30

Computational Biology:
The Fundamentals of Sequence-Based Techniques

30.1 Introduction

The subdisciplines within computational biology, especially those of bioinformatics, are relatively new, the word “bioinformatics” not being coined until the early to mid-1970s by Hesper and Hogeweg, 2011. With roots in computer science, information theory, biochemistry, and biology, it was and is a natural nexus of the four disciplines. However, before it could really take off, two essential factors had to become a reality, that of readily and freely available sequence and 3D structural databases and the development of powerful and relatively inexpensive computing power. Both of these are now very much a reality, having followed and/or exceeded the 18 month doubling exponential growth rate of Moore’s law (e.g., Moore, 1965; Wetterstrand, 2012; and see GenBank growth statistics at ftp://ftp.ncbi.nih.gov/genbank/gbrel.txt) since their development. Within this entire field, one of the most important developments has been the ability for researchers to gain biological understanding of molecules and organismal systems based solely on sequence data.

So, given the nucleotide or amino acid sequence of a biological molecule, what can be known about that molecule? Searching for small relevant patterns in sequences that may reflect some function allows for the inference of biologically relevant information. These can be cataloged motifs ascribed to catalytic activities; restriction enzyme or protease cut sites; regional physical attributes, such as secondary structure predictions, hydrophobicities; or even a sequence’s overall content and composition, as is used in some of the gene finding techniques. However, what about comparisons with other sequences? Can one molecule tell another’s story? Yes, naturally it can; inference through homology is a fundamental principle to all the biological sciences. A tremendous amount of knowledge can be gained by comparing one sequence against others. When a sequence is found to fall into a preexisting
biological gene family, function, enzymatic mechanism, evolution, and possibly even structure can all be inferred based on homology with its neighbors.

However, in order to learn anything by comparing sequences, one needs to know how to compare them. Constrained sequence portions, common patterns between them, could be used as “anchors” to create sequence alignments allowing comparison, but this brings up the alignment problem and "similarity." It is easy to see that sequences are aligned when they have identical symbols at identical positions, but what happens when symbols are not identical and sequences are not the same length? How can anybody know when the most similar portions of sequences are aligned, when is an alignment optimal, and does optimal mean anything as far as biology is concerned?

### 30.2 Databases

Sequence-based methods more often than not compare similar sequences one way or another. Acquiring the similar sequences usually requires sequence databases. These began with Dayhoff et al.’s hardbound *Atlas of Protein Sequence and Structure* in the mid-1960s (1965–1979). Databases have long since outgrown a hardbound atlas. They have become huge and have evolved considerably. In the United States, the National Center for Biotechnology Information (NCBI) (http://www.ncbi.nlm.nih.gov/), a division of the National Library of Medicine (NLM), at the National Institute of Health (NIH), supports and distributes the GenBank nucleic acid sequence database, which began in 1982 (Bilofsky et al., 1986), and the GenPept coding sequence (CDS) translations database, as well as a host of specialized sequence databases (in particular see the non-redundant RefSeq genomic, complementary DNA (cDNA), and protein databases). The National Biomedical Research Foundation (NBRF) (http://www-nbrf.georgetown.edu/), an affiliate of Georgetown University Medical Center, maintains the protein identification resource (PIR) (George et al., 1986) database of polypeptide sequences. In Europe, the European Molecular Biology Laboratory (EMBL) (http://www.embl.de/) and the European Bioinformatics Institute (EBI) (http://www.ebi.ac.uk/) maintain the EMBL nucleotide sequence database, which began in 1980 (Hamm and Cameron, 1986), and the excellently annotated Swiss-Prot (Bairoch, 1991) protein sequence database (also supported by the Swiss Institute of Bioinformatics [SIB] http://www.isb-sib.ch/), as well as the minimally annotated translations from EMBL (TrEMBL)—those EMBL translations not yet in Swiss-Prot—protein sequence databases, in Heidelberg, Germany; Cambridge, United Kingdom; and Geneva, Switzerland. EBI, SIB, and PIR coordinate to maintain the Universal Protein Resource (UniProt) (http://www.uniprot.org/), which is a single, nearly non-redundant, comprehensive, fully classified, richly and accurately annotated protein sequence knowledge base, with extensive cross-references and querying interfaces, all freely accessible to the scientific community. Additional, less well-known, sequence databases include sites with the military, with private industry, and in Japan (the DNA Data Bank of Japan, DDBJ, http://www.ddbj.nig.ac.jp/). In most cases data are openly exchanged between the databases so that many sites “mirror” one another. This is particularly true with GenBank, EMBL, and DDBJ, organized under the auspices of the International Nucleotide Sequence Databases Collaboration (INSDC). There is never a need to look in more than one of these databases; they all contain the same organized, reliable, comprehensive, and openly available libraries of genetic sequence data.

Additionally, a myriad of specialized sequence databases exist. These include sequence pattern databases such as restriction enzyme (e.g., http://rebase.neb.com/rebase/rebase.html) and proteinase (e.g., http://merops.sanger.ac.uk/) cleavage sites, promoter sequences and their binding regions (e.g., http://www.gene-regulation.com/pub/databases.html and http://epd.vital-it.ch/), protein motifs (e.g., http://prosite.expasy.org/) and profiles (e.g., http://pfam.janelia.org/), and organism or system-specific databases such as the sequence portions of A. C. elegans database (ACeDb) (http://www.acedb.org/), FlyBase (the Drosophila database, http://flybase.org/), the Saccharomyces Genome Database (SGD) (http://www.yeastgenome.org/), and the Ribosomal Database Project (RDP) (http://rdp.cme.msu.edu/). Many sites present their data in the context of a genome map browser, for example, the University of California, Santa Cruz, bioinformatics group’s genome browser (http://genome.ucsc.edu/),
and the Ensembl project (http://www.ensembl.org/), jointly hosted by the Wellcome Trust Sanger Institute and EBI. Genome map browsers attempt to tie together as many data types as possible using a physical map of a particular genome as a framework. They are a particularly effective way to explore bioinformatics and a great starting point for many analyses. This is but the “tip of the iceberg” when it comes to specialized sequence databases—visit the major sites and explore.

Two other types of databases are commonly accessed in bioinformatics: reference and 3D structure. Reference databases run the gamut from Online Mendelian Inheritance in Man (OMIM) (Pearson et al., 1994), which catalogs human genes and phenotypes, particularly those associated with human disease states, to PubMed (http://www.ncbi.nlm.nih.gov/pubmed), which provides free public access to more than 22 million citations from MedLine (the NLM’s citation and author abstract bibliographic database of over 5500 biomedical research and review journals), other life science journals, and online books. In nearly all cases, complete abstracts are supplied, and in many cases, links are provided to full-length text content. Other databases that could be put in this class include things like proprietary medical records databases and population studies databases. Finally, the Research Collaboratory for Structural Bioinformatics (RCSB) (http://home.rcsb.org/), a consortium of three institutions: the State University of New Jersey, Rutgers; the San Diego Supercomputer Center at the University of California, San Diego; and the University of Wisconsin–Madison, supports the 3D structure Protein Data Bank (PDB) (Bernstein et al., 1977). RCSB PDB is now a member of a World Wide PDB organization (http://www.wwpdb.org/, Berman et al., 2003), along with centers in Japan and Europe, whose mission “is to maintain a single Protein Data Bank Archive of macromolecular structural data that is freely and publicly available to the global community.” The NIH maintains “Molecules To Go” (http://helixweb.nih.gov/cgi-bin/pdb) as a very easy-to-use interface to PDB. Other 3D structure databases include the Nucleic Acid Databank (NDB) at Rutgers (http://ndbserver.rutgers.edu/) and the proprietary Cambridge small molecule crystallographic structural database (CSD) (http://www.ccdc.cam.ac.uk/products/csd/).

30.3 Primary Algorithms: Just the Basics

Even with the ready availability of sequence databases, the methods of computational biology can be daunting. The bases of the necessary algorithms were already developed within and before the time period when bioinformatics began. Many are discussed elsewhere in this computing handbook series. In particular, various pattern recognition algorithms have been adopted for use in bioinformatics. Computer science concepts used in bioinformatics include dynamic programming, hash tables, edit distances, hierarchical clustering, suffix-trees, hidden Markov models (HMMs), statistical methods of maximum likelihood and Bayesian inference, neural networks, and genetic algorithms. This chapter will concentrate on only those fundamental concepts upon which the entire discipline has grown, that of pair-wise and multiple-sequence alignment, database searching, and their applicability.

Many other realms of computational biology not reviewed in this chapter overlap with the basic concepts of bioinformatics discussed. These include 3D data analysis such as x-ray crystallography, nuclear magnetic spectroscopy, and high-resolution microscopy; “next-generation” DNA sequence assembly; protein and RNA microchip analysis; all the “omics”—genomics, transcriptomics, and proteomics; and, importantly, molecular phylogenetics, population biology, and biogeography. Some of these topics will be discussed elsewhere in this series; all have foundations laid in the fundamental bioinformatics principles discussed here.

30.3.1 Pair-Wise Methods

This category includes all of the programs that perform full length and best segment pair-wise sequence alignment and the numerous database similarity searching algorithms that incorporate alignment. A fundamental aspect of most of these types of programs is a variation of a standard computer science dynamic programming algorithm. The biggest problem is that a “brute force” approach just won’t work. Even without considering the introduction of gaps, the computation required to compare all possible
alignments between two sequences requires time proportional to the product of the lengths of the two sequences. Therefore, if the two sequences are approximately the same length (N), this is a \( N^2 \) problem. To include gaps, one would have to repeat the calculation 2N times to examine the possibility of gaps at each possible position within the sequences, now a \( N^4 \) problem. Waterman (1989) pointed out that using this naïve approach to align two sequences, each 300 symbols long, would require \( 10^{88} \) comparisons, more than the number of elementary particles estimated to exist in the universe and clearly impossible to solve! Part of the solution to this problem is the dynamic programming algorithm, as applied to sequence alignment.

However, first, a very simple method that provides a “gestalt” of all the various ways that two sequences can be compared will be described. This is the dot matrix or “dot plot” technique of sequence analysis, and it can often show things not seen by any other technique, as well as provide a great introduction to all of the pair-wise comparison techniques.

### 30.3.1.1 Dot Matrix (Plot) Methods

Dot matrix analysis is performed by plotting one sequence on a vertical axis against another on a horizontal axis, upon a grid, using a very simple approach: wherever the two match according to some specified scoring and filtering criteria, a dot is generated in the intersecting cell. So, why use dot matrix analysis? Dot matrices can point out areas of similarity between two sequences that all other methods might miss. This is because most other methods align either the overall length of two sequences or just the “best” parts of each to achieve optimal alignments. Dot matrix methods enable the user to visualize the entirety of both sequences all at the same time; all alignments can be seen, the “less than best” comparisons as well as the main one. Regions of interest can then be “zoomed in” on using more detailed procedures, if desired. The human eye and brain are still better than a computer at discerning complex visual patterns, especially when more than one pattern is being considered. This is what makes the technique such a powerful analytic tool with sequences, from short fragments through complete genes or even chromosomes, all the way to entire genomes (see, e.g., MUMmer, Delcher et al., 1999).

Even comparing a sequence to itself can be informative. A main identity diagonal is obvious; however, other features such as palindromes can also be seen. Perfect palindromes directly cross the main diagonal, and if comparing double-stranded DNA or RNA to itself, these inverted repeat regions could be indicative of potential cruciform pseudoknots at that point. Inverted repeats that are not palindromes show up as perpendicular lines to the diagonals but lie off the main diagonal. Direct internal repeats, duplicated sequence regions, will show up as parallel diagonals off of the main diagonal. Rows or columns of diagonals in any dot plot, whether the sequences are the same or different, clearly point out multiple duplications. Comparing different sequences to one another allows for the immediate recognition of insertions and deletions between the two. It is impossible to tell whether an evolutionary event that caused this sort of discrepancy between two sequences was an insertion or a deletion, and hence this phenomenon is called an “indel.” Jumps or shifts in the register of the main diagonal on a dot plot clearly points out the existence of an indel. Diagonals displaced off the center of the plot can also show the occurrence of “transposition.” In fact, dot matrix analysis is one of the few ways transpositions can be recognized in sequences.

However, the use and interpretation of dot plots is entirely up to the user—one must know how to successfully filter out extraneous background noise with appropriate parameters and what the plots mean—to identify areas between sequences that may have significant matches that no other method would ever notice. Much of sequence analysis is all about balancing signal to noise, and this is particularly important with dot plots. Often a filtered windowing approach is used—a dot will only be placed on the plot if some “stringency” is met. Within some defined window size, and when some defined criteria is met, then and only then will a dot be placed at the middle of that window. The window is then shifted over one position and the entire process is repeated. This process allows extraneous, random noise, due merely to the composition of the sequences, to be eliminated, while allowing any significant
similarity to come through. As a general guide to stringency levels, pick whatever window size is most appropriate for the analysis at hand, that is, about the size of the feature trying to be recognized, if known, and then choose a stringency within that window that produces the cleanest plot, without filtering out too much of the signal.

Dot plots can be particularly helpful in the analysis of small nucleic acid sequences, such as transfer RNAs (tRNAs). Consider the set of examples in Figure 30.1 using the phenylalanine tRNA molecule from yeast, GenBank:K01553, compared against itself. The sequence and structure are both known for this molecule, and the illustration shows how simple dot matrix procedures can quickly lead to functional and structural insights, even without complex folding algorithms, such as Zuker’s (1989). Figure 30.1 illustrates many of the previous concepts and how dot plots work using this yeast tRNA molecule as an example.

30.3.1.2 Dynamic Programming

Dynamic programming is a widely applied computer science technique, often used in many disciplines whenever optimal substructure solutions can provide an optimal overall solution. The technique applied to sequence alignment will be illustrated using an overly simplified visualization method. Instead of calculating the “score matrix” on the fly, as is often taught as one proceeds through the graph, a “match matrix” will be completely filled in first, and then points will be added to those positions that produce favorable alignments next. Therefore, the solution will occur in two stages. The first begins very much like the dot matrix methods previously shown; the second is totally different. Furthermore, the process will be illustrated working through the cells, in spite of the fact that many authors prefer to work through the edges; they provide equivalent solutions. Points will be added based on a “looking-back-over-your-left-shoulder” algorithm rule where the only allowable trace-back is diagonally behind and above any particular cell. Matching sequence characters will be worth one point; nonmatching characters will be worth zero points. Subtracting one point for initially creating a gap, and subtracting another point per gap for extending a gap, will penalize the scoring scheme, unless the gaps are at the beginning or end of the sequence. In other words, end gaps will not be penalized; therefore, both sequences do not have to begin or end at the same point in the alignment. This zero penalty end-weighting scheme is the default for most alignment programs but can often be changed with program options, if desired. The gap penalty function described here, and illustrated in Table 30.1, is called an “affine,” function, the standard “y = mx + b” equation for a line that does not cross the X,Y origin, where “b,” the Y intercept, describes how much initial penalty is imposed for creating each new gap (Gotoh, 1982): total penalty = ([length of gap] × [gap extension penalty]) + gap opening penalty. The affine function is the usual gap penalty scheme in most pair-wise alignment programs; however, the gap “creation” or “opening” penalties and gap “extension” penalties used in the example are not typical of most programs. Furthermore, the one/zero match/mismatch scoring scheme is also not typical. These scores are used here for the sake of simplicity. Available program default values are usually quite appropriate and attempt to balance biological feasibility and sensitivity.

The example in Table 30.1 uses two random sequences that happen to fit the TATA promoter region consensus of eukaryotes and of bacteria. The most conserved bases within the consensus are capitalized by convention. The eukaryote promoter sequence is along the X-axis, and the bacterial sequence is along the Y-axis in the example. There may be more than one best path through the matrix. This time, with the relatively simple scoring function used, starting at the top and working down, and then tracing back, only one optimal alignment, with a final score of five, was discovered. This score is the highest, bottom-right value in the trace-back path graph, the sum of five matches minus no interior gaps. Notice that this would not be the optimal alignment, and the matrices would be all wrong, if end gap penalties had been imposed. Also notice that its 62.5% identity score could be improved by sliding the first “c” on the lower sequence to the left but that the gap cost scheme used prevented that from happening. The alignment score, based on the scoring scheme used, is the only number optimized by the algorithm, not any type of a similarity or identity percentage!
FIGURE 30.1  A dot plot analysis of the yeast phenylalanine tRNA. (a) Here the yeast tRNA is compared to itself with a window size of 7 and a stringency value of 5. Several direct repeats are obvious as off-diagonal alignment segments. (b) The yeast tRNA sequence compared to its reverse complement is shown next, using the same 5 out of 7 stringency setting. Now the potential for inverted repeats becomes obvious; these are the well-known stem-loop structures of the tRNA cloverleaf molecular shape. They appear as clearly delineated diagonals. These diagonals are now perpendicular to an imaginary main diagonal running opposite to the previous case, since the orientation of the second sequence was reversed. For example, examine the middle stem; the region of the molecule centered at approximately base number 38 has a clear propensity to base pair with itself without creating a loop, since it crosses that imaginary main diagonal, and then just after a small unpaired gap, another stem is formed between the region from about base number 24 through 30 with approximately 46 through 40. (c) The middle stem described earlier most likely corresponds to the bottom-most stem represented here in a “ribbon” style model of the tRNA. (Drawn from Rotkiewicz, P., iMol molecular visualization program, http://www.pirx.com/iMol, 2007.) This loop is the anticodon region of the yeast phenylalanine tRNA deposited in the 3D PDB under access code 1TRA. (From Sundaralingam, M. et al., *Nucleic Acids Res.*, 10, 2471, 1976.)
TABLE 30.1 Dynamic Programming (One Point for Match, Zero Points for Mismatch, Less One Point for Creating a Gap, Plus Less One Point per Gap for Extending the Gap)

a. First complete the match matrix using whatever match/mismatch scoring scheme used:

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>a</th>
<th>g</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

b. Now add and subtract points based on the best path through the matrix, working diagonally, left to right and top to bottom. However, whenever a box is jumped to make the path, subtract one point for doing so and an additional point per box jumped, except at the beginning or end of the alignment, so that end gaps are not penalized. Fill in all additions and subtractions, calculate the sums and differences along the way, and keep track of all the best paths. The score matrix is shown with all calculations in the following:

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>t</th>
<th>A</th>
<th>a</th>
<th>g</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>0</td>
<td>+0</td>
<td>=0</td>
<td></td>
<td>0</td>
<td>+0</td>
<td>=0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>+0</td>
<td>1</td>
<td>=1</td>
<td></td>
<td></td>
<td>0</td>
<td>+0</td>
<td>=1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>+0</td>
<td>0</td>
<td>1</td>
<td>=2</td>
<td></td>
<td>0</td>
<td>+1</td>
<td>=2</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>1</td>
<td>+0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>0</td>
<td>+1</td>
<td>=2</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>+0</td>
<td>0</td>
<td>1</td>
<td>=2</td>
<td></td>
<td>0</td>
<td>+1</td>
<td>=2</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>+0</td>
<td>0</td>
<td>1</td>
<td>=2</td>
<td></td>
<td>0</td>
<td>+1</td>
<td>=2</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>+0</td>
<td>1</td>
<td>=1</td>
<td></td>
<td></td>
<td>0</td>
<td>+0</td>
<td>=0</td>
</tr>
</tbody>
</table>

(continued)

c. Clean up the score matrix next. Just the totals are shown in each cell in the matrix in the following.

All best paths are highlighted:

<table>
<thead>
<tr>
<th></th>
<th>c</th>
<th>T</th>
<th>A</th>
<th>T</th>
<th>A</th>
<th>t</th>
<th>A</th>
<th>a</th>
<th>g</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>t</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>
Another way to explore the dynamic programming solution space is to reverse the entire process. This can often discover alternative alignments. To recap, and for those people that like mathematics, an optimal pair-wise alignment is defined as an arrangement of two sequences, $1$ of length $i$ and $2$ of length $j$, such that the following holds true:

1. The number of matching symbols between $1$ and $2$ are maximized.
2. The number of gaps within $1$ and $2$ are minimized.
3. The number of mismatched symbols between $1$ and $2$ are minimized. Therefore, the actual solution can be represented by the following recursion:

$$S_{ij} = s_{ij} + \max \begin{cases} S_{i-1j-1} & \text{or} \\ \max S_{i-xj-1} + w_{x-1} & \text{or} \\ 2 < x < i \\ \max S_{i-1j-y} + w_{y-1} & \text{or} \\ 2 < y < i \end{cases}$$

where

- $S_{ij}$ is the score for the alignment ending at $i$ in sequence $1$ and $j$ in sequence $2$
- $s_{ij}$ is the score for aligning $i$ with $j$
- $w_x$ is the score for making a $x$ long gap in sequence $1$
- $w_y$ is the score for making a $y$ long gap in sequence $2$, allowing gaps to be any length in either sequence

However, just because dynamic programming guarantees an optimal alignment, it is not necessarily the only optimal alignment. Furthermore, the optimal alignment is not necessarily the “right”
or biologically relevant alignment! Significance estimators can provide some handle on this, but always question the results of any computerized solution based on what is known about the biology of the system. The example shown in Table 30.1 illustrates the Needleman and Wunsch (1970) global solution, as generalized by Gotoh (1982). Later refinements (Smith and Waterman, 1981) demonstrated how dynamic programming could also be used to find optimal local alignments. To solve dynamic programming using local alignment (without going into all the details) algorithms, do the following:

1. Scoring functions penalize mismatches by assigning negative numbers for them. Therefore, bad paths quickly become very bad. This leads to a trace-back path matrix with many alternative paths, most of which do not extend the full length of the graph.
2. The best trace-back within the overall graph is chosen. This does not have to begin or end at the edges of the matrix—it’s the best segment of alignment.

### 30.3.1.3 Significance

A particularly common misunderstanding in this field regards the concept of homology versus that of similarity: there is a huge difference! Similarity and identity are merely statistical parameters that describe how much two sequences, or portions of them, are alike according to some set scoring criteria. It can be normalized to ascertain statistical significance, as is done in database searching, but it’s still just a number. Homology, in contrast and by definition, implies an evolutionary relationship—but more than just the fact that all life on Earth evolved from some common ancestor. One needs to be able to demonstrate an evolutionary lineage between the organisms or genes of interest to claim homology. Better yet, provide experimental evidence, structural, morphological, genetic, or fossil, that corroborates the assertion. There really is no such thing as percent homology; something is either homologous or it’s not. Walter Fitch (personal communication) explains with the joke: “Homology is like pregnancy—you can’t be 50% pregnant, just like something can’t be 50% homologous. You either are or you are not.” Do not make the mistake of calling any old sequence similarity homology. Highly significant similarity can argue for homology, but not the other way around.

So, how does one tell if a similarity, in other words, an alignment discovered by some program, means anything? Is it statistically significant, is it truly homologous, and, even more importantly, does it have anything to do with real biology? Many programs generate percent similarity scores; however, as seen earlier, these really don’t mean a whole lot. Don’t use percent similarities or identities to compare sequences except in the roughest way. Alignment quality scores mean more but are difficult to interpret. At least they take the length of similarity, all of the necessary gaps introduced, and the matching of symbols all into account; however, quality scores are only relevant within the context of a particular comparison or search and are relative to the particular scoring matrix and gap penalties used.

A traditional way of ascertaining alignment significance relies on an old statistics trick—Monte Carlo simulations. This type of significance estimation has implicit statistical problems; however, few practical alternatives exist when just comparing two sequences, and they are fast and easy. Monte Carlo randomization options in dynamic programming alignment algorithms compare an original alignment score against the distribution of scores from alignments where one of the sequences is repeatedly shuffled, at least 100 times after the initial alignment, each with its own alignment score. A standard deviation is calculated based on that distribution. Comparing the mean of the randomized sequence alignment scores to the original score using a “Z-score,” calculation can help decide significance. A “rule of thumb” is if the actual score is much more than three standard deviations above the mean of the randomized scores, the analysis may be significant; if it is much more than five, than it probably is significant; and if it is above ten, than it definitely is significant. Many Z-scores measure this distance from the mean using a simplistic Monte Carlo model assuming a normal Gaussian distribution, in spite of the fact that
“sequence space” actually follows an “extreme value distribution”; however, this simplistic approximation actually estimates significance quite well:

\[ Z\text{-score} = \frac{\left(\text{Actual score} - \text{Mean of randomized scores}\right)}{\text{Standard deviation of randomized score distribution}} \]

When the two TATA sequences from the previous dynamic programming example are compared to one another using the same scoring parameters as before, but incorporating a Monte Carlo Z-score calculation, their similarity is found not at all significant. The mean score based on 100 randomizations is around 42, with a standard deviation around 8. Therefore, the Z-score is only one, so there is no significance to the match in spite of 62.5% identity! Composition can make a huge difference—the similarity seen here is merely a reflection of the relative abundance of A’s and T’s in both sequences!

The FastA (Pearson and Lipman, 1988; Pearson, 1990, 1998), Basic Local Alignment Search Tool (BLAST) (Altschul et al., 1990, 1997), Profile (Gribskov et al., 1987, 1989), and HMMer (Eddy, 1996, 1998) database search algorithms and their extensions all use a similar approach but base their statistics on the distance of the query matches from the actual, or a simulated, extreme value distribution of the rest of the, “insignificantly similar,” members of the database being searched. The statistics are well characterized for alignments without gaps. The number of ungapped alignments to be expected between any two sequences of length \( m \) and \( n \), with a score greater than or equal to a particular score \( S \), is generalized such that the expectation value \( E \) relates to \( S \) through the function \( E = K m n e^{-\lambda s} \) (Karlin and Altschul, 1990, and see http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html). This is called the \( E \)-value for score \( S \). In a database search, \( m \) is the length of the query and \( n \) is the size of the database in residues, so \( N = mn \) is the complete search space size. \( K \) and \(-\lambda\) are supplied by statistical theory, dependent on the scoring system and the background amino acid or nucleotide frequencies, calculated from actual or simulated database alignment distributions. These two parameters define the statistical significance of an \( E \)-value. Expectation values are given in scientific notation; the smaller the number, that is, the closer it is to zero, the more significant the match. In other words, the less likely a particular alignment is due to chance alone. Expectation values provide evidence for the inference of homology, or not, by showing how often to expect a particular alignment to occur as a random phenomenon in a search of that size database. Rough, conservative guidelines to Z-scores and expectation values from a typical protein search are shown in Table 30.2.

Be very careful with guidelines such as these, though, because they are entirely dependent on both the size and content of the database being searched as well as how often the search is performed! Think about it: the odds are very different for rolling dice depending on how many dice are rolled at a time, whether they are “loaded” or not and how many times they are rolled. Another very powerful empirical method of determining significance is to repeat a database search with an entry of questionable significance discovered by the previous search. If the new entry finds more significant “hits” of the same biochemical “family” of sequence as the original search did, then the entry in question is undoubtedly homologous to the original entry. That is, homology is transitive. If it finds entirely different types of sequences, then it probably is not homologous. Modular proteins with distinctly separate domains confuse issues

### Table 30.2 Rough, Conservative Guidelines to Z-Scores and Expectation Values from a Typical Protein Search

<table>
<thead>
<tr>
<th>~Z-Score</th>
<th>~( E ) Value</th>
<th>Inference</th>
</tr>
</thead>
<tbody>
<tr>
<td>\geq 3</td>
<td>\leq 0.1</td>
<td>Little, if any, evidence for homology, but impossible to disprove!</td>
</tr>
<tr>
<td>\approx 5</td>
<td>\approx 10^{-2}</td>
<td>Probably homologous, but may be due to convergent evolution</td>
</tr>
<tr>
<td>\leq 10</td>
<td>\geq 10^{-3}</td>
<td>Definitely homologous</td>
</tr>
</tbody>
</table>

Downloaded by [Steven M. Thompson] at 10:34 25 June 2014
considerably, but the principles remain the same and can be explained through domain swapping and other examples of non-vertical transmission. And, finally, the “gold standard” of homology is shared structural folds—if two proteins have the same structural fold, then, regardless of similarity, at least that particular domain is homologous between the two.

30.3.1.4 Scoring Matrices
This is all fine, but what about protein sequences—conservative amino acid replacements, as opposed to identities? The scoring functions discussed so far only allow match or mismatch.

Allowing similarities is certainly an additional complication that would seem important. Particular amino acids are very much alike, structurally, chemically, and genetically. How can the similarity of amino acids be taken advantage of in alignments? People have been struggling with this problem since the late 1960s.

Dayhoff (Schwartz and Dayhoff, 1979) unambiguously aligned closely related protein datasets (no more than 15% difference, and in particular cytochrome c) available at that point in time and noticed that certain residues, if they mutate at all, are prone to change into certain other residues. As it turns out, these propensities for change fell into the same categories that chemists had known for years—those same chemical and structural classes mentioned earlier—conserved through the evolutionary constraints of natural selection. Dayhoff’s empirical observation quantified these changes. Based on the multiple-sequence alignments that she created, the empirical amino acid frequencies within those alignments, and the assumption that estimated mutation rates in closely related proteins can be extrapolated to more distant relationships, she was able to empirically specify the relative probabilities at which different residues mutated into other residues through evolutionary history, as appropriate within some level of divergence between the sequences considered. This is the basis of the famous PAM (corrupted acronym of “accepted point mutation”) 250 (meaning that the matrix has been multiplied by itself 250 times) log-odds matrix.

Since Dayhoff’s time, other biomathematicians (e.g., Henikoff and Henikoff’s [1992] BLOSUM [BLOcks SUBstitution Matrix] series of matrices and the Gonnet et al. matrix [1992]) have created matrices regarded more accurate than Dayhoff’s original, but the concept remains the same. Plus, the PAM series remains a classic as historically the most widely used. Confusingly these matrices are known variously as symbol comparison, log-odds, substitution, or scoring tables or matrices, but they are fundamental to all sequence comparison techniques. The default amino acid scoring matrix for most protein similarity comparison programs is now the BLOSUM62 table (Henikoff and Henikoff, 1992). The “62” refers to the minimum level of identity within the ungapped sequence blocks that went into the creation of the matrix. Lower BLOSUM numbers are more appropriate for more divergent datasets. The BLOSUM62 matrix is shown in Table 30.3.

30.3.1.5 Database Searching
Database searching programs use elements of all the concepts discussed earlier; however, classic dynamic programming techniques are far too slow when used against an entire sequence database. Therefore, the algorithms use tricks to make things happen faster. These tricks fall into two main categories, that of hashing and that of approximation. Hashing is the process of breaking a sequence into small “words” or “k-tuples” of a set size and creating a “lookup,” “hash” table with those words keyed to numbers. Then when any of the words match part of an entry in the database, that match is saved. In general, hashing reduces the complexity of the search problem from \( N^2 \) for dynamic programming to \( N \), the length of all the sequences in the database.

Approximation techniques are collectively known as “heuristics.” Webster’s (1973) defines heuristic as “providing aid or direction in the solution of a problem, but otherwise unjustified or incapable of justification.” In database searching techniques, the heuristic usually restricts the necessary search space by calculating some sort of a statistic that allows the program to decide whether further scrutiny of a particular match should be pursued. This statistic may miss things depending on the
### Table 30.3  BLOSUM62 Amino Acid Scoring Matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A4</th>
<th>B-2</th>
<th>C0</th>
<th>D-2</th>
<th>E-1</th>
<th>F-2</th>
<th>G0</th>
<th>H-2</th>
<th>I-1</th>
<th>K-1</th>
<th>L-1</th>
<th>M-1</th>
<th>N-2</th>
<th>P-1</th>
<th>Q-1</th>
<th>R-1</th>
<th>S1</th>
<th>T0</th>
<th>V0</th>
<th>W-3</th>
<th>X-1</th>
<th>Y-2</th>
<th>Z-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>-2</td>
<td>6</td>
<td>-3</td>
<td>2</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>-4</td>
<td>-3</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>-4</td>
<td>-1</td>
<td>-3</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>-3</td>
<td>9</td>
<td>-3</td>
<td>-4</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>-4</td>
</tr>
<tr>
<td>D</td>
<td>-2</td>
<td>6</td>
<td>-3</td>
<td>2</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>-4</td>
<td>-3</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>-4</td>
<td>-1</td>
<td>-3</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>-1</td>
<td>2</td>
<td>-4</td>
<td>2</td>
<td>5</td>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>-3</td>
<td>1</td>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>6</td>
<td>-3</td>
<td>-1</td>
<td>0</td>
<td>-3</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>-4</td>
<td>-3</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>3</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>6</td>
<td>-2</td>
<td>-4</td>
<td>-2</td>
<td>-4</td>
<td>-3</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>8</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>1</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>0</td>
<td>-4</td>
<td>-3</td>
<td>4</td>
<td>-3</td>
<td>2</td>
<td>1</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>3</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>1</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>5</td>
<td>-2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>-1</td>
<td>-4</td>
<td>-1</td>
<td>-4</td>
<td>-3</td>
<td>0</td>
<td>-4</td>
<td>-3</td>
<td>2</td>
<td>1</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>3</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>2</td>
<td>0</td>
<td>-3</td>
<td>-2</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>-2</td>
<td>1</td>
<td>-3</td>
<td>1</td>
<td>0</td>
<td>-3</td>
<td>0</td>
<td>1</td>
<td>-3</td>
<td>0</td>
<td>-3</td>
<td>-2</td>
<td>6</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-3</td>
<td>-4</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-4</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-4</td>
<td>-1</td>
<td>-3</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>-1</td>
<td>0</td>
<td>-3</td>
<td>0</td>
<td>2</td>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>-3</td>
<td>1</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>-1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>-3</td>
<td>2</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-3</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>4</td>
<td>1</td>
<td>-2</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>0</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td>-3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>4</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>-3</td>
<td>-4</td>
<td>-2</td>
<td>-4</td>
<td>-3</td>
<td>1</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>-4</td>
<td>-4</td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-2</td>
<td>-3</td>
<td>11</td>
<td>-1</td>
<td>2</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>3</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
<td>7</td>
<td>-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Z</td>
<td>-1</td>
<td>2</td>
<td>-4</td>
<td>2</td>
<td>5</td>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>-3</td>
<td>1</td>
<td>-3</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
<td>-3</td>
<td>-1</td>
<td>-2</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Values whose magnitude is $\geq 4$ are drawn in shadowed characters to make them easier to recognize.
parameters set—that’s what makes it heuristic. The exact implementation varies between the different programs, but the basic ideas follow in all of them.

Two predominant versions exist: the Fast and BLAST programs. Both return local alignments. Both are not a single program but rather a family of programs with implementations designed to compare a sequence to a database every which way imaginable: a DNA sequence against a DNA database (not recommended unless forced to do so because of dealing with a nontranslated genomic region), a translated (where the translation is done “on-the-fly” in all six frames) version of a DNA sequence against a translated (“on-the-fly”) version of a DNA database (not available in Fast), a translated (“on-the-fly”) version of a DNA sequence against a protein database, a protein sequence against a translated (“on-the-fly”) version of a DNA database, or a protein sequence against a protein database. Many implementations allow the recognition of frame shifts in translated comparisons. In more detail:

30.3.1.5.1 FastA and Family (Pearson and Lipman, 1988; Pearson, 1990, 1998)

1. Works well for DNA against DNA searches (within limits of possible sensitivity)
2. Can find only one gapped region of similarity per search
3. Relatively slow, should often be run in the background
4. Does not require specially prepared, preformatted databases

The Fast programs are older than BLAST. They were the first widely used, powerful sequence database searching programs available. Pearson continually refines the algorithms such that they remain a viable alternative to BLAST, especially if one is restricted to searching DNA against DNA without translation. The Fast programs also may be more sensitive than BLAST in situations where BLAST finds no significant alignments.

30.3.1.5.1.1 Algorithm

The Fast programs build words of a set k-tuple size, by default two for peptides. The program then identifies all exact word matches between the sequence and the database members. Scores are assigned to each continuous, ungapped, diagonal by adding all of the exact match BLOSUM values. The 10 highest scoring diagonals for each query–database pair are then rescored using BLOSUM similarities as well as identities, and ends are trimmed to maximize the score. The best of each of these is called the Init1 score.

Next the program “looks” around to see if nearby off-diagonal Init1 alignments can be combined by incorporating gaps. If so, a new score, Initn, is calculated by summing up all the contributing Init1 scores, penalizing gaps with a penalty for each. The program then constructs an optimal local alignment for all Initn pairs with scores better than some set threshold using a variation of dynamic programming “in a band” (Pearson and Lipman, 1988). A sixteen residue band centered at the highest Init1 region is used by default with peptides. The score generated from this step is called the opt score.

Then Fast uses a simple linear regression against the natural log of the search set sequence length to calculate a normalized z-score for the sequence pair (Pearson, 1998). It compares the distribution of these z-scores to the actual extreme value distribution of the search. Using this distribution, the program estimates the number of sequences that would be expected to have, purely by chance, a z-score greater than or equal to the z-score obtained in the search. This is reported as the expectation value. Unfortunately, the z-score used in the Fast programs and the Monte Carlo style Z-score discussed previously are quite different and cannot be directly compared; however, the expectation values between Fast and other programs are equivalent. Finally, the program uses full Smith–Waterman local dynamic programming, not “restricted to a band,” to produce its final alignments.

30.3.1.5.2 BLAST: Basic Local Alignment Search Tool (Altschul et al., 1990, 1997)

1. Normally not a good idea to use for DNA against DNA searches (not optimized)
2. Prefilters repeat and “low complexity” sequence regions by default
3. Can find more than one region of gapped similarity
4. Very fast heuristic and parallel implementation
5. Restricted to precompiled, specially formatted databases
30.3.1.5.2.1 Algorithm  After BLAST has sorted its lookup table, it tries to find all double-word hits along the same diagonal within some specified distance using a discrete finite automaton (DFA) (NCBI). Word hits of size $W$ do not have to be identical; rather, they have to be better than some threshold value $T$. To identify double-word hits, the DFA scans through all strings of words (typically $W = 3$ for peptides) that score at least $T$ (usually 11 for peptides). Each double-word hit that passes this step then triggers a process of ungapped extension in both directions, such that each diagonal is extended as far as it can, until the running score starts to drop below a predefined value $X$ within a certain range $A$. The result of this pass is called a high-scoring segment pair (HSP). HSPs then, are those stretches of sequence pairs that cannot be further improved by extension or trimming. Those HSPs that pass this step with a score better than $S$ then begin a gapped extension step utilizing dynamic programming. Those gapped alignments with expectation values better than the user specified cutoff are reported. The extreme value distribution of BLAST expectation values is pre-computed against each precompiled database available to the program—this is one area that speeds up the algorithm considerably.

30.3.1.5.3 Database Searching Conclusions

In review, both the Fast and BLAST family of programs calculate their Expectation “$E$” values on a more realistic “extreme value distribution,” based on either real or simulated “not significantly similar” database alignments, than do most Monte Carlo style Z-scores, since Monte Carlo techniques are often based on the normal distribution, although they do parallel Monte Carlo style Z-scores fairly well. Regardless, the higher an $E$-value, the more probable that the observed match is due to chance in a search of that size database, and the lower its Z-score will be. Conversely, the smaller an $E$-value, that is, the closer it is to zero, the more significant it is, and the higher its Z-score will be. The $E$-value is the number that really matters. A value of 0.01 is usually a decent starting point for significance in most typical searches.

Furthermore, all database searching, regardless of the algorithm, is far more sensitive at the amino acid level than at the DNA level. This is because proteins have twenty match criteria, the 20 naturally occurring amino acids, versus DNA’s four nucleotides, and those four DNA nucleotides can only be identical or not, not similar, to each other. Furthermore, many DNA base changes (especially third position changes) do not change the encoded protein at all because of the redundancy of the genetic code. All of these factors drastically increase the “noise” level of a DNA against DNA search and give protein searches a much greater “look-back” time, typically five to ten times longer back in evolutionary history. Therefore, whenever dealing with coding sequence, it is always prudent to search at the protein level. Furthermore, even without a protein sequence query, one can use programs that take a DNA query, translate it in all six frames, and then compare protein databases.

30.3.2 Multiple-Sequence Analysis

The power and sensitivity of sequence-based computational methods dramatically increase with the addition of more data. More data yield stronger analyses—if done carefully! Otherwise, it can confound the issue. The patterns of conservation become ever clearer by comparing the conserved portions of sequences among a larger and larger dataset. Those areas most resistant to change are most important to the molecule. The basic assumption is that those portions of sequence of crucial structural and functional value are most constrained against evolutionary change. They will not tolerate many mutations. Not that mutation doesn’t occur in these regions, it is just that most mutation in the area is lethal, so it is never seen. Other areas of sequence are able to drift more readily, being less subject to this evolutionary pressure.

Therefore, sequences end up a mosaic of quickly and slowly changing regions over evolutionary time.
However, a big problem exists. As we’ve seen, dynamic programming reduces the pair-wise alignment problem’s complexity down to order $N^2$—the solution of a 2D matrix, so the complexity of the solution is equal to the length of the longest sequence squared. But how are more than just two sequences at a time aligned? It becomes much harder. Sequences can be manually aligned with an editor, but some type of an automated solution is desirable, at least as a starting point to manual alignment. Solving the dynamic programming algorithm for more than just two sequences rapidly becomes intractable. Dynamic programming’s complexity, and hence its computational requirements, increases exponentially with the number of sequences in the dataset being compared (complexity = [sequence length]$^\text{number of sequences}$), an N-dimensional matrix. So a three-sequence dynamic programming alignment would require the solution of a three-axis matrix, with complexity equal to the length of the longest sequence cubed, and so forth. A 3D matrix can at least be drawn, but more dimensions than that quickly become impossible to even visualize!

Several different heuristics have been employed over the years to simplify the complexity of the problem. One classic program, multiple sequence alignment (MSA) (Gupta et al., 1995), attempts to globally solve the N-dimensional matrix recursion using a bounding box trick. However, the algorithm’s complexity precludes its use in most situations, except with very small datasets. Another way to globally solve the algorithm, and yet reduce its complexity, is to restrict the search space to only the most conserved “local” portions of all the sequences involved. This approach is used by the program pattern-induced multi-sequence alignment (PIMA) (Smith and Smith, 1992). Neither program ever really caught on though, as both require relatively small datasets.

30.3.2.1 Heuristic Solutions: How the Algorithms Work

Most multiple-sequence alignment implementations do not attempt to globally solve the algorithm; they modify dynamic programming by establishing a pair-wise order in which to build the alignment. This heuristic modification is known as pair-wise, progressive dynamic programming. Originally attributed to Feng and Doolittle (1987), this variation of the dynamic programming algorithm creates a global alignment but restricts its search space to a local neighborhood of the full length of only two sequences, at any one time. Consider a dataset of sequences. First all are compared to each other, pair-wise, using some quick variation of standard dynamic programming. This establishes an order for the dataset, most to least similar, known as the “guide tree.” The algorithm then takes the top two most similar sequences and aligns them. It creates a quasi-consensus of those two and aligns that to the third sequence. Next it creates the same sort of quasi-consensus of the first three sequences and aligns that to the forth most similar. Subgroups are clustered together similarly. The way the various implementations make and use this “consensus” is one of the biggest differences between them. This process, all using standard, pair-wise dynamic programming, continues until it has worked its way through all of the sequences and/or sets of clusters, to complete the full multiple-sequence alignment. The pair-wise, progressive solution is implemented in several programs. Perhaps the most popular is ClustalW of Thompson et al. (1994) and its graphical user interface ClustalX (Thompson et al., 1997). This program achieved the first major advances over the basic Feng and Doolittle algorithm by incorporating variable sequence weighting, dynamically varying gap penalties and substitution matrices, and a neighbor-joining (NJ, Saitou and Nei, 1987) guide tree. Several more variations on the theme have come along since ClustalW. Tree-based consistency objective function for alignment evaluation (T-Coffee) (Notredame et al., 2000) was one of the first and has gained much favor. Its biggest innovation is the use of a preprocessed, weighted library of all the pair-wise global alignments between the sequences in a dataset plus the ten best local alignments associated with each pair of sequences. This helps build both the NJ guide tree and the progressive alignment. Furthermore, the library is used to assure consistency and help prevent errors, by allowing “forward thinking” to see whether the overall alignment will be better one way or another after particular segments are aligned one way or another. Notredame (2006) uses the analogy of school schedules—everybody, students, teachers, and administrators, with some folk being more important than others, that is, the weighting factor, puts the schedule they desire in a big pile, that is, T-Coffee’s library, and
the trick is to best fit all the schedules to one academic calendar, so that everybody is happiest, that is, T-Coffee's final multiple-sequence alignment. The M-Coffee version (Wallace et al., 2006) can even tie together multiple methods as external modules, making consistency libraries from the results of each. T-Coffee is one of the most accurate multiple-sequence alignment methods available because of this consistency-based rationale, but it is not the fastest.

Muscle (Edgar, 2004; Edgar and Batzoglou, 2006) is another multiple-sequence alignment program that came after ClustalW. It is incredibly fast, yet nearly as accurate as T-Coffee with protein data. Muscle is an iterative method that uses weighted log-expectation profile scoring along with a slew of optimizations. It proceeds in three stages—draft progressive using k-tuple counting, improved progressive using a revised tree from the previous iteration, and refinement by sequential deletion of each tree edge with subsequent profile realignment. Two other programs that claim speed and accuracy beyond standard progressive multiple-sequence alignment are partial order alignment (POA) and profile consistency multiple-sequence alignment (PCMA). POA (Lee et al., 2002) uses graph theory to represent an alignment as a partial order graph and allows for a new edit operator, homologous recombination. PCMA (Pei et al., 2003) combines Clustal-style progressive alignment of very similar sequences with a T-Coffee-like strategy of profile–profile consistency comparison. Perhaps the most accurate multiple-sequence alignment program is ProbCons (Do et al., 2005). It uses HMM techniques and posterior probability matrices that compare random pair-wise alignments to expected pair-wise alignments. Probability consistency transformation is used to reestimate the scores, and a guide tree is then constructed, which is used to compute the alignment, which is then iteratively refined.

The MAFFT (Katoh et al., 2002, 2005) multiple-sequence alignment suite is also quite powerful. It can be run many different ways—a couple of progressive, approximate modes, using a fast Fourier transform (FFT); a couple of iteratively refined methods that add in weighted-sum-of-pairs (WSP) scoring; and several iterative methods that use the WSP scoring combined with a T-Coffee-like consistency-based scoring scheme. Speed and accuracy are inversely proportional for these from fast and rough, to slow and accurate, respectively. MAFFT’s FFT provides a huge speedup over most previous methods. Homologous regions are quickly identified by converting amino acid residues to vectors of volume and polarity, thus changing a 20-character alphabet to six, rather than by using an amino acid similarity matrix. Similarly, nucleotide bases are converted to vectors of imaginary and complex numbers. The FFT trick then reduces the complexity of the subsequent comparison to order N logN. FFT identifies potential similarities though, without localizing them; a sliding window step using the BLOSUM62 matrix is used for this. Then MAFFT constructs a distance matrix, and hence a progressive guide tree, on the number of shared six-tuples from this Fourier transform, rather than on a ranking based on full-length, pair-wise sequence similarity. The user can specify how many times a new guide tree is subsequently recalculated from a previous alignment as many times as desired; the alignment is reconstructed using the Needleman–Wunsch algorithm for each pass.

The iterative refinement modes build on this foundation by adding steps that adjust the alignment back and forth until there is either solely no improvement in the WSP score (or the number of cycles has reached a set limit), or it adds both this WSP score and a T-Coffee-like consistency score between pair-wise and multiple alignments to the refinement procedure. Differences in the iterative methods that combine WSP and consistency scores are based on how the pair-wise scores are calculated, globally, locally with affine gap costs, or locally with generalized affine gap costs. Knowing which to choose, especially when dealing with sequences too diverged for the fast methods to work well, depends on the nature of the data. MAFFT’s algorithm page (http://mafft.cbrc.jp/alignment/software/algorithms/algorithms.html) explains where and when each mode is most appropriate. If sequences have full-length, but low, similarity, then the global option is appropriate (ginsi); if sequences have one similar domain among a bunch of “junk,” then the local affine gap option works best (linsi); whereas if sequence data are composed of multiple, yet alignable domains, then the local generalized affine gap scheme is appropriate (einsi).

MAFFT’s capability to handle large datasets and its speed is similar to or greater than Muscle’s in its faster modes; its results and capabilities are similar to T-Coffee in its slow, iteratively refined, optimized modes.
30.3.2.2 Coding DNA Issues

As discussed in the database searching section, all alignment algorithms, be they pair-wise, multiple, or database similarity searching, are far more sensitive at the amino acid level than at the DNA level. The signal-to-noise ratio is just so much better with amino acids, for all the reasons previously described. Therefore, database searching and sequence alignment both should always be done at the protein level, unless forced otherwise by dealing with noncoding DNA, or if the sequences are so very similar as to not cause any problems. Therefore, usually, if dealing with coding sequences, translate DNA to its protein counterpart, before performing multiple-sequence alignment. Even if dealing with very similar coding sequences, where the DNA can be directly aligned, it is often best to align the DNA along with its corresponding proteins. In addition to the much more reliable alignment, this also insures that alignment gaps are not placed within codons, which would not make sense biologically. DNA indels in coding regions need to be in multiples of three to avoid frameshift errors. Phylogenetic analysis can then be performed on the DNA rather than on the proteins, if desired. This is especially important when dealing with datasets that are quite similar, since the proteins may not reflect differences hidden in the DNA. Furthermore, many people prefer to run phylogenetic analyses on DNA rather than protein regardless of how similar they are—the multiple substitution models have a long and well-accepted history, and yet are far simpler. However, the more diverged a dataset becomes, the more random third and eventually first codon positions become, which introduces noise (error) into the analysis. Therefore, often third positions and sometimes first positions are masked out of datasets for phylogenetic analysis. Just like in most of computational molecular biology, one is always balancing signal against noise. Too much noise or too little signal, both degrade the analysis to the point of nonsense.

Several scripts and programs, as well as some web servers, can perform this sort of codon-based alignment, but they can be a bit tricky to run. Examples include mrtrans (Pearson, 1990) (also available in EMBOSS [Rice et al., 2000] as tranalign and in BioPerl [Stajich et al., 2002] as aa_to_dna_aln), transAlign (Bininda-Emonds, 2005), RevTrans (Wernersson and Pedersen, 2003), PAL2NAL (Suyama et al., 2006), and TranslatorX (Abascal et al., 2010). Some multiple-sequence alignment editors, such as SeaView (Galtier et al., 1996), can also help with this process.

Multiple-sequence alignment is much more difficult if forced to align nucleotides because a sequence does not code for a protein. Automated methods may be able to help as a starting point, but they are certainly not guaranteed to come up with a biologically correct alignment. The resulting alignment will probably have to be extensively edited, if it works at all. Success will largely depend on the similarity of the nucleotide dataset.

30.3.2.3 Reliability?

One liability of most global, progressive, pair-wise methods is they are entirely dependent on the order in which the sequences are aligned. Fortunately, ordering them from most similar to least similar makes biological sense and works well. However, most of the techniques are very sensitive to the substitution matrix and gap penalties specified. Some programs allow “fine-tuning” areas of an alignment by realignment with different scoring matrices and/or gap penalties; this can be extremely helpful. Regardless, any automated multiple-sequence alignment program should be thought of only as a tool for offering a starting alignment that can be improved upon, not the “end-all-to-meet-all” solution, guaranteed to provide the “one-true” answer. Although, in this post-genomics era, especially when dealing with giga-bases of data, it does make sense to start with the “best” solution possible. This is the premise of using a very accurate multiple-sequence alignment package, such as T-Coffee (Notredame et al., 2000), ProbCons (Do et al., 2005), POA (Lee et al., 2002), PCMA (Pei et al., 2003), or MAFFT (Katoh et al., 2002, 2005).

Whichever program used to create an alignment, always use comparative approaches to help assure its reliability. After the program has offered its best guess, try to improve it further. Think about it—a sequence alignment is a statement of positional homology—it is a hypothesis of evolutionary history. It establishes the explicit homologous correspondence of each individual sequence position,
each column in the alignment. Therefore, insure that it is as good as it can be. Be sure that it makes biological sense—align things that make sense to align! Beware of comparing “apples and oranges.” Be particularly suspect of sequence datasets found through text-based database searches such as NCBI’s Entrez (Schuler et al., 1996) or the EMBL/EBI sequence retrieval system (SRS) (Etzold and Argos, 1993). For example, don’t try to align receptors and/or activators with their namesake proteins or with each other. Be wary of trying to align genomic sequences with cDNA when working with DNA; the introns will create big problems, unless using software specifically designed for the task. Similarly, aligning mature and precursor proteins, or alternate splicing forms, from the same organism and locus, doesn’t make evolutionary sense, as one is not evolved from the other, rather one is the other. Watch for redundant sequences; there are lots of them in the databases. Some programs can automatically cull them from a dataset. If creating alignments for phylogenetic inference, make either paralogous comparisons (evolution via gene duplication) to ascertain gene phylogenies within one organism or orthologous (within one ancestral loci) comparisons to ascertain gene phylogenies between organisms (which should imply organismal phylogenies). Try not to mix the two without complete data representation. Otherwise, confusion can mislead interpretation, especially if the sequences’ nomenclature is inconsistent. These are all easy mistakes to make; try to avoid them.

Devote considerable time and energy toward developing the best alignment possible. Use biological understanding to help guide judgment. Look for conserved functional, enzymatic, regulatory, and structural elements and motifs—they should all line up. Searches of the PROSITE database of protein families and domains (Bairoch, 1992) for cataloged structural, regulatory, and enzymatic consensus patterns or “signatures” can help, as can searches of the Pfam protein family database of HMM profiles (Eddy, 1996, 1998), and de novo motif discovery tools like MEME (Bailey and Elkan, 1994) and MotifSearch (Bailey and Gribskov, 1998). Look for columns of strongly conserved residues such as tryptophans, cysteines, and histidines; important structural amino acids such as prolines, tyrosines, and phenylalanines; and conserved isoleucine, leucine, and valine substitutions. The conservation of covarying sites in ribosomal and other structural RNA alignments can be very helpful in refining RNA alignments. That is, as one base in a stem structure changes, the corresponding Watson–Crick paired base will change in a corresponding manner. This principle has guided the assembly of ribosomal RNA (rRNA) structural alignments at the Ribosomal Database Project at Michigan State University (RDP, Cole et al., 2007), at the University of Gent, Belgium European Ribosomal RNA database (Wuyts et al., 2004), and at the German SILVA database system (Pruesse et al., 2007). Editing alignments to insure that all columns are truly homologous should be encouraged, not discouraged. Dedicated sequence alignment editing software such as the genetic data environment (GDE) (Smith et al., 1994), Jalview (Clamp et al., 2004), Se-Al (Rambaut, 1996), and SeaView (Galtier et al., 1996) are great for this, but any editor will do, as long as the sequences end up properly formatted afterward. Structural alignment is the “gold standard,” but the luxury of having homologous, experimentally solved structures is often not available. Even with a structural alignment, there’ll often be questionable regions of sequence data within it.

These highly saturated regions have the property known as “homoplasy.” This is a region of a sequence alignment where so many multiple substitutions have occurred at homologous sites that it is impossible to know if those sites are properly aligned and, thus, impossible to ascertain relationships based on those sites. The primary assumption of all phylogenetic inference algorithms is most violated in these regions, and this phenomenon increasingly confounds evolutionary reconstruction as divergence between the members of a dataset increases.

Therefore, only analyze those sequences and those portions of an alignment that assuredly do align. This often means trimming down or somehow excluding those homoplastic regions, some of the internal gaps, and, minimally, the alignment’s extreme amino- and carboxy-termini (5′ and 3′ in DNA), which seldom align well. Sequence editors such as SeaView can do this. Those portions excluded will not be used in subsequent analyses. These decisions are somewhat subjective by nature, experience helps,
Computational Biology

and some software, such as Gblocks (Talavera and Castresana, 2007), ASaturA (Van de Peer et al., 2002), and T-Coffee (Notredame et al., 2000), has the ability to evaluate the quality of particular regions of an alignment. If, after all else, some region, or an entire sequence, just can’t be aligned, if there is any doubt, then do not use it. Cutting an entire sequence out of an alignment may leave columns of gaps across the entire alignment that will need to be removed. Most alignment editors have a function for closing these common gaps. The validity of all subsequent analyses is absolutely dependent upon the quality and accuracy of the input multiple-sequence alignment. Remember the old adage “garbage in—garbage out!” Some general guidelines to remember (Olsen, 1992) include the following:

- If the homology of a region is in doubt, then throw it out
- Avoid the most diverged parts of molecules; they are the greatest source of systematic error
- Do not include sequences that are more diverged than necessary for the analysis at hand

Biocomputing is always a delicate balance—signal against noise—and sometimes it can be quite the balancing act!

30.3.2.4 Applicability

So what’s the big deal about multiple-sequence alignment; why would anyone want to bother?

Multiple-sequence alignments can be

- Extremely useful in the development of PCR primers and hybridization probes
- Great for producing annotated, publication quality, graphics, and illustrations
- Required for building HMM profiles for remote homology similarity searching and alignment
- Invaluable for structural and functional analyses via homology inference
- Absolutely necessary for molecular evolutionary phylogenetic inference

Multiple-sequence alignments can be very helpful for designing phylogenetic-specific probes and primers by allowing for the clear visualization and localization of the most conserved and the most variable regions within an alignment. Depending on the dataset being analyzed, any level of phylogenetic specificity can be achieved. Areas of high variability in the overall dataset that correspond to areas of high conversation in phylogenetic category subset datasets can differentiate between universal and phylo-specific potential probe sequences. Any of several primer discovery programs, such as MIT’s Primer3 (Rozen and Skaletsky, 2000) or the commercial Oligo program (National Biosciences, Inc.), can be used to find and test the best primers within the target areas localized by this method.

Graphics prepared from multiple-sequence alignments can dramatically illustrate functional and structural conservation as it relates to sequence similarity. These can take many forms of all or portions of an alignment—shaded or colored boxes or letters for each residue or base (e.g., BoxShade [Hofmann and Baron] and PrettyPlot in EMBOSS [Rice et al., 2000]), cartoon representations (e.g., WebLogos [Schneider and Stephens, 1990]), running line graphs of overall similarity (as displayed by ClustalX [Thompson et al., 1997] and others), overlays of attributes, various consensus representations, etc.—all can be printed with high-resolution equipment, usually in color or gray tones. These can make a big difference in a poster or manuscript presentation.

Profiles are position-specific scoring matrices (PSSM) that describe a sequence alignment or a portion of an alignment. These powerful tools are created from an existing alignment of similar sequences and can be used to perform the most sensitive database remote similarity search possible. Another advantage of profile techniques is they can be used to build ever-larger multiple-sequence alignments much faster and more accurately than any of the progressive multiple-sequence alignment methods. This ability to easily create larger and larger multiple-sequence alignments is incredibly powerful and much faster than starting all over each time another sequence needs to be added to an alignment. Originally described by Gribskov et al. (1987, 1989), and then automated by NCBI’s PSI-BLAST (Altschul et al., 1997), later refinements have added more statistical rigor (see, e.g., Eddy’s HMM HMMer profiles and
Pfam profile database [1996, 1998] and Bailey and Elkan’s Expectation Maximization [1994]). Briefly described profiles upweight conserved regions such that the more highly conserved a residue is, the more important it becomes, and, in those profiles that allow gaps, gap insertions are penalized more heavily in conserved areas of the alignment than in variable regions.

Conserved sequence really does matter. In addition to the conservation of primary sequence, structure and function are also conserved in these crucial regions. In fact, recognizable structural conservation between true homologues extends way beyond statistically significant sequence similarity. The serine protease superfamily contains a good example. *Streptomyces griseus* protease A shows remarkably little sequence similarity when compared to the rest of the superfamily (expectation values $10^{-1.8}$ in a typical protein database search), yet it clearly is a serine protease as its 3D structure can be superimposed over most other members of the family with a root mean square deviation (RMSD) of less than 3 Å (Pearson, W.R., personal communication). These comparative principles are the premise of “homology modeling,” which works remarkably well. One of the best applications of these principles is PredictProtein, which uses weighted dynamic programming multiple-sequence alignment methods (MaxHom, Sander and Schneider, 1991), along with neural net technology, to predict protein secondary structure by the profile network method (PHD, Rost and Sander, 1993, 1994) at an expected 70.2% average accuracy for the three states helix, strand, and loop. Furthermore, even 3D modeling without crystal coordinates is possible, if a sequence is similar enough to an experimentally solved structure. In fact, the SWISS-MODEL (Guex and Peitsch, 1997; Guex et al., 1999) system at the ExPASy server in Switzerland, supported by SIB and GlaxoSmithKline, automates the homology modeling process, given similar enough sequence data to at least one solved structure, though as in most cases, a multiple-sequence alignment of solved structure sequences makes the theoretical model even more probable.

Finally, multiple-sequence alignments are required for phylogenetic inference. Many different computational methods can estimate the most reasonable evolutionary tree for a sequence alignment based on the assertion of homologous positions within that alignment. Computational tools that incorporate these methods include Phylogenetic Analysis Using Parsimony (PAUP*) (and other methods) (Swofford, 1989–2012), PHYLogeny Inference Package (PHYLIP) (Felsenstein, 1980–2012), MrBayes (Ronquist and Huelsenbeck, 2003), GARLI (Zwickl, 2006), and RAxML (Stamatakis, 2006). This is a huge and complicated field of study and will not be discussed further here. However, always remember that regardless of the algorithm used—any form of parsimony, all of the distance methods, all maximum likelihood techniques, and even all types of Bayesian phylogenetic inference—all make the absolute validity of the input alignment matrix their first and most critical assumption (but see, e.g., Lunter et al., 2005). The famous Darwinian evolutionist Theodosius Dobzhansky summarized the importance of phylogenetic analyses in 1973, provided as an inscription on the inner cover of the classic organic evolution text *Evolution*: “Nothing in biology makes sense except in the light of evolution” (Dobzhansky et al., 1977). Evolution provides the single, unifying, cohesive force to explain all life.

### 30.4 Complications

Sequence data format is a huge problem in bioinformatics. The major databases all have their own distinct format, plus many of the different programs and packages require their own. Clustal (Higgins et al., 1992) has a specific format associated with it. The Fast package (Pearson and Lipman, 1988) uses a very basic sequence format that many programs recognize. NCBI uses a library standard called Abstract Syntax Notation One (ASN.1), plus it provides GenBank flat file format for all sequence data. PAUP* (Swofford, 1989–2012), MrBayes (Ronquist and Huelsenbeck, 2003), and many other phylogenetic analysis packages have a required format called the NEXUS file (Maddison et al., 1997). Even PHYLIP (Felsenstein, 1980–2012) has its own unique data format. Standards have been argued over for years, such as using XML for everything, but until everybody agrees, which is not likely to happen, it just won’t happen. Fortunately several programs are available to convert formats back and forth between
the required standards; however, it can all get quite confusing. BioPerl’s SeqIO system (Stajich et al., 2002) and ReadSeq (Gilbert, 1990–2008) are two very helpful tools for format conversion. T-Coffee (Notredame et al., 2000) comes with one built in, and the SeaView (Galtier et al., 1996) editor recognizes NEXUS, Clustal, FastA, PHYLIP, and MASE format. Alignment gaps are still another problem. Different program suites may use different symbols to represent them. Most programs use hyphens “-” but some do not. Furthermore, not all gaps in sequences should be interpreted as deletions. Interior gaps are probably okay to represent this way, as regardless of whether a deletion, insertion, or a duplication event created the gap, logically they are the same. These are the indels mentioned previously. However, end gaps should not be represented as indels, because a lack of information before or beyond the length of any given sequence may not be due to a deletion or insertion event. It may have nothing to do with the particular stretch being analyzed at all. It just may not have been sequenced! These gaps are just placeholders for the sequence. Therefore, it is safest to manually edit an alignment to change leading and trailing gap symbols to “x”‘s, which mean “unknown amino acid,” or “n”‘s, which mean “unknown base,” or “?”‘s, which is supported by many programs, but not all, and means “unknown residue or indel.” This will assure that incorrect assumptions are not made, though most phylogenetic inference algorithms treat indels and missing data equivalently by default.

### 30.5 Conclusions

The comparative method is a cornerstone of the biological sciences. Multiple-sequence alignment and database searching are the comparative method on a molecular scale and are a vital prerequisite to some of the most powerful computational biology techniques available. Understanding something about the algorithms and the program parameters of each is the only way to rationally know what is appropriate. Knowing and staying well within the limitations of any particular method will avert much frustration. One point that needs to be emphasized is sequence analysis techniques generally have appropriate default parameters. This will usually work just fine, but it is a good idea to think about what these default values imply and adjust them accordingly, especially if the results seem inappropriate after a first pass with the default parameters intact.

Furthermore, the dramatic importance of a multiple-sequence alignment cannot be understated. All subsequent analyses are absolutely dependent upon it, especially phylogenetic inference. Also, if building multiple-sequence alignments for phylogenetic inference, do not base an organism’s phylogeny on just one gene. Many complicating factors can produce weird phylogenies: bad alignments, insufficient data, abjectly incorrect models, saturated positions (homoplasy), compositional biases, and/or horizontal gene transfer. Therefore, use several genes—the Ribosomal Database Project (Cole et al., 2007) provides a good, largely accepted alignment and phylogenetic framework with which other phylogenies can be compared. Anytime the orthologous phylogenies of organisms based on two different genes do not agree, there is either some type of problem with the analysis, or lateral gene transfer has occurred. Paralogous gene phylogenies are another story altogether and should be based, if at all possible, on sequences all from the same organism.

Gunnar von Heijne (1987) in his quite readable but very dated treatise, Sequence Analysis in Molecular Biology; Treasure Trove or Trivial Pursuit, provides an appropriate conclusion:

Think about what you’re doing; use your knowledge of the molecular system involved to guide both your interpretation of results and your direction of inquiry; use as much information as possible; and do not blindly accept everything the computer offers you.

He continues:

… if any lesson is to be drawn … it surely is that to be able to make a useful contribution one must first and foremost be a biologist, and only second a theoretician …. We have to develop better algorithms, we have to find ways to cope with the massive amounts of data, and above all we have to become better biologists. But that’s all it takes.
### Key Terms

**Affine:** An affine function is a linear function, described by the algebraic formula \( y = mx + b \).

**Expectation value (E-value):** The likelihood that a particular sequence alignment is due to chance. The value is dependent on sequence and database composition and size and on how often a researcher performs database searches. Most modern sequence database similarity programs such as BLAST and FastA provide this statistic based on the extreme value distribution. The closer the value is to zero, the more significant the match.

**Exon:** A defined stretch of DNA within a gene that is transcribed into a corresponding complimentary message RNA molecule and that is maintained after RNA processing into mature mRNA, such that only those portions not excised follow the relevant genetic code to be translated into corresponding amino acids.

**Gap penalties:** In the context of affine gap penalties, the creation or opening penalty is how many points a dynamic programming algorithm is penalized for imposing a gap in an alignment (the \( b \) in the previous equation).

The extension or lengthening penalty describes how many additional points the algorithm is penalized for each additional gap added to the first one after it is introduced (the \( x \) in the previous equation).

**Gene:** A defined stretch of DNA nucleotides that encodes either a protein, with or without introns and exons, or a structural or regulatory RNA (e.g., tRNAs, rRNAs, microRNAs, siRNAs, and snRNAs), which is inherited from one generation to the next.

**Global alignment:** As opposed to local alignment, which is the alignment of only the best regions within sequences, global alignment is the alignment of the full length of a sequence set and generally applies to the multiple-sequence alignment problem. However, it should be realized that global alignment can be restricted to subsequences within sequences, the distinction being that local alignment “picks” the best regions, whereas global alignment uses the full length of whatever is specified.

**HMM:** In sequence analysis, HMMs contain the statistical description of a recognized functional unit of some particular biological gene family or superfamily, assembled under the probabilistic model of hidden Markov chains (see profile).

**Homology:** An organ, trait, gene, or even DNA nucleotide position shared with a common ancestor. Homology, as opposed to sequence identity and similarity, can have no level. A sequence and, in fact, a position within an alignment, is either demonstrably related via evolution to another or not. Statistically significant similarity can argue for homology; however, a lack of statistically significant similarity cannot be used to argue against homology.

**Homology modeling:** The secondary and often tertiary, 3D structure of proteins can be inferred by alignment with proteins whose 3D structure has been experimentally determined. Obviously the more similar the sequences are, the more successful the theoretical model will be.

**Homoplasy:** A region of a sequence alignment where so many multiple substitutions have occurred at homologous sites that it is impossible to know if those sites are properly aligned and, thus, impossible to ascertain relationships based on those sites.

**Indel:** An indel is a gap introduced into a sequence alignment necessary to reconcile differing lengths due to differing evolutionary histories. It is impossible to ascertain whether an insertion or a deletion event created the discrepancy, hence the term indel.

**Intron:** A defined stretch of DNA within a gene that is transcribed into a corresponding complimentary message RNA molecule, but that is not retained after RNA processing into a mature mRNA and, therefore, is not translated into amino acids.

**Matrices:** A match matrix is the first step in dot plot techniques and in the visualization of the dynamic programming algorithm illustrated. Its cells contain the value each position receives for matching (aligning) respective X- and Y-axis characters.
A score matrix also has two meanings. In the context of dynamic programming, it is the matrix in which cell values have received initial match values adjusted by gap penalties and trace-back paths. The alternative meaning describes the values that amino acid residues or nucleotide bases receive for aligning with one another, also known as symbol substitution tables, or log-odds matrices, for example, the PAM and BLOSUM matrix series.

The trace-back path matrix delineates the alignments discovered by the dynamic programming algorithm; it illustrates the optimal paths through the matrix.

**Motif:** A motif is a described and cataloged region of a sequence, usually shorter than a domain and often, but not always, associated with some biological structural, functional, or regulatory role. Motifs are commonly represented as consensus patterns but are often described by HMM profiles as well.

**Orthology and paralogy:** Two major classes of sequence homology exist. Orthology describes homologous sequences present in different organisms as a result of speciation processes. Paralogy describes homologous sequences within the same organism as a result of gene duplication. Major confusion can result from mixing paralogues and orthologues in the same analysis.

**Profile:** A profile is a statistical description of a multiple-sequence alignment, commonly of a region or a motif within a multiple-sequence alignment. Profiles take many forms associated with the particular programs that create them, for example, ProfileBuild, HMMer, MEME, but always increase the importance of conserved residues or bases and decrease the importance of variable areas.

**Z-score and Z-score:** The Z-score is usually based on a normal Gaussian distribution and describes how many standard deviations a particular score is from the distribution’s mean, though it may be based on the extreme value distribution. This is confusingly in contrast to Bill Pearson’s z-score in the FastA programs that is a linear regression of the opt score against the natural log of the search set sequence length. The two values, Z and z, have entirely different magnitudes and should not be correlated.

**References**


Wetterstrand, K.A. 2012. DNA sequencing costs: Data from the NHGRI large-scale genome sequencing program available at: [http://www.genome.gov/sequencingcosts/](http://www.genome.gov/sequencingcosts/)


Zwickl, D.J. 2006. Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion. PhD dissertation, University of Texas, Austin, TX. [https://www.nescent.org/wg_garli/Main_Page](https://www.nescent.org/wg_garli/Main_Page)