The Pairwise Sequence Alignment Problem

Brett Pellock[1] and Brian Tjaden[2]

Motivation

For many human genes, their nucleotide or protein sequence is similar to that of a gene in another organism. Genes from two different organisms can have similar sequences for a variety of reasons, but often the genes share a common ancestor and are said to be related or *homologous*. Figure 1 illustrates two homologous sequences from part of a gene that codes for a leukemia transcription factor found in many vertebrates, including humans and zebra fish. Approximately 30% of human genes have homologs in the genome of a worm, 50% in the genome of a fly, 90% in the genome of a fish, and 99% in the genome of a chimpanzee. Over time, genes evolve and homologous genes are likely to diverge through nucleotide mutations, insertions, and deletions. However, too much change to a gene sequence may result in a loss of its function with negative effects on an organism's fitness, which will be selected against in the process of evolution.

A fundamental problem in genomics is determining whether two sequences, such as two DNA sequences or two protein sequences, are related. Here, we restrict ourselves to primary sequences, though there are many interesting and important problems pertaining to structural and functional relationships. Given two DNA sequences, one whose functional role is known and one whose functional role is unknown, recognizing a relationship between the two sequences might suggest that the sequence with unknown function has a similar role as that of the sequence with known function. Indeed, identification and investigation of homologs of the *pbx1* gene in humans, flies, and fish, shown in part in Figure 1, helped elucidate the role of this oncogene in tumor progression [1, 2]. More broadly, in modern genomics, the annotation of newly sequenced genomes is primarily established through comparative genomics approaches, i.e., through recognizing relationships between a newly sequenced genome and previously annotated genomic sequences [3].

Ideally, we would establish a firm relationship between the sequences, e.g., that the two sequences are derived from a common ancestor and, hence, homologous. Unfortunately, without being able to observe the evolution of the sequences from their common ancestor, it would be all but impossible to prove a homologous relationship between the two sequences. Though we cannot generally prove homology of sequences, we can often estimate the similarity of two sequences, and from this similarity we can hypothesize or infer homology of the sequences. Like many bioinformatics methods, then, computational approaches that assess the similarity of two genomic sequences are generally hypothesis-generating. When viewed through the lens of the scientific method, these approaches emphasize that part of the scientific method relating to *generating* hypotheses in contrast to more experimentally oriented approaches that emphasize that part of the scientific method relating to *testing* hypotheses.

[1] Department of Biology, Providence College, Providence, RI.
[2] Department of Computer Science, Wellesley College, Wellesley, MA.

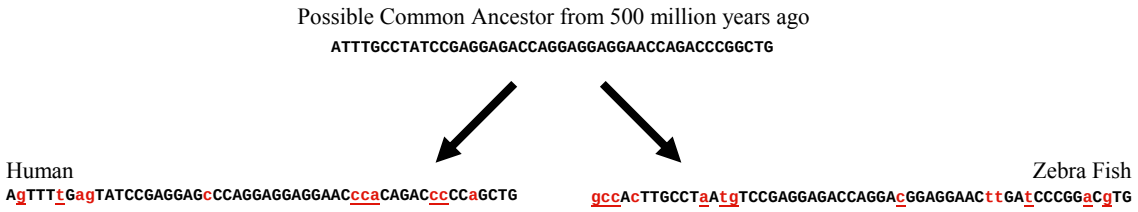

Figure 1: Nucleotide sequences from part of the *pbx1* gene in the human (*homo sapiens*) genome and in the zebra fish (*Danio rerio*) genome are shown at the bottom of the figure. The *pbx1* gene codes for a pre-B-cell leukemia transcription factor. The similarity of the human and zebra fish sequences suggest that the sequences are homologous. While the genome of the common ancestor from about 500 million years ago is unknown, the sequence at the top of the figure represents part of a hypothesized ancestral gene sequence for *pbx1*. In the human and zebra fish sequences, lower case nucleotides in red illustrate portions of the sequence that have mutated from or been inserted into (underlined) the hypothesized ancestral sequence.

The Pairwise Sequence Alignment Problem

Initially, we will focus our attention on *global* sequence alignments and rudimentary scoring models. After establishing a foundation for calculating simpler alignments, we will extend the described methods to include *local* sequence alignments and more sophisticated scoring models.

An *alignment* of two sequences is a one-to-one mapping of characters in the first sequence to characters in the second sequence such that the mapping preserves the order of the characters in the two sequences. Often gaps, meant to reflect insertion or deletion events that may have occurred as two homologous sequences diverged over time, are included as characters of each sequence [4]. For example, a few possible alignments of the two genomic sequences `CGTTACATG` and `TGTCACGT` are shown in Figure 2.

In the alignments in Figure 2, the gaps are represented as hyphens. The alignments in the figure range from having a single gap to having up to eleven gaps. Vertical bars between the alignments illustrate characters in one sequence that are mapped to identical characters in the second sequence. A gap in one sequence is disallowed from mapping to a gap in the second sequence. In each alignment, excluding gaps, the order of characters in each sequence is the same as in the original sequences. The alignments are only a few of the many possible alignments of the two sequences. Assuming we have some measure for scoring alignments, the *pairwise alignment problem* is the problem of finding the best or optimal scoring alignment for a pair of sequences. Finding the optimal pairwise alignment of two sequences is one way to estimate the similarity of two sequences and, thus, form a hypothesis about their relationship.

In order to assess whether one alignment is better than another, we benefit from having some measure for scoring alignments. Later, we will consider the benefits of various different scoring schemes, but for illustrative purposes initially, we will use the following scoring scheme. When a character in one sequence is mapped to an identical character in the other sequence, depicted with a vertical bar, we will denote this pair of

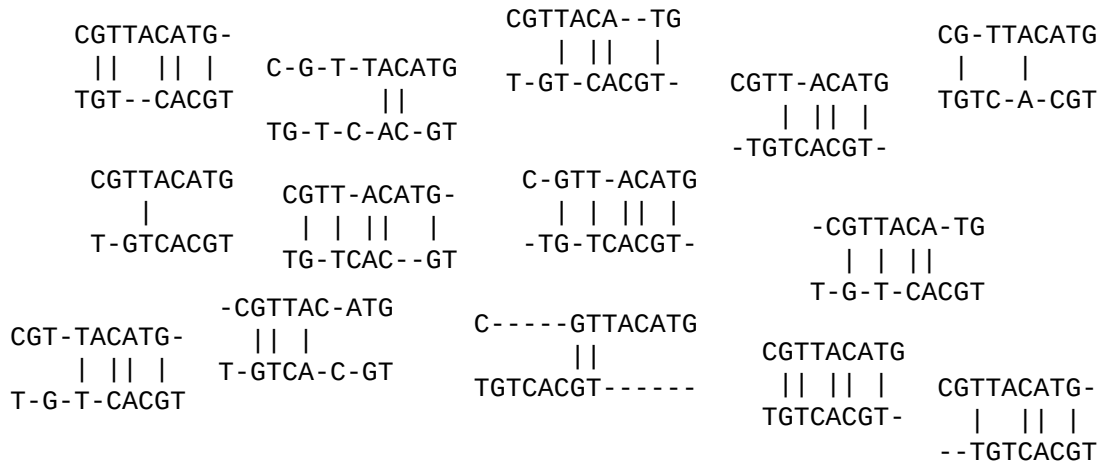

Figure 2. Fourteen of the many possible pairwise alignments of the sequences `CGTTACATG` and `TGTCACGT`. In the alignments, hyphens represent gaps and vertical bars represent matching pairs of aligned characters.

characters as a *match* and add +5 to our cumulative alignment score. When a character in one sequence is mapped to a non-identical, non-gap character in the other sequence, we will denote this pair of characters as a *mismatch* and add -4 to our cumulative alignment score. When a character in one sequence is mapped to a gap character in the other sequence, we will denote this pair of characters as a *gap* and add -6 to our cumulative alignment score. The score of an alignment, then, is the sum of the scores of the individually aligned characters. For example, the alignment in Figure 3a has an alignment score of 7 since it has 5 matches contributing +5 each, 3 mismatches contributing -4 each, and 1 gap contributing -6 each. The alignment in Figure 3b has an alignment score of -1 since it has 5 matches contributing +5 each, 2 mismatches contributing -4 each, and 3 gaps contributing -6 each.

In the pairwise sequence alignment problem, our goal is to determine the best scoring alignment for two sequences out of all possible alignments of the two sequences. If the two sequences are very short, we may be able to align them well by hand. If our goal is to visualize the similarity of the two sequences, then a dot-matrix plot may be used [5]. However, if we are interested in calculating the optimal alignment of the two sequences and if the sequences are not trivially short, then it is useful to consider systematic or algorithmic approaches for doing so. One approach for determining an optimal pairwise alignment for two sequences would be to enumerate all possible alignments of the two sequences, score each of the alignments, and then determine the alignment with maximum score. This approach is problematic in that the number of different alignments for a given pair of sequences can be very large. The number of alignments is an exponential function of the length of the two sequences and, thus, it is computationally intractable to list all possible alignments for two sequences that contain thousands or even hundreds of characters. Certainly, we prefer to design an approach for computing the optimal pairwise alignment of two sequence that is effective on sequences that contain hundreds or thousands of characters, so any approach that requires enumerating exponentially many alignments can be ruled out as computationally

Figure 3. Assuming a score of +5 for each match, -4 for each mismatch, and -6 for each gap, the alignment in (a) has a score of 7 and the alignment in (b) has a score of -1.

infeasible. How then can we find the *best* scoring alignment of two sequences if we do not compute the score of *every possible* alignment of the two sequences?

Algorithmic Properties

Of course, some problems are better solved by computers than others. In fact, many problems can be classified into one of only a few families indicating the problem's amenability to solution using computers. For example, there are problems that computers provably cannot solve at all, no matter how much time they are given. A classic example of such a problem is the *halting problem*. It can be shown that no computer program can be written that, in general, determines whether another computer program runs forever or eventually stops running, i.e., halts [6]. There are other classes of problems that computers can solve but, in general, solving them exactly would require more time than the age of the universe, so for all practical purposes, optimal solutions cannot be obtained computationally [7]. Examples of such problems include various instances of determining an optimal alignment of *multiple* sequences as opposed to a *pair* of sequences [8], finding optimal regulatory sites in a set of sequences [9], and identifying optimal clusters for sets of microarray data [10]. For these problems, *heuristic* methods are often used to efficiently identify approximate or suboptimal solutions. Still other classes of problems can be solved optimally by computers in a reasonable amount of time. Often, it is not immediately obvious which class a problem belongs to. For instance, someone pondering the pairwise sequence alignment problem for the first time might reasonably expect that the problem could not be solved efficiently since it is computationally intractable to enumerate all pairwise alignments for two sequences. However, the problem has commonalities with a class of problems that lend themselves to efficient computational solutions. Specifically, the pairwise alignment problem has two properties that provably enable efficient solution of the problem through a technique known as *dynamic programming* [11]. While a broader investigation of this technique is beyond the scope of this paper, we detail the use of dynamic programming below in the context of efficiently solving the pairwise sequence alignment problem.

The pairwise sequence alignment problem has two important properties that facilitate its solution [12]. First, the solution to the problem of finding the optimal alignment of two sequences can be found by considering the solutions to subproblems, i.e., by utilizing the optimal alignments of various *sub*sequences of the original two sequences. Second, the subproblems often overlap. These two properties suggest that many of the same subproblems present themselves multiple times and solutions to these subproblems can be used to solve the problem. As a result, rather than solve a subproblem each time it presents itself, we can solve the subproblem once and remember the solution by storing it in a table. Then, if the subproblem presents itself again, rather

than re-compute the solution we can simply look the solution up in the table, thereby saving ourselves the costs of re-computation.

To formalize the approach for how the optimal alignment of two sequences can be computed, we introduce some notation. Let $s_{i,j}$ refer to the subsequence[1] of sequence s from the character at index i to the character at index j, inclusive. For example, is s refers to the sequence ACGTGA, consisting of n=6 characters, then $s_{2,4}$ corresponds to the subsequence CGT consisting of 3 characters, $s_{1,1}$ corresponds to the subsequence A consisting of 1 character, and $s_{1,n}$ corresponds to the subsequence ACGTGA consisting of 6 characters. Also, let $\overline{s \cdot t}$ refer to the optimal alignment score for two sequences s and t. Finally, let *align*(c, d) refer to the alignment score of two individual characters c and d. For example, if matches have an alignment score of +5, mismatches have an alignment score of -4, and gaps have an alignment score of -6, then *align*(G, G) would correspond to +5, *align*(G, C) would correspond to -4, and both *align*(G, -) and *align*(-, G) would correspond to -6.

It can be shown that the optimal alignment score $\overline{s \cdot t}$ of two sequences, s and t, can be calculated from three subproblems, i.e., from the optimal alignment scores of aligning three pairs of subsequences from s and t [12]. Specifically, for two sequences s and t with lengths m and n, respectively, the optimal alignment score for the two sequences can be calculated as

$$\overline{s \cdot t} = \max \begin{cases} \overline{s_{1,m-1} \cdot t_{1,n}} + align(s_{m,m}, -) \\ \overline{s_{1,m} \cdot t_{1,n-1}} + align(-, t_{n,n}) \\ \overline{s_{1,m-1} \cdot t_{1,n-1}} + align(s_{m,m}, t_{n,n}) \end{cases} . \quad \textbf{(1)}$$

Figure 4 illustrates how the optimal pairwise alignment score can be computed for two example sequences, s=AGCGTTA and t=ACGTGA. As Figure 4a depicts, the optimal score for aligning AGCGTTA and ACGTGA is the largest of three quantities: (1) the optimal score for aligning AGCGTT and ACGTGA summed with the score of aligning A, the last character of AGCGTTA, with a gap, (2) the optimal score for aligning AGCGTTA and ACGTG summed with the score of aligning a gap with A, the last character of ACGTGA, or (3) the optimal score for aligning AGCGTT and ACGTG summed with the score of aligning A, the last character of AGCGTTA, with A, the last character of ACGTGA.

We haven't yet described how to compute each of the three quantities, namely the optimal score of aligning AGCGTT and ACGTGA, the optimal score of aligning AGCGTTA and ACGTG, and the optimal score of aligning AGCGTT and ACGTG. However, suppose that we somehow learned the answers, namely that the optimal score of aligning AGCGTT and ACGTGA is 4, the optimal score of aligning AGCGTTA and ACGTG is 4, and the optimal score of aligning AGCGTT and ACGTG is 10. If we assume that matched pairs of characters have

[1] We use the term *subsequence* to mean a *consecutive* ordered subset of characters in a sequence. In contrast, the term subsequence is sometimes defined as an ordered subset of characters in a sequence, where the characters are not necessarily consecutive. By our definition, ACG is a subsequence of ACGTGA but AG is not a subsequence of ACGTGA.

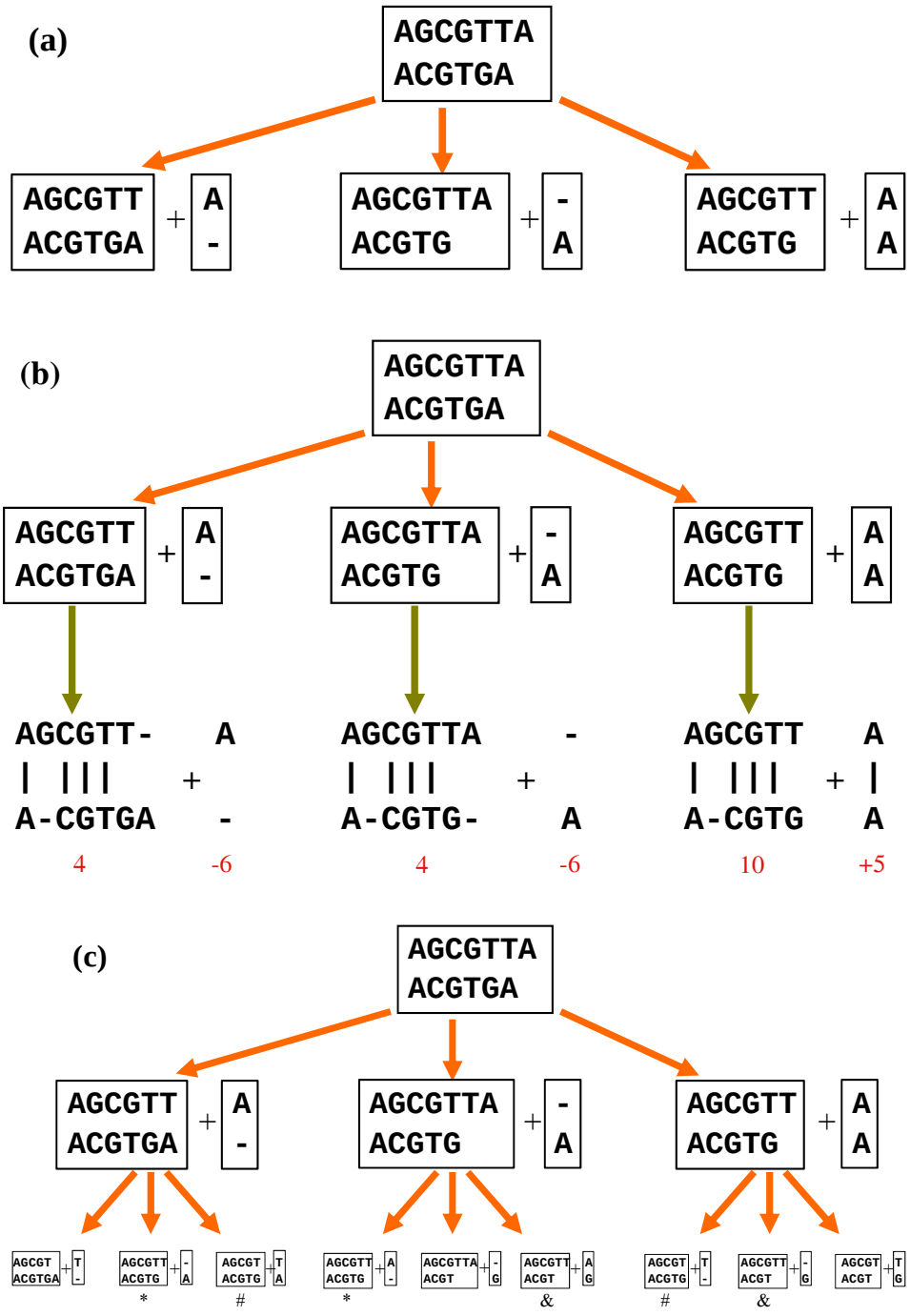

Figure 4. The figure illustrates the steps involved in calculating the optimal alignment score of sequences AGCGTTA and ACGTGA. (a) The optimal alignment score of the two sequences is equal to the maximum of three values, each of which is a sum of two terms. (b) Assuming a match score of +5, a mismatch score of -4, and a gap score of -6, the optimal alignment score of the two sequences is equal to the maximum of three values, 4+(-6), 4+(-6), and 10+(+5). Thus, the optimal alignment score is equal to 15. (c) The optimal alignment score of the two sequences can be decomposed into three subproblems, each of which in turn can be decomposed into three subproblems. The two subproblems of subproblems marked with an asterisk, *, are similar, the two marked with a pound sign, #, are similar, and the two marked with an ampersand, &, are similar.

an alignment score of +5, mismatched pairs of characters have an alignment score of -4, and characters aligned with a gap have an alignment score of -6, then as Figure 4b depicts, the optimal score for aligning AGCGTTA and ACGTGA is the largest of three quantities: (1) the optimal score for aligning AGCGTT and ACGTGA, which is 4, summed with the score of aligning A, the last character of AGCGTTA, with a gap, which is -6 (2) the optimal score for aligning AGCGTTA and ACGTG, which is 4, summed with the score of aligning a gap with A, the last character of ACGTGA, which is -6, or (3) the optimal score for aligning AGCGTT and ACGTG, which is 10, summed with the score of aligning A, the last character of AGCGTTA, with A, the last character of ACGTGA, which is +5. The largest of these three quantities, namely 4 + (-6), 4 + (-6), and 10 + (+5), is 15. Thus, given our assumptions, the optimal alignment score of the two sequences is 15.

Notice, we have decomposed the problem of calculating the optimal alignment score of two sequences into three subproblems. Each of the three subproblems also involves calculating the optimal alignment score for a pair of sequences. However, the subproblems are *smaller* than the original problem in the sense that one or both sequences in the subproblems are one character shorter than the sequences in the original problem. While we have not proved that the optimal alignment score for two sequences can be calculated from the solutions of three subproblems [12], such a proof is beyond the scope of this article, we are relying on this fact and illustrating its application.

The astute reader will notice that we have not yet described how the three subproblems may be solved. But, of course, the subproblems involve computing the optimal pairwise alignment score for two sequences, just as the original problem did, so the subproblems can be solved by the same approach. Each of the three subproblems can, itself, be solved by solving three subproblems of the subproblems, as shown in Figure 4c. And each of the subproblems' subproblems can be solved by decomposing the problem into three further subproblems. Each time we generate a new subproblem, one or both of the sequences involved is one character smaller than it was previously. If we keep decomposing a problem into subproblems involving smaller and smaller sequences, eventually we will be dealing with subproblems where one or both of the sequences contain no characters. In such a case, we needn't continue decomposing problems into three subproblems, we can trivially compute the optimal alignment score for two sequences if one or both of the sequences contains no characters.

One might wonder whether characterizing the solution to a problem with three subproblems, each of which in turn requires solving three subproblems, and so forth, might generate a large number of subproblems needing solution. In general, the answer is *yes*. For two sequences that are each several hundred characters in length, we could easily generate an intractable number of subproblems requiring solution. Fortunately, as suggested by the second property of the pairwise sequence alignment problem, many of the subproblems overlap. As Figure 4c illustrates, many of the subproblems are identical to other subproblems. If we solved a subproblem once, rather than re-solve it again, assuming we have stored the solution in a table, we can look up the solution to the subproblem in the table. The number of unique subproblems that require solution is, in fact, tractable, so our table of subproblem solutions can be of manageable size.

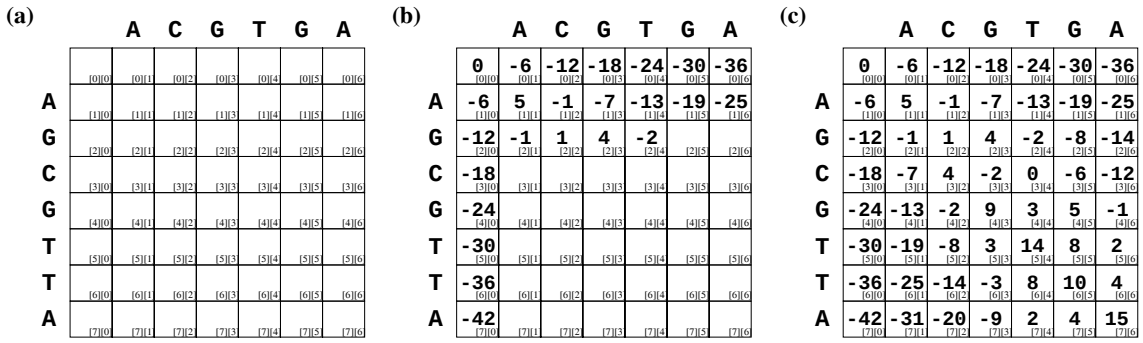

Figure 5. Tables are depicted for the sequences *s*=AGCGTTA and *t*=ACGTGA. The row and column index of each table entry are shown in square brackets in the bottom right corner of each table entry to enable easy visual reference. Each entry in a table corresponds to the optimal score for a particular subsequence of *s* aligned with a particular subsequence of *t*. For the tables in the figure, a match score of +5, a mismatch score of -4, and a gap score of -6 are assumed. (a) None of the entries in the table are filled in. (b) The table is partially filled in. (c) The table is completely filled in.

Storing Solutions to Subproblems in a Table

Given two sequences *s* and *t* of lengths *m* and *n*, respectively, we will create a table, *T*, consisting of *m*+1 rows and *n*+1 columns. Figure 5 illustrates such a table for the two sequences *s*=AGCGTTA and *t*=ACGTGA. We will use the notation *T*[*i*][*j*] to refer to the entry in the i^{th} row and j^{th} column of table *T*. Each entry in the table corresponds to the optimal score for a particular subsequence of *s* aligned with a particular subsequence of *t*. Specifically, *T*[*i*][*j*] corresponds to $\overline{s_{1,i} \cdot t_{1,j}}$. For example, for the table in Figure 5, *T*[3][5] corresponds to the optimal score of aligning $s_{1,3}$, which is the sequence AGC, with $t_{1,5}$, which is the sequence ACGTG. Similarly, in Figure 5, *T*[7][1] corresponds to the optimal score of aligning AGCGTTA with A, *T*[0][3] corresponds to the optimal score of aligning an empty sequence with ACG, i.e., aligning three gaps with ACG, and *T*[7][6] corresponds to the optimal score of aligning AGCGGTA with ACGTGA. Thus, if our goal is to calculate the optimal alignment score for *s* and *t*, we need only determine the table entry *T*[*m*][*n*]. However, as we will see, computing *T*[*m*][*n*] requires first computing values for all other entries in the table.

Populating the Table

As Figure 4 illustrates, the optimal score of aligning AGCGTTA with ACGTGA can be expressed as a function of the optimal scores of aligning AGCGTT with ACGTGA, aligning AGCGTTA with ACGTG, and aligning AGCGTT with ACGTG. Using our table notation and our example in Figure 5, the optimal score of aligning AGCGTTA with ACGTGA, which is *T*[7][6], can be expressed as a function of *T*[6][6], *T*[7][5], and *T*[6][5]. Reformulating expression (1) using our table notation, we have

$$T[7][6] = \max\begin{cases} T[6][6] + align(s_{7,7}, -) \\ T[7][5] + align(-, t_{6,6}) \\ T[6][5] + align(s_{7,7}, t_{6,6}) \end{cases} \quad \textbf{(2)}$$

and more generally

$$T[i][j] = \max\begin{cases} T[i-1][j] + align(s_{i,i}, -) \\ T[i][j-1] + align(-, t_{j,j}) \\ T[i-1][j-1] + align(s_{i,i}, t_{j,j}) \end{cases} . \quad \textbf{(3)}$$

Expression (3) suggests that any entry in a table can be calculated as a function of three other table entries, namely the table entry above, to the left, and diagonally above and left of the entry to be filled in. For instance, in Figure 5b, *T*[2][4] is filled in with the score -2 because -2 is the maximum of the following three quantities, (1) *T*[1][4] plus the score of aligning `G` with a gap, which is -6, (2) *T*[2][3] plus the score of aligning `T` with a gap, which is -6, and (3) *T*[1][3] plus the score of aligning `G` with `T`, which is -4.

To fill in a table completely, we begin by filling in the first row and column of the table. Since each table entry in the first row and first column corresponds to the optimal score of aligning some sequence with an empty sequence, the optimal score of the alignment can be calculated as the score of aligning characters in one sequence with the appropriate number of gaps. For example, filling in table entry *T*[5][0] using the sequences in Figure 5 corresponds to the optimal score of aligning `AGCGT` with an empty sequence, which corresponds to aligning `AGCGT` with 5 gaps, and if each gap alignment contributes -6, then the optimal alignment score is -30. Once the first row and first column of the table are filled in, we proceed from left to right across each row, starting at row 1 and ending at row *m*, filling in each table entry based on expression (3). Our approach can be expressed in pseudocode as follows

```
// Assumes the first row and column of the table have been filled in
Repeat for each row i = 1 to m
        Repeat for each column j = 1 to n
                T[i][j] = max( T[i-1][j] + align(s_i,i, -),
                               T[i][j-1] + align(-, t_j,j),
                               T[i-1][j-1] + align(s_i,i, t_j,j))
```

Once the table has been completely filled in, the optimal score for aligning sequences *s* and *t* in their entirety can be found at table entry *T*[*m*][*n*].

Notice that the use of a table to calculate the optimal alignment score for two sequences is effective precisely because the alignment problem has two properties, namely the solution to the problem can be expressed as a function of solutions to subproblems and the subproblems overlap. The first property suggests that earlier entries in the table can be used to calculate later entries. The second property suggests that it is worth storing solutions to subproblems to avoid re-computation since the subproblem solutions may be used multiple times. Rather than compute solutions to more than 2^{n+m}

Figure 6. A table is depicted for the sequences *s*=AGCGTTA and *t*=ACGTGA. The row and column index of each table entry are shown in square brackets in the bottom right corner of each table entry to enable easy visual reference. Each entry in the table corresponds to the optimal score for a particular subsequence of *s* aligned with a particular subsequence of *t*, assuming a match score of +5, a mismatch score of -4, and a gap score of -6. In the top left corner of each table entry, the indicated row index and column index represent the subproblem from which the optimal score was calculated for that table entry.

		A	C	G	T	G	A
	[-1][-1] 0 [0][0]	[0][0] -6 [0][1]	[0][1] -12 [0][2]	[0][2] -18 [0][3]	[0][3] -24 [0][4]	[0][4] -30 [0][5]	[0][5] -36 [0][6]
A	[0][0] -6 [1][0]	[0][0] 5 [1][1]	[1][1] -1 [1][2]	[1][2] -7 [1][3]	[1][3] -13 [1][4]	[1][4] -19 [1][5]	[1][5] -25 [1][6]
G	[1][0] -12 [2][0]	[1][1] -1 [2][1]	[1][1] 1 [2][2]	[1][2] 4 [2][3]	[2][3] -2 [2][4]	[2][4] -8 [2][5]	[2][5] -14 [2][6]
C	[2][0] -18 [3][0]	[2][1] -7 [3][1]	[2][1] 4 [3][2]	[2][3] -2 [3][3]	[2][3] 0 [3][4]	[2][4] -6 [3][5]	[2][4] -12 [3][6]
G	[3][0] -24 [4][0]	[3][1] -13 [4][1]	[3][2] -2 [4][2]	[3][2] 9 [4][3]	[4][3] 3 [4][4]	[3][4] 5 [4][5]	[4][5] -1 [4][6]
T	[4][0] -30 [5][0]	[3][1] -19 [5][1]	[4][2] -8 [5][2]	[4][3] 3 [5][3]	[4][3] 14 [5][4]	[5][4] 8 [5][5]	[5][5] 2 [5][6]
T	[5][0] -36 [6][0]	[5][1] -25 [6][1]	[5][2] -14 [6][2]	[5][3] -3 [6][3]	[5][3] 8 [6][4]	[5][4] 10 [6][5]	[5][5] 4 [6][6]
A	[6][0] -42 [7][0]	[6][0] -31 [7][1]	[6][2] -20 [7][2]	[6][3] -9 [7][3]	[6][4] 2 [7][4]	[6][4] 4 [7][5]	[6][5] 15 [7][6]

non-unique subproblems, we need only compute solutions to (*n*+1)*(*m*+1) unique subproblems.

The pairwise alignment problem is not the only problem that exhibits the abovementioned two properties. Many problems have these two properties and can be solved using the technique of dynamic programming. Dynamic programming is the problem solving approach of creating a table with solutions to subproblems, and using the solutions to subproblems to solve larger problems [11]. Another problem that is often addressed using dynamic programming is the problem of calculating an energetically favorable secondary structure for a given RNA sequence [13, 14].

Alignment

While we have described how to compute the optimal pairwise alignment *score* for two sequences, we have not yet discussed how to compute the pairwise *alignment* associated with an optimal score. Fortunately, the table that is constructed to calculate an optimal alignment score can also be used to calculate an alignment associated with the optimal score. By *backtracking* through the table, we can construct an alignment, starting at the end of the alignment and progressing toward the beginning of the alignment. First, we modify our approach for filling in a table so that each table entry keeps track not only of an optimal score for a subproblem but also how that score was determined, i.e., which of the three possible subproblems yielded the maximum value. Figure 6 shows an example table filled in with both optimal scores and, in the top left corner of the table entries, an indicator of which of three subproblems yielded the maximum value. For example, in Figure 6, table entry *T*[3][2] shows an optimal alignment score of 4, and the index [2][1] in the top left corner of the table entry indicates that the optimal score of 4 was calculated as the sum of *T*[2][1] and the score of aligning the appropriate pair of characters, *align*(C, C). Similarly, for table entry *T*[6][2], the optimal score of -14 was calculated based on the value at index [5][2] in the table. For some table entries, more than one subproblem could lead to an optimal score, i.e., when computing the maximum of three values, more than one of the three values could achieve the maximum. For instance, the score of -6 in table

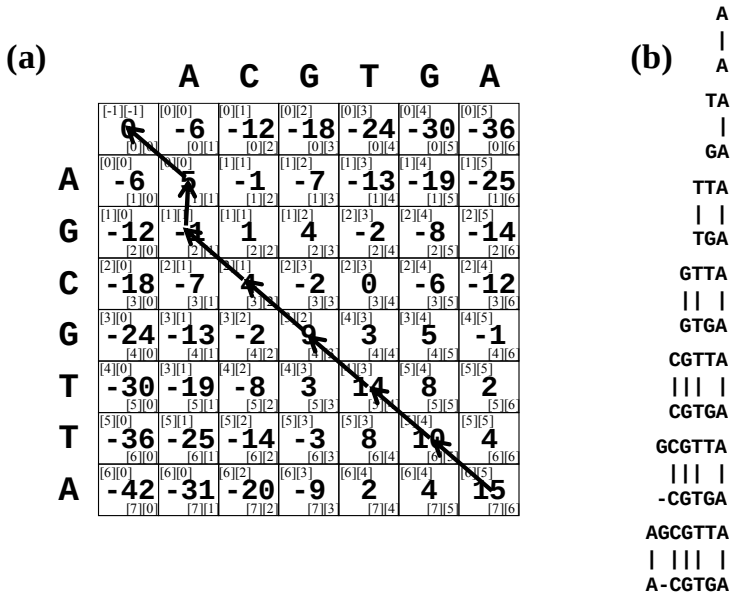

Figure 7. A filled-in table is depicted for the sequences *s*=AGCGTTA and *t*=ACGTGA. (a) Arrows in the table indicate a backtracking path from the last table entry, *T*[7][6], to the first, *T*[0][0], based on the indices in the top left corner of table entries. Each arrow represents a single backtracking step. (b) With each of the seven backtracking steps, a pair of characters is added to the alignment, starting at the end of the alignment and working toward the beginning.

entry *T*[3][5] in Figure 6 could have been achieved either from *T*[3][4]+*align*(-, G)=-6 or from *T*[2][4]+*align*(C, G)=-6. The top left corner of the table entry was filled in arbitrarily with [2][4] rather than [3][4]. Such a situation represents a scenario where two sequences can be aligned in different ways with each achieving an optimal score.

To determine an alignment once a table has been filled in, we begin at the last table entry *T*[*m*][*n*] and proceed to earlier table entries based on the indices in the top left corners of the table entries until we reach *T*[0][0]. Figure 7 illustrates the construction of an alignment based on a filled in table. In the figure, *T*[7][6] is derived from *T*[6][5] and aligning A with A, which is shown in the first of seven alignments in Figure 7b. *T*[6][5], in turn, is derived from *T*[5][4] and aligning G with T, which is shown in the second of seven alignments in Figure 7b. *T*[5][4] is derived from *T*[4][3] and aligning T with T, *T*[4][3] is derived from *T*[3][2] and aligning G with G, *T*[3][2] is derived from *T*[2][1] and aligning C with C, *T*[2][1] is derived from *T*[1][1] and aligning G with a gap, and *T*[1][1] is derived from *T*[0][0] and aligning A with A. The growing alignment is shown in Figure 7b after each of the seven backtracking steps.

Local Pairwise Sequence Alignment

Thus far, we have concentrated on global alignments. Global alignments align the entirety of the first sequence with the entirety of the second sequence. Figure 8a depicts the optimal global alignment for the two homologous *pbx1* sequences shown in Figure 1. More common than computation of global alignments is computation of local alignments. In fact, computation of local alignments is a core component of the popular BLAST tool [15]. Indeed, the name BLAST is an acronym for Basic *Local Alignment* Search Tool (emphasis added). Local alignments allow for the identification of subregions, domains, and motifs that may be similar within two sequences.

Local alignments align a subsequence of one sequence with a subsequence of a second sequence. The local alignment problem is the problem of finding the optimal

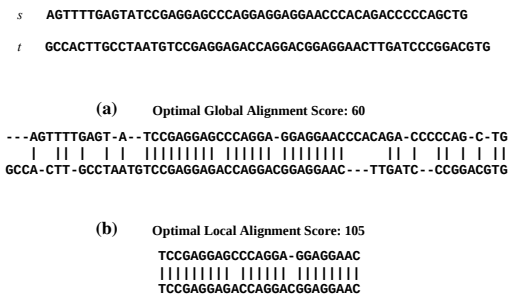

Figure 8. Alignments for two sequences, *s* and *t*, are shown, assuming a match score of +5, a mismatch score of -4, and a gap score of -6. (a) The optimal global alignment for *s* and *t* is shown. The optimal global alignment has a score of 60. (b) The optimal local alignment for *s* and *t* is shown. The optimal local alignment has a score of 105.

scoring alignment out of all possible alignments of any subsequence of one sequence with any subsequence of the other sequence [16]. Intuitively, the local alignment problem attempts to align subregions of sequences rather than complete sequences. Thus, a solution to the local alignment problem can help identify domains of similarity between sequences even if two sequences are not similar in their entirety. Figure 8b depicts the optimal local alignment for the two homologous *pbx1* sequences from Figure 1. The highly conserved portion of *pbx1* shown in the local alignment of Figure 8b corresponds to a DNA binding region of the Pbx1 leukemia transcription factor [17]. Notice that the optimal local alignment score for two sequences should always be greater than or equal to the optimal global alignment score for two sequences since the optimal local alignment score is optimal over all pairs of subsequences, including the sequences in their entirety.

To compute the optimal local alignment score for two sequences, we could enumerate all possible local alignments of the two sequences and determine the alignment or alignments with highest score. However, as in the case of global alignments, as a function of the lengths of the two sequences, there are exponentially many different possible local alignments of two sequences. As a result, enumerating all possible local alignments of two sequences would be computationally intractable for most sequences of interest. Fortunately, like the global alignment problem, the local alignment problem has the two properties that solutions to the problem can be determined from solutions to subproblems and that subproblems are overlapping [16]. Thus, like our approach for solving the global alignment problem, we can solve the local alignment problem by filling in a table with solutions to subproblems using dynamic programming. For the local alignment problem, filling in the table is similar to filling in the table for the global alignment problem. The critical difference is that each table entry is computed as a maximum of four values, as shown in equation (4), as opposed to the three values used in equation (3) for global alignment.

$$T[i][j] = \max\begin{cases} T[i-1][j] + align(s_{i,i}, -) \\ T[i][j-1] + align(-, t_{j,j}) \\ T[i-1][j-1] + align(s_{i,i}, t_{j,j}) \\ 0 \end{cases}. \quad \textbf{(4)}$$

For local alignment, the fourth value of zero in the maximum in equation (4) reflects the fact that optimal local alignments cannot have a negative score. Any pair of subsequences that yield a negative alignment score cannot be optimal because there is always a pair of subsequences with a higher score, a score of zero, namely two empty sequences. When

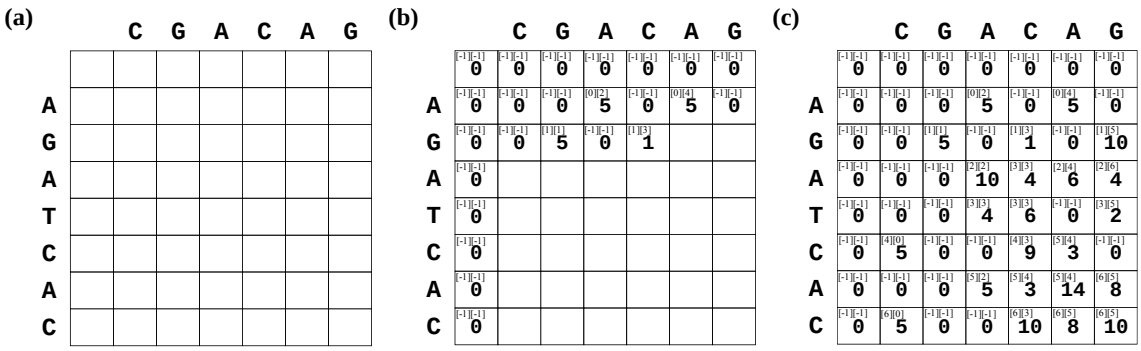

Figure 9. Tables are depicted for the sequences *s*=AGATCAC and *t*=CGACAG. Each entry in a table corresponds to the optimal score for a particular subsequence of *s locally* aligned with a particular subsequence of *t*. For the tables in the figure, a match score of +5, a mismatch score of -4, and a gap score of -6 are assumed. (a) None of the entries in the table are filled in. (b) The table is partially filled in. (c) The table is completely filled in. In the top left corner of each table entry, the indicated row index and column index represent the subproblem from which the optimal score was calculated for that table entry.

filling in an entry in the local alignment table, a lower, and hence suboptimal, score will result from constructing an alignment from a negative scoring local alignment than from constructing an alignment from a zero scoring alignment of two empty subsequences.

Figure 9 shows tables for computing the optimal local alignment score for two example sequences. None of the table entries in Figure 9 are negative. In the local alignment tables in Figure 9, unlike in the global alignment tables in Figure 5, the row and column index of each table entry have not been included in the bottom right corner of the table entry. However, to enable re-constructing an optimal alignment from the table via backtracking, the row and column index in the top left corner of each entry have been included to indicate the subproblem from which the optimal score was calculated for that table entry. A score of zero with index [-1][-1] corresponds to an optimal local alignment of two empty subsequences.

For global alignments, the optimal alignment score was in the bottom rightmost table entry. For local alignments, the optimal alignment score is the maximum value in the table, which could be at any of the entries in the table. In Figure 9, for example, the maximum value in the table and, hence, the optimal local alignment score is 14, which is found at table entry [6][5]. Once the table has been populated and the optimal local alignment score has been determined, by backtracking through the table we can construct a local alignment. To determine an alignment once a table has been filled in, we begin at the table entry with the optimal score and proceed to earlier table entries based on the indices in the top left corners of the table entries until we reach a table entry with a value of 0. Figure 10 illustrates the construction of an alignment based on a filled in table. The growing alignment is shown in Figure 10b after each of the five backtracking steps.

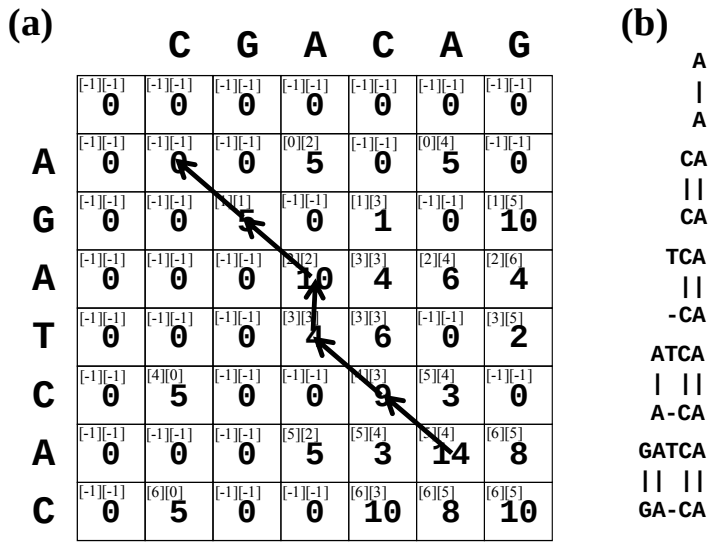

Figure 10. A filled-in local alignment table is depicted for the sequences *s*=AGATCAC and *t*=CGACAG. (a) Arrows in the table indicate a backtracking path from the maximum table entry, *T*[6][5], to an entry with value 0, based on the indices in the top left corner of table entries. Each arrow represents a single backtracking step. (b) With each of the five backtracking steps, a pair of characters is added to the alignment, starting at the end of the alignment and working toward the beginning.

Affine Gap Scoring

As two homologous sequences diverge over time, portions of either sequence may be excised or portions of new sequences may be spliced in. The gaps in our pairwise sequence alignment model are meant to reflect these evolutionary deletion and insertion events. All of the examples above have employed a *linear* gap score, which is to say that all gaps in an alignment have contributed the same amount, e.g., c = -6, to the alignment score. Thus, in the above examples, excising (or splicing) a region of seven nucleotides is seven times more costly than excising (or splicing) a single nucleotide. However, for a cell, the act of excising (or splicing) a portion of DNA may be costly, but the length of the portion excised (or spliced) may be less important to the cell. Ideally, our pairwise sequence alignment model should account for the different costs associated with an excision (or splice) event and the length of the region excised (or spliced). As an alternative to the *linear* gap score, an *affine* gap score allows for different gaps contributing different amounts to the alignment score. A common affine gap scoring model is to have the first gap in a series of consecutive gaps contribute one score, α, and each subsequent gap in the same series of consecutive gaps contributes a different score, β. If gaps represent insertion/deletion events between related biological sequences, an affine gap scoring model is meant to reflect the relatively greater cost of the existence of a gap as compared to the extension of a gap. In affine gap scoring, typically, the existence of a gap is represented by the first character in a series of gaps and is somewhat more costly, α, whereas increasing the length of the gap is somewhat less costly, β. Figure 11 illustrates a sample alignment using both a linear gap scoring model and an affine gap scoring model. Empirical evidence suggests that an affine gap scoring system may better capture the relationships between biological sequences than a linear gap scoring model [4].

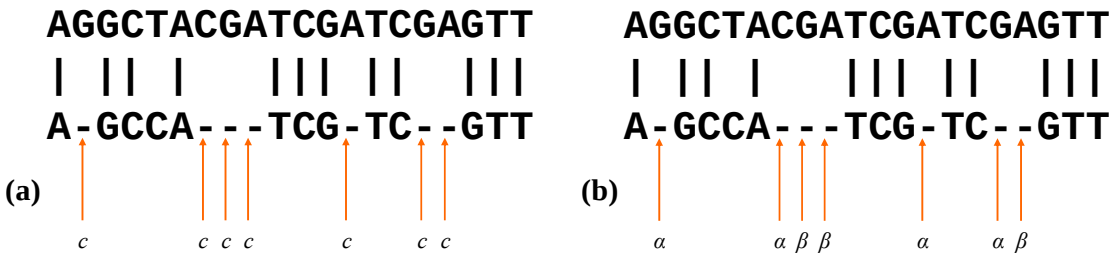

Figure 11. Two identical alignments are shown. (a) Assuming a match score of +5, a mismatch score of -4, and a linear gap score of c=-6, the alignment has a score of 14. (b) Assuming a match score of +5, a mismatch score of -4, and an affine gap scoring model with a gap opening score of α=-7 and a gap extension score of β=-2, the alignment has a score of 22.

Alternative Scoring Models

Just as affine gap scoring offers a more flexible and realistic scoring model than linear gap scoring, so too can we extend our scoring model for pairs of matched and mismatched characters in an alignment. All of the examples above have employed a fixed score for all pairs of aligned matching characters and a fixed score for all pairs of aligned mismatching characters. In the case of DNA sequences composed of the 4 DNA nucleotides, we can construct a scoring matrix of all 16 possible pairs of characters and the alignment score for each pair. As Figure 12a illustrates, using a fixed score for matches and a fixed score for mismatches, the 4 pairs of matching characters, A|A, C|C, G|G, and T|T, all have the same score and the 12 pairs of mismatching characters, A|C, A|G, A|T, C|A, C|G, C|T, G|A, G|C, G|T, T|A, T|C, and T|G, all have the same score. Alternatively, we could assign different scores to different matches or different mismatches, with the aim of better representing various properties of the sequences that we are modeling [18]. For example, for DNA sequences, an adenine aligned with a guanine might contribute a less negative score to an alignment than an adenine aligned with cytosine, under the assumption that replacing one purine with another should penalize an alignment less than replacing a purine with a pyrimidine. Figure 12b illustrates a scoring model where not all pairs of mismatched characters contribute the same score, rather purines mismatched with pyrimidines contribute -4 whereas purines mismatched with other purines contribute -1

Figure 12. Two different scoring matrices are shown reflecting two different sets of possible match and mismatch alignment scores for the 16 pairs of DNA nucleotides. (a) All matches contribute a score of 5. All mismatches contribute a score of -4. (b) All matches contribute a score of 5. A purine mismatched with a pyrimidine contributes a score of -4 whereas a purine mismatched with another purine or a pyrimidine mismatched with another pyrimidine contributes a score of -1.

(a)

	A	C	G	T
A	5	-4	-4	-4
C	-4	5	-4	-4
G	-4	-4	5	-4
T	-4	-4	-4	5

(b)

	A	C	G	T
A	5	-4	-1	-4
C	-4	5	-4	-1
G	-1	-4	5	-4
T	-4	-1	-4	5

and pyrimidines mismatched with other pyrimidines contribute -1. For protein sequences, the cost of replacing one amino acid with another should depend on the properties of the amino acids. A scoring matrix for protein sequences should have 20 rows and 20 columns to reflect the 20 amino acids. BLOSUM [19] and PAM [20] matrices are commonly used as scoring matrices when aligning protein sequences. As a frame of reference, when comparing nucleotide sequences, the BLAST program employs a match score of +1, a mismatch score of -3, and affine gap scoring with a gap opening score of -5 and gap extension score of -2 with one of its default settings [21, 22]. When comparing protein sequences, the BLAST program employs the BLOSUM62 scoring matrix and affine gap scoring with a gap opening score of -11 and gap extension score of -1 with one of its default settings [21, 22].

While many different scoring models are reasonable for aligning pairs of sequences, one desirable property of scoring models is that the expected score of random sequences is negative [23]. Why? Intuitively, two sequences independently generated at random do not have a relationship and should not be deemed similar, i.e., they should have a negative global alignment score. For local alignments, random and unrelated portions of two sequences preferably should not contribute positively toward the optimal alignment score since one of the goals of local pairwise alignment is to distinguish similar portions of two sequences from dissimilar or unrelated portions of two sequences. Practically, if the expected score of random sequences were positive, longer local alignments would be advantaged over shorter local alignments and optimal local alignments would approximate optimal global alignments, rather than distinguish similar subsequences.

Significance of an Optimal Alignment

Thus far, we have investigated how to compute an optimal alignment score and an optimal alignment of two sequences. But we have not broached the topic of whether an optimal alignment is meaningful, i.e., whether an optimal alignment is suggestive that two sequences are similar and likely to have a relationship. If we know that the optimal alignment score for two sequences is 60, can we say with confidence that the two sequences are similar? What if the two sequences have an optimal alignment score of 600 or 6000? The answer, of course, is “it depends”. Whether an alignment score is significant depends on such factors as the length of the sequences, the distribution of characters in the sequences, and the scoring model. For example, for two sequences of 100 characters, an alignment score of 490 might be significant if matched pairs of characters received a score of +1 and mismatched pairs of characters received a score of -1. However, an alignment score of 490 might not be significant for the same two sequences if matched pairs of characters received a score of 200 and mismatched pairs of characters received a score of -5.

In order to assess the significance of an alignment score, often randomized data is used to estimate how likely it is that such an alignment score would occur merely by chance [23]. The underlying assumption here is that the optimal alignment score for two similar or related sequences should have a higher probability of being larger than the optimal alignment score for two random or unrelated sequences. The biological motivation for this assumption is that random mutations are more likely to be deleterious

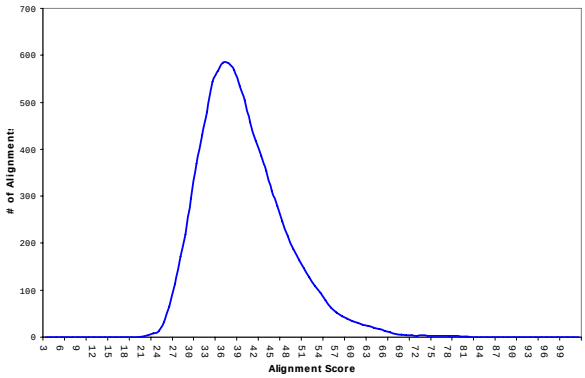

Figure 13. A histogram of 10,000 optimal alignment scores for 10,000 pairs of randomly generated sequences. Each sequence in each pair of randomly generated sequences is 50 characters in length and has an expected GC content of 50%. Of the 10,000 alignment scores, 631 or 6.31% have a score of 60 or higher.

to related pairs of sequences than to random or unrelated pairs of sequences and, thus, less likely to be preserved owing to selective evolutionary pressure [24].

One computational approach that uses randomized data to estimate the significance of an alignment score of two sequences, *s* and *t*, proceeds as follows. A large number, say 10,000, of pairs of random sequences are generated. Each pair of random sequences should be similar to the original pair of sequences, *s* and *t*, in that the two sequences in the random pair should have the same lengths as *s* and *t*. Further, the distribution of characters in each random pair should approximate the distribution of characters in *s* and *t*. For instance, if *s* has a GC content of 46% then so too should one of the sequences in the random pair. A random sequence with such a distribution of characters might be generated by randomly permuting the order of characters in *s* or by generating a random sequence *de novo* with the expected distribution of characters. Then, the optimal alignment score, either global or local depending on the application, can be computed for each pair of random sequences. Figure 13 shows a histogram of 10,000 optimal alignment scores for 10,000 pairs of randomly generated sequences. As can be seen from the histogram, almost all of the 10,000 alignment scores fall between 20 and 80. 6.31% of the 10,000 alignments achieved a score of 60 or greater. Thus, the original sequences, *s* and *t*, had approximately a 6% chance of achieving an optimal alignment score of 60 or greater merely be chance. The *p-value* of an optimal local alignment score, *S*, for two sequences is the likelihood that two random sequences, of the same lengths and compositions as the original sequences, would have an optimal local alignment score greater than or equal to *S*. Thus, if the optimal local alignment score for *s* and *t* is S=60, then the *p*-value for this alignment score is approximately p=0.06. A larger *p*-value suggests that an alignment score has a higher likelihood of occurring merely by chance. A smaller *p*-value suggests that an alignment score has a lower likelihood of occurring merely by chance. Smaller *p*-values generally suggest higher significance, i.e., that an alignment score is less likely to have occurred by chance and more likely to have occurred as a result of some other factor such as selective evolutionary pressure.

Generally, *p*-values rather than alignment scores are used to assess the statistical significance of an alignment of two sequences. Alignment scores depend on the lengths of the sequences being aligned and the distribution of characters in the sequences. An alignment score of 60 may be significant for two sequences of length 30 characters but not for two sequences of length 100 characters. In Figure 13, the histogram would shift to the left for two sequences of length 30 characters and to the right for two sequences of 100 characters, assuming the same scoring model. Thus, without knowing such details

about the sequences, the alignment score may be meaningless. Similarly, an alignment score of 60 may be significant for one scoring model but not for another. *p*-values have the advantage over alignment scores that they account for these different sequence properties and scoring models. While there is no firm threshold for *p*-values, often as a guide researchers consider *p*-values less than 0.01 to be significant.

Conclusions

Determining the similarity of two sequences is a fundamental problem in genomics. The similarity of a pair of sequences may suggest a relationship between the sequences. Often, when a scientist determines the nucleotide sequence of a previously uncharacterized genomic region or the amino acid sequence of a previously uncharacterized protein, the next step is comparison of the sequence to other sequences in order to identify possibly related elements. Pairwise alignment indicates the similarity of two sequences. As described in this article, the problem of computing the optimal pairwise alignment for two sequences can be solved efficiently. Calculation of both the optimal global alignment and the optimal local alignment of a pair of sequences was illustrated. Though the examples in this article focused primarily on nucleotide sequences, the same approaches are directly applicable to protein sequences.

While this article deals specifically with the pairwise alignment problem, there are a number of broader themes throughout the article that apply to other bioinformatics problems. We conclude with explicit statement of some of the more important of these themes.

- Experimental approaches often validate or refute hypotheses. Computational approaches often suggest hypotheses and, in some cases, indicate levels of confidence in the hypotheses. Computing the optimal pairwise alignment for two sequences, for example, may indicate the similarity of the sequences, which can suggest a relationship between the sequences, i.e., a hypothesis. Experimental and computational approaches are not competing approaches for addressing the same problem but rather are complimentary approaches in the application of the scientific method.
- When computational approaches generate hypotheses, it is often useful if the probabilities that the generated hypotheses are correct can be estimated. For many problems, randomly generated data can be used to estimate the significance of an observation, i.e., how likely the observation is to have occurred merely by chance.
- Some problems cannot be solved computationally. Some problems can be solved computationally, but not efficiently. Some problems can be solved efficiently computationally. It is not always obvious which of the abovementioned classes a problem may fall into. Identifying various properties of a problem can be the difference between finding a solution to the problem or not, and identifying such properties often requires algorithmic insights.
- Some problems are solved exactly or optimally. Other problems are addressed using a *heuristic* approach. Heuristic methods for solving a problem may not solve a problem exactly or optimally, but they can generate good possibly suboptimal solutions, and often they can do so very efficiently. The

abovementioned solution to the pairwise alignment problem is not heuristic, it provably generates an optimal solution. BLAST is a heuristic tool in that, for the purposes of efficiency, it does not align a query sequence to *every* target sequence in a large database, but rather aligns a query sequence to a subset of target sequences in the database. Heuristically, BLAST identifies a subset of target sequences in the database that are likely to be similar to the query sequence. Heuristic approaches often offer a trade-off between sensitivity and efficiency.

- Many computational approaches are improved by incorporating additional biological insights into their underlying method or model. In the case of the pairwise alignment problem, an alignment is more meaningful if the scoring matrix or model is well-founded biologically.
- Recent advances have enabled scientists to gather large amounts of, often heterogeneous, data. One of the roles of many bioinformatics tools is efficient analysis of large data sets with the aim of extracting new biological insights from the wealth of data.

References

1. Chang, C.P., W.F. Shen, S. Rozenfeld, H.J. Lawrence, C. Largman, and M.L. Cleary, (1995). *Pbx proteins display hexapeptide-dependent cooperative DNA binding with a subset of Hox proteins.* Genes and Development, **9**: p. 663-674.
2. Lam, S.H., Y.L. Wu, V.B. Vega, L.D. Miller, J. Spitsbergen, Y. Tong, H. Zhan, K.R. Govindarajan, S. Lee, S. Mathavan, K.R.K. Murthy, D.R. Buhler, E.T. Liu, and Z. Gong, (2006). *Conservation of gene expression signatures between zebrafish and human liver tumors and tumor progression.* Nature Biotechnology, **24**(1): p. 73-75.
3. Thomas, J.W., J.W. Touchman, R.W. Blakesley, G.G. Bouffard, S.M. Beckstrom-Sternberg, E.H. Margulies, M. Blanchette, A.C. Siepel, P.J. Thomas, J.C. McDowell, B. Maskeri, N.F. Hansen, M.S. Schwartz, R.J. Weber, W.J. Kent, D. Karolchik, T.C. Bruen, R. Bevan, D.J. Cutler, S. Schwartz, L. Elnitski, J.R. Idol, A.B. Prasad, S.Q. Lee-Lin, V.V.B. Maduro, T.J. Summers, M.E. Portnoy, N.L. Dietrich, N. Akhter, K. Ayele, B. Benjamin, K. Cariaga, C.P. Brinkley, S.Y. Brooks, S. Granite, X. Guan, J. Gupta, P. Haghighi, S.L. Ho, M.C. Huang, E. Karlins, P.L. Laric, R. Legaspi, M.J. Lim, Q.L. Maduro, C.A. Masiello, S.D. Mastrian, J.C. McCloskey, R. Pearson, S. Stantripop, E.E. Tiongson, J.T. Tran, C. Tsurgeon, J.L. Vogt, M.A. Walker, K.D. Wetherby, L.S. Wiggins, A.C. Young, L.H. Zhang, K. Osoegawa, B. Zhu, B. Zhao, C.L. Shu, P.J. De Jong, C.E. Lawrence, A.F. Smit, A. Chakravarti, D. Haussler, P. Green, W. Miller, and E.D. Green, (2003). *Comparative analyses of multi-species sequences from targeted genomic regions.* Nature, **424**(6950): p. 788-793.
4. Vingron, M. and M.S. Waterman, (1994). *Sequence alignment and penalty choice: review of concepts, case studies and implications.* Journal of Molecular Biology, **235**(1): p. 1-12.
5. Gibbs, A.A. and G.A. McIntyre, (1970). *The diagram: a method for comparing sequences. Its use with amino acid and nucleotide sequences.* European Journal of Biochemistry, **16**: p. 1-11.
6. Turing, A., (1936). *On computable numbers, with an application to the Entscheidungsproblem.* Proceedings of the London Mathematical Society, **2**(42): p. 230-265.
7. Cook, S.A., (1971). *The complexity of theorem-proving procedures.* Proceedings of the third annual ACM Symposium on Theory of Computing: p. 151-158.
8. Wang, L. and T. Jiang, (1994). *On the complexity of multiple sequence alignment.* Journal of Computational Biology, **1**(4): p. 337-348.
9. Lawrence, C.E. and A.A. Reilly, (1990). *An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences.* Proteins, **7**(1): p. 41-51.
10. Eisen, M.B., P.T. Spellman, P.O. Brown, and D. Botstein, (1998). *Cluster analysis and display of genome-wide expression patterns.* Proceedings of the National Academy of Sciences of the United States of America, **95**(25): p. 14863-14868.
11. Bellman, R., (1952). *On the theory of dynamic programming.* Proceedings of the National Academy of Sciences of the United States of America, **38**: p. 716-719.

12. Needleman, S.B. and C.D. Wunsch, (1970). *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* Journal of Molecular Biology, **48**(3): p. 443-453.
13. Nussinov, R. and A.B. Jacobson, (1980). *Fast algorithm for predicting the secondary structure of single-stranded RNA.* Proceedings of the National Academy of Sciences of the United States of America, **77**(11): p. 6309-6313.
14. Zuker, M. and P. Stiegler, (1981). *Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information.* Nucleic Acids Research, **9**(1): p. 133-148.
15. Altschul, S.F., W. Gish, W. Miller, E.W. Myers, and D.J. Lipman, (1990). *Basic local alignment search tool.* Journal of Molecular Biology, **215**: p. 403-410.
16. Smith, T.F. and M.S. Waterman, (1981). *Identification of common molecular subsequences.* Journal of Molecular Biology, **147**: p. 195-197.
17. Piper, D.E., A.H. Batchelor, C.P. Chang, M.L. Cleary, and C. Wolberger, (1999). *Structure of a HoxB1-Pbx1 heterodimer bound to DNA: role of the hexapeptide and a fourth homeodomain helix in complex formation.* Cell, **96**: p. 587-597.
18. Altschul, S.F., (1991). *Amino acid substitution matrices from an information theoretic perspective.* Journal of Molecular Biology, **219**: p. 555-565.
19. Henikoff, S. and J.G. Henikoff, (1992). *Amino acid substitution matrices from protein blocks.* Proceedings of the National Academy of Sciences of the United States of America, **89**: p. 10915-10919.
20. Dayhoff, M.O., ed. *A model of evolutionary change in proteins*. Atlas of Protein Sequence and Structure, ed. M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. Vol. 5. 1978, Natl. Biomed. Res. Found.: Washington, D.C.
21. States, D.J., W. Gish, and S.F. Altschul, (1991). *Improved sensitivity of nucleic acid database searches using application-specific scoring matrices.* Methods, **3**(66-70).
22. http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml#other_advanced.
23. Karlin, S. and S.F. Altschul, (1990). *Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes.* Proceedings of the National Academy of Sciences of the United States of America, **87**: p. 2264-2268.
24. Frazer, K.A., L. Elnitski, D.M. Church, I. Dubchak, and R.C. Hardison, (2003). *Cross-species sequence comparisons: a review of methods and available resources.* Genome Research, **13**: p. 1-12.