be determined by considering only the amino acids affected by the event.

Lemma 1 *Assume a nucleotide event changes $A = UXV$ to $A' = UX'V$. Let $n = |A|$ and $k = ||A| - |A'||$. If there for any amino acids σ, τ and for all $0 < l \leq n - k$ exists $0 \leq j \leq k$ such that $c_p(\sigma, \tau) + g_p(j) + g_p(k - j) \leq g_p(l) + g_p(l + k)$, then $dist_p(A, A')$ is the cost of an alignment describing exactly k insertions or deletions. Furthermore $dist_p(A, A')$ only depends on X and X'.*

Proof. We will argue that the assumption stated in the lemma implies that $dist_p(A, A')$ is the cost of one of the alignments illustrated in figure 3. These alignments all describe the minimum number of insertions or deletions and only the sub-alignment of X and X', as illustrated by the shaded parts, contributes to the cost. We split the argumentation depending on the event. Since $dist_p(A, A')$ is equal to $dist_p(A', A)$ then the cost of an insertion transforming A to A' is equal to the cost of a deletion transforming A' to A. We thus only consider substitutions and deletions.

A substitution of a nucleotide in the ith codon of A transforms A_i to A'_i. The alignment in figure 3(a) describes no insertion-deletions and has cost $c_p(A_i, A'_i)$. Any other alignment of A and A' must describe an equal number of insertions and deletions, so by convexity of g_p the cost is at least $2 \cdot g_p(l)$ for some $0 < l \leq n$. The assumption in the lemma implies that $c_p(A_i, A'_i) \leq 2 \cdot g_p(l)$ for any $0 < l \leq n$, so the protein level cost of the substitution is $c_p(A_i, A'_i)$.

A deletion of $3k$ nucleotides affects k or $k+1$ consecutive codons. If the deletion affects exactly k codons then it transforms $A = A_1 A_2 \cdots A_n$ to $A' = A_1 A_2 \cdots A_i A_{i+k+1} A_{i+k+2} \cdots A_n$. Any alignment of A and A' must describe l insertions and $l+k$ deletions for some $0 \leq l \leq n-k$, so the cost is at least $g_p(l) + g_p(l+k)$. The alignment in figure 3(b) describes k deletions and has cost $g_p(k)$. The assumption in the lemma and the convexity of g_p implies that $g_p(k) \leq g_p(j) + g_p(k-j) \leq g_p(l) + g_p(l+k)$ for all $l > 0$, so the protein level cost of a deletion affecting k codons is $g_p(k)$.

If the deletion affects $k+1$ codons, say by deleting the $3k$ nucleotides $a_3^i a_1^{i+1} a_2^{i+1} a_3^{i+1} \cdots a_1^{i+k} a_2^{i+k}$, then it transforms $A = A_1 A_2 \cdots A_n$ to $A' = A_1 A_2 \cdots A_{i-1} v A_{i+k+1} \cdots A_n$ where v is the amino acid coded by $a_1^i a_2^i a_3^{i+k}$. We say that v is the remaining amino acid and $a_1^i a_2^i a_3^{i+k}$ is the remaining codon. Any alignment of A and A' describing exactly k deletions must align v with A_{i+j} for some $0 \leq j \leq k$, so by convexity of g_p the cost is at least $g_p(j) + c_p(A_{i+j}, \sigma) + g_p(k-j)$. The alignment in figure 3(c) illustrates one of the $k+1$ alignments of A and A' where v is aligned with an affected amino acids and all non-affected amino acids are aligned. Such an alignment describes exactly k deletions and the cost of the optimal alignment among them has cost

$$\min_{j=0,1,\ldots,k} \{g_p(j) + c_p(A_{i+j}, \sigma) + g_p(k-j)\}, \tag{4}$$

and is thus optimal for any alignment describing exactly k deletions. Any other alignment of A and A' must describe l insertions and $l+k$ deletions for some $0 < l \leq n-k$, so the cost is at least $g_p(l) + g_p(l+k)$. The assumption in the lemma implies that the cost given by expression (4) is less than or equal to $g_p(l) + g_p(l+k)$, so that the protein level cost of a deletion affecting $k+1$ codons is given expression (4). □

The assumption in lemma 1 is sufficient to ensure that we can compute the protein level cost of a nucleotide event efficiently, but the formulation of the lemma it is to general to make the assumption necessary. The following example however suggests when the assumption is necessary. Consider a deletion of three nucleotides that transforms the six amino acids ABEFCD to ABGCD, i.e. $X = \text{EF}$ and $X' = \text{G}$. If we assume that $c_p(\text{E}, \text{G}) \leq c_p(\text{F}, \text{G})$ then the cost of the alignment in figure 4 (left) is $c_p(\text{E}, \text{G}) + g_p(1)$ while the cost of the alignment in figure 4 (right) is $g_p(2) + g_p(1)$. If the assumption in lemma 1 does not hold then $g_p(2) + g_p(1)$ might be less than $c_p(\text{E}, \text{G}) + g_p(1)$ because $c_p(\text{E}, \text{G})$ can be arbitrary large. Hence, the protein level cost of the deletion would not be the cost of an alignment describing the minimum number of insertion-deletions.

$$\begin{bmatrix} \text{A} & \text{B} & \text{E} & \text{F} & \text{C} & \text{D} \\ \text{A} & \text{B} & \text{G} & - & \text{C} & \text{D} \end{bmatrix} \qquad \begin{bmatrix} \text{A} & \text{B} & \text{E} & \text{F} & - & \text{C} & \text{D} \\ \text{A} & \text{B} & - & - & \text{G} & \text{C} & \text{D} \end{bmatrix}$$

Figure 4: Two alignments of the amino acids ABEFCD and ABGCD.

An often used gap cost is the affine gap cost function $g_p(k) = \alpha_p + \beta_p k$ for some $\alpha_p, \beta_p \geq 0$. With affine gap cost the assumption in lemma 1 becomes $c_p(\sigma, \tau) + \alpha_p + \beta_p k \leq 2 \cdot \alpha_p + \beta_p(k + 2l)$ for any amino acids σ, τ and all lengths $0 < l \leq n - k$. This simplifies to $c_p(\sigma, \tau) \leq \alpha_p + 2\beta_p$ for all amino acids σ, τ. This is a biological reasonable assumption since it reflects that insertions and deletions are rare compared to substitutions.

3 A simple alignment algorithm

Let $a_1 a_2 \cdots a_{3n}$ and $b_1 b_2 \cdots b_{3m}$ be two coding sequences of DNA. We want to compute an *optimal alignment* of a and b in the DNA/protein model. An alignment of a and b describes a set of events between a and b but not the order of the events. The DNA level cost of an alignment is easy to determine as it is independent of the order, so it is just the sum of the cost of all the events described by the alignment. The protein level cost however depends on the order of the events. An obvious way to determine the protein level cost of an alignment is to minimize over all possible sequences of the events described by the alignment. This method is, however, not feasible in practice due to the factorial number of possible sequences one has to consider. If the substitution cost c_p and gap cost g_p fulfill lemma 1, then the protein level cost of a nucleotide event only depends on the affected codons. We can use this property to decompose the computation of the protein level cost of an alignment into smaller subproblems.

We decompose the alignment into *codon alignments*. A codon alignment is a minimal part of the alignment that corresponds to a path connecting two nodes $(3i', 3j')$ and $(3i, 3j)$ in the alignment graph. The alignment in figure 2 is a codon alignment. We can decompose any alignment uniquely into codon alignments as illustrated in figure 6. The assumption that an insertion or deletion has length a multiple of three implies that there are eleven distinct types of codon alignments as illustrated in figure 5.

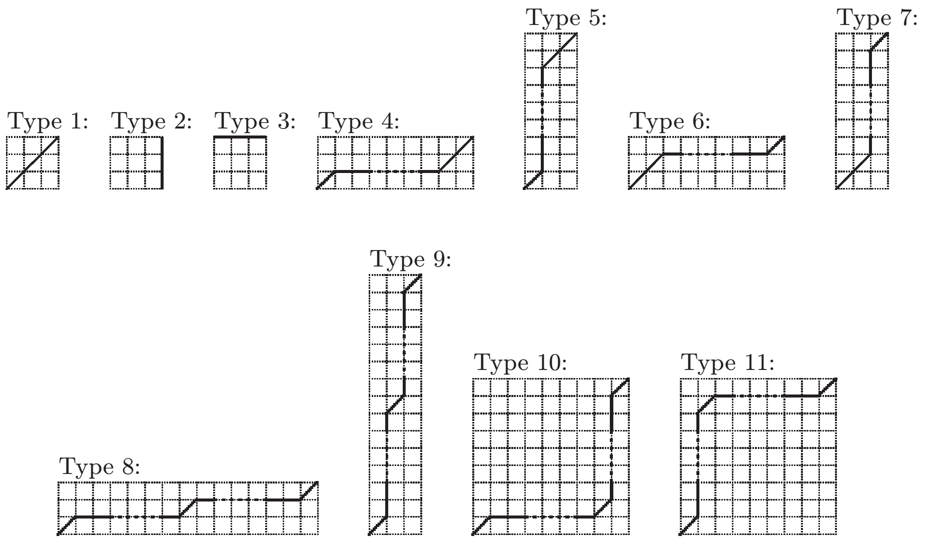


Figure 5: The eleven types of codon alignments. The length of an insertion or deletion is a multiple of three.

We observe that nucleotide events described by two different codon alignment in the decomposition of an alignment do not affect the same codons. Hence, the protein level cost of the alignment is the sum of the protein level cost of each of the codon alignments in the decomposition. The protein level cost of a codon alignment can be determined by minimizing over the possible sequences of the up to five nucleotide events described by the codon alignment. Hein [3] describes how the decomposition into codon alignments makes it possible to compute the optimal alignment of a and b in time $\mathrm{O}(n^2m^2)$. The algorithm can be summarized as follows.

Let $D(i, j)$ denote the cost of an optimal alignment of $a_1a_2 \cdots a_{3i}$ and $b_1b_2 \cdots b_{3j}$. If $i \leq 0$ or $j \leq 0$ then we define $D(i, j)$ to be infinity. An optimal alignment of $a_1a_2 \cdots a_{3i}$ and $b_1b_2 \cdots b_{3j}$ can be decomposed into codon alignments $ca_1, ca_2, \ldots, ca_k$. If the last codon alignment ca_k is an alignment of $a_{3i'+1}a_{3i'+2} \cdots a_{3i}$ and $b_{3j'+1}b_{3j'+2} \cdots b_{3j}$ for some $(i', j') < (i, j)$[2], then $D(i, j)$ is equal to $D(i', j') + \mathit{cost}(ca_k)$. We can thus compute $D(i, j)$ by minimizing the expression $D(i', j') + \mathit{cost}(ca)$ over all $(i', j') < (i, j)$ and the up to four possible codon alignments ca of $a_{3i'+1}a_{3i'+2} \cdots a_{3i}$ and

[2]We say that $(i', j') < (i, j)$ iff $i' \leq i \wedge j' \leq j \wedge (i' \neq i \vee j' \neq j)$.

$b_{3j'+1}b_{3j'+2}\cdots b_{3j}$. If we assume that $D(i',j')$ is known for all $(i',j') < (i,j)$, then Hein [3] argues that we can compute $D(i,j)$ in time $\mathrm{O}(ij)$. By dynamic programming this implies that we can compute $D(n,m)$ in time $\mathrm{O}(n^2m^2)$.

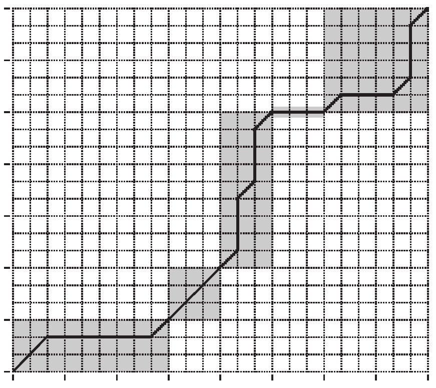


Figure 6: An alignment of two sequences decomposed into codon alignments.

4 An improved algorithm

The improved algorithm is similar to the simple algorithm in the sense that we compute the cost of an optimal alignment by minimizing over all possible last codon alignments. We define $D^t(i,j)$ to be the cost of an optimal alignment of $a_1a_2\cdots a_{3i}$ and $b_1b_2\cdots b_{3j}$ under the assumption that the last codon alignment is of type t. If $i \leq 0$ or $j \leq 0$ then we define $D^t(i,j)$ to be infinity. We compute $D(i,j)$ as

$$D(i,j) = \min_{t=1,2,\dots,11} D^t(i,j). \tag{5}$$

In the rest of this paper we assume that the amino acid substitution cost c_p and gap cost g_p fulfill lemma 1 and that the combined gap cost function $g(k) = g_d(3k) + g_p(k)$ is affine $\alpha + \beta k$ for some $\alpha, \beta \geq 0$. These assumptions make it possible to compute $D^t(i,j)$ in constant time if $D(k,l)$ has been computed (by the above expression) for all $(k,l) < (i,j)$. This implies that we can compute $D(n,m)$ in time $\mathrm{O}(nm)$.

We divide the explanation of how to compute $D^t(i,j)$ in constant time according to the number of gaps within a codon (internal gaps) described by a codon alignment of type t. Codon alignments of type 1–3 describe no internal gaps, codon alignments of type 4–7 describe one internal gap and codon alignments of type 8–11 describe two internal gaps. In each case we use the fact that c_p and g_p fulfill lemma 1 to compute $D^t(i,j)$ as the cost of the last codon alignment (of type t) plus the cost of the remaining alignment.

Codon alignments with no internal gaps

A codon alignment of type 1 describes three substitutions. We introduce $c_p^* : \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}^3 \times \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}^3 \to \mathbb{R}$ such that $c_p^*(\sigma_1\sigma_2\sigma_3, \tau_1\tau_2\tau_3)$ is the cost of a codon alignment of type 1 of codon $\sigma_1\sigma_2\sigma_3$ and codon $\tau_1\tau_2\tau_3$. The cost $c_p^*(\sigma_1\sigma_2\sigma_3, \tau_1\tau_2\tau_3)$ is the minimum cost[3] of a sequence of the three substitutions $\sigma_1 \to \tau_1$, $\sigma_2 \to \tau_2$ and $\sigma_3 \to \tau_3$. The cost $D^1(i,j)$ is the cost of the last codon alignment of type 1 plus the cost of the remaining alignment.

$$D^1(i,j) = D(i-1, j-1) + c_p^*(a_1^i a_2^i a_3^i, b_1^j b_2^j b_3^j) \tag{6}$$

A codon alignment of type 2 or type 3 describes a gap between codons. Since the combined gap cost function is affine we can use the technique introduced in [2] saying that a gap ending in (i,j) is either a continuation of an existing gap ending in $(i-1,j)$ or $(i,j-1)$, or a start of a new gap.

$$D^2(i,j) = \min\{D(i,j-1) + \alpha + \beta, D^2(i,j-1) + \beta\} \tag{7}$$
$$D^3(3i,3j) = \min\{D(i-1,j) + \alpha + \beta, D^3(i-1,j) + \beta\} \tag{8}$$

Codon alignments with one internal gap

We will describe how to compute $D^6(i,j)$. The other three cases where the last codon alignment describes one internal gap are handled similarly. The last codon alignment of type 6 describes three substitutions and one deletion. If the deletion has length k (a deletion of $3k$ nucleotides), then the last codon alignment is an alignment of $a_1^{i'} a_2^{i'} a_3^{i'} \cdots a_1^i a_2^i a_3^i$ and $b_1^j b_2^j b_3^j$ where $i' = i - k$. This is illustrated in figure 7.

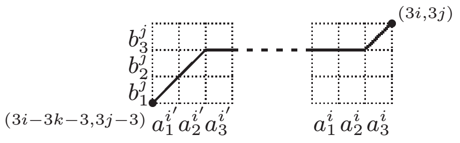
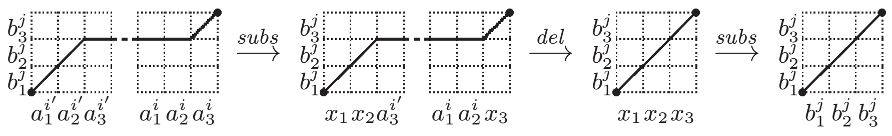


Figure 7: The last codon alignment of type 6.

The cost $D^6(i,j)$ is the cost of the last codon alignment plus the cost of the remaining alignment. The cost of the remaining alignment is $D(i-k-1, j-1)$ and the cost of the last codon alignment is the minimum cost of a sequence of the four described events. Any sequence of the four events can be divided into three steps: The substitutions occurring before the deletion, the deletion and the substitutions occurring after the deletion. Figure 8 illustrates the three steps of the evolution of $a_1^{i'} a_2^{i'} a_3^{i'} \cdots a_1^i a_2^i a_3^i$ to $b_1^j b_2^j b_3^j$. The nucleotides x_1, x_2 and x_3 are the result of the up to three substitutions

[3]We use the term *cost* to denote the DNA level cost plus the protein level cost.

before the deletion. For example, if the substitution $a_1^{i'} \to b_1^j$ occurs before the deletion, then x_1 is b_1^j, otherwise it is $a_1^{i'}$. We say that $x_1 \in \{a_1^{i'}, b_1^j\}$, $x_2 \in \{a_2^{i'}, b_2^j\}$ and $x_3 \in \{a_3^i, b_3^j\}$ are the status of the three substitutions before the deletion. The status of the substitutions before the deletion is used extensively in the computation of the cost of the last codon alignment.



b_3^j b_2^j b_1^j
$a_1^{i'} a_2^{i'} a_3^{i'}$ $a_1^i a_2^i a_3^i$
subs
$x_1 x_2 a_3^{i'}$ $a_1^i a_2^i x_3$
del
$x_1 x_2 x_3$
subs
$b_1^j b_2^j b_3^j$

Figure 8: The evolution of $a_1^{i'} a_2^{i'} a_3^{i'} \cdots a_1^i a_2^i a_3^i$ to $b_1^j b_2^j b_3^j$ described by the last codon alignment.

To compute the cost of the three substitutions in the last codon alignment we use the status of the substitutions before the deletion to split the occurrence into two steps. We say that the substitutions $a_1^{i'} \to x_1$, $a_2^{i'} \to x_2$ and $a_3^i \to x_3$ occur before the deletion and that the substitutions $x_1 \to b_1^j$, $x_2 \to b_2^j$ and $x_3 \to b_3^j$ occur after the deletion. Since an identical substitution has cost zero, then the cost of the three substitutions in the last codon alignment is equal to the cost of the six substitutions obtained by splitting the occurrence of the three substitutions into two steps. The substitutions occurring before the deletion change codon $a_1^{i'} a_2^{i'} a_3^{i'}$ to $x_1 x_2 a_3^{i'}$ and codon $a_1^i a_2^i a_3^i$ to $a_1^i a_2^i x_3$. The substitutions occurring after the deletion change codon $x_1 x_2 x_3$ to $b_1^j b_2^j b_3^j$. We recall that the cost of changing codon $\sigma_1\sigma_2\sigma_3$ to codon $\tau_1\tau_2\tau_3$ by a sequence of the substitutions $\sigma_1 \to \tau_1$, $\sigma_2 \to \tau_2$ and $\sigma_3 \to \tau_3$ is $c_p^*(\sigma_1\sigma_2\sigma_3, \tau_1\tau_2\tau_3)$, so the cost of the three substitutions is

$$\begin{aligned} cost(subs) = {} & c_p^*(a_1^{i'} a_2^{i'} a_3^{i'}, x_1 x_2 a_3^{i'}) + \\ & c_p^*(a_1^i a_2^i a_3^i, a_1^i a_2^i x_3) + c_p^*(x_1 x_2 x_3, b_1^j b_2^j b_3^j). \end{aligned} \tag{9}$$

The cost of the deletion of $3k$ nucleotides in the last codon alignment is the sum of the DNA level cost $g_d(3k)$ and the protein level cost as given by expression (4). By using the combined gap cost function $g(k) = g_d(3k) + g_p(k) = \alpha + \beta k$ and our knowledge of the status of the three substitutions before the deletion, i.e. the remaining codon of the deletion, we can formulate this sum as

$$cost(del) = \min \begin{cases} \alpha + \beta k + c_p(a_1^i a_2^i x_3, x_1 x_2 x_3)^4 \\ 2\alpha + \beta k + \min_{0<l<k} c_p(a_1^{i-l} a_2^{i-l} a_3^{i-l}, x_1 x_2 x_3) \\ \alpha + \beta k + c_p(x_1 x_2 a_3^{i'}, x_1 x_2 x_3) \end{cases} . \tag{10}$$

The cost of the deletion depends on the deletion length, the remaining codon $x_1 x_2 x_3$ and a witness. The witness encodes the amino acid aligned with the

[4]We use $c_p(\sigma_1\sigma_2\sigma_3, \tau_1\tau_2\tau_3)$ as a convenient notation for $c_p(\sigma, \tau)$ where σ and τ are the amino acids coded by the codons $\sigma_1\sigma_2\sigma_3$ and $\tau_1\tau_2\tau_3$ respectively.

remaining amino acid (see figure 3(c)). The witness can be the end-codon $a_1^i a_2^i x_3$, the start-codon $x_1 x_2 a_3^{i'}$ or one of the internal codons $a_1^{i-l} a_2^{i-l} a_3^{i-l}$ for some $0 < l < k$.

We compute $D^6(i,j)$ by minimizing the cost of the last codon alignment plus the cost of the remaining alignment over all possible last codon alignments of type 6. This is done by minimizing the sum $\mathit{cost}(\mathit{subs}) + \mathit{cost}(\mathit{del}) + D(i-k-1, j-1)$ over all possible combinations of deletion length k and remaining codon $x_1x_2x_3$. A combination of deletion length k and remaining codon $x_1x_2x_3$ is possible if $x_1 \in \{a_1^{i'}, b_1^j\}$, $x_2 \in \{a_2^{i'}, b_2^j\}$ and $x_3 \in \{a_3^i, b_3^j\}$ where $i' = i - k$. The terms $c_p^*(a_1^i a_2^i a_3^i, a_1^i a_2^i x_3)$ and $c_p^*(x_1x_2x_3, b_1^j b_2^j b_3^j)$ of $\mathit{cost}(\mathit{subs})$ do not depend on the deletion length, so we can split the minimization as

$$D^6(i,j) = \min_{x_1x_2x_3} \{ c_p^*(a_1^i a_2^i a_3^i, a_1^i a_2^i x_3) + c_p^*(x_1x_2x_3, b_1^j b_2^j b_3^j) + D^6_{x_1x_2x_3}(i,j) \} \quad (11)$$

where

$$D^6_{x_1x_2x_3}(i,j) = \min_{0<k<i} \{ D(i-k-1, j-1) + c_p^*(a_1^{i'} a_2^{i'} a_3^{i'}, x_1 x_2 a_3^{i'}) + \mathit{cost}(\mathit{del}) \} \quad (12)$$

is the minimum cost of the terms that depend on both the deletion length and the remaining codon under the assumption that the remaining codon is $x_1x_2x_3$. The cost $D^6_{x_1x_2x_3}(i,j)$ is defined if there exist a deletion length k such that k and $x_1x_2x_3$ is a possible combination of deletion length and remaining codon. If we expand the term $\mathit{cost}(\mathit{del})$ we get

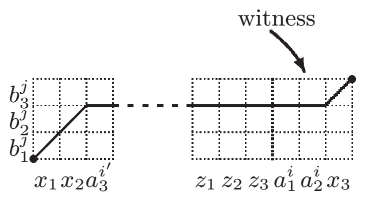
$$D^6_{x_1x_2x_3}(i,j) = \min_{0<k<i} \{ \mathrm{len}^6_{x_1x_2}(i,j,k) + \min \left\{ \begin{array}{l} c_p(a_1^i a_2^i x_3, x_1x_2x_3) \\ \alpha + \min_{0<l<k} c_p(a_1^{i-l} a_2^{i-l} a_3^{i-l}, x_1x_2x_3) \\ c_p(x_1 x_2 a_3^{i'}, x_1x_2x_3) \end{array} \right. \} \quad (13)$$

where

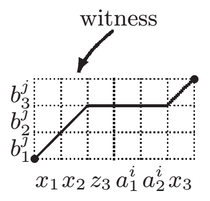
$$\mathrm{len}^6_{x_1x_2}(i,j,k) = D(i-k-1, j-1) + c_p^*(a_1^{i'} a_2^{i'} a_3^{i'}, x_1 x_2 a_3^{i'}) + \alpha + \beta k \quad (14)$$

is the cost of the remaining alignment plus the part of the cost of the last codon alignment that does not depend on the codon $a_1^i a_2^i a_3^i$ and the witness. The cost $\mathrm{len}^6_{x_1x_2}(i,j,k)$ is defined if $x_1 \in \{a_1^{i'}, b_1^j\}$ and $x_2 \in \{a_2^{i'}, b_2^j\}$ where $i' = i - k$.

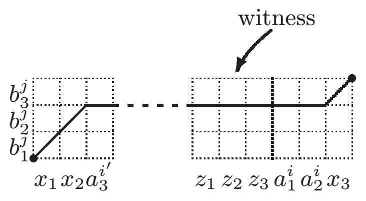
Since only x_1 and x_2 depend on the deletion length (because $a_1^{i'}$ and $a_2^{i'}$ depend on the deletion length), then there are at most 32 possible remaining codons $x_1x_2x_3$. If we can compute $D^6_{x_1x_2x_3}(i,j)$ in constant time for each of these possible remaining codons, then we can compute $D^6(i,j)$ in constant time. To compute $D^6_{x_1x_2x_3}(i,j)$ we must determine a combination of witness and deletion length that minimizes the cost. This combination must be one of the four cases illustrated in figure 9, so all we have to do is to compute the minimum cost of these four cases. The cost of case 1–3 is obtained by simplifying expression (13) for a particular witness and deletion length.



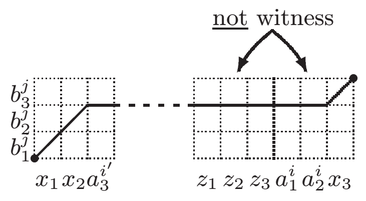
(a) Case 1



(c) Case 3



(b) Case 2



(d) Case 4

Figure 9: The four cases in the computation of $D^6_{x_1x_2x_3}(i,j)$. We use $z_1z_2z_3$ as notation for $a_1^{i-1}a_2^{i-1}a_3^{i-1}$.

Case 1: The end-codon is the witness and the deletion length is at least one. The cost is: $\min_{0<k<i} \text{len}^6_{x_1x_2}(i,j,k) + c_p(a_1^i a_2^i x_3, x_1x_2x_3)$.

Case 2: The last internal codon is the witness and the deletion length is at least two. The cost is: $\min_{1<k<i} \text{len}^6_{x_1x_2}(i,j,k) + \alpha + c_p(a_1^{i-1}a_2^{i-1}a_3^{i-1}, x_1x_2x_3)$.

Case 3: The start-codon is the witness and the deletion length is one. The cost is: $\text{len}^6_{x_1x_2}(i,j,1) + c_p(x_1x_2a_3^{i-1}, x_1x_2x_3)$.

Case 4: The witness is neither the end-codon nor the last internal codon and the deletion length is at least two. If the witness of $D^6_{x_1x_2x_3}(i-1,j)$ is not the end-codon $a_1^{i-1}a_2^{i-1}x_3$, then by optimality of $D^6_{x_1x_2x_3}(i-1,j)$ this witness must also be the witness of case 4. If this is the case then the cost of case 4 is $D^6_{x_1x_2x_3}(i-1,j) + \beta$.

Case 4 suggests that we can use dynamic programming to keep track of $D^6_{x_1x_2x_3}(i,j)$ under the assumption that the end-codon is not the witness, i.e. use dynamic programming to keep track of the minimum cost of case 2–4. We introduce tables $F^6_{x_1x_2x_3}$ corresponding to the 64 combinations of $x_1x_2x_3$. We maintain that if $x_1x_2x_3$ is a possible remaining codon and the end-codon $a^i_1a^i_2x_3$ is not the witness of $D^6_{x_1x_2x_3}(i,j)$, then $F^6_{x_1x_2x_3}(i,j)$ is equal to $D^6_{x_1x_2x_3}(i,j)$. If we define $F^6_{x_1x_2x_3}(0,j)$ to infinity, then we can compute table entry (i,j) as

$$F^6_{x_1x_2x_3}(i,j) = \min \begin{cases} \text{cost of } \textit{Case 2} \\ \text{cost of } \textit{Case 3} \\ F^6_{x_1x_2x_3}(i-1,j) + \beta \end{cases} \tag{15}$$

In order to compute the cost of case 1 and 2 in constant time we maintain the minimum of $\text{len}^6_{x_1x_2}(i,j,k)$ over k by dynamic programming. We introduce tables $L^6_{x_1x_2}$ corresponding to the 16 combinations of x_1x_2 such that $L^6_{x_1x_2}(i,j)$ is equal to $\min_{0<k<i} \text{len}^6_{x_1x_2}(i,j,k)$. If we define $L^6_{x_1x_2}(0,j)$ to infinity, then we can compute table entry (i,j) as

$$L^6_{x_1x_2}(i,j) = \min \begin{cases} \text{len}^6_{x_1x_2}(i,j,1) \\ L^6_{x_1x_2}(i-1,j) + \beta \end{cases} \tag{16}$$

We are now finally in a position where we can formulate how to compute $D^6_{x_1x_2x_3}(i,j)$ in constant time. The cost is the minimum cost of case 1–4. The cost of case 1 is $L^6_{x_1x_2}(i,j) + c_p(a^i_1a^i_2x_3, x_1x_2x_3)$ and the minimum cost of case 2–4 is $F^6_{x_1x_2x_3}(i,j)$, so

$$D^6_{x_1x_2x_3}(i,j) = \min \begin{cases} L^6_{x_1x_2}(i,j) + c_p(a^i_1a^i_2x_3, x_1x_2x_3) \\ F^6_{x_1x_2x_3}(i,j) \end{cases} \tag{17}$$

To compute $D^6(i,j)$ in constant time by expression (11) we must compute $D^6_{x_1x_2x_3}(i,j)$ for each of the 32 possible remaining codons. To do this we must compute entry (i,j) in the 16 tables $L^6_{x_1x_2}$ and entry (i,j) in the 64 tables $F^6_{x_1x_2x_3}$. The other three cases where the last codon alignment describes one internal gap (type 4, 5 and 7) are handled similarly. However, if the last codon alignment is of type 4 or 5, then only the first nucleotide x_1 in the remaining codon depends on the deletion (or insertion) length. This limits the number of possible remaining codons to 16 and implies that only four tables are needed to keep track of $\min_{0<k<i} \text{len}^t_{x_1}(i,j,k)$ for $t = 4,5$. Hence, to compute $D^t(i,j)$ for $t = 4,5,6,7$, we must compute $2\cdot4+2\cdot16+4\cdot64 = 296$ table entries in total.

Codon alignments with two internal gaps

We will describe how to compute $D^8(i,j)$. The other three cases where the last codon alignment describes two internal gaps are handled similarly. The last codon alignment of type 8 describes three substitutions and two deletions. If the first deletion has length k' and the second deletion has length k, then the last codon alignment is an alignment of $a_1^{i''}a_2^{i''}a_3^{i''} \cdots a_1^{i'}a_2^{i'}a_3^{i'} \cdots a_1^{i}a_2^{i}a_3^{i}$ and $b_1^j b_2^j b_3^j$ where $i' = i - k$ and $i'' = i' - k'$. This is illustrated in figure 10.

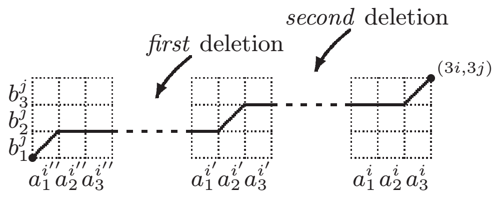
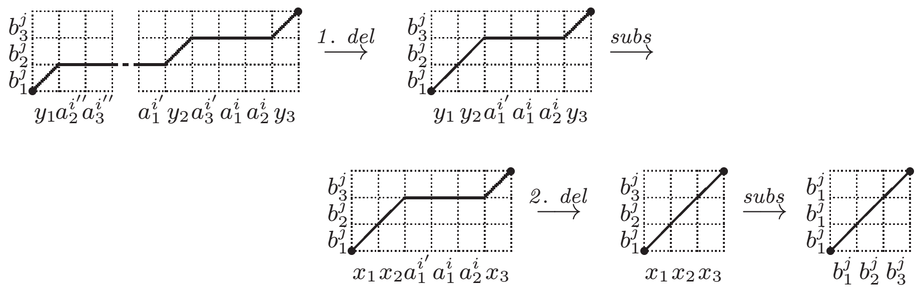


Figure 10: The last codon alignment of type 8

We will compute $D^8(i,j)$ in the same way as we computed $D^6(i,j)$. We will minimize the cost over all possible combinations of deletion length and remaining codon of the second deletion. This reduces the problem to computing $D^8_{x_1x_2x_3}(i,j)$, the cost under the assumption of a certain remaining codon of the second deletion, for each of the 32 possible remaining codons of the second deletion. We will compute $D^8_{x_1x_2x_3}(i,j)$ similar to the method described by expression (15), (16) and (17). The method can be used with almost no modifications. All we essentially have to do is to use $\text{len}^8_{x_1x_2}(i,j,1)$ instead of $\text{len}^6_{x_1x_2}(i,j,1)$ in expression (16).

The cost $\text{len}^8_{x_1x_2x_3}(i,j,k)$ is the part of the cost $D^8(i,j)$ that does not depend on the codon $a_1^i a_2^i a_3^i$ and the witness of the second deletion, under the assumption the second deletion has length k and remaining codon $x_1x_2x_3$. The cost depends on the order of the two deletions in the last codon alignment, so we introduce $\text{len}^{8'}_{x_1x_2x_3}(i,j,k)$ and $\text{len}^{8''}_{x_1x_2x_3}(i,j,k)$ to denote the cost when the first deletion occurs before the second deletion and vice versa. We define $\text{len}^8_{x_1x_2x_3}(i,j,k)$ as $\min\{\text{len}^{8'}_{x_1x_2x_3}(i,j,k), \text{len}^{8''}_{x_1x_2x_3}(i,j,k)\}$. We only have to compute $\text{len}^8_{x_1x_2x_3}(i,j,1)$. In the following we will therefore examine the cost $D^8(i,j)$ under the assumption that the second deletion has length one and remaining codon $x_1x_2x_3$. We split the examination depending on the order of the first and second deletion.

Figure 11 illustrates the evolution of the last codon alignment (of type 8) when the second deletion has length one and occurs after the first deletion. The nucleotides y_1, y_2 and y_3 are the status of the substitutions before the first deletion and the nucleotides x_1, x_2 and x_3 are the status of the substitutions before the second deletion. We can regard the substitution $a_1^{i''} \to b_1^j$ as occurring in three steps: $a^{i''} \to y_1$, $y_1 \to x_1$ and $x_1 \to b_1^j$ where $y_1 \in \{a_1^{i''}, x_1\}$ and $x_1 \in \{a_1^{i''}, b_1^j\}$. Similarly, we can regard the other two substitutions as occurring as $a_2^{i'} \to y_2 \to x_2 \to b_2^j$ and $a_3^i \to y_3 \to x_3 \to b_3^j$.



b_3^j
b_2^j
b_1^j
$y_1 a_2^{i''} a_3^{i''}$
$a_1^{i'}\, y_2 a_3^{i'}\, a_1^i\, a_2^i\, y_3$
$\xrightarrow{1.\ del}$
$y_1\, y_2 a_1^{i'}\, a_1^i\, a_2^i\, y_3$
$\xrightarrow{subs}$
$x_1 x_2 a_1^{i'}\, a_1^i\, a_2^i\, x_3$
$\xrightarrow{2.\ del}$
$x_1 x_2 x_3$
$\xrightarrow{subs}$
$b_1^j\, b_2^j\, b_3^j$

Figure 11: The first deletion occurs before the second deletion and the second deletion has length one.

The cost of the substitutions in the last codon alignment is

$$\begin{aligned} cost(subs) = c_p^*(a_1^{i''} a_2^{i''} a_3^{i''}, y_1 a_2^{i''} a_3^{i''}) + c_p^*(a_1^{i'} a_2^{i'} a_3^{i'}, a_1^{i'} y_2 a_3^{i'}) + \\ c_p^*(a_1^i a_2^i a_3^i, a_1^i a_2^i y_3) + c_p^*(y_1 y_2 a_1^{i'}, x_1 x_2 a_1^{i'}) + \\ c_p^*(a_1^i a_2^i y_3, a_1^i a_2^i x_3) + c_p^*(x_1 x_2 x_3, b_1^j b_2^j b_3^j). \quad (18) \end{aligned}$$

The deletion length of the first deletion is k' and the deletion length of the second deletion is one. Recall that $i' = i - 1$ and $i'' = i' - k'$. Similar to expression (10) we can formulate the cost of the two deletions as

$$cost(del_1) = \min \left\{ \begin{array}{l} \alpha + \beta k' + c_p(a_1^{i'} y_2 a_3^{i'}, y_1 y_2 a_3^{i'}) \\ 2\alpha + \beta k' + \min_{0<l<k'} c_p(a_1^{i'-l} a_2^{i'-l} a_3^{i'-l}, y_1 y_2 a_3^{i'}) \\ \alpha + \beta k' + c_p(y_1 a_2^{i''} a_3^{i''}, y_1 y_2 a_3^{i'}) \end{array} \right. \quad (19)$$

$$cost(del_2) = \alpha + \beta + \min \left\{ \begin{array}{l} c_p(x_1 x_2 a_3^{i'}, x_1 x_2 x_3) \\ c_p(a_1^i a_2^i x_3, x_1 x_2 x_3) \end{array} \right. \quad (20)$$

The cost of the remaining alignment is $D(i' - k' - 1, j - 1)$, so the minimum cost under the assumption of $x_1 x_2 x_3$ as the remaining codon of the second deletion is given by the sum $cost(subs) + cost(del_1) + cost(del_2) + D(i' - k' - 1, j - 1)$ minimized over all possible combinations of $y_1 y_2 y_3$ and k'. The cost $\text{len}^8_{x_1 x_2 x_3}(i, j, 1)$ is the part of this cost that does not depend on $a_1^i a_2^i a_3^i$ or the witness of the second deletion. This part includes everything except the terms $c_p^*(a_1^i a_2^i a_3^i, a_1^i a_2^i y_3) + c_p^*(a_1^i a_2^i y_3, a_1^i a_2^i x_3)$ of $cost(subs)$ and $\min\{c_p(x_1 x_2 a_3^{i'}, x_1 x_2 x_3), c_p(a_1^i a_2^i x_3, x_1 x_2 x_3)\}$ of $cost(del_2)$. It is easy to verify that the minimum of $D(i' - k' - 1, j - 1) + c_p^*(a_1^{i''} a_2^{i''} a_3^{i''}, y_1 a_2^{i''} a_3^{i''}) + cost(del_2)$ over the length k' of the first deletion is $D^4_{y_1 y_2 a_3^{i'}}(i', j)$, so

$$\begin{aligned} \text{len}^{8'}_{x_1 x_2 x_3}(i, j, 1) = \alpha + \beta + c_p^*(x_1 x_2 x_3, b_1^j b_2^j b_3^j) + \\ \min_{y_1 y_2} \{ c_p^*(a_1^{i'} a_2^{i'} a_3^{i'}, a_1^{i'} y_2 a_3^{i'}) + D^4_{y_1 y_2 a_3^{i'}}(i', j) + c_p^*(y_1 y_2 a_3^{i'}, x_1 x_2 a_3^{i'}) \} \quad (21) \end{aligned}$$

where we minimize over $y_1 \in \{a_1^{i''}, x_1\}$ and $y_2 \in \{a_2^{i'}, x_2\}$. The cost $\text{len}^{8'}_{x_1x_2x_3}(i,j,1)$ is defined if $x_1x_2x_3$ allows the second deletion to have length one, i.e. if $x_1 \in \{a_1^{i''}, b_1^j\}$, $x_2 \in \{a_2^{i'}, b_2^j\}$ and $x_3 \in \{a_3^i, b_3^j\}$. The nucleotide $a_1^{i''}$ depends on the unknown length of the first deletion, so we must assume that it can be any of the four nucleotides.

Figure 12 illustrates the evolution of the last codon alignment when the second deletion has length one and occurs before the first deletion. The nucleotides z_1, x_2 and x_3 are the status of the substitutions before the second deletion and the nucleotides y_1, y_2 and y_3 are the status of the substitutions before the first deletion. The first nucleotide x_1 in the remaining codon of the second deletion is just $a_1^{i'}$. A detailed description of the cost can be done as above. It would reveal that $\text{len}^{8''}_{x_1x_2x_3}(i,j,1)$ includes everything except the cost of the substitution $a_3^i \to x_3 \to y_3$ and $c_p(x_1x_2x_3, w_1w_2w_3)$ where $w_1w_2w_3$ is the witness of the second deletion. It would also reveal the minimum cost of the remaining alignment, the substitution $a_1^{i''} \to y_1$ and the first deletion is given by $D^4_{y_1y_2y_3}(i',j)$, so

$$\begin{aligned}\text{len}^{8''}_{x_1x_2x_3}(i,j,1) &= \alpha + \beta + c_p^*(a_1^{i'}a_2^{i'}a_3^{i'}, x_1x_2a_3^{i'}) + \\ &\min_{y_1y_2y_3}\{c_p^*(a_1^{i'}x_2x_3, a_1^{i'}y_2y_3) + D^4_{y_1y_2y_3}(i-1,j) + c_p^*(y_1y_2y_3, b_1^jb_2^jb_3^j)\} \quad (22)\end{aligned}$$

where we minimize over $y_1 \in \{z_1, b_1^j\}$, $y_2 \in \{x_2, b_2^j\}$ and $y_3 \in \{x_3, b_3^j\}$. The nucleotide z_1 depends on the unknown length of the first deletion, so we must assume that z_1 can be any of the four nucleotides. The cost $\text{len}^{8''}_{x_1x_2x_3}(i,j,1)$ is defined if $x_1x_2x_3$ allows the second deletion to have length one, i.e. if $x_1 = a_1^{i'}$, $x_2 \in \{a_2^{i'}, b_2^j\}$ and $x_3 \in \{a_3^i, b_3^j\}$.

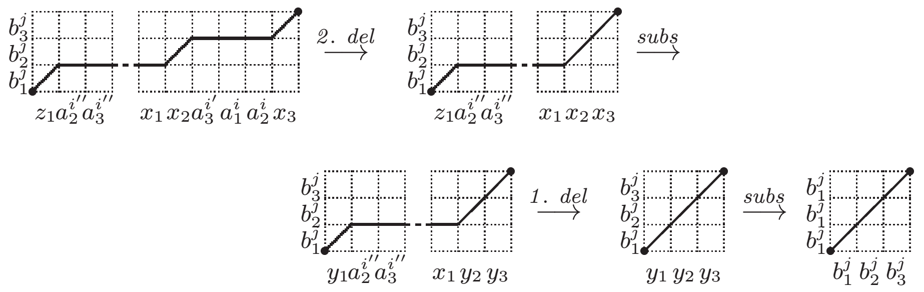


Figure 12: The second deletion occurs before the first deletion and the second deletion has length one.

We are finally in a position where we can formulate how to use the method from the previous section to compute $D^8(i,j)$. The cost $\text{len}^8_{x_1x_2x_3}(i,j,k)$ depends on x_1, x_2 and x_3, so instead of 16 tables we need 64 tables to keep track of $\min_{0<k<i} \text{len}^8_{x_1x_2x_3}(i,j,k)$. We still need 64 tables

to keep track of the cost under the assumption that the end-codon $a_1^i a_2^i x_3$ is not the witness (of the second deletion). We compute table entry (i,j) as

$$L^8_{x_1x_2x_3}(i,j) = \min \left\{ \begin{array}{l} \mathrm{len}^8_{x_1x_2x_3}(i,j,1) \\ L^8_{x_1x_2x_3}(i-1,j)+\beta \end{array} \right. \tag{23}$$

$$F^8_{x_1x_2x_3}(i,j) = \min \left\{ \begin{array}{l} L^8_{x_1x_2x_3}(i-1,j)+\beta+\alpha+c_p(a_1^{i-1}a_2^{i-1}a_3^{i-1}, x_1x_2x_3) \\ \mathrm{len}^8_{x_1x_2x_3}(i,j,1)+c_p(x_1x_2a_3^{i-1}, x_1x_2x_3) \\ F^8_{x_1x_2x_3}(i-1,j)+\beta \end{array} \right. \tag{24}$$

We compute $D^8_{x_1x_2x_3}(i,j)$ by using the above tables, and we compute $D^8(i,j)$ by minimizing over the 32 possible remaining codons of the second deletion.

$$D^8_{x_1x_2x_3}(i,j) = \min \left\{ \begin{array}{l} L^8_{x_1x_2x_3}(i,j)+c_p(a_1^i a_2^i x_3, x_1x_2x_3) \\ F^8_{x_1x_2x_3}(i,j) \end{array} \right. \tag{25}$$

$$D^8(i,j) = \min_{x_1x_2x_3} \{ c_p^*(a_1^i a_2^i a_3^i, a_1^i a_2^i x_3) + D^8_{x_1x_2x_3}(i,j) \} \tag{26}$$

This computes $D^8(i,j)$ in constant time. The computation require us to compute entry (i,j) in 128 tables. The other three cases where the last codon alignment describes two internal gaps, type 9–11, are handled similarly, so to compute $D^t(i,j)$ for $t = 8, 9, 10, 11$ we compute entry (i,j) in $4 \cdot 128 = 512$ tables. Finally, to compute $D(i,j)$ by expression (5) we compute entry (i,j) in $1 + 11 + 296 + 512 = 820$ tables. The 512 entries are explained in this section, the 296 entries was explained in the previous section, the 11 entries correspond to $D^t(i,j)$ for $t = 1, 2, \ldots, 11$ and the last entry corresponds to $D(i,j)$.

5 Improvements and future work

The only real difference between the computation of $D^6(i,j)$ and $D^8(i,j)$ is between $\mathrm{len}^6_{x_1x_2}(i,j,1)$ and $\mathrm{len}^8_{x_1x_2x_3}(i,j,1)$. The similarity stems from the fact that a codon alignment of type 6 and type 8 ends in the same way. By "end in the same way" we mean that the events described on the codon $a_1^i a_2^i a_3^i$ are the same (see figure 5). A codon alignment of type 11 also ends in the same way as a codon alignment of type 6 or 8. The similarity between the computation of $D^t(i,j)$ for $t = 6, 8, 11$ makes it possible to replace the six tables $L^t_{x_1x_2x_3}$ and $F^t_{x_1x_2x_3}$ for $t = 6, 8, 11$ by two tables $L^{6,8,11}_{x_1x_2x_3}$ and $F^{6,8,11}_{x_1x_2x_3}$ in which entry (i,j) equals the minimum of entry (i,j) in the tables being replaced. We compute $L^{6,8,11}_{x_1x_2x_3}(i,j)$ and $F^{6,8,11}_{x_1x_2x_3}(i,j)$ similar to expression (23) and (24), except that we use

$$\mathrm{len}^{6,8,11}_{x_1x_2x_3}(i,j,1) = \min_{t=6,8,11} \mathrm{len}^t_{x_1x_2x_3}(i,j,1) \tag{27}$$

instead of $\text{len}^8_{x_1x_2x_3}(i,j,1)$. We use $\text{len}^6_{x_1x_2x_3}(i,j,1)$ defined as $\text{len}^6_{x_1x_2}(i,j,1)+c^*_p(x_1x_2x_3,b^j_1b^j_2b^j_3)$ to ensure that $\text{len}^t_{x_1x_2x_3}(i,j,1)$ for $t=6,8,11$ describes the same part of the total cost. The cost $D^{6,8,10}(i,j)$ is the minimum of $D^t(i,j)$ over $t=6,8,10$. To compute $D^{6,8,11}(i,j)$ by expressions similar to expression (25) and (26) we compute entry (i,j) in $1+64+64=129$ tables. To compute $D^t(i,j)$ for $t=6,8,10$ we compute entry (i,j) in $3+80+128+128=339$ tables. The combined computations thus saves the computation of 210 table entries. Codon alignments of type 7, 9 and 10 also end in the same way, so we can also combine the computation of $D^t(i,j)$ for $t=7,9,10$. We can thus reduce the number of table entry updates in the computation of $D(i,j)$ to $820-2\cdot(339-129)=400$.

We are working on implementing the alignment algorithm in order to compare it to the heuristic alignment algorithm described in [4]. The heuristic algorithm allows frame shifts, so an obvious extension of our exact algorithm would be to allow frame shifts, e.g. to allow insertion-deletions of arbitrary length. This however makes it difficult to split the evaluation of the alignment cost into small independent subproblems (codon alignments) of known size. Another interesting extension would be to annotate the DNA sequence with more information. For example, if the DNA sequence codes in more than one reading frame (overlapping reading frames) then the DNA sequence should be annotated with all the amino acid sequences encoded and the combined cost of a nucleotide event should summarize the cost of changes induced on all the amino acid sequences encoded by the DNA sequence. This extension also makes it difficult to split the evaluation of the alignment cost into small independent subproblems. To implement these extensions efficiently it might be fruitful to investigate reasonable restrictions of the cost functions.

References

[1] Arvestad, L. Aligning coding DNA in the presence of frame-shift errors. In *Combinatorial Pattern Matching* (1997), vol. 1264 of *LNCS*, pp. 180–190.

[2] Gotoh, O. An improved algorithm for matching biological sequences. *Journal of Molecular Biology 162* (1981), 705–708.

[3] Hein, J. An algorithm combining DNA and protein alignment. *Journal of Theoretical Biology 167* (1994), 169–174.

[4] Hein, J., and Støvlbæk, J. Genomic alignment. *Journal of Molecular Evolution 38* (1994), 310–316.

[5] Hein, J., and Støvlbæk, J. An algorithm combining DNA and protein alignment. *Methods in Enzymology 266* (1996), 402–418.

[6] Hua, Y., Jiang, T., and Wu, B. Aligning DNA sequences to minimize the change in protein. Manuscript, October 1997.

[7] Needleman, S. B., and Wunsch, C. D. A general method applicable to the search for similarities in the amino acid seqeunce of two proteins. *Journal of Molecular Biology 48* (1970), 433–443.

[8] Peltola, H., Söderlund, H., and Ukkonen, E. Algorithms for the search of amino acid patterns in nucleic acid sequences. *Nuclear Acids Research 14*, 1 (1986), 99–107.

[9] Sankoff, D. Matching sequences under deletion/insertion constraints. In *Proc. Nat. Acad. of Sci. U.S.A.* (1972), vol. 69, pp. 4–6.

[10] Sellers, P. H. On the theory and computation of evolutionary distance. *SIAM Journal of Applied Mathematics 26* (1974), 787–793.

[11] Wagner, R. A., and Fisher, M. J. The string to string correction problem. *Journal of the ACM 21* (1974), 168–173.

[12] Zhang, Z., Pearson, W. R., and Miller, W. Aligning a DNA sequence with a protein sequence. In *Proceedings of the First Annual International Conference on Computational Molecular Biology* (1997), ACM, pp. 337–343.