# Aligning a DNA Sequence with a Protein Sequence

Zheng Zhang* William R. Pearson † Webb Miller*

**Abstract**

We develop several algorithms for the problem of aligning a DNA sequence with a protein sequence. Our methods account for frameshift errors, but not for introns in the DNA sequence. Thus, they are particularly appropriate for comparing a cDNA sequence that contains sequencing errors with an amino acid sequence or a protein sequence database. We describe techniques for efficient implementation, verify sufficient conditions for equivalence of several definitions of alignment, and discuss experience with these ideas in a new release of the `fasta` suite of database-searching programs.

## 1 Basic Definition and Algorithm

We begin with a plausible definition of an alignment of a DNA sequence and an amino acid sequence, then give an algorithm for computing an optimal alignment. This section restates ideas originally developed ten years ago by Peltola *et al.* [9].

Alignment scores are based on (1) a penalty $\pi > 0$ for inserting or deleting a nucleotide (i.e., a frameshift error), (2) an amino-acid substitution-score matrix, $S$, (such as a PAM or Blosum matrix) and (3) gap-open and gap-extension penalties, $gop > 0$ and $gep > 0$ (i.e., a gap of $g$ amino acids is penalized $gop + gep \cdot g$). Let $S[x, y]$ denote the substitution score for amino acids $x$ and $y$. To score the replacement of a short string of nucleotides by an amino acid, we proceed as follows. Let $w$ be a nucleotide string of length 2, 3 or 4, and let $y$ be an amino acid. Define an amino acid, denoted $x$, as follows. If $|w| = 3$ ($|w|$ denotes the length of $w$), then $x$ is the translation of $w$ according to the genetic code. If $|w| = 2$, pick $x$ such that $w$ is a subsequence of some codon for $x$ and $S[x, y]$ is maximal over all such $x$. If $|w| = 4$, pick $x$ such that some codon for $x$ is a subsequence of $w$ and $S[x, y]$ is maximal over all such $x$. Then, we define $R[w, y] = S[x, y]$ if $|w| = 3$, and $R[w, y] = S[x, y] - \pi$ otherwise.

A *codonification* of a nucleotide sequence $B = b_1 b_2 \ldots b_N$, with $N \neq 1$, is a sequence $c_1 c_2 \ldots c_L$ where ($i$) each $c_k$ is either a nucleotide sequence with $2 \leq |c_k| \leq 4$ or a single dash character, and ($ii$) concatenating all the non-dash $c_k$ sequences, in that order, gives $B$. An *alignment* between a nucleotide sequence $B = b_1 b_2 \ldots b_N$ and an amino acid sequence $A = a_1 a_2 \ldots a_M$ consists of a codonification, $c_1 c_2 \ldots c_L$, of $B$ with each $c_k$ placed above an amino acid or a dash character, and where (1) removing all dashes from the second row gives $A$ and (2) if $c_k$ appears over a dash, then $|c_k| = 3$. The score of such an alignment is the sum of the $R[w, y]$, where nucleotide string $w = c_k$ is placed above amino acid $y$, minus the penalties for gaps.

An alignment of the DNA sequence **AGTGTAGTGCCCTCTAGTTCA** and the amino acid sequence **TVSPGS** follows. The alignment's score is $R[AGT, T] + R[GTA, V] + R[GT, S] + R[CT, P] + R[CTAG, G] + R[TTCA, S] - gop - gep$.

```
codonification    AGT GTA GT GCC CT CTAG TTCA
Protein Sequence   T   V  S   -  P   G    S
```

Notice that this definition of an alignment does not directly model sequencing errors that cause nucleotide substitutions or that occur more than once within a codon. In Section 3 we show that nothing is sacrificed by those omissions, at least under the plausible assumption that sequencing errors are relatively infrequent.

Our definition leads directly to an algorithm for computing optimal alignments. Let $B_i$ denote the prefix $b_1 b_2 \ldots b_i$, and let $A_j$ denote $a_1 a_2 \ldots a_j$. Let $C(i, j)$ be the highest score over all (global) alignments of $B_i$ and $A_j$, let $D(i, j)$ be the highest score over all alignments of $B_i$ and $A_j$ that end with $c_k = b_{i-2} b_{i-1} b_i$ positioned above a dash, and let $I(i, j)$ be the highest score over all alignments of $B_i$ and $A_j$ that end with $c_k =$ "$-$". $D(i, j)$, $I(i, j)$ and $C(i, j)$ satisfy the recurrences of Figure 1, which are similar to those developed by Gotoh [2]. In it, $progap(j)$ and $dnagap(j)$ are the penalties for gaps of $j$ amino acids or nucleotides, respectively. Namely, $progap(j) = gop + gep \cdot j$, $dnagap(i) = gop + gep \cdot \frac{i}{3}$ when $i$ is divisible by 3, and $dnagap(i) = \infty$ otherwise.

Such dynamic-programming recurrences are traditionally pictured as solving an optimal path problem in a grid graph. Figure 2 shows the edges entering the three nodes at grid point $(i, j)$ of our alignment graph when $i \geq 4$ and $j > 0$.

*Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802. Supported by grant LM05110 from the National Library of Medicine.

†Department of Biochemistry, University of Virginia, Charlottesville, VA 22908. Supported by grant LM04969 from the National Library of Medicine.

$$D(i,j) = \begin{cases} \max(D(i-3,j), C(i-3,j) - gop) - gep & \text{if } i \geq 3 \\ -\infty & \text{if } i < 3 \end{cases}$$

$$I(i,j) = \begin{cases} \max(I(i,j-1), C(i,j-1) - gop) - gep & \text{if } j > 0 \\ -\infty & \text{if } j = 0 \end{cases}$$

for $i \geq 4$ and $j > 0$

$$C(i,j) = \max \begin{cases} D(i,j) \\ I(i,j) \\ C(i-2,j-1) + R[b_{i-1}b_i, a_j] \\ C(i-3,j-1) + R[b_{i-2}b_{i-1}b_i, a_j] \\ C(i-4,j-1) + R[b_{i-3}b_{i-2}b_{i-1}b_i, a_j] \end{cases}$$

and when $j > 0$, the boundary conditions are:

$$\begin{aligned} C(0,0) &= 0 \\ C(j,0) &= -dnagap(j) \\ C(0,j) &= -progap(j) \\ C(1,j) &= -\infty \\ C(2,j) &= \max(I(2,j), C(0,j-1) + R[b_1b_2, a_j]) \\ C(3,j) &= \max(D(3,j), I(3,j), C(0,j-1) + R[b_1b_2b_3, a_j]) \end{aligned}$$

Figure 1: Recurrence relations for $D(i,j)$, $I(i,j)$ and $C(i,j)$.

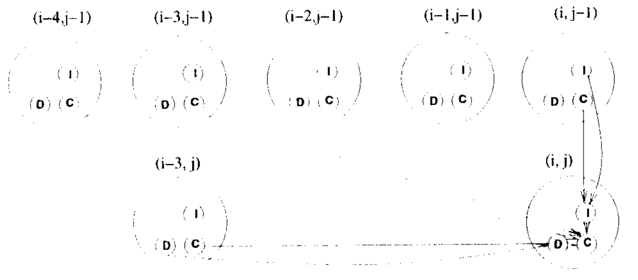

Figure 2: Graph model of the recurrences.

These recurrences can be adapted to deliver an alignment using only linear space by applying appropriate variants of the method of Hirschberg [6] (for the rectangular case) and of Chao *et al.* [1] (for the banded case). Indeed, the **fastx** package (described below) includes a linear-space variant.

## 2 An efficient implementation

Several techniques for increasing the efficiency of an implementation of the basic algorithm are considered in this section. The first type of improvement, though it performs all the computations corresponding to edges in the graph of Figure 2, reduces the number of address computations and array accesses. Notice that in general a given $C$-value, say $C(i,j)$, is used to compute five subsequent values, *viz.* $D(i+3,j)$, $I(i,j+1)$, $C(i+2,j+1)$, $C(i+3,j+1)$ and $C(i+4,j+1)$. The goal is to reduce the number of times that arithmetic is done to locate $C(i,j)$ and that $C(i,j)$ is fetched from the computer's memory.

As pictured in Figure 2, the grid point $(i,j)$ contains $C(i,j)$, $I(i,j)$ and $D(i,j)$. Computing those values involves fetching seven values from other grid points (corresponding to incoming edges). Alternatively, we can store $C(i,j)$, $I(i,j+1)$ and $D(i+3,j)$ at grid point $(i,j)$, and computing those values involves fetching only five values from other grid points. Namely, one value is fetched from each of $(i-3,j)$ and $(i,j-1)$.

Further efficiency can be gained by thinking of the nodes

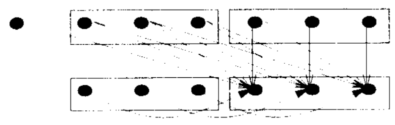

Figure 3: Grouping grid points by threes.

as grouped by threes. To compute the nine values in the lower right block in Figure 3, we need the $C$ values from the upper left block, the $D$ values from the lower left block, and the $I$ values from the upper right block. (The value on the extreme left of the top row is carried over from an earlier step.) Actually, those $D$ values don't need to be stored because the computation is done row-by-row. Thus six array fetches are used, instead of the 21 suggested by the graph edges, at the cost of increasing the number of statements in the inner loop by a factor of around three. The approach, including the ideas of the previous paragraph, speeds up the computation by a factor of perhaps three over a completely simple-minded implementation.

The second source of execution-time efficiency is, in intuitive terms, to reduce the number of graph edges that need to be considered. Following the lead of Phil Green's SWAT program (an implementation of the Smith-Waterman algorithm for database searching), one can exploit the fact that some edges can be eliminated from consideration in those regions of the dynamic-programming grid where scores are small enough. This techique is applicable to the *local* variant of the *global* algorithm presented above. The local alignment problem is to determine a highest-scoring alignment between arbitrary substrings of the given sequences. In terms of the graph model, one seeks highest-scoring paths to $(i,j)$ *that start at any grid point.* The recurrence equations for $C(i,j)$ in Figure 1 need only be modified to add "0" to the terms over which the maximum is taken. Thus, (edges corresponding to) terms that are known *a priori* to contribute a non-positive term to the maximization can be ignored.

The following result is applicable when the frameshift penalty $\pi$ is sufficiently large. It gives a testable condition under which the only edges entering a block that need to be considered are the horizontal edges (which do not require an array reference) and the "exact codon" edges of the form $(i-3,j-1) \rightarrow (i,j)$.

**Theorem 1** *Assume that* $\pi \geq 2gop + 2gep + M_S$, *where* $M_S$ *is the maximum value of the amino acid substitution-score matrix. If all three I-values stored in the block labeled* B *in Figure 4 (i.e., the I-values defined for block* A*) are non-positive, then the computation of C-values in block* A *can ignore the solid edges. If, furthermore,* $\pi \geq 2gop + 3gep + M_S$, *then the dashed edge can be ignored.*

*Proof.* The vertical edges can be ignored since they contribute non-positive $I(i,j)$ in the recurrence relations for $C(i,j)$ with $(i,j)$ in block A. For $(i,j)$ in B, $0 \geq I(i,j+1) \geq C(i,j) - gop - gep$, so $C(i,j) \leq gop + gep$.

$C$-values in block E satisfy $C \leq 2gop + 2gep$. To see this, let $(i,j)$ be in E. The recurrence for $(i+3,j)$ in B shows that $gop + gep \geq C(i+3,j) \geq D(i+3,j) \geq C(i,j) - gop - gep$, which can be rearranged to verify the claim.

The non-vertical edges in Figure 4 have the form $(i-2,j-1) \rightarrow (i,j)$ or $(i-4,j-1) \rightarrow (i,j)$. Each non-vertical dark edge begins at a node with $C$-value at most $2gop+2gep$, and for some amino-acid pair, $a, b$, it contributes a term $C+$

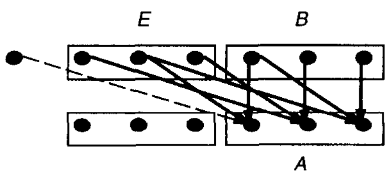

Figure 4: Edges that can sometimes be eliminated.

$S[a,b] - \pi \le 2gop + 2gep + M_S - \pi \le 0$ to the maximization defining $C(i,j)$. Thus those edges can be ignored.

The $C$-value at the node immediately to the left of block E satisfies $C \le 2gop + 3gep$, since $gop + gep \ge C(i+6,j) \ge D(i+6,j) \ge D(i+3,j) - gep \ge C(i,j) - gop - 2gep$. The claim about the dashed edge follows as in the previous paragraph. □

In an implementation, a bit can be associated with each block telling if all $I$ values stored in the block are non-positive. The bit is tested when values in the block beneath it are computed. For computations where almost all $I$ values are non-positive, this strategy achieves an additional 35% improvement in efficiency.

## 3 A more general model

The above definition of an alignment between a DNA sequence and an amino acid sequence is just one from a range of possibilities. For other approaches see papers [3, 4, 5, 7, 8, 9]. There is a tradeoff between the completeness of transformations that are directly modeled by a definition of a DNA-protein alignment and the execution time for computing an optimal alignment. In this section we describe a more general notion of aligning a DNA sequence to a protein sequence. The recurrence relations of Figure 6 give an $O(MN)$-time algorithm for computing an optimal alignment, though in practice that method requires perhaps 10 times more time and space than the method described above. We then give a sufficient condition to guarantee that the two definitions yield the same optimal alignments.

As before, we model sequencing errors at the DNA level and sequence evolution at the amino acid level, though now the sequencing errors are not limited to a single insertion or deletion per codon. Define a *generalized alignment* of a DNA sequence $B$ and a protein sequence $A$ to consist of a pair $(\beta, \alpha)$, where $\beta$ is an alignment between $B$ and some other DNA sequence (called the *intermediate*) of length a multiple of 3, and $\alpha$ is an alignment between the translation of the intermediate sequence and $A$. An example appears in Figure 5. $\beta$ is scored by 0 for a match and $-\pi$ for a mismatch, an insertion, or a deletion, $\alpha$ is scored using $S[.,.]$, $gop$ and $gep$, as before, and the score of the generalized alignment is the sum of the scores of $\beta$ and $\alpha$.

For the computation of optimal alignments, we assume existence of a pre-computed table $T_x[b,a]$, defined whenever $b$ is a nucleotide, $a$ is an amino acid, and $x$ is a nucleotide sequence with $|x| \le 2$. $T_x[b,a]$ gives the maximum, taken over all nucleotide sequences $y$ with $|y| = 3 - |x|$, of $s[b,y] + S[z,a]$, where $s[b,y]$ is the optimal score of an alignment of $b$ and $y$ (using the above scores for DNA alignments) and $z$ is the translation of the codon $xy$ ($x$ followed by $y$). Informally, $T_x[b,a]$ is the score for aligning $b$ and $a$, given that nucleotides $x$ will be attached at the front of the intermediate to produce exactly one codon.

```
DNA sequence       AGT GTA G-T GCC -CT CTAG TTCA
Intermediate       AGT GTA GAT GCC CCT -TAG -TCA
Translation         T   V   G   A   P   X    S
Protein Sequence    T   V   S   -   P   G    S
```

Figure 5: A more general kind of DNA-protein alignment.

$$D(i,j) = \max \begin{cases} C(i-2,j) - gop - gep - \pi \\ C(i-3,j) - gop - gep \\ D(i-3,j) - gep \end{cases}$$

$$I(i,j) = \max(I(i,j-1), C(i,j-1) - gop) - gep$$

$$C(i,j) = \max \begin{cases} I(i,j) & \text{(a)} \\ D(i,j) & \text{(b)} \\ C(i-1,j) - \pi & \text{(c)} \\ C(i,j-1) - 3\pi + \max_c S[c,a_j] & \text{(d)} \\ \max_{|x|\le 2}(C_x(i-1,j-1) + T_x[b_i,a_j]) & \text{(e)} \end{cases}$$

$$\text{if } |x| = 1 \text{ then } C_x(i,j) = \max \begin{cases} C(i-1,j) + s[b_i,x] \\ C(i,j) - \pi \\ C_x(i-1,j) - \pi \end{cases}$$

and

$$\text{if } |x| = |y| = 1 \text{ then } C_{xy}(i,j) = \max \begin{cases} C_x(i-1,j) + s[b_i,y] \\ C_x(i,j) - \pi \\ C_{xy}(i-1,j) - \pi \end{cases}$$

Figure 6: Recurrence relations for a more general model of alignment.

We now develop recurrence relations analogous to Figure 1. Sequences $A$, $A_j$, $B$ and $B_i$ are as before. Let $\beta$ and $\alpha$ denote the constituent DNA and protein alignment, respectively, of a generalized alignment. Let $C(i,j)$ be the maximum score over all generalized alignments of $B_i$ and $A_j$, and let $I(i,j)$ be the maximum of all such scores where $\alpha$ ends with a dash in row 1.

Deletions at the end of $\alpha$ (i.e., with a dash in the lower row) require special handling. Suppose $\alpha$ ends with a run of deletions, and let $\beta_\delta$ denote a terminal portion of $\beta$ that produces the deleted codons. We say that $(\beta,\alpha)$ *ends with a proper amino acid deletion* if (1) no mismatch or deletion occurs in $\beta_\delta$ and (2) no nucleotide insertion occurs in $\beta_\delta$ except perhaps for its first column. Define $D(i,j)$ to be the maximum score over all generalized alignments of $B_i$ and $A_j$ that end with a proper amino acid deletion.

Our recurrences also rely on defining a generalization of $C$. For any nucleotide sequence $x$ with $|x| \le 2$, define an $x$-alignment of $B_i$ and $A_j$ to consist of a DNA alignment of $B_i$ to an intermediate $B'x$ (i.e., $B'$ followed by $x$) and a protein alignment of the translation of $B'$ and $A_j$. Informally, such an alignment leaves $B_i$ with the untranslated tail $x$ that will contribute to the next codon. Define $C_x(i,j)$ to be the maximum score of all $x$-alignments of $B_i$ and $A_j$. Note that if $x$ is the empty sequence, then $C_x(i,j) = C(i,j)$. Recurrence relations for $D$, $I$ and the $C_x$ are given in Figure 6.

**Theorem 2** *Assuming correct definition of the initialization cases, the recurrences of Figure 6 correctly determine $D(i,j)$, $I(i,j)$ and the $C_x(i,j)$.*

*Proof.* Suppose that the equations of Figure 6 and the initializations correctly determine $C(p,q)$ whenever $p \le i$, $q \le j$, and $(p,q) \ne (i,j)$. Strictly speaking, we need to show that ($i$) for any term of a maximization in Figure 6, there is an appropriate alignment with at least that score and ($ii$) any optimal alignment's score is given by at least one of the terms. Here we treat only the most difficult part of this. Namely, assuming that $(\beta, \alpha)$ is an optimal generalized alignment of $B_i$ and $A_j$, we locate a corresponding term in the definition of $C(i,j)$. The remainder of the proof is left to the reader.

Terms (a) and (b) handle cases where $\alpha$ ends with an insertion or a proper deletion. If $b_i$ is deleted in $\beta$ (i.e., appears above a dash), then (c) applies. Suppose that $\alpha$ ends with arbitrary amino acid deletions; we will show that case (b) or (c) applies. Optimality of $(\beta, \alpha)$ implies that $\beta_\delta$ (the suffix of $\beta$ producing the deleted codons) contains no mismatches. Suppose that it contains a nucleotide deletion. The score of $\beta_\delta$ is unchanged by the transformation that replaces a deletion-match column pair by a match-deletion pair. Repeated application of this transformation moves the nucleotide deletions to the right end of $\beta_\delta$, where they are handled by case (c). Thus, suppose that $\beta_\delta$ contains a insertion columns. Those can all be moved to the left end of $\beta_\delta$ without decreasing the score, whence it is straightforward to show that optimality of $(\beta, \alpha)$ guarantees that there cannot be more than one nucleotide insertion.

Suppose that none of these previous cases holds. Let $c$ be aligned to $a_j$ by $\alpha$, i.e., $c$ appears at the top of $\alpha$'s last column, and let $pqr$ be the last codon of the intermediate. If $p$ appears in $\beta$ after the column containing $b_i$ (i.e., if all of $p$, $q$ and $r$ are inserted), then case (d) applies (and $S[c,a] = max_z S[z,a]$). Otherwise, let $d$ denote the nucleotide immediately before $p$ in the intermediate, and consider the prefix $\beta'$ of $\beta$ up to and including the column containing $d$ and the column containing $b_{i-1}$, whichever comes later. Let $x$ denote the string of nucleotides that follows $d$ in the second row of $\beta'$. If $\alpha'$ is just $\alpha$ with its last column removed, then $(\beta', \alpha')$ is an optimal $x$-alignment of $B_{i-1}$ and $A_{j-1}$, and the score of $(\beta, \alpha)$ is $C_x(i-1, j-1) + T_x[b_i, a_j]$. □

Evaluation of the recurrences in Figure 6 requires considerably more time and space than needed for the recurrences of Figure 1, primarily because there are values $C_x$ for 21 different strings $x$. Theorem 3 gives conditions on the alignment-scoring parameters under which the two notions of alignment are equivalent. The following notation will be useful. A *codon block* is a run of columns of $\beta$ whose second row (with dashes removed) constitutes a codon of the intermediate; the first row is a *precodon* of $B$. A codon block is a *perfect codon* if it consists of three exact-match columns. Let $M_S$ denote the largest amino-acid substitution score, define

$$M_1 = \max_a(\max_z S[z,a] - \min_z S[z,a])$$

and define

$$M_2 = \max_{a,xy}(\max_z S[z,a] - Q[xy,a])$$

where $Q[xy,a]$ is the maximum score from matching amino acid $a$ with a codon obtained by inserting one nucleotide into the dinucleotide $xy$. $M_1$ bounds the effect of replacing the first entry of a substitution column ($z$ over $a$) in $\alpha$, while $M_2$ bounds the effect of replacing $z$ by the translation of a codon where two of the nucleotides ($x$ and $y$) are fixed.

**Theorem 3** *Assume that*

$$\pi \geq \max(gep + M_1, gop + gep + \max(M_S, M_2))$$

*where $M_S$, $M_1$ and $M_2$ are defined above. Then any DNA sequence $B$ and amino acid sequence $A$ have an optimal generalized alignment $(\beta, \alpha)$, where*

1. *$\beta$ contains no mismatches (i.e., columns aligning differing nucleotides) and*
2. *$\beta$ can be partioned into codon blocks such that*
   - *(a) the precodons cover $B$ (i.e., nucleotides are deleted only within a codon block),*
   - *(b) each codon block that matches an amino acid involves at most one nucleotide insertion or deletion, and*
   - *(c) each amino acid deletion involves a perfect codon.*

*Proof.* We show how to transform an arbitrary generalized alignment of $B$ and $A$ into the desired form without decreasing its score. First, replace each mismatch column of $\beta$ with an exact match. Each such transformation raises $\beta$'s score by $\pi$, while the resulting change in the codon can lower $\alpha$'s score by at most $M_1$, for a net gain. This proves statement 1.

Next, consider codons that are aligned to an amino acid. Codons of the intermediate with more than one inserted nucleotide are transformed by (1) deleting the 0 or 1 nucleotides from $B$ to remove that codon and (2) adding an amino-acid insertion to $\alpha$, for a net gain of $\pi - gop - gep - M_S \geq 0$. For any codon of the intermediate containing an inserted nucleotide, we can assume that the other two nucleotides are consecutive in $B$, since otherwise some $b_k$ between them is deleted in $\beta$, and the deletion and the insertion can be replaced by an exact match of $b_k$, for a net change of $2\pi - M_1 > 0$.

Codons of the intermediate can be transformed so that the leftmost and rightmost nucleotides of the codon are at most three positions apart in $B$, i.e., at most one nucleotide is deleted in this precodon of $B$. To see this, suppose that amino acid $a$ matches a codon $pqr$ with positions $i_p$, $i_q$ and $i_r$ in $B$. If $i_r - i_p > 4$ (so three or more intervening nucleotides are deleted), we can delete $b_{i_p}$ to $b_{i_r-3}$ by amino-acid deletions and at most two nucleotide deletions, then match $b_{i_r-2}b_{i_r-1}b_{i_r}$ and $a$. This cannot decrease the score, since $3\pi \geq gop + gep + M_1$.

It remains to describe the case where $i_r - i_p = 4$. First suppose that either $i_q = i_r - 1$ or $i_q = i_p + 1$, i.e., two of the nucleotides are adjacent in $B$. The two adjacent nucleotides can be matched with $a$ by inserting an advantageous nucleotide, so the score of the match is at least $R[pqr, a]$. The additional gap penalties total $\pi + gop + gep < 2\pi$, verifying an improvement. The other possibility is that $i_q = i_p + 2 = i_r - 2$. Then $B$ has the substring is $pxqyr$. We match $a$ with $px$ or $yr$ and delete the other three nucleotides at the amino acid level. Since $\pi \geq gop + gep + M_2$, this cannot decrease the score, proving 2(b).

After the above transformations are applied, each amino acid matches a codon of the intermediate arising from a precodon of $B$ of length 2, 3 or 4. Between those precodons are groups of nucleotides that are deleted, either in $\beta$ or by an amino acid deletion. To prove 2(a) and 2(c) it is sufficient to show that the length of each of these deleted strings can be assumed to be a multiple of 3, so the deletion can be performed more economically in $\alpha$, which will complete the construction. Suppose that there is a block of deleted nucleotides whose length is $3k+1$. (Blocks of length $3k+2$ are handled similarly.) The only case needing some attention is when a gap of length 1 follows a precodon of length 4. We can delete two nucleotides from the precodon to form a precodon of two, and since $\pi \geq gop + gep + M_2$, this cannot decrease the score. □

```
DNA sequence       AGTGTAGTGCCCTCTAGTTCA...
Frame 1             T  V  V  P  S  S  S
Frame 2              V  X  C  P  L  V
Frame 3               C  S  A  L  X  F
Protein Sequence    T  V S  - P   G   S
                         *    *   *   *
```

Figure 7: A three frame DNA-protein alignment. An asterisk indicates a frameshift.

$$I(i,j) = \max(I(i,j-1) - gep, C(i,j-1) - gop - gep)$$

$$D(i,j) = \max \begin{cases} D(i-2,j) - \pi - gep \\ C(i-2,j) - \pi - gop - gep \\ D(i-3,j) - gep \\ C(i-3,j) - gop - gep \\ D(i-4,j) - \pi - gep \\ C(i-4,j) - \pi - gop - gep \end{cases}$$

$$C(i,j) = \max \begin{cases} I(i,j) \\ D(i,j) \\ C(i-2,j-1) + S[a_i,b_j] - \pi \\ C(i-3,j-1) + S[a_i,b_j] \\ C(i-4,j-1) + S[a_i,b_j] - \pi \end{cases}$$

Figure 8: Recurrence relations for three-frame alignment.

## 4 Three-frame alignment

There is also a formulation that is *less* general than the one described in Section 1 and which has some practical advantages. Indeed, we used this approach in the original release of **fastx**. Fix a DNA sequence and an amino acid sequence. Generate the translation of the DNA sequence in each of the three reading frames, and place each amino acid below the central nucleotide of its codon, as pictured in Figure 7. An alignment consists, essentially, of positioning the entries of the original amino acid sequence underneath these translations and separating each entry by between 1 and 3 blanks.

In the abstract, our problem is as follows. We are given amino acid sequences $B$ and $A$, where $B$ can be thought of as three interleaved sequences. The positions of $B$ are interpreted as the nodes of a graph where, for a general position $x$ in $B$, there is a 0-weighted edge to $x+3$ and $\pi$−weighted edges to $x+2$ and $x+4$. In addition, there is a 0-weighted edge from position 0 to 1 and a $\pi$-weighted from 0 to 2. Finally, add a node at the end of $B$, and call this position $N$. No letter is associated with node 0 or node $N$. Node $N$ is entered by a 0-weighted edge from $N-1$ and a $\pi$-weighted edge from $N-2$. A *three-frame alignment* is a pair $(E,\alpha)$, where $E$ is the sequence of letters along some path from 0 to $N$ in $B$ and $\alpha$ is an amino acid alignment of $E$ and $A$; its score is the sum of the score of the path and the score of $\alpha$.

Let $C(i,j)$ be the highest alignment score between letters along a path ending at node $i$ and $A_j$, and let $D(i,j)$ and $I(i,j)$ be the best such scores where $b_i$ or $a_j$ (resp.) is aligned with dash. It is straightforward to verify the recurrence relations of Figure 8, which omit boundary cases. Efficiency can be gained as follows.

**Theorem 4** *Suppose $\pi \geq gep$. There is an optimal alignment in which no deletion gap has a frameshift, with the possible exception of the edge from node $N-3$ to $N-1$.*

*Proof.* For alignment $(E,\alpha)$, fix a subpath $x_0, x_1, \ldots, x_p, x_{p+1}$ such that the labels from $x_1, \ldots, x_p$ are deleted, but each of $x_0$ and $x_{p+1}$ is either not deleted or a terminal node (0 or $N$). Suppose that $(x_{i-1}, x_i)$ is a frameshift edge (i.e., its weight is $\pi$) but $(x_i, x_{i+1})$ is normal (weight 0). Except when $x_{i-1} = N-3$ and $x_i = N-1$, $x_i$ can be moved one position to the left or right to produce a normal edge followed by a frameshift edge. (This changes the deleted symbol in $E$, which is of no consequence.) Repeating this transformation moves frameshift edges to the right side of the deletion gap. Two "short" frameshifts (positions $x$ and $x+2$) can be replaced by one long one, a short one and a long one cancel (the edge from $N-2$ to $N$ is considered "long"), and two long frameshifts can be replaced by two normal edges and a short frameshift. The first two kinds of replacements improve the score, while the third at least breaks even since $\pi \geq gep$. In the end, we're down to at most one frameshift edge, which either enters a node not corresponding to a deleted symbol (i.e., a node with an aligned symbol or node $N$) or is the edge from $N-3$ to $N-1$. □

Theorem 4 permits simplification of the recurrence relation for $D(i,j)$ in Figure 8, to the following.

$$D(i,j) = \max \begin{cases} D(i-3,j) - gep \\ C(i-3,j) - gop - gep \end{cases}$$

Of course, the recurrences are invalid when $i = N$, given that edges enter node $N$ only from $N-1$ and $N-2$. $C(N,j)$ is needed only for $j = M$, where it is determined by:

$$C(N,M) = \max \begin{cases} C(N-1,M) \\ C(N-2,M) - \pi \\ C(N-3,M) - gop \\ D(N-3,M)) - \pi - gep \end{cases}$$

The last two terms account for the special nature of the edge from $N-3$ to $N-1$, as described by Theorem 4.

Boundary conditions need to be specified to derive a dynamic programming algorithm for the recurrences. For $I$ the only boundary condition is:

$$I(j,0) = -\infty$$

For $D$, the boundary condition are:

$$D(0,j) = D(2,j) = D(3,j) = -\infty$$
$$D(1,j) = C(0,j) - gop - gep$$

For $C$, assuming $j > 0$, we have:

$$C(0,0) = 0 \qquad C(0,j) = I(0,j) \qquad C(j,0) = D(j,0)$$

$$C(1,j) = \max(C(0,j-1) + S[a_1,b_j], I(1,j), D(1,j)),$$
$$C(2,j) = \max(C(0,j-1) + S[a_2,b_j] - \pi, I(2,j)),$$
$$C(3,j) = \max(C(1,j-1) + S[a_3,b_j] - \pi, I(3,j)),$$

$$C(4,j) = \max \begin{cases} I(i,j) \\ D(4,j) \\ C(1,j-1) + S[a_4,b_j] \\ C(2,j-1) + S[a_4,b_j] - \pi \end{cases}$$

All other values can be computed using the general formulas.

In essence, three-frame alignment restrict frameshift errors to fall between two codons (not within a codon). Its main advantages are that it can be implemented to run about 30% faster (the inner loop has one access to $S$, rather than three accesses to $R$) and the print-out of the alignment is easier to interpret.

## 5 Still other approaches

Knecht [8] considers a very general notion of sequence alignment, in which the substitution scores $S[x, y]$ are defined for $p$-tuples $x$ and $q$-tuples $y$ of sequence entries. Some of his results rely on a definition rather like our $C_x$, resulting in a running time that is similar to an algorithm based on Figure 6 (i.e., the product of the sequence lengths times the square of the alphabet size). However, his model is not general enough to capture our results. First of all, our model permits matching of an amino acid with 2, 3, or 4 nucleotides, whereas Knecht uses fixed $p$ and $q$. More importantly, our penalty for a gap in the DNA sequence depends in part on the context within the alignment. Namely, if the gap is within a precodon it will be penalized at the nucleotide level; otherwise it can be penalized at either the amino-acid or nucleotide level. Knecht's result assumes an additive gap-penalty ($gop = 0$) and handles only nucleotide-level deletions.

Guan and Uberbacher [3] give an algorithm that solves the three-frame alignment problem. However, their recurrence relations are substantially more complicated than ours. To compute the three values at a grid point, they need 15 earlier values, where we need only seven. The complexity of these recurrences would make it harder to optimize the implementation.

Huang and Zhang [7] study DNA-protein alignment when introns are allowed in the DNA sequence, and they model all evolutionary events at the DNA level. Their algorithm for optimal alignments allows only one deletion gap within (what we call) a precodon, whereas our general model allows arbitary gaps. Their recurrence relation dealing with an intra-codon deletion suffers a much larger constant factor in the running time per grid point than does ours (roughly 138 vs. 20). On the other hand, they present a simplified algorithm, for which they do not guarantee optimality of the solution. We note that our approach can be modified to assess a constant penalty for large gaps, thereby providing a mechanism for handling introns.

## 6 Experience

We have incorporated both the algorithm of Section 1 and the simpler, between-codon-frameshift algorithm in `fastx` and `tfastx`, new programs in the *FASTA* package of protein and DNA sequence comparison programs. `fastx` compares a DNA sequence to a protein sequence database, translating the DNA sequence in three frames and calculating a similarity score that allows frameshifts. `tfastx` is an extension of `tfasta` and compares a protein sequence to a DNA sequence library, translating the DNA library in three forward and three reverse frames "on-the-fly", calculating similarity scores that include frameshifts.

`fastx` was implemented by extending `fasta` [10] by translating the DNA query sequence in all three frames, concatenating these three sequences separated by a flag character, and building the initial lookup table used by `fasta` from the concatenated protein sequences. Thus, the standard `fasta` algorithm is used, but on a sequence that is three times as long. `fasta` includes a step in which "consistent" high-scoring segment pairs (alignments without gaps that do not overlap) are threaded together to produce a pseudo-alignment with gaps; `fastx` modifies this process slightly so that high-scoring segment pairs that are from different reading frames but start in the same codon appear to share the same diagonal. The diagonal with the best similarity score is identified and used to center a band-limited alignment using the algorithms described above.

`tfastx` uses a complementary approach, building a three-frame translation from each library sequence (and its reverse complement) and looking up residues from the resulting three-fold longer protein sequence in the protein query sequence. Again, a "consistent" pseudo-alignment is produced and the diagonal with the best similarity score rescanned with the band-limited alignment algorithm.

Comparing a protein sequence to a protein database with `fasta` is faster than comparing a cDNA sequence to the same database with `fastx`; both because the cDNA sequence is longer and because there are additional paths that must be considered in the grid graph. In one test, it took 29s for `fasta`, *ktup=2* to compare a 218 amino acid mouse glutathione transferase sequence (SwissProt GTM1_MOUSE) against the entire SwissProt protein sequence database. The same search using `fastx` and the cDNA sequence that encodes the glutathione transferase sequence took about three times as long: 95s. This time difference is almost exactly that expected based on the lengths of the two sequences; a cDNA sequence must be three times the length of the amino acid sequence it encodes. Using a 684 amino-acid query sequence (SNOB_HUMAN) took 51s to search SwissProt; the encoding cDNA sequence required 173s. (These timings are the best possible because the cDNA sequence was edited to match exactly the protein coding region; full-length cDNA sequences typically contain from 500 to more than 1000 nucleotides of non-coding sequence that would increase the time required.)

Naturally, `fastx` is not the best strategy for identifying a cDNA sequence by sequence similarity if the cDNA sequence is 100% accurate. `fastx` is designed for cDNA sequences that contain frameshift errors, typically because of high-throughput single-pass cDNA sequencing projects. For protein sequences, `fasta` is not only faster, it can identify more distantly related sequences because it is less likely that an unrelated sequence will produce a high similarity score by chance, simply because the amino-acid sequence is 1/3 the length of the translated cDNA sequence.

However, `fastx` is much more effective at comparing an error-prone cDNA sequence than traditional DNA–DNA sequence comparison. For example, a search of the entire GenPept protein sequence database (52, 659, 744 amino acids in 170, 134 sequences) with a "mutated" (26 errors in 1089 nucleotides) GTM1_MOUSE cDNA sequence required 319s; a `fasta` DNA database search, which was restricted to mammalian and vertebrate sequences (130, 261, 868 nucleotides in 124, 436 sequences), required 838s but missed about 1/2 of the clearly related sequences detected by `fastx`. Thus, by comparing sequences at the protein level, where information is available about conservative and non-conservative amino-acid changes, additional distant sequence homologies can be identified.

`tfastx` is a considerable improvement over `tfasta` [10], which it replaces, because it calculates a single score, which may include frameshifts, for the protein query vs DNA library sequence comparison. Calculation of a single score that reflects the best match to all three translation frames greatly simplifies the statistics and interpretation of a protein vs DNA database search. We have used `tfastx` to routinely scan EST databases for members of specific protein families and to confirm the assignment of open reading frames in microbial genomic DNA sequences.

## References

[1] Chao, K.-M., W. Pearson and W. Miller (1992) Aligning two sequences within a specified diagonal band. *CABIOS* **8**, 481-487.

[2] Gotoh, O. (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.* **162**, 705–708.

[3] Guan, X., and E. Uberbacher (1996) Alignments of DNA and protein sequences containing frameshift errors. *CABIOS* **12**, 31–40.

[4] Hein, J. (1994) An algorithm combining DNA and protein alignment. *J. theor. Biol.* **167**, 169–174.

[5] Hein, J., and J. Stovlbaek (1994) Genomic alignment. *J. Mol. Evol.* **38**, 310–316.

[6] Hirschberg, D. (1975) A linear space algorithm for computing maximal common subsequences. *Comm. ACM* **18**, 341–343.

[7] Huang, X., and J. Zhang (1996) Methods for comparing a DNA sequence with a protein sequence. To appear in *CABIOS.*

[8] Knecht, L. (1995) Pairwise alignment with scoring on tuples. *Combinatorial Pattern Matching* (Springer Lecture Notes in Computer Science, 937), 215–229

[9] Peltola, H., H. Soderlund and E. Ukkonen (1986) Algorithms for the search of amino acid patterns in nucleic acid sequences. *Nucleic Acids Res.* **14**, 99–107.

[10] Pearson, W. R., and Lipman, D. J. (1988) Improved tools for biological sequence comparison *Proc. Natl. Acad. Sci. USA* **85**, 2444–2448.