The Evolution of Sequence Comparison Algorithms
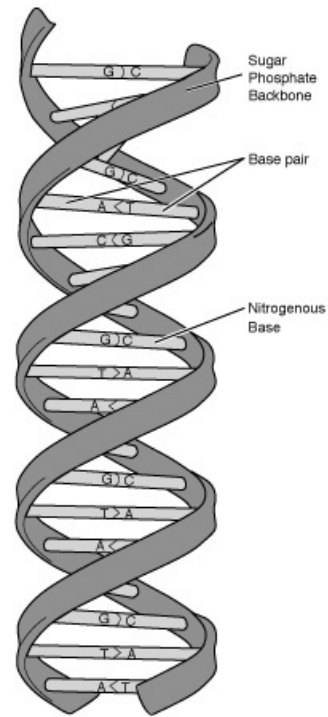
Article

1 author:

J Christopher Bare
Sage Bionetworks
51 PUBLICATIONS **1,513** CITATIONS

SEE PROFILE

The Evolution of Sequence Comparison Algorithms

J. Christopher Bare

CSEP 521 - Applied Algorithms - Spring 2005

University of Washington

Abstract

We explore algorithms for comparison of biological sequences. After digressing into some basic background on molecular biology, we review the literature on sequence comparison from biology and computer science journals. For the sake of starting at the beginning, we cover dynamic programming algorithms and continue through to the heuristic algorithms FASTA and BLAST.

Introduction

The analysis of nucleic acid and protein sequences is one of the oldest problems in computational biology. The comparison of biological sequences has become a fundamental aspect of molecular biology and its success as a scientific tool has transformed the science of biology.

Sequence comparison has proven to be an enormously powerful means to infer evolutionary and functional relationships. It has enabled biologists to find families of related genes and proteins and to clarify the workings evolution at the molecular level.

Biologists use sequence comparison algorithms to:
- measure the degree of difference of similarity between sequences
- construct optimal alignments
- search for similar sequences in databases.

Large databases of sequence data have been compiled and it is standard practice to compare and submit newly discovered sequences to the databases. The human genome and the genomes of hundreds of (mostly microbial) model organisms have been completely sequenced [GOLD]. The Genbank nucleic acid database at the NCBI held roughly 45 billion base pairs as of 2004 [NCBIb]. The human genome alone is 3 billion base pairs long.

The compilation of sequence data is widely seen as a first step to understanding the developmental, regulatory, and metabolic processes of the cell. The current frontier is to integrate the understanding of the complex networks of interacting biochemical pathways within the cell into a systems level model of cellular processes.

The increasingly vast quantity of available sequence data has motivated the development of sequence comparison algorithms of increasing sophistication and algorithmic efficiency.

Biological Background [MBC]

The Central Dogma of Molecular Biology in a Nutshell

Common to all life is the flow of information within a cell from DNA to RNA to protein, first established in the 1950's by Francis Crick. DNA serves as the storage repository of genetic information. Genes are transcribed to mRNA, which carries the information to ribosomes. A ribosome is a molecular factory that constructs proteins using mRNA as a template in a process called translation.

Nucleic acids, DNA and RNA, along with proteins are the central molecules of biology. Nucleic acids consist of long chains of chemical subunits called nucleotides or bases. Similarly, a protein is a long chain of amino acids. In either case, the sequence of a molecule defines the exact order of the subunits along the length of the chain. Biological sequences can be represented as strings over an alphabet that contains one letter for each

type of chemical subunit.

Molecule	Alphabet
DNA	{a,t,g,c}
RNA	{a,u,g,c}
Protein	{A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y}

A three letter chunk of nucleic acid, called a codon, translates into one amino acid. The mapping is defined by the genetic code. A gene is said to code for a protein if the sequence of the gene translates into the sequence of the protein. With few exceptions, the genetic code is universal.

Similarity and Homology

Evolution relies on genetic mechanisms that allow genes to be duplicated, modified, and recombined. Proteins that perform identical or related functions often have similar sequences. Such proteins are believed to have evolved from a single ancestral gene which duplicated. Over the course of evolution, mutations accumulate to produce related proteins possibly with new functions. Additionally, many proteins seem to have derived by a process of domain shuffling, joining preexisting blocks of protein sequence into new combinations.

Similarity due to shared ancestry is called homology. The term homology is frequently used as a synonym for similarity, but it is important to note that the true meaning of homology is a statement about evolutionary history. It is entirely possible for two sequences to be related by descent but have relatively little sequence similarity and conversely for proteins with similar structure and function to arise separately. Properly speaking, a high degree of sequence similarity constitutes evidence for homology.

Evolutionary Distance

If we have two homologous sequences, we may want to know how much divergence has taken place since the duplication event that separated the two sequences from the common ancestor. It is precisely this evolutionary distance that sequence comparison algorithms attempt to compute.

Evolutionary distance can be thought of in several equivalent ways:
- the minimal number of mutation events that separates one sequence from the other.
- the minimal number of mutations between the sequences and their common ancestor.
- the amount of time since divergence from the common ancestor.

These are all essentially equivalent, given the assumption that mutations accumulate at a constant rate over time on average.

Nonrandomness of Biological Sequences

It would be a mistake to think of biological sequences or their evolution as completely random. They are highly nonrandom in interesting ways. For instance, mutations that

result in loss of function are likely to effect the survival of the organism and are therefore less likely to become established. Some sequence changes are more likely to damage a functioning protein than others. This is true in at least three ways – some mutations are silent due to degeneracy of the genetic code, some amino acids substitutions are more drastic others, and some regions of a protein's sequence are more important to its function than others.

The genetic code has a property called degeneracy which means that one or more distinct codons may code for the same amino acid, for example GAA and GAG both code for glutamic acid. Degeneracy comes about because there are 4^3 or 64 possible codons but only 20 amino acids plus the stop codon. As a result, an error at the DNA or RNA level may be silent. It may have no effect at all on the protein product.

Some pairs of amino acids are more similar in their chemical properties than others. Substitutions of one amino acid for another with very different chemical properties is more likely to result in a non-functional protein, while substitutions between amino acids of similar properties will be more likely to preserve function.

Over the course of evolution, some regions along the length of a protein may change much less others. These highly conserved regions are often chemically active sites, binding sites, or are otherwise crucial to the function of the protein. Changes to these regions would be likely to result in a loss of functionality. Changes to other less critical regions are more tolerated and tend to be more abundant.

As a result of the evolutionary process of duplication and modification comes the central biological importance of sequence comparison: Sequence similarity often implies related structure and function. [Gus97 Ch.10]

Computation on Biological Sequences

In looking at molecular biology in a simplified (or oversimplified) way, we ignore significant and fascinating biological detail. But, the biological ideas necessary for basic understanding sequence comparison are just these:
· Biological sequences can be thought of as strings over the appropriate alphabet.
· All mutations are not equal. Some have greater biological effect than others.
· Evolutions proceeds by duplication and modification.
These ideas make computation on biological sequences immediately tractable. This tractability has led to some phenomenally successful applications of computer science to biological problems.

The Problem

The sequence comparison problem is to quantify the degree of similarity or, equivalently the distance between the sequences. An alignment may be constructed as an intermediate step or as a goal in itself.

Edit Distance

The exact definition of similarity or distance varies by application, but is usually formulated as a set of edit operations (or mutations to stick with biological metaphors) that are then used to transform one sequence into the other.

One such scoring system is the edit distance. Here the operations are single character insertions, deletions, and substitutions. The edit distance is the smallest number of such operations required to transform one string into the other. More generally we can assign a cost to each operation and find the sequence of operations with the minimal cost. The set of possible operations and the cost of each operation constitute a scoring system. [JP04 Ch.6.4]

Optimization and Duality

While there can be any number of sequences of operations that have the desired outcome, finding the distance requires that we find a sequence of operations with minimal cost. This means the problem of computing distance is an optimization problem.

Depending on the scoring system, the finding the optimal may involve minimizing or maximizing. From the minimal distance point of view, we may assign a distance of 0 for an exact match and assess a cost for mismatches and gaps. From the perspective of maximizing similarity, the we may award points for exact matches and extract a penalty for mismatches and gaps. Such scoring systems can be inverted and scaled to switch between the perspectives of maximizing similarity or minimizing distance. So we can conclude that maximum similarity and minimum distance are dual problems and computing either one is equivalent to computing the other. [Mye91]

The consequence of this duality is that finding evolutionary distance and optimal sequence alignment involve essentially the same computation.

The Sequence Comparison Problem

Formally stated, the problem of sequence comparison (aka approximate string matching) is: [Gus97 Ch.11, Nav01]

Given an alphabet Σ, a distance function $d : \Sigma^* \times \Sigma^* \to \mathbb{R}$, and two strings on the alphabet $s \in \Sigma^*$ and $t \in \Sigma^*$ and compute the value of $d(s,t)$.

In addition, comparison enables similarity search. If t is large in comparison to s, we may want to find all substrings $t_{i,j}$ of t such that $d(s, t_{i,j}) \leq r$ where r is a threshold or cutoff value.

Scoring Systems

As stated earlier, a scoring system defines a set of operations with an associated cost for each. Several related problems are special cases of the sequence comparison problem which vary mainly in the scoring systems.

The **longest common subsequence**, often abbreviated LCS, is a well known problem in computer science. A subsequence of a string s is a subset of the characters in s arranged in the same order as in the original string. Computing the LCS can be thought of as performing sequence comparison allowing insert and delete operations but no mismatches. The score computed will be the length of the longest common subsequence and the LCS is the alignment with the highest possible number of matches.

The **Hamming distance** is simply the number of positions that differ between two strings of equal length. So, the only operation allowed is substitution and the score for any such operation is one.

Edit distance, already mentioned, is also called **Levenshtein distance**, after Vladimir Levenshtein who published on the topic in 1965. The edit distance between two strings *s* and *t* is the minimal number of single character deletions, insertions, and substitutions required to transform *s* into *t*.

Other kinds of operations are possible, for example transposition. In the scoring systems above, the operations work on one character at a time. We might define operations that work on several characters at once. This is especially important when considering gaps in biological sequences. [Gus97, Nav01]

Substitution Matrices and Gap Penalties

Scoring systems for biological applications are usually framed as a substitution matrix and possibly a gap penalty function. A substitution matrix gives an individual score to each possible substitution from one character in the alphabet to another. A gap character can be added to the substitution matrix and scored like any other character. Alternatively, gap penalties may be calculated as a function of the length of the gap. This usually involves a large penalty to open a gap but a much smaller penalty for extending the gap.

The substitution matrix is a table with a substitution score for each possible pair of letters in the alphabet. For amino acids comparisons in particular, we make use of a substitution matrix that scores some mismatches as being more similar than others. Margaret Dayhoff pioneered this approach with the PAM [DSO78] series of matrices. The BLOSUM [HH92, HH93] series was developed later with the benefit of more sequence data.

Both of these scoring matrices are based on observed substitution rates. These scoring matrices capture information about the chemical similarity among amino acids and about their relative frequency in nature and also the influence of the molecular and evolutionary mechanisms at work. Because amino acid sequence comparisons can take advantage of this extra information, they are generally more sensitive than comparing nucleic acid sequences.

The BLOSUM 62 matrix [HH92], for example, is popular choice. Matches given a positive similarity score. Most mismatches correspond to negative numbers, but some conservative changes also have positive scores. Note also that some identities are scored higher than others. This reflects the relative frequencies of the occurrence of amino acids in nature.

```
    A   R   N   D   C   Q   E   G   H   I   L   K   M   F   P   S   T   W   Y   V
A   4  -1  -2  -2   0  -1  -1   0  -2  -1  -1  -1  -1  -2  -1   1   0  -3  -2   0
R  -1   5   0  -2  -3   1   0  -2   0  -3  -2   2  -1  -3  -2  -1  -1  -3  -2  -3
N  -2   0   6   1  -3   0   0   0   1  -3  -3   0  -2  -3  -2   1   0  -4  -2  -3
D  -2  -2   1   6  -3   0   2  -1  -1  -3  -4  -1  -3  -3  -1   0  -1  -4  -3  -3
C   0  -3  -3  -3   9  -3  -4  -3  -3  -1  -1  -3  -1  -2  -3  -1  -1  -2  -2  -1
Q  -1   1   0   0  -3   5   2  -2   0  -3  -2   1   0  -3  -1   0  -1  -2  -1  -2
E  -1   0   0   2  -4   2   5  -2   0  -3  -3   1  -2  -3  -1   0  -1  -3  -2  -2
G   0  -2   0  -1  -3  -2  -2   6  -2  -4  -4  -2  -3  -3  -2   0  -2  -2  -3  -3
H  -2   0   1  -1  -3   0   0  -2   8  -3  -3  -1  -2  -1  -2  -1  -2  -2   2  -3
I  -1  -3  -3  -3  -1  -3  -3  -4  -3   4   2  -3   1   0  -3  -2  -1  -3  -1   3
L  -1  -2  -3  -4  -1  -2  -3  -4  -3   2   4  -2   2   0  -3  -2  -1  -2  -1   1
K  -1   2   0  -1  -3   1   1  -2  -1  -3  -2   5  -1  -3  -1   0  -1  -3  -2  -2
M  -1  -1  -2  -3  -1   0  -2  -3  -2   1   2  -1   5   0  -2  -1  -1  -1  -1   1
F  -2  -3  -3  -3  -2  -3  -3  -3  -1   0   0  -3   0   6  -4  -2  -2   1   3  -1
P  -1  -2  -2  -1  -3  -1  -1  -2  -2  -3  -3  -1  -2  -4   7  -1  -1  -4  -3  -2
S   1  -1   1   0  -1   0   0   0  -1  -2  -2   0  -1  -2  -1   4   1  -3  -2  -2
T   0  -1   0  -1  -1  -1  -1  -2  -2  -1  -1  -1  -1  -2  -1   1   5  -2  -2   0
W  -3  -3  -4  -4  -2  -2  -3  -2  -2  -3  -2  -3  -1   1  -4  -3  -2  11   2  -3
Y  -2  -2  -2  -3  -2  -1  -2  -3   2  -1  -1  -2  -1   3  -3  -2  -2   2   7  -1
V   0  -3  -3  -3  -1  -2  -2  -3  -3   3   1  -2   1  -1  -2  -2  -2  -3  -1   4
```
The BLOSUM 62 matrix.

Biologically, a scoring system constitutes a model of molecular evolution.

The Algorithms

The algorithms we will discuss fall into two categories. The earlier three articles describe dynamic programming algorithms that find exact optimal solutions for a given scoring system. FASTA and BLAST are heuristic algorithms motivated by the specific problem of similarity search against large sequence databases.

Dynamic Programming Algorithms	
1970	Needleman Wunsch
1974	Sellers
1981	Smith Waterman
Heuristic Algorithms	
1985	FASTP
1988	FASTA
1990	BLAST
1997	Gapped BLAST

The Basic Dynamic Programming Algorithm

Alignment of two sequences and their distance can be efficiently computed in O(mn) time by algorithms based on the technique of dynamic programming. If m is the length of string s and n is the length of string t, the algorithm constructs an $m{\times}n$ table with each axis labeled by one of the strings.[Gus97, JP04]

```
           t    g    c    a    t    g    c    a    t    a
    ---------------------------------------------------
  |    0   -5  -10  -15  -20  -25  -30  -35  -40  -45  -50
 t|   -5    5    0   -5  -10  -15  -20  -25  -30  -35  -40
 g|  -10    0   10    5    0   -5  -10  -15  -20  -25  -30
 g|  -15   -5    5    6    1   -4    0   -5  -10  -15  -20
 a|  -20  -10    0    1   11    6    1   -4    0   -5  -10
 t|  -25  -15   -5   -4    6   16   11    6    1    5    0
 c|  -30  -20  -10    0    1   11   12   16   11    6    1
 g|  -35  -25  -15   -5   -4    6   16   11   12    7    2
 a|  -40  -30  -20  -10    0    1   11   12   16   11   12
 t|  -45  -35  -25  -15   -5    5    6    7   11   21   16
 a|  -50  -40  -30  -20  -10    0    1    2   12   16   26
```

Table produced by the dynamic programming algorithm with a scoring system of +5 for a match, -4 for a mismatch, and -5 for a gap character.

The value of each cell in the table is the distance between the two substrings $s_{0..i}$ and $t_{0..j}$ given by the following recurrence: [Mye91]

$$d(i,j)=\begin{cases} \text{if } i=0 \ \&\ j=0: & 0 \\[4pt] \text{if } i>0 \ \&\ j=0: & d(i-1,0)+\delta(s(i),\text{-}) \\[4pt] \text{if } i=0 \ \&\ j>0: & d(0,j-1)+\delta(\text{-},t(j)) \\[4pt] \text{otherwise:} & min\begin{pmatrix} d(i-1,j)+\delta(s(i),\text{-}), \\ d(i,j-1)+\delta(\text{-},t(j)), \\ d(i-1,j-1)+\delta(s(i),t(j)) \end{pmatrix} \end{cases}$$

The dependencies in the recurrence are such that any cell in the table depends only on the three other cells – the cells above, to the left, and diagonally to the upper left. To remain consistent with the dependencies, we can construct the table row by row, column by column, or in anti-diagonals. The final value entered in cell (*m,n*) is the distance between the two strings.

The alignment can be spelled out by saving backpointers as we fill in the table or working backwards through the table after it is constructed. If there is more than one optimal alignment, the backpointers will branch forming a directed acyclic graph. Traversing the DAG gives the optimal alignments.

```
tggatcg-ata
tgcat-gcata

tggat-cgata
tgcatgc-ata
```

Optimal Alignments

All of the dynamic programming algorithms presented here are variations of this algorithm. The heuristic algorithms are conceptually based on the dynamic programming algorithm and make use of its table and of modified forms of the algorithm itself.

This algorithm is usually attributed to Needleman and Wunsch [NW70] when applied to global alignment, although it was discovered separately in the context of signal processing [Nav01] and an almost identical algorithm for longest common subsequence is called a "folk algorithm" in [CLRS].

One of the indications for the application of dynamic programming is the property of **optimal substructure**. [CLRS Ch.15] This means that the solution to the problem can be defined recursively in terms of solutions to smaller instances of the same problem. This can be observed here by noticing that the distance of two strings $d(s_{0..i},\ t_{0..j})$ is defined in terms of $d(s_{0..i-1},t_{0..j-1})$, $d(s_{0..i},t_{0..j-1})$, and $d(s_{0..i-1},t_{0..j})$. [Mye91]
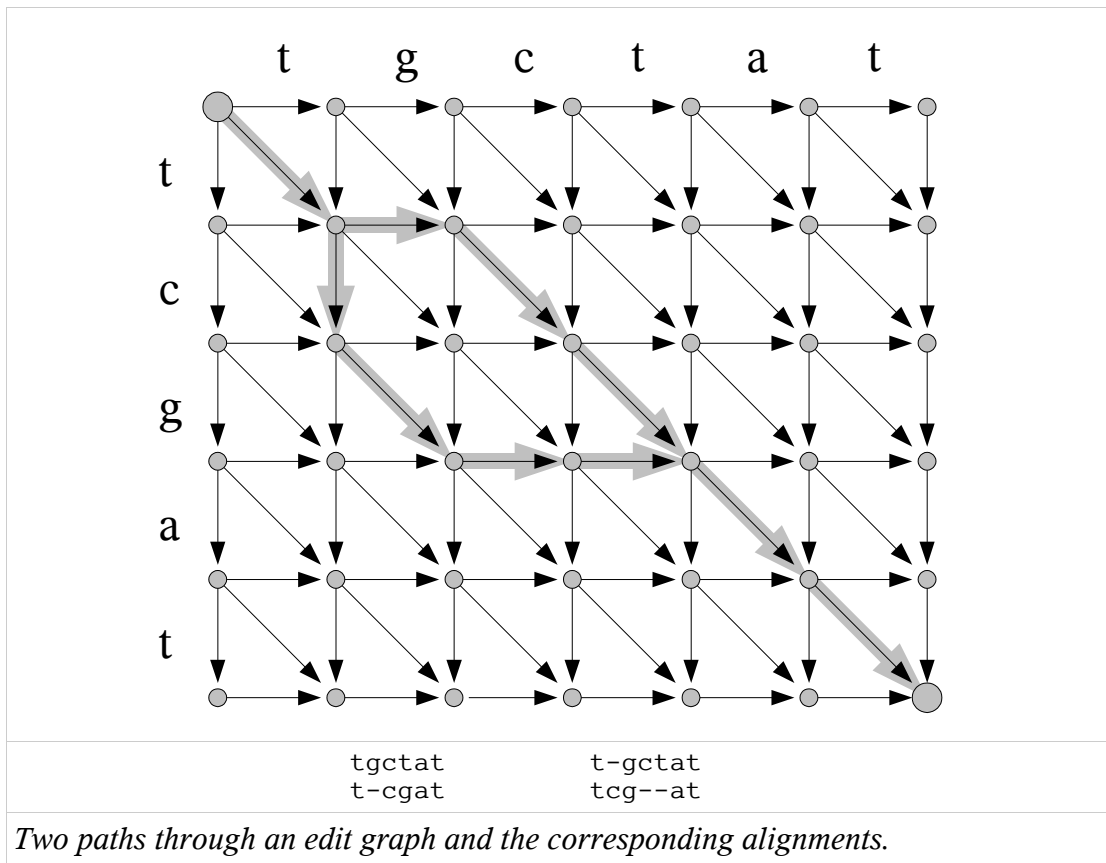
Edit Graph

The sequence comparison problem has an interesting graph theoretic interpretation called an edit graph. [Gus97, JP04] We can construct the edit graph by creating a node for each pair of characters (si, tj) from the sequences s and t to be compared. Arranging the nodes in an $(m+1)\times(n+1)$ grid gives a graph with a similar structure to the dynamic programming table. We now a add three kinds of directed edges:

Direction	For each node where	Edge	Label	Weight
↘	i>0 and j>0	(i-1,j-1)→(i,j)	(si, tj)	δ(si, tj)
↓	i>0	(i-1,j)→(i,j)	(si,-)	δ(si, -)
→	j>0	(i,j-1)→(i,j)	(-,tj)	δ(-, tj)

Diagonal edges represent aligning character si with character tj. Vertical edges represent deletions and horizontal edges represent insertions. We then label the node at (0,0) as the source node and the node at (m,n) as the sink.

The graph can be thought of as a finite automaton that accepts all alignments of the two strings. Assigning weight to the edges using the scoring system, we can find the optimal alignment using by finding the shortest path through the graph. [Mye91]


```
        tgctat              t-gctat
        t-cgat              tcg--at
```

Two paths through an edit graph and the corresponding alignments.

Needleman and Wunsch

The general dynamic programming algorithm described previously is commonly referred to as "the Needleman Wunsch algorithm" when it is applied to **global sequence alignment**. The exact algorithm described in the 1970 paper [NW70] is a variant which runs in O(nm(n+m)) time and allows for the consideration of blocks of several gap characters as a single unit. It differs from the general dynamic programming algorithm in that the recurrence used is: [SW81a]

$$C(i,j) = \begin{cases} \text{if } i=0 \ \& \ j=0: & 0 \\ \text{if } i>0 \ \& \ j=0: & gap(i-1) \\ \text{if } i=0 \ \& \ j>0: & gap(j-1) \\ \text{otherwise:} & \delta(s(i),t(j)) + \\ & max\begin{pmatrix} C(i-1,j-1), \\ max_{1<k\leq i}(C(i-k,j-1)+gap(k)), \\ max_{1<k\leq j}(C(i-1,j-k)+gap(k)) \end{pmatrix} \end{cases}$$

In this formulation, C(i,j) is the cumulative score at position i,j in the table. $\delta(s(i), t(j))$ is the score given for matching the ith symbol in string *s* with the jth symbol in string *t*. The penalty for a gap of size k is defined by *gap*(k). In its simplest form, $\delta(a,b)$ can be defined as 1 for a match and 0 for a mismatch. With a gap penalty of zero, this score is equivalent to the length of the LCS, but more biologically accurate gap penalties can be represented.

The difference between this algorithm and the general one can be seen by considering the maximization over the preceding row and column, giving the longer running time. Because we maximize score over parts of the previous row and column, our alignment may trace a more complicated path through the table. Gaps are represented not as vertical or horizontal edges, but by edges that make longer jumps through the table. It is therefore necessary to keep backpointers.

In the original paper, the table was filled in starting at the end of the sequences at position (m,n) in the table. Probably, this was done so that traversing the backpointers would give the alignment in forwards order. The direction makes no difference to the final result and it is reversed the above recurrence for easier comparison with the other recurrences.

The original illustrations from the Needleman and Wunsch paper showing a partially completed table and a complete table with backpointers. In the first illustration a match between two R characters is being considered. The cell gets one point for the match. To be added is the maximum score attainable by aligning the substrings OCLCRPM and CKCRBP, which we can see is 4. The final score for the cell will therefore be 1+4 = 5.

Remarkably, Needleman and Wunsch anticipated many of the major developments in sequence comparison that came afterwards. For example, they state that, "The sophistication of the comparison is increased if [...] each cell value is made a function of [...] any theory concerned with the significance of a pair of amino acids." Such theories motivated the development of substitution matrices such as PAM and BLOSUM. Needleman and Wunsch also comment on complex block gap penalty functions, multiple alignments, local alignment, and evaluating the statistical significance of a match. Not bad for 1970.

Sellers

The algorithm described by Sellers [Sel74] is identical to the general dynamic programming algorithm previously described. Sellers set biological sequence comparison on mathematically rigorous foundations by showing that evolutionary distance conforms to the definition of a metric on biological sequences.

The formation of a evolutionary distance as a metric also becomes important in the construction of **phylogenetic trees**, which are essentially family trees for sequences. [JP04, SW81a]

> **Definition:** a metric space (*M*, *d*) is a set *M* and a function *d* describing the distance between elements of *M*. The metric is a function $d : M \times M \rightarrow \mathbf{R}$ (where $\mathbf{R}$ is the set of real numbers) which satisfies the following conditions:
>
> for all *x*, *y*, *z* in *M*,
>
> $d(x, x) = 0$
> $d(x, y) \geq 0$
> $d(x, y) = 0$ implies *x=y*
> symmetry: $d(x, y) = d(y, x)$
> triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$

Later, Sellers formulated the dynamic programming algorithm as a solution to the similarity search problem. This is significant mainly because the O(mn) dynamic programming algorithm is too slow for the job, motivating the later development of faster heuristic algorithms for similarity search.

Smith Waterman

It is often the case that one or more regions of high similarity will exist in two sequences that are otherwise dissimilar. This can come about because functionally important parts of a protein tend to be more highly conserved during the course of evolution. These highly conserved regions usually indicate a chemically active site or other biologically important part of the protein. Also, successful protein domains tend to get reused. Common motifs appear to have been repeatedly shuffled and recombined. For these reasons, local alignment is often more biologically relevant than global alignment.

The Smith-Waterman algorithm [SW81] differs from the previously discussed algorithms in that it computes **local alignments**. To illustrate the difference between global and local alignments, we have two alignments of the same DNA sequences. The first shows a weak global alignment while the second shows a stronger local alignment.

```
        TTGACACCCTCC-CAATTGTA
          ::   ::    ::   :
        ACCCCAGGCTTTACACAT---


        ---------TTGACACCCTCCCAATTGTA                 TTGACAC
             ::  :::::                    or          :: ::::
        ACCCCAGGCTTTACACAT-----------                 TTTACAC
```

Global vs. Local Alignment

The change in the algorithm is very small. We simply add a zero term into maximization in the recurrence.

$$C(i,j) = \begin{cases} \text{if } i=0 \text{ or } j=0: & 0 \\ \\ \text{otherwise:} & max \begin{pmatrix} 0, \\ C(i-1,j)+\delta(s(i),-), \\ C(i,j-1)+\delta(-,t(j)), \\ C(i-1,j-1)+\delta(s(i),t(j)) \end{pmatrix} \end{cases}$$

The effect is to allow regions of low similarity at the beginning of the strings to be ignored. The alignment starts "keeping score" only when the score is above zero. We then select the cell in the table with the highest value as the end of the highest scoring local alignment. This algorithm requires that the scoring function be negatively biased so that regions of low similarity will have negative scores.

Returning to the graph theoretic view, the Smith-Waterman algorithm can be formulated by adding a new set of edges to the edit graph. We add a zero weight edge from the source to every other node. It is these "free ride" edges that allow low similarity regions to be skipped over.

The **k-best** variation of the Smith-Waterman algorithm [WE87] returns non-overlapping local alignments that score at or above some preset level. This is particularly useful if two sequences share multiple regions of similarity interrupted by dissimilar regions and with the order of the similar regions rearranged.

The Similarity Search Problem

Given a pattern p of length n and a text t of length m, a scoring function d, and a cutoff k, find all positions j where $1 \leq j \leq m-n+1$ and $t_i t_{i+1}..t_j$ and $p_1 p_2.. p_n$ have a distance of at most k, or $d(t_i t_{i+1}..t_j, p_1 p_2.. p_n) \leq k$. [Nav01]

Both FASTA and BLAST are heuristic similarity search algorithms motivated by the problem of finding sequences in a large database that are similar to a query sequence. Both employ a heuristic filtration strategy. [Gus97, Kah04] These algorithms still require O(mn) time, but greatly reduce the constants of proportionality. [Mye91]

FASTP and FASTA

FASTA [PL88] was created as a refinement to the ideas of its predecessor, FASTP [LP85]. Both algorithms work in a complicated series of stages:

1. Perform filtration using k-mers and "diagonal" method.
2. Re-score best diagonals using substitution matrix to build "initial regions"
3. [FASTA only] Join high scoring initial regions
4. Optimize alignments using dynamic programming

The first and most important stage is filtration, which quickly finds positions of potential

matches or "hits". The filtration stage works on the premise that any substring of the text t that approximately matches the query pattern p is very likely to contain a smaller substring of length k that is an exact match for some substring of p. The term "k-mer" is used to refer to the substrings of length k. Typical values for k are 2 for protein sequences and 6 for nucleic acid.

FASTA and FASTP generate all k-mers of the query string and perform exact string matching on the database using the hashing technique of the Rabin Karp algorithm [CLRS Ch.32]. The hash function has the essential property that if the hash for a substring $s_{j-k+1}..s_j$ is known, the hash for the substring $s_{j-k+2}..s_{j+1}$ can be computed in constant time. We could imagine hashing either the query string, the text, or both. Because of the size of the databases, in practice the hash table is constructed for the query string and not for the database. The matching k-mers in the database can be found in time $O(rm)$ where r is the number of k-mers in the query string and m is the length of the database.

The filtering stage continues by considering the diagonals of the dynamic programming table. A positive score is assigned to each k-mer match and a negative score is assigned to spaces between matches. The algorithm then selects a fixed number of diagonals with highest scores.

Exactly matching k-mers plotted on the dynamic programming table.
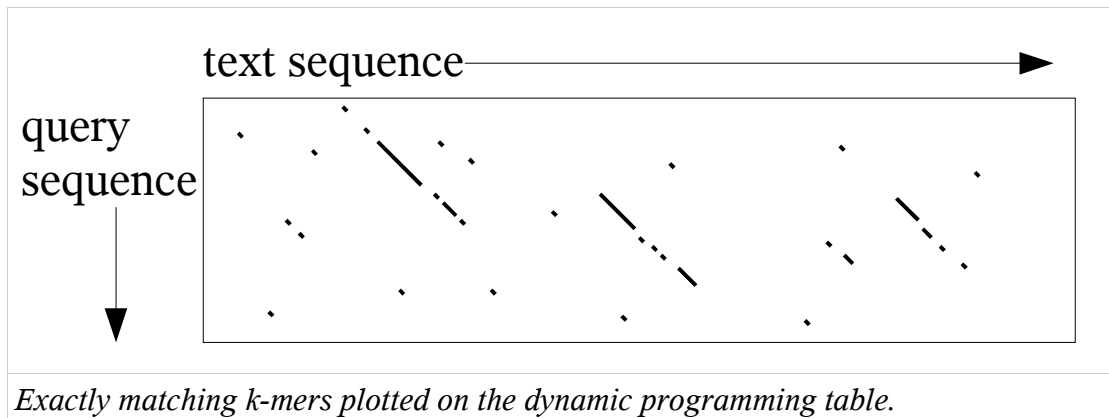
In the second stage, the high scoring diagonals are re-scored using a substitution matrix for greater sensitivity, but still without considering insertions or deletions. For each diagonal, the maximally scoring subregion is selected and called the "initial region". Here, FASTA introduces a third step not present in FASTP. In this stage, the algorithm tries to join together initial regions from nearby diagonals to give a larger higher scoring substring alignments which can include insertions and deletions. In the final stage, the high scoring regions of the sequences are realigned using a modified dynamic programming method.

The FASTP/FASTA algorithm is heuristic in the filtration stage because it is not guaranteed that a high scoring match actually does contain an exactly matching k-mer. The scoring and join of diagonals seems to have a more ad-hoc feel as well.

The BLAST Algorithm

The BLAST [AGM90] algorithm arrived in 1990 and quickly became the standard method for searching the major biological sequence databases. Like FASTA, BLAST proceeds in stages.

1. Create a list of all short sequences, called words, that score above a threshold value T when aligned with the query sequence.
2. Scan the database for exact matches with these words.
3. Extend each exact matches to see if it is contained within an approximately matching substring of score S or greater.

Also like FASTA, the primary speedup comes from a heuristic filtering stage which involves performing exact matches on smaller pieces of the query string.

BLAST applies some sophistication to the filtering stage by using the concept of a neighborhood. The T-neighborhood of a string w is the set of all strings that align with w with a score greater than the threshold value T. For each of the m-k+1 k-mers in the query string p, where m=|p|, BLAST generates the T-neighborhood using a substitution matrix without the possibility of gaps. The algorithm avoids considering insertions or deletions at this stage because the goal is to enable exact string matching. We end up with a list of words, each of which is in the T-neighborhood of some k-mer of the query string. The key idea behind BLAST is that if we precompute the word list, we can use an exact multiple string matching algorithm to find all occurrences the words in the database.

In order to scan the database for matching k-mers, BLAST constructs a finite automaton which accepts any string from the word list. The finite automaton can be constructed in O(kz) time where z is the size of the word list. The database can then be scanned in time linear in the size of the database. This part of the algorithm is related to the multiple pattern matching algorithm of Aho and Corasick which makes use of a keyword tree.

When a match is found between a word and a k-mer in the database, BLAST forms a segment pair. The segment pair is a pair of matching strings of equal length, one from the query sequence and the other from the database. The algorithm tries to extend the segment pair by lengthening the strings in either direction. In other words, the match is extended along its diagonal in the dynamic programming table. When the score can no longer be improved either by lengthening or shortening the segment pair, we have found the locally maximal segment pair. If the score of the locally maximal segment pair is higher than the cutoff value S, the match is reported.

In the original BLAST, each matching segment pair found is used as a seed for an ungapped alignment. In a newer revision described in a 1997 paper [AMS97], the ability to generate gapped alignments in the extension phase was added [BA]. The newer algorithm uses heuristic modification of dynamic programming for extension of hits into gapped local alignments.

```
// BLAST Algorithm
//   parameters:  seq     query sequence, a string
//                db      database, a long string
//                w       size of words
//                d       scoring function
//                T       threshold score for words
//                S       cutoff score for acceptable matches

BLAST(seq, db, w, d)

        // eliminate low complexity regions from the query sequence
        seq = filter(seq)

        // generate word list
        words = empty_list
        m = length(seq)
        for i = 1 to m-w+1
                word = substring(seq,i,w)
                words.append( generate_T_neighborhood(word, T, d) )

        // find words in database
        fa = construct_finite_automaton(words)
        fa_state = fa.start_state()
        n = length(db)
        for j = 1 to n

                // transition to next state in the FA
                // based on the next character in the db sequence
                fa_state = fa.transition(fa_state, db[j])

                // if the FA has found an exact match with a word
                if fa.isAccepting(fa_state)

                        // find locally maximal segment pair
                        // and return start and end indices
                        // in seq and in db (s1,e1,s2,e2)
                        i = fa.keyword_position(fa_state)
                        (s1,e1,s2,e2) = find_LMSP(seq,i,db,j,w,d)

                        score = compute_score(seq,s1,e1,db,s2,e2,d)
                        if score >= S
                                report_match(seq,s1,e1,db,s2,e2,score)
```

An attempt at a rough sketch of the BLAST algorithm in pseudo-code. A few substantial steps are left as subroutines.

BLAST Statistics

One key feature of BLAST is that it determines the statistical significance of its results. For each match, BLAST reports an E-value. The Expect value (E) describes the number of matches with score at least S expected to occur by chance in a database of a particular size. A smaller E-value indicates more significant results. There is extensive theory behind BLAST, based on work by Karlin and Altschul [KA90] on the statistical properties of sequence alignments. Readers interested in this topic are referred to [KA90, Alt91, EG01, NCBIa, Gish].

Conclusions and Directions for Further Study

There has been quite a bit of interdisciplinary crossover surrounding biological sequence comparison. The biological problem has motivated the developments in algorithms and statistics.

Statistics

The statistical aspects of sequence comparison are worthy of further study. This includes the statistics supporting BLAST and the statistical basis of substitution matrices based on log-odds ratios. [KA90, Alt91, EG01, NCBIa, Gish, DSO78, HH92, HH93]

Related Problems in Computer Science

The problem of biological sequence comparison is essentially a variant of the **approximate string matching** problem. Algorithms based on three main strategies – dynamic programming, automata, and suffix trees – have been developed. Results from Ukkonen, and Landau and Vishkin in the 1980's and from Myers, Wu and Manber in the 1990's show this to be an active area of research. **Approximate pattern matching** expands the scope to include searching for regular expressions or query strings specified by grammars. [Nav01, Mye91]

Approximate string matching is, itself, a generalization of **exact string matching**. Looking backwards into the development of the Knuth-Morris-Pratt algorithm or Boyer-Moore would certainly be interesting. [CLRS, Nav01]

Related Problems in Molecular Biology

Biologists frequently encounter sets of sequences that perform the same function or that are related by descent. It can be informative to compare the whole set simultaneously. This is the **multiple sequence alignment** problem [Gus97, JP04], known to be NP-hard [Mye91]. Multiple alignment is another opportunity for heuristic algorithms to come into play.

Multiple alignment enables the construction of **phylogenetic trees**. A phylogenetic tree graphically shows the evolutionary relationships among the sequences. The leaves of the tree are the sequences being compared. Each interior node represents the putative most recent common ancestor of its child nodes. The lengths of the edges are proportional to the evolutionary distance between the sequences.

As for the sequence comparison problem, BLAST is hardly the last word. Approaches based on preprocessing or indexing the database is one current focus of attention.

As biology transforms itself into more of an information based science, sequence comparison is only the beginning of the ways in which computer science can be applied to biological problems.

Bibliography

Original Articles

[AGM90]   Altschul, S., W. Gish W., Miller W., Myers E., and Lipman D., (1990) "A Basic Local Alignment Search Tool", *J. Mol. Biology* 215:403-410.

[Alt91]   Altschul SF. (1991) "Amino acid substitution matrices from an information theoretic perspective", *J. Mol. Biology* 219:555-565.

[AMS97]   Altschul,S.F., Madden,T.L., Schaffer,A.A., Zhang,J., Zhang,Z., Miller,W. and Lipman,D.J. (1997) "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", *Nucleic Acids Res.* 25, 3389–3402.

[DSO78]   Dayhoff M, Schwartz RM, Orcutt BC. (1978) "A model of evolutionary change in proteins", In: Dayhoff M, ed. Atlas of protein sequence and structure, vol5 (suppl3). Silver Spring, Maryland: National Biomedical Research

[HH92]   Henikoff S, Henikoff JG. (1992) "Amino acid substitution matrices from protein blocks", *Proc Nail Acad Sci USA* 89:10915-10919.

[HH93]   Henikoff, S. & Henikoff, J.G. (1993) "Performance evaluation of amino acid substitution matrices", *Proteins* 17:49-61.

[KA90]   Karlin, S. & Altschul, S.F. (1990) "Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes", *Proc. Natl. Acad. Sci. USA* 87:2264-2268.

[LP85]   Lipman, D.J. and Pearson, W.R., (1985) "Rapid and sensitive protein similarity searches", *Science*, 227:1435--1441.

[NW70]   Needleman, S.B. and Wunsch, C.D. (1970) "A general method applicable to the search for similarities in the amino-acid sequence of two proteins," *J. Mol. Biology* 48:443-453.

[PL88]   Pearson, W.R. & Lipman, D.J. (1988) "Improved tools for biological sequence comparison", *Proc. Natl. Acad. Sci. USA* 85:2444-2448.

[Sel74]   Sellers, P.H., (1974) "On the theory and computation of evolutionary distances", *SIAM J. App. Math.* 26:787-793.

[SW81]   Smith, T.F. and M.S. Waterman, (1981) "Identification of common molecular subsequences", *J. Mol. Biol* 147:195-197.

[SW81a]   Smith, T.F. and M.S. Waterman, (1981) "Comparison of biosequences", *Adv. Appl. Math.*

[WE87]   Waterman, M.S. and M. Eggert, (1987) "A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons", *J. Mol. Biol.* 197:723-728.

Review Articles:

[Mye91]   Myers, E. (1991) "An overview of sequence comparison algorithms in molecular biology", Tech. Rep. TR-91-29, Dept. of Computer Science, Univ. of Arizona.

[Nav01]   Navarro, G. (2001) "A guided tour to approximate string matching", *ACM Computing Surveys*

Books:

[MBC]   Alberts, B., et. al. (2002) "Molecular Biology of the Cell, 4th ed.", Garland Publishing, New York, N.Y.

[EG01]   Ewens, W.J. & Grant G.R. (2001) "Statistical Methods in Bioinformatics: An Introduction", Springer-Verlag

[Gus97]   Gusfield, D. (1997) "Algorithms on Strings, Trees and Sequences", Cambridge Univ. Press, Cambridge.

[JP04]   Neil C. Jones, Pavel A. Pevzner (2004) "An Introduction to Bioinformatics Algorithms", MIT Press

[CLRS]   Cormen, T.H., Leiserson C.E., Rivest R.L., & Stein C. (2001) "Introduction to Algorithms, 2nd ed." MIT Press

Internet Resources

[NCBIa]   Altschul, S.F. The Statistics of Sequence Similarity Scores, http://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html

[NCBIb]   http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html

[GOLD]   http://www.genomesonline.org/

[Gish]   http://blast.wustl.edu/doc/infotheory.html

[Kah04]   Lecture Notes on Bioinformatics by Tamer Kahveci, http://www.cise.ufl.edu/~tamer/teaching/fall2004/

[BA]   Website companion to the Jones and Pevzner book, http://www.bioalgorithms.info/

[MT]   "Lecture Notes on Biological Sequence Analysis" and "Basics of Molecular Biology" by Martin Tompa, http://www.cs.washington.edu/homes/tompa/papers/

Appendix: Implementations

As part of my project, I implemented both the original and the more general variants of the Needleman-Wunsch algorithm for global alignment. The interesting part of the code for the original variant is listed here.

```
# NeedlemanWunsch

# computes the global alignment of 2 sequences using a scoring
# system of 1 for a match, 0 for mismatch or gap.

# implements the algorithm described in
# Needleman, S.B. and Wunsch, C.D.
# "A general method applicable to the search for similarities
# in the amino-acid sequence of two proteins,"
# J. Molecular Biology 48 (1970), 443-453.

# gap penalties are zero and are omitted

# by: J. Christopher Bare
# date: 2005-5-24

# align two sequences given by strings s1 and s2
def align(s1, s2):
    n = len(s1)
    m = len(s2)

    # initialize the table to n x m
    table = [0]*(m*n)
    coord = Coord(m)

    # initialize table of back pointers to n x m
    # each entry will be a list of (i,j) tuples
    backpointers = [[]]*(m*n)

    for i in reversed(range(0,n)):
        for j in reversed(range(0,m)):

            if i+1 < n and j+1 < m:

                max_score = table[coord(i+1,j+1)]
                max_coords = [(i+1,j+1)]

                # find max score in previous row
                # in cols to the right of j+1
                for k in range(j+2, m):
                    if table[coord(i+1,k)] > max_score:
                        max_score = table[coord(i+1,k)]
                        max_coords = [(i+1,k)]
                    elif table[coord(i+1,k)] == max_score:
                        max_coords.append((i+1,k))

                # find max score in previous col
                # in rows below i+1
                for k in range(i+2, n):
                    if table[coord(k,j+1)] > max_score:
                        max_score = table[coord(k,j+1)]
                        max_coords = [(k,j+1)]
                    elif table[coord(k,j+1)] == max_score:
                        max_coords.append((k,j+1))

                table[coord(i,j)] = score(s1[i],s2[j]) + max_score
                backpointers[coord(i,j)] = max_coords

            else:

                table[coord(i,j)] = score(s1[i],s2[j])

[... boring code omitted ...]

def generate_alignments_from_paper():
    align("AJCJNRCKCRBP","ABCNJROCLCRPM")
```

Output of the original variant run on the sequences from the original paper is shown here:

```
     A   B   C   N   J   R   O   C   L   C   R   P   M
   -----------------------------------------------
A|   8   7   6   6   5   4   4   3   3   2   1   0   0
J|   7   7   6   6   6   4   4   3   3   2   1   0   0
C|   6   6   7   6   5   4   4   4   3   3   1   0   0
J|   6   6   6   5   6   4   4   3   3   2   1   0   0
N|   5   5   5   6   5   4   4   3   3   2   1   0   0
R|   4   4   4   4   4   5   4   3   3   2   2   0   0
C|   3   3   4   3   3   3   3   4   3   3   1   0   0
K|   3   3   3   3   3   3   3   3   3   2   1   0   0
C|   2   2   3   2   2   2   2   3   2   3   1   0   0
R|   2   1   1   1   1   2   1   1   1   1   2   0   0
B|   1   2   1   1   1   1   1   1   1   1   1   0   0
P|   0   0   0   0   0   0   0   0   0   0   0   1   0

AJC-JNR-CKCRBP-
ABCNJ-ROCLCR-PM

AJCJN-R-CKCRBP-
ABC-NJROCLCR-PM
```

For comparison, here is the output of the general variant on a short snippet of protein sequence from human and mouse insulin. The alignment was scored using the BLOSUM62 matrix.

```
        G    S    P    G    D    L    Q    T    L    A    L    E    V    A    R
   ------------------------------------------------------------------
  |   0   -5  -10  -15  -20  -25  -30  -35  -40  -45  -50  -55  -60  -65  -70  -75
G|  -5    6    1   -4   -9  -14  -19  -24  -29  -34  -39  -44  -49  -54  -59  -64
G| -10    1    6    1    2   -3   -8  -13  -18  -23  -28  -33  -38  -43  -48  -53
P| -15   -4    1   13    8    3   -2   -7  -12  -17  -22  -27  -32  -37  -42  -47
G| -20   -9   -4    8   19   14    9    4   -1   -6  -11  -16  -21  -26  -31  -36
A| -25  -14   -8    3   14   17   13    8    4   -1   -2   -7  -12  -17  -22  -27
G| -30  -19  -13   -2    9   13   13   11    6    1   -1   -6   -9  -14  -17  -22
S| -35  -24  -15   -7    4    9   11   13   12    7    2   -3   -6  -11  -13  -18
L| -40  -29  -20  -12   -1    4   13    9   12   16   11    6    1   -4   -9  -14
Q| -45  -34  -25  -17   -6   -1    8   18   13   11   15   10    8    3   -2   -7
P| -50  -39  -30  -18  -11   -6    3   13   17   12   10   12    9    6    2   -3
L| -55  -44  -35  -23  -16  -11   -2    8   12   21   16   14    9   10    5    0
A| -60  -49  -40  -28  -21  -16   -7    3    8   16   25   20   15   10   14    9
L| -65  -54  -45  -33  -26  -21  -12   -2    3   12   20   29   24   19   14   12
E| -70  -59  -50  -38  -31  -24  -17   -7   -2    7   15   24   34   29   24   19
G| -75  -64  -55  -43  -32  -29  -22  -12   -7    2   10   19   29   31   29   24
S| -80  -69  -60  -48  -37  -32  -27  -17  -11   -3    5   14   24   27   32   28
L| -85  -74  -65  -53  -42  -37  -28  -22  -16   -7    0    9   19   25   27   30

GGPGAGSLQPLALEGSL
GSP--GDLQTLALEVAR

GGPGAGSLQPLALEGSL
GSPG--DLQTLALEVAR
```