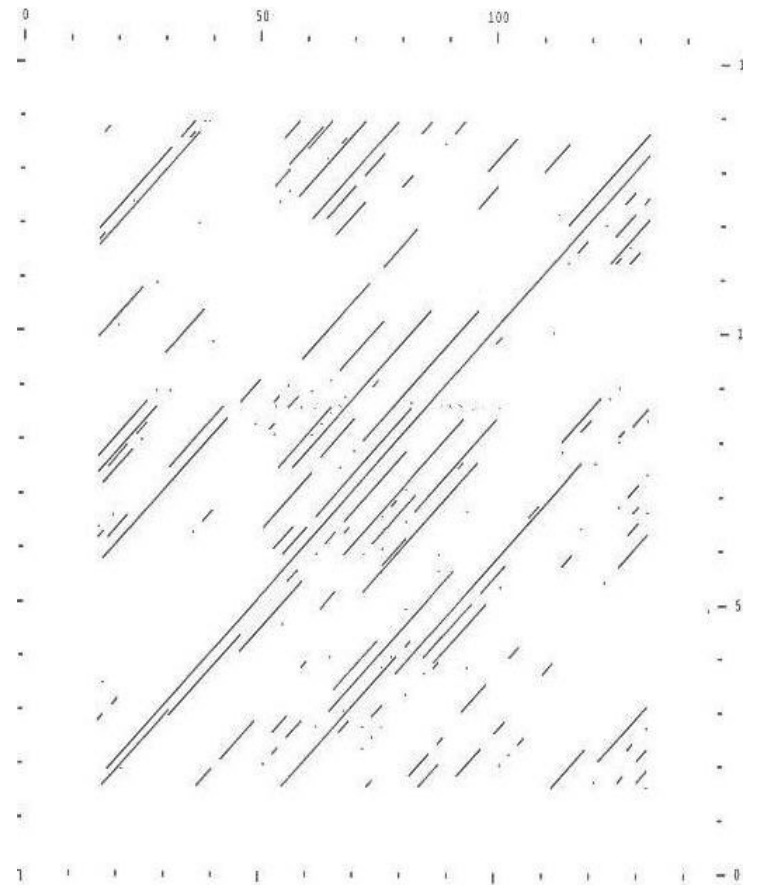


Plot 1: 2- α haptoglobin² against itself.



Plot 2: Human β globin v. human myoglobin. $w=30, s=9$

II. Global Alignment

An alignment of two sequences a and b is an arrangement of a and b by position where both can be padded with gap symbols, and the length of both counting the gap symbols is the same:

Given a : AGCACACA and b : ACACACTA we can have, among others, the alignments

AGCACAC-A
A-CACACTA

or

AG-CACACA
ACACACT-A

By reading the alignment from left to right we have a list of edit operations required to transform a into b :

Left: Match (A,A)
Delete(G,-)
Match (C,C)
Match (A,A)
Match (C,C)
Match (A,A)
Match (C,C)
Insert(-,A)
Match (A,A)

Right: Match (A,A)
Replace(G,C)
Insert (-,A)
Match (C,C)
Match (A,A)
Match (C,C)
Replace(A,T)
Delete (C,-)
Match (A,A)

The optimal global alignment is one which minimizes the total cost of transforming a into b , where each operation has a specific cost. We will first talk about the costs for matches and mismatches (scoring matrices), and then discuss the costs for insertions and deletions (gap penalties).

II.1 Scoring Matrices

Scoring matrices are used to determine the penalties for mismatches and payoffs for matches for the alignment. Given an alphabet Σ , the element M_{ij} of the scoring matrix M will have the match payoff (if $i = j$) or the mismatch penalty (o.w.) The choice of the scoring matrix will have a major effect on the alignment produced. The matrix can be built based on one of many principles:

- Physical/Chemical similarities:
 - Comparing two sequences according to the properties of their residues may highlight regions of structural similarity
- Identity matrices
 - by stressing only identities and not penalizing divergence are better able to locate the remaining common features
- Mutation Matrices
 - matrix entries represent the likelihood of the particular letter changing into another over time. The most common matrices of this type are PAM and BLOSUM. The matrix that performs best is the one that most closely resembles the evolutionary distance between the two sequences being aligned.

II.2 Statistical Motivation

There is a statistical motivation for alignment scores. Given a gap-free alignment, we have two potential hypotheses: **H**omologous and **R**andom.

If we are aligning DNA sequences, we will assume independent sites and the Jukes-Cantor model for the homology hypothesis and independent sites and equal frequencies for all base pairs for the random hypothesis. We can calculate the log-likelihood as follows:

Sample Alignment:

AGCTGA . . .
AACCGG . . .

$$P(\text{data}|\mathbf{H}) = P\left(\begin{matrix} \text{A} \\ \text{A} \end{matrix} \middle| \mathbf{H}\right) \times P\left(\begin{matrix} \text{G} \\ \text{A} \end{matrix} \middle| \mathbf{H}\right) \times \dots = (1-p)^a p^d$$

$$\text{where } d = \text{\#disagreements}, a = \text{\#agreements}, \text{ and } p = \frac{3}{4}(1 - e^{-8\alpha}).$$

$$P(\text{data}|\mathbf{R}) = P\left(\begin{matrix} \text{A} \\ \text{A} \end{matrix} \middle| \mathbf{R}\right) \times P\left(\begin{matrix} \text{G} \\ \text{A} \end{matrix} \middle| \mathbf{R}\right) \times \dots = \left(\frac{1}{4}\right)^a \left(\frac{3}{4}\right)^d.$$

$$\log \left\{ \frac{P(\text{data}|\mathbf{H})}{P(\text{data}|\mathbf{R})} \right\} = a \times \log \frac{1-p}{1/4} + d \times \log \frac{p}{3/4}$$

$$\text{Because } p < \frac{3}{4}, \log\left(\frac{p}{3/4}\right) < 0, \log\left(\frac{1-p}{1/4}\right) > 0,$$

$score = a \times \sigma + d \times (-\mu)$, where σ is the match score and $-\mu$ is the mismatch penalty.

Note that if $\alpha t \approx 0$, $\sigma \approx \log 4$, while $-\mu \approx \log 8\alpha t$, which is massively negative. We have a big difference between the two scores. Conversely, if αt is large $p = \frac{3}{4}(1 - \epsilon)$, $\frac{p}{3/4} = 1 - \epsilon$, and $\mu = \log(1 - \epsilon) \approx -\epsilon$, while $1 - p = \frac{1}{4}(1 + 3\epsilon)$, $\frac{1-p}{1/4} = 1 + 3\epsilon$, and so $\sigma = \log(1 + 3\epsilon) \approx 3\epsilon$. Thus the scores are about 3:1.

This is easily generalized to any other Markov Substitution matrix:

Gap free alignment:

$a_1 a_2 \dots a_m$
 $b_1 b_2 \dots b_m$

$$P(data|\mathbf{H}) = \prod_1^m \pi_{a_i} p_{a_i b_i} (2t)$$

$$P(data|\mathbf{R}) = \prod_i \pi_{a_i} \pi_{b_i}$$

$$\log \left\{ \frac{P(data|\mathbf{H})}{P(data|\mathbf{R})} \right\} = \sum_i \log \left\{ p_{a_i b_i} (2t) / \pi_{b_i} \right\}$$

The elements of the log-odds score matrix are usually positive on the diagonal and negative off the diagonal, but not always.

The relative size of the match and mismatch penalties increase as αt decreases. In the PAM(αt) matrix PAM(120) is more stringent than PAM(250), while PAM(360) is even less stringent. PAM(0), the identity matrix, is the most stringent.

There are many other scoring matrices based on other principles, but in general the matrix that works best is the one that best reflects the evolutionary distance between the two sequences. To some extent this creates a chicken and egg problem, since we want to align the sequences in order to detect whether the two are related, and if so how long ago they split.

The two most commonly used scoring matrices are PAM and BLOSUM. The following chart has the BLOSUM(62) substitution matrix below the diagonal and BLOSUM(62)-PAM(160) above the diagonal. As the relatively small values in the top half indicate, the differences between the two matrices are not large.

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

II.3 Gap penalties

Gap Penalties are generally composed of two parts: the gap opening penalty and the gap extension penalty.

The gap opening penalty reduces the alignment score so that any gap must create a better alignment downstream than would be present without the gap. The gap opening penalty is usually of the order of one to three times the size of the values in the scoring matrix.

The gap extension penalty is usually much smaller, because biologically once an insertion is created, its size can vary. At the same time we need some gap-extension penalty in order to limit the size of the gap to practical lengths. One model commonly used is the affine gap penalty, where a gap length k is penalized $o+ek$, where o is the gap opening penalty and e is the gap extension penalty. A smaller gap extension penalty may allow an alignment to resolve situations where complete loops may be missing between the two structures.

As with the scoring matrix, the choice of the gap opening and gap extension penalties will have a large role in determining the alignment produced. The following are two alignments run with different gap-start penalties:

to: LGB1_PEA.pep check: 2970 from: 1 to: 147

Seq: LGB1_PEA

Gap Weight:	3.000	Average Match:	0.540
Length Weight:	0.100	Average Mismatch:	-0.396
Quality:	63.1	Length:	153
Ratio:	0.429	Gaps:	4
Percent Similarity:	46.809	Percent Identity:	21.986

LGB1_PEA.pep x HBHU.pep May 24, 1999 15:06 ..

```

1  ..GFTDKQEALVNSSSEFKQNLPGYSILFYTIVLEKAPAAKGLFSFLKD. 47
   :|...|.. :| |::| .. :| .. :|. :|
1  MVHLTPEEKSAVTALWG.KVNVDEVGGEALGRLLVVYPWTQRFFESFGDL 49

48 ..TAGVEDSPKLQAHAEQVFLVRDSAAQLRTKGEVVLGNATLGAIHVQK 95
   ..:| :.||:..|:|..|:| ..|:| :|:| :. ||:..|:|
50 STPDAMVGNPKVKAHGKVKLGAFSDGLAHL...DNLKGTFFATLSELHCDK 96

96 .GVTNPHFVVVKEALLQTIKASGNNWSEELNTAWEVAYDGLATAIKKAM 144
   I..:| :: :. |: . : |::...:|:|: . |:|. |: .
97 LHVDPENFRLGNVLCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKY 146

145 KTA 147
   .
147 H.. 147

```

Gap Weight:	20.000	Average Match:	0.540
Length Weight:	0.100	Average Mismatch:	-0.396
Quality:	48.3	Length:	151
Ratio:	0.329	Gaps:	0
Percent Similarity:	34.965	Percent Identity:	15.385

HBHU.pep x LGB1_PEA.pep May 24, 1999 18:43 ..

```

1  MVHLTPEEKSAVTALWGKVNVDVGEALGRLLVVYPWTQRFFESFGDLS 50
   . :| .. :. :|:|:| .. :|:|:|:|:|
1  ....GFTDKQEALVNSSSEFKQNLPGYSILFYTIVLEKAPAAKGLFSFLK 46

51  TPDAMVGNPKVKAHGKVKLGAFSDGLAHLNHLKGTFFATLSELHCDKLHVD 100
   ...:| :.||:..|:|..|:| ..|:| :|:| . :. . .| . :|:|
47  DTAGVEDSPKLQAHAEQVFLVRDSAAQLRTKGEVVLGNATLGAIHVQKG 96

101 PENFRLGNVLCVLAHHFGKEFTPPVQAAYQKVVAGVANALAHKYH... 147
   ..| :|:| . : . : : . : . . || :| | |
97  VTNPHFVVVKEALLQTIKASGNNWSEELNTAWEVAYDGLATAIKKAMKTA 147

```

The gap-start penalty was 3 in the first alignment and 20 in the second one. As a result the second alignment has no gaps but a lower similarity score.

II.4 Edit distance

We can now turn the edit protocol into a measure of distance between the sequences by assigning a “cost” two every edit operations. One of the simplest of these is the unit cost model, also called Levenshtein distance:

$$S(u,u)=0; S(u,v)=1 \text{ for } u \neq v; S(u,-)=S(-,v)=1.$$

In general, however, it is necessary to use a more sophisticated cost model which would take into account the biochemical similarities between the elements being compared.

The *cost of an alignment* of two sequences **a** and **b** is the sum of the costs of all the edit operation which lead from **a** to **b**.

The *optimal alignment* of **a** and **b** is an alignment with minimal cost among all possible alignments.

Using the unit cost model, in our previous example we obtain the following costs:

AGCACAC-A	or	AG-CACACA
A-CACACTA		ACACACT-A
cost: 2		cost: 4

III Dynamic Programming

Dynamic programming is a method for obtaining an optimal alignment. It is an extension of the dotplot method, but instead of dots the comparison matrix entries are assigned values that reflect the scores in the scoring matrix. The optimum alignment is obtained by finding the highest scoring path from the top left-hand corner to the bottom right-hand corner. When the alignment steps away from the diagonal (right or down) this implies an insertion or deletion event, the impact of which can be assessed by applying the gap penalty.

Given two sequences $\mathbf{a} = (a_1, a_2, \dots, a_m)$ and $\mathbf{b} = (b_1, b_2, \dots, b_m)$, we denote by d_{ij} the edit distance between the initial segments $\mathbf{a} = (a_1, a_2, \dots, a_i)$ and $\mathbf{b} = (b_1, b_2, \dots, b_j)$, and define $d_{00} = 0$. Now assuming that insertions incur a penalty of +1, we have the following formula for d_{ij} :

$$d_{ij} = \min(d_{i-1,j-1} + s(a_i, b_j), d_{i,j-1} + 1, d_{i-1,j} + 1)$$

The following matrix shows the implementation of this algorithm (under the unit cost model) on our test sequences (arrows indicate the least-cost path):

		$b \rightarrow$								
			A	C	A	C	A	C	T	A
		0	1	2	3	4	5	6	7	8
$a \downarrow$	A	1	0	1	2	3	4	5	6	7
	G	2	1	1	2	3	4	5	6	7
	C	3	2	1	2	2	3	4	5	6
	A	4	3	2	1	2	2	3	4	5
	C	5	4	3	2	1	2	2	3	4
	A	6	5	4	3	2	1	2	3	3
	C	7	6	5	4	3	2	1	2	3
	A	8	7	6	5	4	3	2	2	2

The formula above can be generalized to any gap model by setting

$$d_{ij} = \min \begin{cases} d_{i-1,j} + s(a_i b_j) \\ d_{ij} + \text{gap}(i, h) \quad 0 \leq h < i \\ d_{ih} + \text{gap}(j, h) \quad 0 \leq h < j \end{cases}$$

This, however, requires $O(n^3)$ calculation, unlike $O(n^2)$ for the constant gap penalty. It is also possible to calculate the optimal alignment for the affine gap model in $O(n^2)$ time by storing three values for each entry in the matrix:

Let M_{ij} be the best score up to (i, j) given that a_i is aligned to b_j ;

Let N_{ij} be the best score up to (i, j) given that a_i is aligned to a gap (an insertion with respect to b);

Let O_{ij} be the best score up to (i, j) given that b_j is aligned to a gap (an insertion with respect to a);

Given a gap opening penalty o and gap extension penalty e , we can define

$$\begin{aligned} M_{ij} &= \min(M_{i-1,j-1} + s(a_i, b_j), N_{i-1,j-1} + s(a_i, b_j), O_{i-1,j-1} + s(a_i, b_j)) \\ N_{ij} &= \min(M_{i-1,j} + o, N_{i-1,j} + e) \\ O_{ij} &= \min(M_{i,j-1} + o, O_{i,j-1} + e) \end{aligned}$$

We can then backtrack through the matrix in the usual manner, setting our back pointer to (i, j) if M_{ij} was the smallest of the three, to $(i-1, j)$ if N_{ij} was, and to $(i, j-1)$ if O_{ij} was the smallest.

III.1 Local Alignment

Local Alignment is very similar to the preceding, except the goal is different. Instead of trying to find the similarity between the two sequences, we are trying to find the best fit for one of the sequences inside another. This is useful for database searching, where we can think of the database as one long sequence. The Smith-Waterman algorithm for local alignment has

$$d_{ij} = \min(d_{i-1,j-1} + s(a_i, b_j), d_{i,j-1} + 1, d_{i-1,j+1}, 0)$$

The only difference is the 0 as the last term in the min. Because of the 0 the values in the matrix can never become negative, hence we will see areas of similarity even if there are long mismatches or gaps in between them.

IV Chance or Common Ancestry?

Given an optimal alignment, it is necessary to determine whether the alignment occurred by chance or because of common ancestry. In order to estimate the likelihood of some alignment occurring by chance we can randomize (shuffle) one of the two input sequences and look at the distribution of scores representing chance similarity rather than homology. The score from our original sequences can be referred to this distribution and assigned a Z-score, or a p-value.

This approach has been criticized because shuffled versions of one of the two sequences may have plausible amino acid composition, but are not similar to real sequences. It is possible to partially address these concerns by either a) restricting the randomization to blocks; or b) creating the distribution of random scores from real amino acid sequences known not to be homologous to our query sequence.

V More on Dynamic Programming

Algorithms we now call Dynamic Programming (DP) date back to Bellman in the 1950s. They were rediscovered in the context of coding independently by Levenstein and Viterbi, and similar algorithms arose in the theory of HMMs (Baum et al) 1966-1972. DP was first used for sequence alignment by Needleman and Wunsch in 1970, and since then many DP algorithms have appeared in computational biology.

DP algorithms usually include a) a recurrence relation; b) tabular computation; and c) a traceback, though the third part is sometimes skipped. A natural formulation of DP algorithms involves directed graphs. What follows is based on the notes of Phil Green that were on the web.

A **directed graph** is defined as a finite set of vertices and edges, $G = (V, E)$, with $V = \{v_1, v_2, \dots, v_n\}$ and E a set of ordered pairs from V .

A **path** of length $l-1$ is a sequence of vertices $P = v_{i_1}, v_{i_2}, \dots, v_{i_l}$ with $(v_{i_k}, v_{i_{k+1}}) \in E \ \forall k$, or equivalently, as the corresponding sequence of edges.

A **cycle** is a path that starts and ends at the same vertex. We will be concerned with Directed Acyclic Graphs (DAGs), which are graphs which do not have cycles.

A graph may be **weighted** by a real function W , defined on the edges and/or vertices. The weight of the path P is defined to be

$$W(P) = \sum_{e \in P} W(e) + \sum_{v \in P} W(v)$$

Given a DAG, $G=(V,E)$, and two vertices u and t within a graph, we can find a minimum weight path between the two vertices in time $O(|V|+|E|)$. Proof of this contains the essence of many DP algorithms in computational biology.

The major idea underlying the proof is that the minimum weight path to a vertex v is easy to calculate once the weight of the minimum path to $\forall v'$ s.t. $(v', v) \in E$ is known.

Proof For each vertex v define $M(v)$ to be the minimum weight over all paths ending in v . Let $P(v)$ be a path achieving this maximum, and $L(v)$ be the vertex preceding v in P , i.e. $((L(v), v) \in P$, or $L(v)=v$ if $|P|=0$. These can be written recursively as:

$$M(v) = \begin{cases} \min_{(v', v) \in E} M(v') + W(v', v) + W(v) & \text{if } v \neq u \\ W(v) & \text{o.w.} \end{cases}$$

$$L(v) = \begin{cases} \arg_{v'} \min_{(v', v) \in E} M(v') + W(v', v) + W(v) & \text{if } v \neq u \\ v & \text{o.w.} \end{cases}$$

$$P(v) = \begin{cases} (P(L(v)), v) & \text{if } v \neq u \\ v & \text{o.w.} \end{cases}$$

Using these formulae we can calculate $M(v_j)$ for any j in time $O(|V|+|E|)$. $L(v)$ and $P(v)$ come for free from calculating $M(v)$. Most of the dynamic programming algorithms in computational biology (Smith-Waterman, Viterbi, etc.) are variations on this theme.