CHAPTER 1

Sequence Alignment

Concepts and History

MICHAEL S. ROSENBERG
Arizona State University

Sequence alignment is a fundamental procedure (implicitly or explicitly) conducted in any biological study that compares two or more biological sequences (whether DNA, RNA, or protein). It is the procedure by which one attempts to infer which positions (sites) within sequences are homologous, that is, which sites share a common evolutionary history (see the section "Homology" in this chapter for more detail). For the majority of scientists, alignment is a task whose automated solution was solved years ago; the alignment is of little direct interest but is rather a necessary step that allows one to study deeper questions, such as the identification and quantification of conserved regions or functional motifs (Kirkness et al. 2003; Thomas et al. 2003), profiling of genetic disease (Miller and Kumar 2001; Miller et al. 2003), phylogenetic analysis (Felsenstein 2004), and ancestral sequence profiling

and prediction (Cai et al. 2004; Hall 2006). For other scientists, alignment is an active area of research, where basic questions on how one should construct and evaluate an alignment are under heavy scrutiny and debate. Because alignment is the first step in many complex, high-throughput studies (Lecompte et al. 2001), it is important to remember that alignment algorithms produce a hypothesis of homology (just as a phylogenetic tree is a hypothesis of evolutionary history). Like other hypotheses, these alignments may contain more or less error depending on the nature of the data, some of which may have huge downstream effects on other analyses (Kumar and Filipski 2007; Ogden and Rosenberg 2006; Rosenberg 2005a, b).

In a casual survey, most researchers guess that two or three dozen alignment programs and algorithms have been published. The true number is actually in the hundreds, with the numbers increasing each year. Figure 1.1 shows a summary of the number of named alignment programs released over the 20-year period from 1986 to 2005 (alignment algorithms go back to 1970, but prior to the mid-1980s and the advent of the personal computer, most were simply published as logical descriptions or as source code rather than as compiled executables). Just counting named programs and not papers describing algorithmic

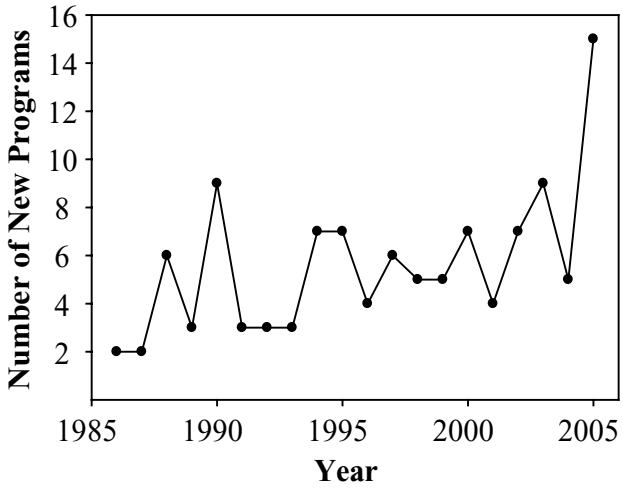

Figure 1.1. Number of new named alignment programs released each year from 1986 to 2005.

advances or unnamed code or software, there has been an average of over five programs released per year, with a generally increasing trend through time.

This chapter provides a brief historical overview of sequence alignment with descriptions of the common basic algorithms, methods, and approaches that underlie most of the way sequence alignments are performed today. More thorough treatments of many of these topics are discussed throughout the text.

PAIRWISE ALIGNMENT AND DYNAMIC PROGRAMMING

To be able to compare potential sequence alignments, one needs to be able to determine a value (or score) that estimates the quality of each alignment. The formulas behind an alignment score are generally known as *objective functions*; they range from simple cost-benefit sums to complex maximum-likelihood values. This introduction will use a simple cost-benefit approach, but much more advanced scoring algorithms and mechanisms are available, many of which are discussed in detail throughout this book.

When using a cost-benefit approach to evaluating a pairwise alignment, one must specify scores for the various ways in which a pair of sites can be compared. In the simplest case, three scores are specified: (1) the benefit of aligning a pair of sites that contain the same character (state) in both sequences; (2) the cost of aligning a pair of sites that contain different characters in the sequences; and (3) the cost of aligning a character in one sequence with a gap in the other sequence. Depending on how one defines the scores, the eventual goal could be to find the alignment that maximizes the benefit or to find the alignment that minimizes the cost. This is essentially an arbitrary choice based on how one chooses to define the costs and benefits. Many of the popular alignment programs in use today (e.g., ClustalW) find the maximum score, a convention that will be retained throughout this description. In computer science, one simple, cost-based scoring function is the *edit distance*, that is, the minimum number of changes necessary to convert one sequence into another. In this case, the goal of alignment is to minimize the edit distance.

Given a scoring function, one can compare any set of alignments for the same set of initial sequences. The one with the best score would be considered the best alignment. For example, let us set the benefit of a match to +1, the cost of a mismatch to −3, and the cost of aligning

```
ACCTGATCCG      ACCTGATCCG
||  |||||  |    ||    ||  |
AC-TGATCAG      ACTGA-TCAG
S=8-4-3=1       S=5-4-12=-11
```

Figure 1.2. Alternate alignments of a pair of sequences illustrating a simple scoring function with matches = +1, mismatches = –3, and gaps = –4. The alignment on the left is better than the alignment on the right because its overall score is larger (1 vs. –11).

a character to a gap to –4. Figure 1.2 shows two possible alignments of sequences ACCTGATCCG and ACTGATCAG. In the first potential alignment, there are 8 matches ($8 \times +1 = +8$), one mismatch ($1 \times -3 = -3$), and one site aligned with a gap ($1 \times -4 = -4$), for a total score of +1. In the second potential alignment, there are 5 matches (+5), 4 mismatches (–12), and one site aligned with a gap (–4), for a total score of –11. The first alignment has a higher score than the second and would be considered a better alignment. But how do we know that it is the best possible alignment?

It is impossible to evaluate all possible alignments. Take the simple case where a sequence of 100 characters is being aligned with a sequence of 95 characters. If all we do is add 5 gaps to the second sequence (to bring it to 100 total sites), there are approximately 55 million possible alignments (Krane and Raymer 2003). Because we may need to add gaps to both sequences, the actual number of possible alignments is significantly greater.

The number of potential alignments made automated procedures for aligning sequences a critical aspect of molecular sequence comparison (but see Chapter 7 for a discussion of why one should not rely solely on automated methods). The first attempts at developing a computational method for the alignment of sequences were undertaken in the mid-1960s with studies such as those of Fitch (1966) and Needleman and Blair (1969), but it was not until 1970 that the first elegant solution to the alignment problem was produced (Needleman and Wunsch 1970). It is this solution, using dynamic programming, that has made their procedure the grandfather of all alignment algorithms.

Dynamic programming is a computational approach to problem solving that essentially works the problem backwards. Dynamic programming is best illustrated with a simple mathematical example,

say calculating the *n*th value of a Fibonacci sequence. The Fibonacci sequence is a series of numbers in which each value is equal to the sum of the two values preceding it, $F_n = F_{n-1} + F_{n-2}$ (by definition, the first two values of the sequence must be specified). Thus, to calculate the 10th value of the Fibonacci sequence, one needs to know the 8th and 9th values; to calculate the 9th value, one needs to know the 7th and 8th values; to calculate the 8th value one needs to know the 6th and 7th values; and so on. Although it is not particularly difficult to solve this problem with a straightforward recursive algorithm, finding the *n*th value by a recursive approach is very inefficient. Note that if each value is determined independently, one needs to compute the same value more than one time (e.g., calculation of the 9th and 8th values both require calculating the 7th value). Using a standard recursive algorithm, determination of the 10th value of the Fibonacci sequence would require 109 steps (Figure 1.3A); the formal complexity of the recursive approach to the Fibonacci sequence is exponential. A dynamic programming approach, on the other hand, is simpler and more efficient. It works the problem in the opposite direction from the recursive approach by starting with the first value in the sequence rather than the *n*th value. In the Fibonacci sequence, the first two values must be predefined; for this example we will set the first value to 0 and the second value to 1. Given the first two values, we can easily determine the 3rd value, 0 + 1 = 1. The 4th value is the sum of the already determined 2nd and 3rd values and therefore is 1 + 1 = 2. At this point it is trivial to keep moving forward until one reaches the 10th value (5th value = 1 + 2 = 3; 6th value = 2 + 3 = 5; 7th value = 3 + 5 = 8; 8th value = 5 + 8 = 13; 9th value = 8 + 13 = 21, and 10th value = 13 + 21 = 34). By avoiding the redundant determination of the lower values, the dynamic programming approach takes only 10 steps. That is, the complexity is linear, requiring only *n* steps (Figure 1.3B).

Pairwise sequence alignment is more complicated than calculating the Fibonacci sequence, but the same principle is involved. The alignment score for a pair of sequences can be determined recursively by breaking the problem into the combination of single sites at the end of the sequences and their optimally aligned subsequences (Eddy 2004). If sequences x and y have m and n sites, respectively, the last position of their alignment can have three possibilities: x_m and y_n are aligned, x_m is aligned with a gap with y_n somewhere upstream, or y_n is aligned with a gap with x_m somewhere upstream. The alignment score for each case is the score for that final position plus the score of the optimal alignment

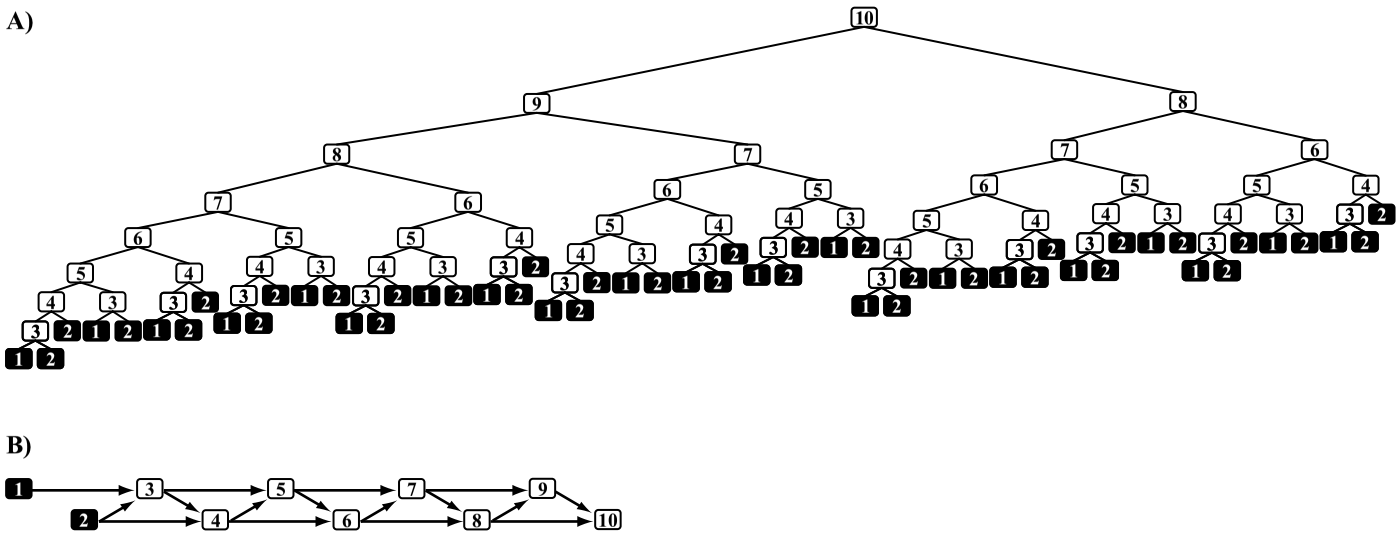

Figure 1.3. Calculating the 10th value of the Fibonacci sequence. The numbers in the cells represent the *n*th number in the sequence. The actual value of the *n*th number is equal to the sum of the previous two values ($f_n = f_{n-1} + f_{n-2}$). The first two values of the sequence are predefined (represented by black cells). (A) The top-down recursive approach has duplication of effort since many values have to be determined multiple times (*e.g.*, the 3rd value is determined 21 times). It requires 109 steps, including looking up the predefined values 55 times. (B) The bottom-up dynamic programming approach starts with the first two values and works forward to the desired 10th value. There is no duplication of effort, and the 10th value can be determined in just 10 steps.

of the upstream subsequence (each site is scored independently, so the score of nonoverlapping alignment segments can be added for a total score). The overall optimal score is the maximum of the score for the three cases. Critically, the optimal score for each of the aligned subsequences can be determined the same way: three possible cases for the final position plus the optimal alignment of the upstream subsequence for each case. Thus, a formula for the optimal alignment score can be written in a simple recursive format:

$$S(i, j) = \max \begin{vmatrix} S(i-1, j-1) + \sigma(x_i, y_j) \\ S(i-1, j) + \gamma \\ S(i, j-1) + \gamma \end{vmatrix},$$

where $S(i, j)$ is the score for the optimal alignment from position 1 to i in sequence x and 1 to j in sequence y, γ is the gap cost, and $\sigma(x_i, y_j)$ is the mismatch/match score for the pair of states at positions x_i and y_j (Eddy 2004).

Although simple to write, this formula is very inefficient to solve and, as for the Fibonacci sequence, leads to redundant determination of the optimal score for subsequence alignments that will show up over and over again. The dynamic programming solution works by starting with the optimal alignment of the smallest possible subsequences (nothing in sequence x aligned to nothing in sequence y) and progressively determining the optimal score for longer and longer sequences by adding sites one at a time. By keeping track of the optimal score for each possible aligned subsequence in a matrix, the optimal score and alignment of the full sequences can easily and efficiently be determined.

This method is illustrated in Figure 1.4 with the alignment of a pair of short sequences: ATG and GGAATGG, using a match score of +1, a mismatch score of –1, and a gap score of –2 (example adapted from Lesk 2002). The first step is to construct a matrix that contains each sequence along an axis, with an extra empty row and column at the top and left sides of the matrix. Each cell in the matrix will be filled with the maximum of three possible values: (1) the sum of the score of the cell that is diagonally to its upper left and the match or mismatch score (depending on whether the characters in the row/column of the cell match or mismatch); (2) the sum of the score of the cell that is directly above the cell and the gap score; or (3) the sum of the score of the cell directly to the left of the cell and the gap score. (There is essentially a hidden rule that allows one to skip any rule which does not apply, for

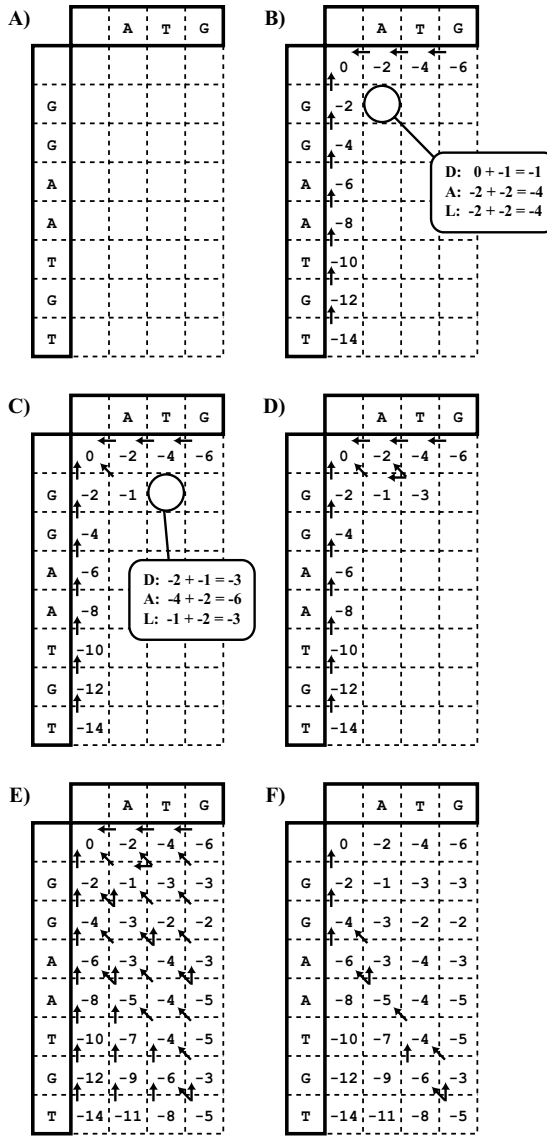

Figure 1.4. Illustration of Needleman–Wunsch (1970) global alignment algorithm. (A) Setting up the matrix. (B) The first row and column are filled with increasing multiples of the gap cost. The first cell will be given the maximum of three possible values. (C) The value for the first cell is entered along with the path that led to the value. The possible values for the second cell are illustrated. (D) The value for the second cell is entered; multiple paths are recorded since multiple paths led to the maximum score. (E) The completed matrix. (F) The completed matrix with all suboptimal paths removed. Tracing the arrows from the bottom right corner to the upper left leads to four possible paths and (therefore) four equally optimal alignments.

example, the cells in the top row do not have any cells above them, so rule 2 cannot apply). The practical upshot of these rules is that the first step is to place a zero in the upper left corner of the matrix and then to fill the first row and column with increasing multiples of the gap cost. The rest of the cells in the matrix are filled in one by one. An important point is that one needs to keep track of which rule was applied, that is, which neighboring cell (diagonal, above, or left) led to the value that was filled in. Computationally this is usually stored in a second matrix called a trace-back matrix; in Figure 1.4 this is illustrated with arrows for simplicity. A second important point is that it is possible for multiple rules to produce the identical maximum value; in such a case the trace-back matrix should properly include *all* possible paths to the maximum value.

In our example, the first cell (A vs. G) can have three values: (1) the score of the cell to the upper left, 0, plus the mismatch cost (A/G is a mismatch), −1 = −1; (2) the score of the cell above it, −2, plus the gap score, −2, = −4; or (3) the score of the cell to the left, −2, plus the gap score, −2, = −4. The maximum of these is the first score, −1, which is entered in the cell, along with an arrow to the upper left to remind us that it is that path from which the score was derived. We repeat for the next cell (to the right). The possible values for this cell are −2 + −1 = −3, −4 + −2 = −6, or −1 + −2 = −3. The maximum value is −3, but this can be achieved from two separate paths, so we record both paths using our trace-back arrows.

This procedure is repeated until the entire matrix is filled with values. The value in the last cell, the lower right (−5 in our example), represents the score of the best alignment (given the score function). To find this alignment, one starts with this cell and follows the arrows back to the upper left corner of the matrix. Following a diagonal arrow indicates that the sites represented by that row and column of the matrix should be aligned. Following a vertical arrow indicates that the character in the sequence along the vertical axis (the character represented by the row of the matrix) should be aligned with a gap in the sequence represented by the horizontal axis. Following a horizontal arrow indicates that the character in the sequence along the horizontal axis (the character represented by the column of the matrix) should be aligned with a gap in the sequence represented by the vertical axis.

If there is more than one possible path back to the top of the matrix, this indicates that multiple pairwise alignments lead to the identical score and are equally optimal. In our example, there are four possible

```
GGAATGG    GGAATGG    GGAATGG    GGAATGG
---ATG-    ---AT-G    --A-TG-    --A-T-G
```

Figure 1.5. Four equally optimal global alignments of sequences GGAATGG and ATG derived from the alignment matrix shown in Figure 1.2.

paths, leading to four possible alignments of these sequences, shown in Figure 1.5.

In this specific case, one might argue that the first of these appears to be a subjectively better alignment than the others, but, based solely on the specified cost function, all four of these alignments are equally good (each of these alignments contains three matches and four gaps). Changing the cost function may change the result (see below). Very few alignment programs produce more than one alignment, even if there are multiple equally optimal alignments; the sim algorithms of Huang et al. (1990) and Huang and Miller (1991) are a notable exception. How the single resultant alignment produced by most programs is chosen from the universe of possible optimal alignments is usually not clear.

This is the simplest approach to pairwise sequence alignment. Obvious enhancements include the use of more complicated scoring functions. Not all mismatches are necessarily equal, and different types of mismatches could be given different scores depending on the properties of the characters. For DNA sequences, these differential scores might be based on standard models of sequence evolution. For example, it is well known that transitional substitutions occur more often than transversional substitutions, therefore a transversional mismatch might be given a higher cost than a transitional mismatch. For protein sequences, empirically derived substitution matrices are usually used to determine relative costs of various mismatches; these matrices include the PAM (Dayhoff et al. 1978), JTT (Jones et al. 1992), and BLOSUM (Henikoff and Henikoff 1992) matrices. These matrices usually include estimated biological factors such as the conservation, frequency, and evolutionary patterns of individual amino acids. In principle, one could imagine giving different benefits to different matches (e.g., perhaps a larger benefit should come from aligning a pair of cysteine residues because of the extreme conservation and structural constraints of this amino acid).

Another enhancement to the scoring function has to do with the gap costs. As described above, all gaps are treated as identical single

position events. Biologically, we recognize that single insertion-deletion events may (and often do) cover multiple sites. We therefore may not want the cost of a gap that covers three sites to be triple the cost of a gap that covers only one site (a linear gap cost). The general solution is to use one cost score for opening (starting) a gap and a second score for extending (lengthening) a gap (an affine gap cost) (Altschul and Erickson 1986; Gotoh 1982, 1986; Taylor 1984). In this case the total cost of the gap is $O + nE$ where O is the gap opening cost, E is the gap extension cost, and n is the length of the gap (or the length of the extension, depending on how the algorithm is defined). Much more complicated schemes for scoring gaps of varying lengths are possible, although the majority of modern alignment programs appear to use some form of an affine cost structure. Some algorithms will also vary in how they treat terminal gaps (that is, gaps that occur at the very beginning or ends of a sequence); some algorithms will give these reduced cost (even zero) since they are not inferred to occur between observed characters (this is sometimes known as semi-global alignment).

In our previous example, the use of an affine gap cost would eliminate the third and fourth alignments in Figure 1.5 from the optimal set, since these each have three putative independent gaps (of length 2, 1, and 1) while the first two alignments have only two gaps (of length 3 and 1). Lowering the cost of terminal gaps would then eliminate the second alignment, since one of its gaps is internal to the observed characters, while the first alignment contains only terminal gaps.

Beyond changes in how alignments are scored, there have been numerous improvements in efficiency of the basic dynamic programming approach described above, including, for example, decreasing memory requirements (Myers and Miller 1988) and the number of computational steps (Gotoh 1982).

GLOBAL ALIGNMENT VS. LOCAL ALIGNMENT

The procedure described thus far is a global alignment algorithm; that is, it assumes that the entirety of the sequences are sequentially homologous and tries to align all of the sites optimally within the sequences. This assumption may be incorrect as a result of large-scale sequence rearrangement and genome shuffling. In such a case, only subsections of the sequences may be homologous, or the homologous sections may be in a different order. For example, a long sequence may be ordered ABCDEF (where each letter represents a section of sequence and not an

```
AB--CDEF      ABCDEF      ABCDE--F
ABEDC--F      ABEDCF      AB--EDCF
```

Figure 1.6. Illustration of global alignment problem. Sequences ABCDEF and ABEDCF cannot be properly aligned because the homologous sections of the sequences are not in the same order.

individual site). A sequence inversion of section CDE may change the sequence in another species to ABEDCF. Although each section of the first sequence is homologous with a section of the second sequence, they cannot be globally aligned, because of the rearrangement. Figure 1.6 shows possible global alignments if section C, D, or E is aligned. In each case, the other two sections cannot be aligned properly.

An alternative approach to global alignment is local alignment. In a local alignment, subsections of the sequences are aligned without reference to global patterns. This allows the algorithm to align regions separately regardless of overall order within the sequence and to align similar regions while allowing highly divergent regions to remain unaligned.

Early approaches for local alignment were developed by Sankoff (1972) and Sellers (1979, 1980), but the basic local alignment procedure most widely used was proposed by Smith and Waterman (1981b). It is a simple adaptation to the standard Needleman–Wunsch algorithm. The first difference is in the determination of values for a cell. In addition to the three possible values described by the Needleman–Wunsch algorithm, the local alignment algorithm allows for a fourth possible value: zero. This prevents the alignment score from ever becoming negative; if this rule is invoked, no trace-back arrow is stored for the cell. Figure 1.7 shows the score matrix for a local alignment of the same sequences that were globally aligned in Figure 1.4. The addition of the fourth rule substantially changes the structure of the scores and the trace-back arrows.

Once the matrix has been filled, the second change in the local alignment algorithm is that rather than starting in the lower right corner of the matrix, one uses the cell with the largest value as the starting position for the trace-back. In our example, this is the cell directly above the lower right cell, with a score of 3. The final difference in this method is that the trace-back does not continue to the upper-left corner, but rather terminates when the trace-back arrows end. The result of this procedure is that only a portion of the sequences may be aligned; the remainder

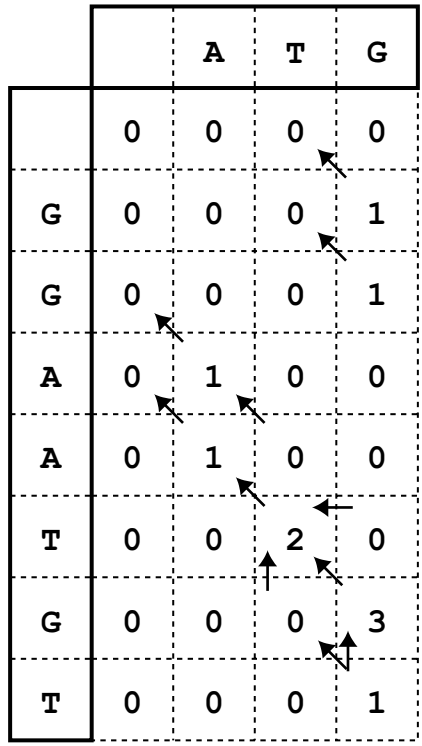

Figure 1.7. Completed score and trace-back matrix for local alignment using the Smith and Waterman (1981b) algorithm.

of the sequences are left as unaligned. For this example, the local alignment is simply that shown in Figure 1.8.

Only the aligned parts of the sequences are reported. Of course, additional local alignments of the sequences could be found if there are multiple cells with the same maximal score or by choosing submaximal starting points. As with global alignment, there have been major advances in approaches for local alignment; major local alignment programs and algorithms in use today include DIALIGN (Morgenstern 1999; Morgenstern, Frech, et al. 1998) and CHAOS (Brudno, Chapman, et al. 2003; Brudno and Morgenstern 2002).

With the recent sequencing revolution, one bioinformatic challenge has been the comparison of full genome sequences. Because genomes are so readily rearranged, alignment of entire genomes is a specialized

```
ATG
ATG
```

Figure 1.8. The local alignment of sequences GGAATGG and ATG derived from the alignment matrix shown in Figure 1.2.

case of local alignment applied on a very large scale. Many specialized programs for producing local alignments of entire genomes have been produced recently, include BlastZ (Schwartz et al. 2003), MUMMER (Delcher et al. 1999; Delcher et al. 2002), GLASS (Batzoglou et al. 2000), WABA (Kent and Zahler 2000), MAUVE (Darling et al. 2004), GRAT (Kindlund et al. 2007), MAP2 (Ye and Huang 2005), and AuberGene (Szklarczyk and Heringa 2006).

Local Alignment vs. Database Searching

Much of the work on local alignment has focused on database searching rather than simple sequence comparison. It was recognized very early on that algorithms would be necessary to retrieve sequences from a database with a pattern similar to that of a query sequence (e.g., Korn et al. 1977). Comparing sequences for similar patterns requires, in some form, local alignment, and local alignment methods form the basis of all database searching algorithms. The most famous sequence search algorithm, BLAST (Altschul et al. 1990), contains this phrase in its name: Basic *Local Alignment* Search Tool. Although not discussed in any detail within this book, must of the major work on local alignment derives from interests in database searching, particularly in the development of both the BLAST and FASTx (Lipman and Pearson 1985; Pearson and Lipman 1988) families of algorithms.

IMPORTANCE OF THE COST FUNCTION

The algorithms described above allow one to search the population of possible alignments efficiently for the most optimal alignment(s), but it is the cost function that is most important in determining the actual best alignment. As one would expect, changing the cost function may change which alignment is considered to be most optimal. Take for example, the pair of sequences CAGCCTCGCTTAG and AATGCCATTGACGG. If we perform global and local alignments of these sequences using the parameters

A)

```
CA-GCC-TCGCTTAG  CA-GCC-TCGCTTAG  CA-GCC-TCGCTTAG  CA-GCC-TCGCTTAG
AATGCCATTGACG-G  AATGCCATTGAC-GG  AATGCCATTGA-CGG  AATGCCATTG-ACGG
```

B)

```
GCC
GCC
```

Figure 1.9. Optimal alignments of sequences CAGCCTCGCTTAG and AATGCCATTGACGG with a cost function with matches = +1, mismatches = –1, and gaps = –2. (A) Four equally optimal global alignments. (B) The single optimal local alignment.

as before (+1 match, –1 mismatch, –2 gap), we find four equally optimal global alignments, shown in Figure 1.9A (the only difference is the position of the gap in the second sequence), and one local alignment (Figure 1.9B).

If we change the parameters of our cost function so that the match benefit is still +1, but the mismatch cost is now –0.3 and the gap cost is –1.3, we would find the same four global alignments, but our local alignment would change to that shown in Figure 1.10. Different sets of scoring values may lead to different optimal alignments; the best alignment is not only dependent on the algorithm (global vs. local) but on parameter choices.

How does one determine what values should be used for the cost function? Most users tend to use program defaults, in which case the problem of determining the proper values is just left to the program authors rather than the end user. It should be noted that the absolute magnitudes of the values are unimportant; it is the relative values that matter (that is, multiplying all of the scores by a constant will not affect the resultant best alignment). As mentioned previously, relative match and mismatch values are usually determined from empirical substitution matrices (for proteins) or models of sequence evolution for DNA.

```
GCC-TCG
GCCATTG
```

Figure 1.10. The optimal local alignment of sequences CAGCCTCGCTTAG and AATGCCATTGACGG with a cost function with matches = +1, mismatches = –0.3, and gaps = –1.3. Contrast with the local alignment in Figure 1.9B.

However, the most important value in scoring alignments is often considered to be the ratio of the mismatch cost and gap cost (a more complex function may include different mismatch costs and affine gap costs, but the general principle still holds). The values in the above example indicate that a gap is twice as costly as a mismatch. One could take this to mean that point mutations occur twice as often as insertion/deletion events; however, biologically we know that indels tend to be much rarer than that, relative to point mutations. There is remarkably little data available on the observed ratio of indel events to point mutations. For DNA, indels appear to be about 12 to 70 times less common than point mutations, depending on the specific taxa and evolutionary divergences examined (Mills, Luttig, et al. 2006; Ophir and Graur 1997; Sundström et al. 2003); one would expect protein sequences to have a different ratio. It is easy to imagine how one can force an optimal alignment to be more or less "gappy" by manipulating the gap cost: A very high gap cost will increase the number of mismatches and decrease the number of gaps in the optimal alignment, while a lower gap cost will decrease the number of mismatches and increase the number of gaps.

Additionally, there appears to be a bit of a discrepancy between the biological ratio of point mutations and indels and the actual cost structure used in the alignment. Subjective evaluation of alignments produced with different cost ratios, as well as objective examination of both empirical and simulated benchmarks (unpublished) tend to find that the gap costs that produce the best alignments (those that most closely resemble the true alignment) are usually less than those that would be predicted from a straightforward evaluation of the actual mutation rates. Methods for optimizing gap costs (as well as other aspects of the cost function) and their effects are an understudied aspect of sequence alignment.

MULTIPLE ALIGNMENTS

Up to this point, the described algorithms have been for comparing pairs of sequences. In general, we are often interested in aligning more than two sequences (in general, called multiple alignment). In principle, one could use a Needleman–Wunsch approach for more than two sequences (for example, constructing a three-dimensional cubic matrix for three sequences) (Jue et al. 1980; Murata et al. 1985), but this quickly becomes computationally intractable and inefficient as a result of constraints in computational power and memory.

Early alternate approaches for multiple alignment required a known phylogenetic tree. Sankoff and colleagues (Sankoff 1975; Sankoff et al. 1976; Sankoff et al. 1973) developed parsimony-based approaches. Waterman and Perlwitz (1984) suggested an alternate approach that used weighted averaging. Instead of using a known tree, Hogeweg and Hesper (1984) suggested an iterative method where one starts with a putative tree, aligns the data, uses the alignment to estimate a new tree, and then uses the new tree to realign the data, and so forth.

The approach for multiple sequence alignment that eventually really caught on is known as progressive alignment (Feng and Doolittle 1987, 1990). In progressive alignment, one generally starts by constructing all possible pairwise alignments (for n sequences, there are $n \times (n - 1)/2$ pairs). These pairwise alignments are used to estimate a phylogenetic tree using a distance-based algorithm such as the unweighted pair group method with arithmetic mean (UPGMA) or neighbor joining. Using the tree as a guide, the most similar sequences are aligned to each other using a pairwise algorithm. One then progressively adds sequences to the alignment, one sequence at a time, based on the structure of the phylogenetic tree. Numerous multiple alignment programs have been based on a progressive alignment adaptation of the Needleman–Wunsch algorithm, including ClustalW (Thompson et al. 1994), perhaps the most widely used global multiple alignment program.

Unlike the pairwise algorithm, multiple alignment algorithms are heuristic rather than exact solutions; by searching only a subset of the population of alignments, they efficiently find an alignment that is approximately optimal but is not guaranteed to be the most optimal alignment possible for the given cost function. For example, a general disadvantage of the progressive alignment approach is that it is what is known as a greedy algorithm; any mistakes that are made in early steps of the procedure cannot be corrected by later steps. For example, take the case (adapted from Duret and Abdeddaim 2000), with three short sequences whose optimal alignment is as shown in Figure 1.11A. Assuming the guide tree indicates we should start by aligning sequences 1 and 2, there are three possible alignments with the same score (one transversional mismatch and one gap), shown in Figure 1.11B. When adding sequence 3, the position of the gap cannot be changed. Thus, adding sequence 3 could lead to three possible multiple alignments, shown in Figure 1.11C, only the first of which is optimal. At the first step, only one of the three alignments can be used for the next step, and if the wrong one is chosen, the end results will not be the most optimal solution.

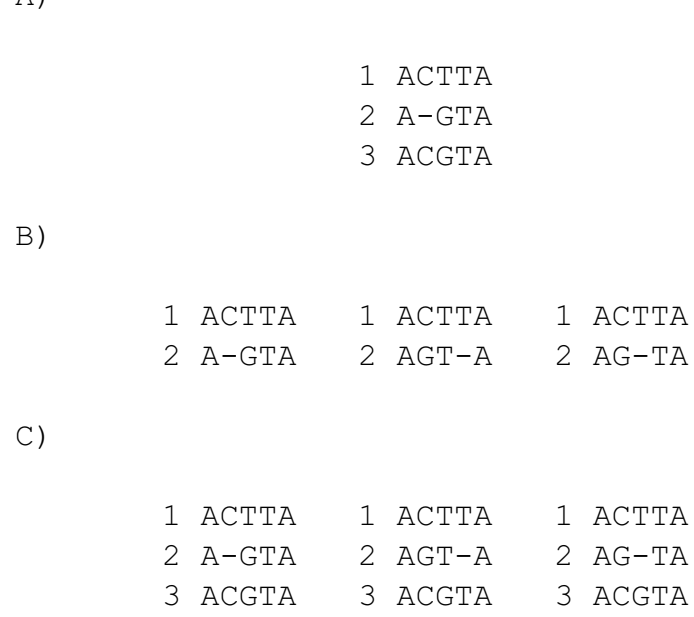

Figure 1.11. Illustration of the progressive alignment problem. (A) The optimal multiple alignment of three sequences. (B) The three possible optimal alignments that would be constructed in the first step of the alignment, depending on which pair were chosen to be aligned first. (C) The multiple alignments resulting from each of the three starting points from part B. Only one of these is equal to the actual optimal alignment illustrated in A. Example adapted from Duret and Abdeddaim (2000).

A number of less-greedy algorithms have been designed to try to get around this problem. For example, T-Coffee (Notredame et al. 2000) starts by using pairwise local alignments to find high scoring regions of similarity. Although a basic greedy progressive alignment is used to produce the global multiple alignment, these local alignments have an effect on the relative scoring at the early progressive stages and may help avoid errors that would otherwise be introduced into the multiple alignment. Another approach is to use iterative methods in which the alignment generated from one pass of an algorithm is used to construct a new guide tree, which can then be used to form a new alignment. Some of the better known iterative alignment programs include MultAlin (Corpet 1988), PRRP (Gotoh 1996), and DIALIGN (Morgenstern 1999; Morgenstern, Frech, et al. 1998).

STATISTICAL APPROACHES TO SEQUENCE ALIGNMENT

The methods described thus far could be considered "character-based" alignments in which the algorithms optimize a cost/benefit function. An alternate approach to alignment uses statistical estimation based on either maximum-likelihood or Bayesian methods. Although a statistical approach to alignment was suggested as far back as the mid-1980s (Bishop and Thompson 1986), this approach did not really begin to gain traction for another 15 years because of the computational complexity of the problem. The most influential contribution to statistical alignment has certainly been a pair of papers by Thorne et al. (1991, 1992) that describe the first tractable stochastic models for the insertion-deletion process. Although these models suffer from issues of realism (the first paper requires all insertion-deletion events to be a single character in length), the models suggested in these papers have formed the basis of most statistical alignment procedures in use today.

An important early advance was the formal development of hidden Markov models to describe the insertion-deletion process (Baldi et al. 1993, 1994; Krogh et al. 1994), including the release of the software HMMER (Eddy 1995) for statistical alignment. These were followed by a number of advances by Hein and colleagues for maximum-likelihood solutions to the multiple alignment problem (Hein 2001; Hein et al. 2003; Hein et al. 2000; Steel and Hein 2001).

Allison and colleagues (Allison and Wallace 1994; Allison et al. 1992a, b; Allison and Yee 1990) set the stage for Bayesian approaches for alignment through the formal modeling of and comparison of sequences. Mitchison (1999) describes one of the first statistical approaches to simultaneously estimating an alignment and a phylogeny, which helped lead to Handel (Holmes and Bruno 2001), one of the first software packages for Bayesian alignment.

In recent years, there has been an explosion of development in statistical alignment, in general, and simultaneous estimation of alignment and phylogeny specifically (Fleißner et al. 2005; Lunter et al. 2005; Redelings and Suchard 2005). Modern advances in statistical alignment are discussed in more detail in Chapters 5 and 10.

HOMOLOGY

The biological goal of alignment is the inference of site *homology*. Homology is similarity in a character or trait due to inheritance from a common ancestor. With respect to comparing biological sequences,

homology can have three different interpretations: (1) The sequences can be homologous; (2) the sites within homologous sequences can be homologous; (3) the observed characters at a homologous site can be homologous. Sequence alignment (as discussed in this book) is mostly concerned with the second of these. The general purpose of alignment is to identify positions in homologous sequences that are descended from a common ancestral sequence, that is, to identify which sites in a pair (or more) of sequences are themselves homologous. A pair of sites is "homologous" if the position in both sequences corresponds to the identical position in the common ancestral sequence. A pair of sites is "identical" if both sequences contain the same nucleotide (or amino acid for protein sequences); identity could be due to homology (i.e., the specific nucleotide was inherited by both sequences from the common ancestral sequence with no substitutions) but may often be due to convergent or parallel substitutions or by misalignment (when two sites are thought to be homologous but are not). Thus, character homology is first dependent on site homology (homologous characters can be found only at homologous sites) and secondarily on inheritance of the character from the common ancestor. We assume that the sequences we wish to align are homologous (in the first sense), although local alignment is often used as part of the inference of sequence homology using database search algorithms such as BLAST (Altschul et al. 1990) and FASTA (Pearson and Lipman 1988).

An additional level of homology that plays an important role with respect to sequence alignment is structural homology. Inferred homology of the secondary and/or tertiary structure of a protein or RNA is often used to guide the inferred sequence alignment. This is often done because structural homology should be conserved more than sequence similarity, making it easier to infer than direct sequence homology. Very little discussion has been raised about whether structural homology and sequence homology need to be congruent; it seems logically possible to have structural homologs whose underlying sequences are not themselves homologous. How common this incongruence may be is very difficult to determine.

An extremely important issue to remember is that there is a fundamental difference between the biological and computational goals of alignment algorithms. The biological goal is the inference of homology. The computational goal is the (efficient) optimization of an objective function. Many investigators forget that the fact that a solution is computationally optimal does not mean it is biologically correct. This is a

fundamental problem throughout computational biology, not just alignment. For example, phylogenetic methods attempt to find the tree that is most optimal based on a given criterion (such as parsimony, distance, or likelihood). It has been shown through simulation (Kumar 1996; Nei et al. 1998; Takahashi and Nei 2000; many unpublished studies) that the true tree is often not the most optimal tree for a given dataset, although this fact does not appear to be widely appreciated by the phylogenetics community at large. The same holds true for alignment.

CHALLENGES FOR THE FUTURE

Despite all of the progress that has been made in sequence alignment over the last four decades, there are still many challenges facing the sequence alignment community (both users and developers). In some sense, the primary purpose of this book is to highlight these challenges, and thus many of these challenges are discussed in detail throughout the remaining chapters. The following serves as a summary of some of these major challenges, as well as what to look for in the future.

- Generating objective functions (and parameters) that lead to the most biologically realistic homology predictions
- Generally improving methods for aligning highly diverged sequences
- Developing better methods for aligning larger data sets, both more and longer sequences, particularly at the scale of entire genomes
- Developing a better understanding of the molecular mechanisms leading to indel formation in the first place and find better ways to integrate these into context-dependent alignment
- Developing more realistic and computationally tractable models of indel mutation for use in statistical (maximum-likelihood or Bayesian) alignment and analysis
- Developing more efficient methods for statistical approaches to alignment so that they will become more useful and practical for larger data sets
- Further exploring the relationship between sequence homology and structural (or functional) morphology
- Developing better methods for the automatic incorporation of structural information into alignment

- Developing better methods for avoiding the circularity problem inherent in progressive multiple sequence alignment and phylogeny reconstruction (most likely by improving methods for simultaneous alignment and phylogeny recovery)
- Developing better and broader benchmarks for testing alignment algorithms, both in general and for specific alignment problems and situations
- Developing better approaches for comparing and contrasting alternate alignments
- Developing better methods for recognizing and displaying ambiguities in an alignment
- Developing better methods for incorporating alignment ambiguity into other analyses
- Exploring in much greater detail the effects of alignment error on downstream analyses in bioinformatics and genomics