Sequence Similarity

In these notes, we will talk about *sequence* ***similarity*** *or* ***alignment*** *of two sequences*. The sequences we are concerned with are DNA over the alphabet $\Sigma=(A,C,T,G)$, RNA over the alphabet $\Sigma=(A,C,U,G)$ or amino acid sequences making up a protein molecule.

Biological Motivation

The biological motivation for studying this problem comes from the fact that high degree of similarity of bimolecular sequences usually implies **significant structural and functional similarity**.

Example

- The first successful sequencing of the genome of a living organism in 1995 of the bacterium *Haemophilus influenza rd* (Fleishmann et al, 1995).
- After that, researchers identified 1743 sites as prospective gene sites.
- In order to determine whether these sites are actually involved in protein synthesis, the coding regions were translated into amino acid sequences using the genetic code.
- The resulting amino acid sequences were then compared with a protein database that contains for each known protein the corresponding amino acid sequences.
- The search identified about 1007 close matches. Since the protein database annotated with functions, the close matches allowed coming up with strong conjectures about the functions of these genes.

Motivation

- The sequence similarity is also relevant in the context of understanding the **molecular basis of evolution**.
- It is well known that the closely related organisms have high similarity between their genomes.
- Study of conserved sequences among the organisms reveal past speciation and the structure of ancestral family trees and the role of mutation in the evolution of these trees.
- Studying similarities within the individual organisms in a species might also reveal whether certain individuals are prone to **inherited diseases**.
- There are many other examples from biology that illustrates the use of sequence similarity.

Alignment of Two Sequences

We will discuss the similarity algorithms w.r.t. DNA sequence. A simplified model of change in DNA sequence during evolution is to assume that the following three events might have happened at any location in the sequence:

- A **deletion** operation, D, of one base.
- A **replacement or substitution** operation, R, of one base by another base.
- An **insertion** operation, I, of one base

Alignment Example (1)

- Given sequence *S=ATAGCCAT* and assume that a sequence of operations (R,D,I) and has taken place as follows:

R	D	I
ATAGCCAT	*ATAGTCAT*	*AAGTC--AT*
ATAGTCAT	*A--AGTCAT*	*AAGTCTAT*

  - Biologists call these operations "**alignment**" and represent them by writing the two sequences, one over the other.

Alignment Example (2)

- If *X* and *Y* are two distinct symbols, then the operations can be denoted by the ordered pair in any vertical column of the alignment as

  R=(*X,Y*), D=(*X*,--) and I=(--,*X*),

  - where – denote a null sequence.
- Obviously, (--,--) is a useless operation aligning null sequence with null sequence.
- Symbols pair that are identical in a vertical column represent matched symbols and sometime an operation *M* is defined for this situation.
- Sometimes, an ***indel*** operation is used denoting either a delete or insert operation when the direction of transformation is not known.

Alignment Example(3)

- For this example, the combined effect of the three operations can be captured by the alignment

ATAGCC--AT
A--AGTCTAT

- In the evolutionary history, the accumulated changes may obscure the exact sequence of operations.
- E.g., the same final sequence may be obtained by the alignment that needs 5, rather than 3 operations:

ATAGCCAT—
A--AGTCTAT

Goal of Sequence Alignment

- The goal of sequence alignment is to discover the possible evolution of sequences without actual knowledge of the evolutionary events.
- Naturally, the alignment with minimum number of operations involving minimum energy may be the Nature's choice. This transformation, as we will see, soon corresponds to ***edit distance*** between the sequences.

Definition: Alignment

- Let *S1* and *S2* two sequences of length *n* and *m* , respectively , over a finite alphabet Σ. An **alignment** maps the strings *S1* and *S2* into strings and that may contain space ("—") characters such that $|S_1'| = |S_2'| = l$ and removal of all space characters leaves *S1* and *S2* intact.

Number of Alignments

- It is clear that $\max(n,m) \le l \le n+m$. The case *l=n+m* occurs when the alignment corresponds to deleting all characters in *S1* followed by insertion of all characters of *S2*.
- Let *f(i,j)* denote the number of alignments of one sequence of *i* letters with another of *j* letters. Then, it has been proved that

$$f(n,m) \approx (1+\sqrt{2})^{2n+1} n^{-1/2}$$

- For n=1000, ***f*(1000,1000)=** $10^{767.4}$ **alignments!** The number of elementary particles in the universe is about 10^{80} , and Avogadro's number is 10^{23}.

Definition: Edit Distance

- Given two strings S_1 and S_2, the minimum number of edit operations (I: insert, D: delete, R: replace) required to transform S_1 to S_2 is called the **edit distance** between the strings. It is also called **Levenstein** distance.
  - A *replacement* or *substitution* operation can be conceived of a *delete* operation followed by an *insert* operation. Thus the edit distance can be expressed only in terms of only insert and delete operations.

Edit Transcript

- A string over the alphabet (R,I,D,M) of length *l* that transforms S_1 to S_2 is called an ***edit transcript***.
- For the alignment:

A--TCCGAT--
TAT--C—ATC

The edit transcript is: RIMDMDMMI which converts *ATCCGAT* to *TATCATC*.

Technical comments on edit distance

- Symmetrical: D(A,B) = D(B,A)
  - Can “reverse the movie”
  - Substitution X→Y becomes substitution Y→X
  - Insertion becomes deletion
  - Deletion becomes insertion
- “Parsimony” principle often used in computational biology
  - Simplest explanation for an observation
  - minimum number of edits = fewest mutations

Courtesy : Bob Edgar, UC Berkeley

Optimal Alignment

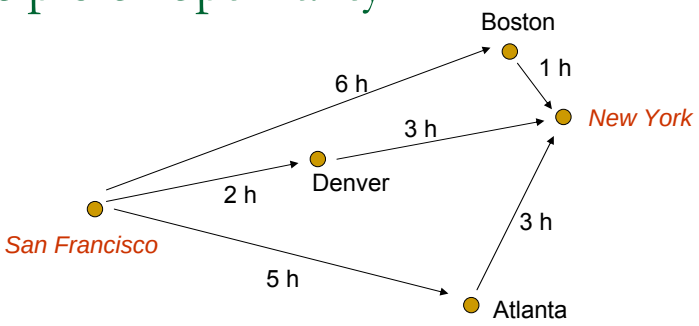
- Given the sequences and the edit transcripts, it is easy to find the alignment for the transcript.
- Alignment and edit transcript are equivalent. The transcript explicitly shows the mutational events and alignment displays the relationship between the strings.
- An alignment corresponding to the minimum edit distance between the two strings is called an ***optimal alignment***.

Principle of optimality

- In some optimization problems...
- ...components of a globally optimal solution are themselves globally optimal
- Then can optimize by recursively optimizing sub-problems

Courtesy : Bob Edgar, UC Berkeley

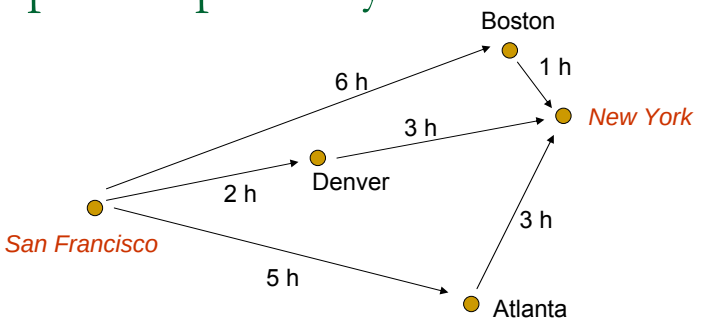
Principle of optimality



- Want fastest time San Francisco to NY, given:
  (1) You must fly via Denver (D), Boston (B) or Atlanta (A)
  (2) Fastest times from SF to D, B or A, and
  (3) Fastest times from D, B or A to NY.

Courtesy : Bob Edgar, UC Berkeley

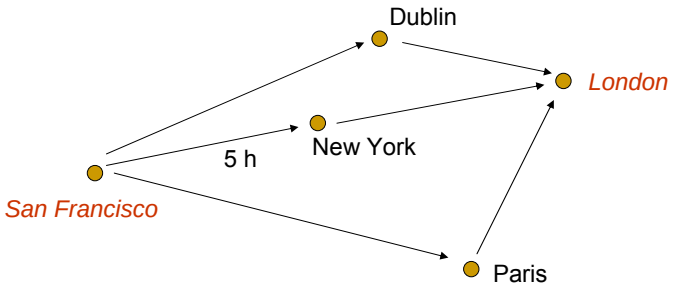
Principle of optimality



- Answer: find minimum of the three possible routes:
  - SF to B + B to NY
  - SF to D + D to NY
  - SF to A + A to NY
- = min (6 + 1, 2 + 3, 5 + 3) = min (7, 5, 8) = 5.

Courtesy : Bob Edgar, UC Berkeley

Principle of optimality



- Now want fastest time to London
- Must fly via New York, Dublin or Paris
- Doesn't matter how we get to NY, already know best time is 5h
- If we solve same problem for Dublin and Paris, can find the answer in the same way as for NY

Courtesy : Bob Edgar, UC Berkeley

Dynamic Programming

- Principle of optimality holds
- Solve simpler sub-problems
- Remember the results
- Use recursion to solve the next-biggest problem

Courtesy : Bob Edgar, UC Berkeley

Edit Distance matrix D

$D[i,j]$ = edit distance between
first i letters in the first string (S_1) and
first j letters in the second string (S_2).

(In other words, the edit distance between the prefix of S_1 with length i and the prefix of S_2 with length j.)

Computing Optimal Alignment by Dynamic Programming.

- **Definition**: For two strings S_1 and S_2, $D(i, j)$ is defined to be the edit distance of $S_1[1....i]$ and $S_2[1....j]$. Then, $D(n, m)$ is the edit distance of S_1 and S_2.
- Let $0 \le i \le n$ and $0 \le j \le m$, where the index 0 will denote null string. The idea is to compute the values of D for increasing values of i and j, using values corresponding to smaller values of i and j

Recurrence Relations

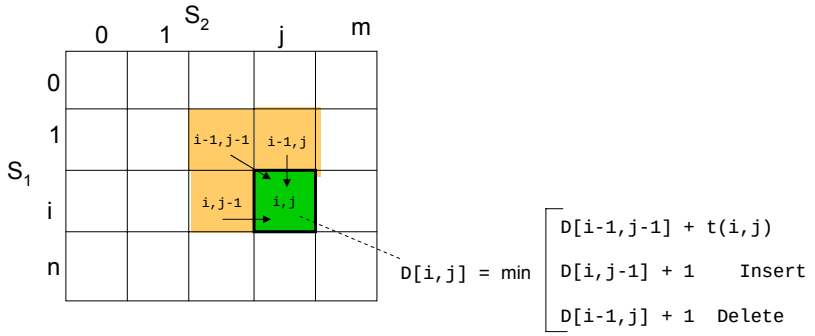
- To start the process, we need a "basis" for i=0 and j=0.

$$D(i,0) = i$$
$$D(0, j) = j$$
$$D(0,0) = 0$$

Where $i = 1...n$ for the first equation and $j = 1...m$ for the second equation.

  - The first equation signifies that i deletion operations are needed to convert the prefix of $S1$ of length i to a null string and the second equation states that j insertion operations are needed to convert a null string to the prefix of $S2$ of length j. The third equation corresponds to a null string being converted to a null string with 0 operation.

The Recurrence Relation



Consider a minimum edit transcript for $D(i,j)$. If the last operation of this transcript is an **insert** I operation, then the alignment must have been at this point (--, $S_2(j)$) corresponding to the **horizonta**l arrow in the matrix. If the last operation of this transcript is a **delete** D operation, then the alignment must have been at this point ($S_i(i)$, --) corresponding to the **vertical** arrow in the matrix. Otherwise, the computation must have taken the diagonal arrow in the matrix which might correspond to either a match or a replacement of $S_1(i)$ by $S_2(j)$.

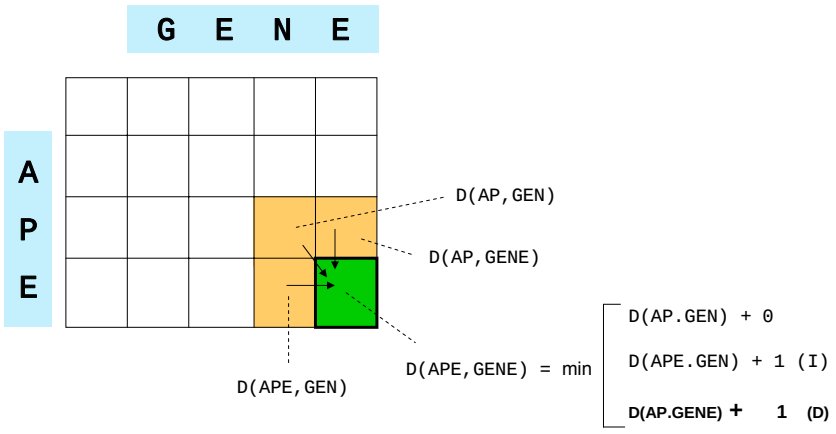
Minimum Edit Transcript for $D(i,j)$.

- If the last operation of this transcript is an insert I operation, then the alignment must have been at this point (--, $S_2(j)$) corresponding to the horizontal arrow in the matrix.
- If the last operation of this transcript is a delete D operation, then the alignment must have been at this point ($S_i(i)$, --) corresponding to the vertical arrow in the matrix.
  - Otherwise, the computation must have taken the diagonal arrow in the matrix which might correspond to either a match or a replacement of $S_1(i)$ by $S_2(j)$.

Cost of Operations

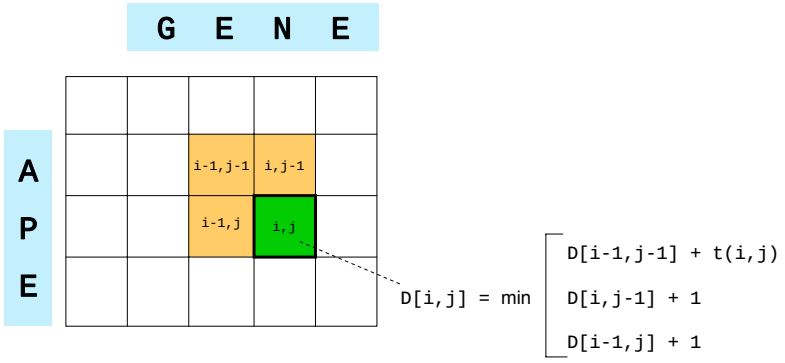
- We assign a cost value of 1 for either insert or delete operation.
- If it is a match the cost *t*(*i*,*j*) is zero;
  - otherwise, we are assuming the cost is 1 but it could be determined by other conditions
- Recursively, we have assumed that *D*(*i-1*,*j*), *D*(*i*,*j-1*) and *D*(*i-1*,*j-1*) are all minimum values of edit distances up to those points in the computation. Then *D*(*i*,*j*) has to be optimal if we take the minimum cost path from these three neighboring points.
- Note the minimum cost path may not be unique, as we will see in our example.

Edit distance matrix M



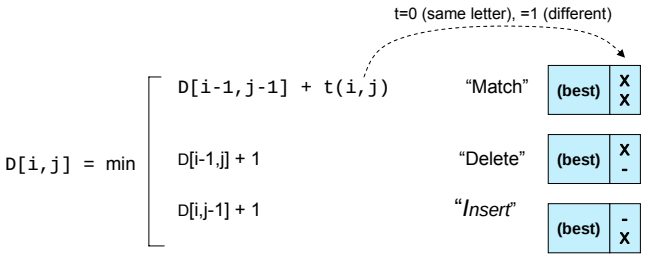
Courtesy : Bob Edgar, UC Berkeley

Edit distance matrix M



Courtesy : Bob Edgar, UC Berkeley

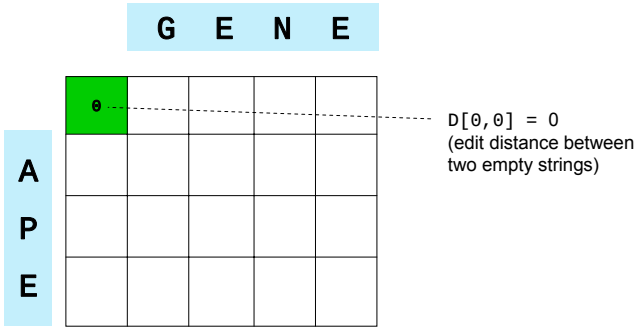
Recursion relations



"Match" = no edit or a substitution
"Insert", "Delete" relative to string S_1

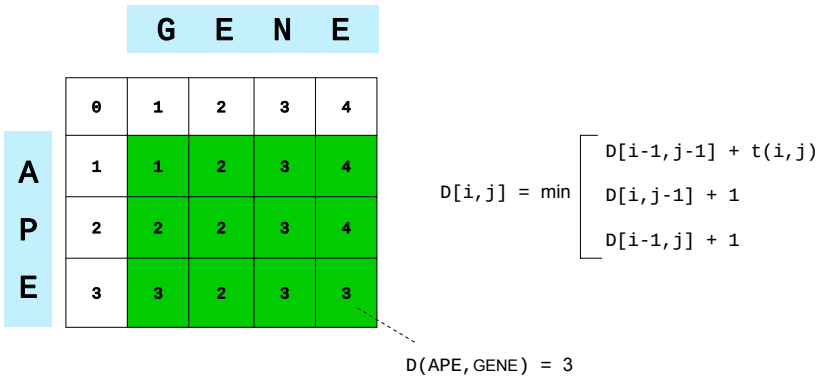
Courtesy : Bob Edgar, UC Berkeley

Initialization



Courtesy : Bob Edgar, UC Berkeley

Rest by recursion



Courtesy : Bob Edgar, UC Berkeley

Recursive Procedure

- The recursive procedure is a top-down approach. That is, the computation starts at the lower rightmost point.In practical implementation, it might need an **exponential** number of calls.
- A bottom-up tabular computation is more efficient.
- To compute the value at any point in the matrix, it is sufficient if we know the minimum edit distances of its **north, north-west and west neighbors** and the pairs of characters from the two sequences under consideration.
- We know how to compute the 0th row and the 0th column of the matrix (the minimum edit distance is simply the index of the row or column), then we can compute the rest of the matrix one row at a time consecutively with increasing row indices or one column at a time consecutively with increasing column index.

Example

D(i,j)			A	T	C	C	G	A	T
		0	1	2	3	4	5	6	7
	0	0	←1	←2	←3	←4	←5	←6	←7
T	1	↑1	↖1	↖1	←2	←3	←4	←5	←↖6
A	2	↑2	↖1	←↖↑2	↖2	←3	←4	↖4	←5
T	3	↑3	↑2	↖1	←2	←↖3	←↖4	←↖↑5	↖4
C	4	↑4	↑3	↑2	↖1	←2	←3	←4	←↑5
A	5	↑5	↖↑4	↑3	↑2	↖2	←↖3	↖3	←4
T	6	↑6	↑5	↖↑4	↖↑3	↖↑3	↖3	←↖↑4	↖3
C	7	↑7	↑6	↑5	↖↑4	↖3	←↖↑4	↖4	↑4

Path

D(i,j)			A	T	C	C	G	A	T
		0	1	2	3	4	5	6	7
	0	**0**							
T	1	**↑1**							
A	2		**↖1**						
T	3			**↖1**					
C	4				**↖1**	**←2**	**←3**		
A	5							**↖3**	
T	6								**↖3**
C	7								**↑4**

Alignment:

S1 T A T C _ _ A T C
S2 _ A T C C G A T _

Time Complexity

- **Theorem**: The dynamic programming algorithm computes a minimum edit distance in time $O(nm)$.
  - Proof. The algorithm needs an $(n+1)(m+1)$ table to be computed. Any particular entry in the table involves three additions, one character comparison operation and one three-way minimum value computation, all requiring $O(1)$ time. Hence the total time is $O(nm)$.

Time Complexity

- **Theorem**: Once the dynamic programming table with pointers has been computed, an optimal edit transcript can be found taking $O(n+m)$ time.
  - Proof. During the construction of the table, the back pointers to neighboring cells having minimum edit distance values can be set up taking $O(nm)$ storage and time. Then a directed path of back pointers originating from (n,m) to (0,0) , called a *trace,* can be constructed taking only $O(n+m)$ time since at each step the path must extend to north, west or north-west. The maximum possible path length is $n+m$.

Time Complexity

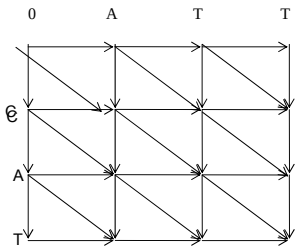
- **Theorem**: Every path from (n,m) to (0,0) corresponds to an optimal edit transcript in one-to-one fashion.
  - Proof: Every point in the trace represents a minimum edit distance from (0,0) to that point.

Weighted Edit Graph

- ***Weighted edit graph*** is an alternate way to represent the dynamic programming problem.
- It can be formulated as a ***shortest path problem***.
- The graph has (n+1)(m+1) vertices corresponding to all possible pairs of indices of the rows and columns. The specific weights and edges depend on the specific string problem.
- For the edit distance problem, vertex (i,j) is connected to vertices ($i,j+1$),($i+1,j$) and ($i+1,j+1$) by ***directed*** arcs having weights corresponding to the cost of insert, delete, replace or match operations, respectively .
- The graph is obviously acyclic. For our edit distance problem, we have assumed , to have value 1 for delete and insert and to have value 1 for replacement and 0 for match.

Weighted Edit Graph

			A	T	T
		0	1	2	3
	0	0	←0	←1	←2
C	1	↑1	↖1	←↖2	←↖3
A	2	↑2	←↖1	←↖2	←↖3
T	3	↑3	↑2	↖1	←2



Shortest path

- **Theorem:** With the weights as defined, an edit transcript for S_1 and S_2 has minimum number of edit operations if and only if it corresponds to a shortest path from (0,0) to (n,m) in the edit graph.

Operation-Weight Alignment

- With arbitrary weights, the solution will correspond to a minimum weighted path between these two points. This allows us to define more complex alignment problems between two strings.
- We can assign weights based on operation (I,D,R, or M), called ***operation-weight alignment*** or based on specific symbols involved in the operation called ***alphabet-weight alignment***.

Dynamic Programming Rules with Weights

- If we assume that the insert and delete operations have weights *u* and *d*, respectively, the replacement operation (or equivalently a mismatch) has a weight *r*, a match operation has a weight *w, we can* re-write the dynamic programming rules as:

$$D(i,0) = iu$$
$$D(0, j) = jd$$
$$D(0,0) = 0$$

Dynamic Programming Rules with Weights

- The general recurrence relations is:

$$D(i, j) = \min[D(i-1, j) + u, D(i, j-1) + d, D(i-1, j-1) + t(i, j)]$$

where

$$t(i, j) = r \quad \text{if} \quad S_1(i) \neq S_2(j)$$
$$t(i, j) = w \quad S_1(i) = S_2(j)$$

Obviously, if r is defined, $r \leq u + d$; otherwise, the replacement operation can be realized by first deleting and then doing an insertion operation to obtain minimum edit distance.

Alphabet Weight Edit Distance

- The ***alphabet weight edit distance*** can be computed using exactly the same set of equations as given above except that the weights are now given by a set of look-up tables viz.
  - a look-up table for insertion cost of each character in the alphabet,
  - a table for deletion cost
  - a table for match cost and a table for replacement cost for every pair of symbols.
- These values have to plugged in as the computation proceeds.

String Similarity

- Finding differences between two sequences can be alternately formulated as finding similarity between two sequences.
- Biologists usually prefer using similarity measures to study relationship between strings.
- Earlier we gave a definition of alignment as follows:
  - **Definition**: Let S_1 and S_2 two sequences of length n and m , respectively , over a finite alphabet $\sum$. An **alignment** maps the strings S_1 and S_2 into strings and that may contain space ('—") characters $|S_1'| = |S_2'| = l$ such that and removal of all space characters leaves S_1 and S_2 intact.

Similarity Definition

- We enlarge the alphabet to include the space symbol '—", so that . Then for any two characters *x* and *y* in Σ', we define a *score* or *value* obtained by aligning *x* against *y*. For a given alignment of *S1* and *S2,* let S'_1 and S'_2 denote the strings after the chosen insertion of spaces. And let *k* denote the equal length of these two strings. Then value *V* of alignment between S'_1 and S'_2 is defined as

$$\sum_{i=1}^{k} s(S_1'(i), S_2'(i))$$

Maximization Problem

- In string similarity problems, the value of *s* is usually set greater than zero for matched symbols and less than zero for symbol pairs that do not match or when a symbol is aligned with a '—' character.
- This reduces the problem to the problem of maximization of *V* for all possible alignments.

Dynamic Programming Solution

- Let $V(i, j)$ be the optimal alignment of prefixes $S_1[1...i]$ and $S_2[1...j]$.
- Basis:

$$V(0, j) = \sum_{k=1}^{j} s(-, S_2)$$

$$V(i,0) = \sum_{k=1}^{i} s(S_1,-)$$

$$V(0,0) = 0$$

Dynamic Programming Solution

- recurrence relation is:

$$V(i, j) = \max[V(i-1, j-1) + s(S_1(i), S_2(j)), \quad \leftarrow \text{replacement}$$
$$V(i-1, j) + s(S_1(i),-), \quad \leftarrow \text{deletion}$$
$$V(i, j-1) + s(-, S_2(j))] \quad \leftarrow \text{insertion}$$

The value of the optimal alignment is given by $V(n, m)$.

Like for the computation of the edit distance, we can use a bottom-up method to compute the alignment matrix. The complexity is *O(nm)* since at each point we perform 3 comparisons, 3 look-up operations and 3 additional operations.

Maximum similarity path

- By setting up suitable pointers, once the matrix is computed, we can obtain a trace for the optimal alignment by constructing any path from the cell (*n,m*) to the cell (0,0).
- Also, the problem can be formulated as finding a ***maximum weighted path*** in a ***weighted acyclic graph*** similar to one discussed earlier. (In general, computing a longest path in arbitrary graph is NP complete).

- The weights of the edges must correspond to specific values of *s* for the pair of symbols. The algorithm takes *O(nm)* space.
- This is quite expensive if the sequences are large.
- If one were interested only in the value of the alignment and not obtaining a trace, this could easily be done by keeping only the last two rows of the matrix to compute the next row.
- This will need only *O(n+m)* space. Is it possible to reconstruct an alignment using only linear space?

Example

j *i*	0	1 *C*	2 *A*	3 *G*	4 *T*	5 *G*
0	0	-1	-2	-3	-4	-5
1 *A*	-1	-1	1	0	-1	-2
2 *C*	-2	1	0	0	-1	-2
3 *T*	-3	0	0	-1	2	1
4 *C*	-4	-1	-1	-1	1	1
5 *G*	-5	-2	-2	1	0	3
6 *T*	-6	-3	-3	0	3	2

j *i*	0	1 *C*	2 *A*	3 *G*	4 *T*	5 *G*
0	0	← -1	-2	-3	-4	-5
1 *A*	↑ -1	-1	↖ 1	0	-1	-2
2 *C*	-2	↖ 1	0	↖ 0	-1	-2
3 *T*	-3	↑ 0	↖ 0	-1	↖ 2	1
4 *C*	-4	-1	↖ ↑ -1	-1	↑ 1	1
5 *G*	-5	-2	-2	↖ 1	0	↖ 3
6 *T*	-6	-3	-3	0	↖ 3	← ↑ 2

- The optimal alignment corresponding to these three paths are:

A	C	T	C	G	T	_
_	C	_	A	G	T	G

A	C	T	C	G	T	_
_	C	_	A	G	T	G

A	C	T	C	G	T	_
_	C	_	A	G	T	G

Beyond edit distance – Another Perspective to maximizing similarity

- Edit distance works quite well, especially for closely related DNA
- Can do better, especially for highly diverged proteins

```
GRB2_CHICK    ...SVKFGND-----VQQFKV...
SRC_RSVSR     ...SIRDWDDMKGDHVKHYKI...
```

(a)

```
HBA_HUMAN   GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL
              G+  +VK+ HGKKV    A+++++ AH+D+ +  + ++++ LS+LH    KL
HBB_HUMAN   GNPKVKAHGKKVLGAFSDGLAHLDNL KGTFATLSELHCDKL
```

This shows clear similarity of human alpha globin to beta globin. There are manypositions where the residues are identical. Many others are 'functionally conservative', indicated by a '+' sign

(b)

```
HBA_HUMAN GSAQVKGHGKKVADALTNAVAHV- - - D- - DMPNALSALSDLHAHKL
              ++  + + ++H+  KV     +   +A    ++                +L+  L+ ++H+  K
LGB2_LUPLU  NNPELQAHAGKVFKLVYEAAIQLQVTGVVVT DATLKNLGSVHVSKG
```

This shows a biologically meaningful alignment between leghaemoglobin and yellow lupin. These two sequences are evolutionary related and have same three Dimensional structure , and function in oxygen binding. Note much fewer match Characters and 'gaps' inserted to maintain alignment

(c)

```
HBA_HUMAN GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSD- - - - LHAHKL
              GS+  +   G   +      +D  L    ++   H+   D+    A   +AL  D      ++AH+
F11G11.2        GSGYLVGDSLTFVDLL - - VAQHTADLLAANAALLDEFPQFKAHQE
```

A spurious high-scoring alignment to a nematode glutathione S-tranferase homolog named F11G11.2

Objective score

- Function that converts an alignment into a single number
- Goal: higher number = better alignment
  - "better" = more biologically plausible
- "Objective" = a computer can calculate it
  - as opposed to "subjective" (a personal opinion), nothing to do with object-oriented programming
- (Negative) edit distance is a biologically reasonable objective score
  - Want minimum, not maximum, number of edits, so need a minus sign

$$\text{Score}\left(\begin{matrix} G & E & N & E \\ A & P & - & E \end{matrix}\right) = -3$$

Amino acid similarities

```
GRB2_CHICK    ...SVKFGND-----VQQFKV...
SRC_RSVSR     ...SIRDWDDMKGDHVKHYKI...
```

- Leucine (L) and Isoleucine (I) biochemically similar
  - High score for subsitution = +2
  - But not as high as no change (LL or II) = +4
- Leucine (hydrophobic) and Aspartic Acid (D) (hydrophilic) biochemically different
  - Low score for substitution = -4

Substitution Matrices

- Given a pair of aligned sequences, how do we assign a **score** that gives relative likely hood that the sequences are related?
- Random model (R): Assumes that every letter in the sequence occurs independently. Thus, the probability of the two sequences is the product of probabilities of each amino acid.
- $P(S_1,S_2/R)=\Pi p_i\ \Pi\ p_j$ where p_i and p_j *denote probabilities of ith and* jth symbols in the two sequences.

Substitution matrices

- In the alternate match model *M,* the aligned pair of residues occur with a joint probability p_{ij} , which can be thought of as the probability that the residues *i* and *j* have been derived from some unknown originl residue *k* in their common ancestor *(k* could be same as *i* and/or *j).* Thus

$$P(S_1,S_2/R)=\Pi p_{ij}$$

Substitution matrices

- The ratio of these two likelihood is known as the *odds-ratio*

$$(\Pi p_i\ \Pi\ p_j) / \Pi p_{ij}$$

In order to arrive at an additive scoring system, we take the logarithm of this ratio, known as the log-odds ratio:

$$\sum s(S_1(i), S_2(i))$$

Where $s=log\{(\Pi pi\ \Pi\ pj) / \Pi pij\}$

is the log likelihood of the residue pair occuring as an aligned pair. These scores can then be arranged as a 20X20 matrix Called the *score matrix or substitution matrix.*

Substitution Matrix: Blosum 50 matrix

Substitution matrix: BLOSUM62

	A	C	D	E
A	**4**	0	−2	−1
C	0	**9**	−3	−4
D	−2	−3	**6**	2
E	−1	−4	2	**5**

Subsititions, e.g. C+E
score depends on how likely the two amino acid types are to substitute for each other.

Identities, e.g. E+E
High scores (main diagonal).

Matrix is symmetrical

Optimal Alignment with Gap

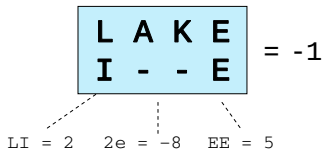
- Definition:
  - A ***gap*** in an alignment between two strings is a run of contiguous spaces.
- An insertion or deletion of a character was represented by a space.
- Each occurrence of such a space character in the alignment is considered to be a mutation.
- Sometimes a gap of more than one space can be created by only one mutational or evolutionary event. To handle this kind of situation, we need to develop a model of alignment cost function that does not attribute a negative cost or penalty based on the length of the gap.

- Examples of 'gaps' in biological context is numerous.
- The case of cDNA is a good biology application.
  - In a genome, not all DNA are responsible for the production of proteins or hormones;
  - those that carry these functions are said to be expressed.
- To study this phenomenon, biologists make DNA , called **cDNA**, corresponding to **mRNA** that leaves the nucleolus to cytoplasm for translation, by replacing **uracil (*U*)** in RNA by **thyamine (*T*)** .
- Concatenation of these DNA strands then corresponds to the **exon** of the gene.

- If we now sequence the cDNA and compare this with similar DNA in the chromosomal DNA, we would have obtained a map of chromosomal genes that are expressed.
- While doing this similarity search, the **introns** have to be aligned with long gaps.
- Recall a gene may be distributed over several segments with interleaving introns. If we used our scoring scheme for similarity search here, we would have penalized heavily our total score for the alignment (since gap will translate into a set of contiguous delete operations) and the similarity of the cDNA with some segment of chromosomal DNA would be missed.
- The alignment that best reflect the relationship consists of a few regions of strong similarity interspersed with long regions of gaps.

Gap penalties

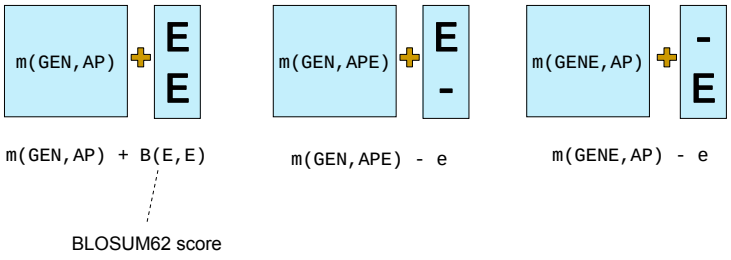
- Gaps subtract a value from the objective score
- Simplest design: “linear” penalties
- a fixed parameter (e) multiplied by length of gap
  - “e” for “extension”
  - e = 4
- Subtract e for every “-” in the alignment



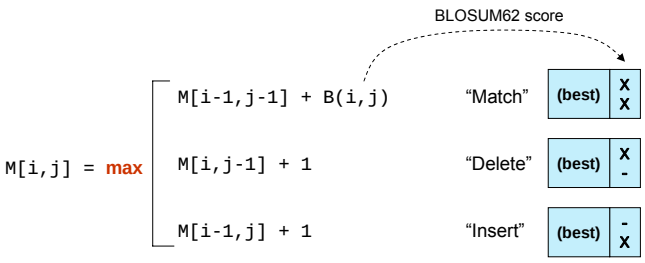
Computing max objective score

max Score(GENE,APE) = ?

Notation: max Score(A,B) = V(A,B)=m(A,B)



Recursion relations



M[i,j] is same as V[i,j] as described earlier

- Dynamic programming very similar to edit distance
  - max instead of min
  - BLOSUM62 score instead of S = 1 or 0

Problem with linear gap penalties

```
GRB2_CHICK    ...SVKFGN---D-VQQFKV...
SRC_RSVSR     ...SIRDWDDMKGDHVKHYKI...

GRB2_CHICK    ...SVKFGND----VQQFKV...
SRC_RSVSR     ...SIRDWDDMKGDHVKHYKI...
```

- These alignments have same objective score with linear penalties
- But lower alignment is more biologically reasonable
  - One gap instead of two = one insertion / deletion event instead of two

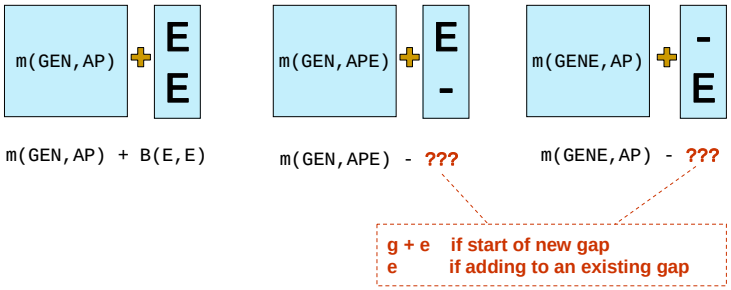
Affine gap penalties

- Prefer fewer gaps (parsimony: fewer insert / delete events)
- Penalty = g + ek
  - g = “gap open” or “per-gap” penalty, typical g = 9
  - e = “gap extension” penalty, typical e = 2
  - k = gap length (number of consecutive “-” symbols)

```
GRB2_CHICK    ...SVKFGN----D-VQQFKV...
SRC_RSVSR     ...SIRDWDDMKGDHVKHYKI...
```
gap penalty = -2g - 5e = -28

```
GRB2_CHICK    ...SVKFGND-----VQQFKV...
SRC_RSVSR     ...SIRDWDDMKGDHVKHYKI...
```
gap penalty = -g - 5e = -19

Problem with recursion relations



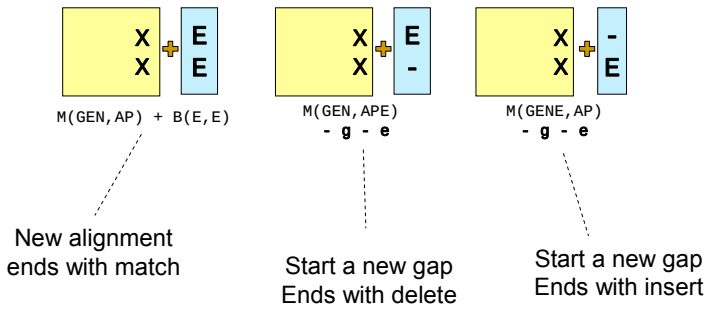
Solution: three similarity matrices

M[i,j] = objective score of best alignment of
first i letters of A to first j letters of B
that ends in a match

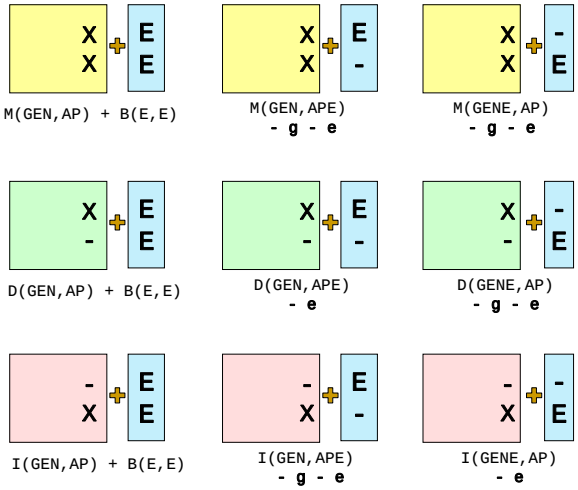
D[i,j] = objective score of best alignment of
first i letters of A to first j letters of B
that ends in a delete

I[i,j] = objective score of best alignment of
first i letters of A to first j letters of B
that ends in an insert

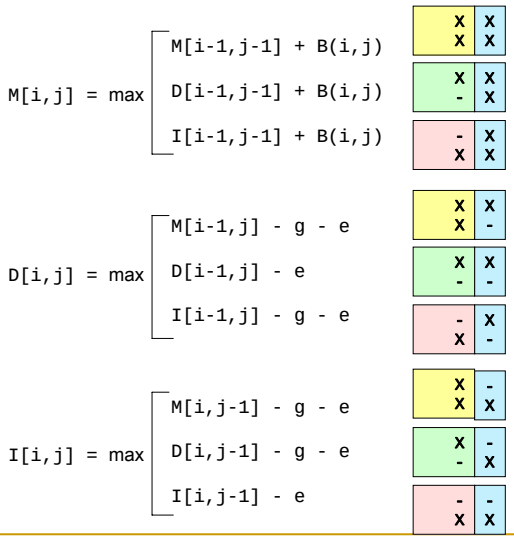
E.g. sub-alignment ends in match



Recursion relations



Recursion relations



Global alignment

- Global alignment
  - all letters from both sequences
- Objective score: substitution matrix + affine gap penalties
- Three similarity matrices M,D,I
- Three trace-back matrices (if alignment needed as well as score)

Needleman-Wunsch Algorithm

- Global alignment by dynamic programming often called “the Needleman-Wunsch algorithm”
  - 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 Biol* **48**(3): 443-53.
  - Paper describes an algorithm with fixed gap penalty (independent of length)
  - First application of dynamic programming to biological sequences

Local Alignments

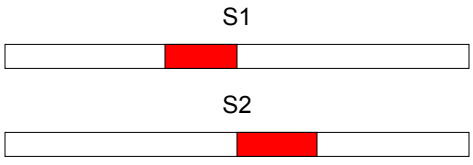
- A particularly interesting variant of similarity search is local alignment or similarity.
- Suppose we have two long DNA sequences in which there is a particularly interesting subsequence representing a gene that are common between the sequences.
  - Doing a global alignment or similarity search will not be able to identify this because there may be a lot of dissimilarity in the rest of the sequence which yield a low value for similarity and a large value of edit distance, none of which say anything about this interesting region.
- If the regions of highly similar local alignment are small, they might get lost in the context of global alignment.

Local alignment

- Often called “the Smith-Waterman algorithm”
  - Smith, T.F. and Waterman, M.S. (1981) Identification of common molecular subsequences. *J Mol Biol* **147**(1): 195-7.
  - Introduces the critical “all prefixes of all suffixes” trick.
- Surprisingly, only small modification of global case
  - Many more local alignments than global alignments
  - Prior to Smith-Waterman paper, algorithms were much slower

Problem Definition

- Given two strings *S1* and *S2*, find substrings α of *S1* and β of *S2* such that the similarity of these two substrings has maximum value over all pairs of substrings from *S1* and *S2*.



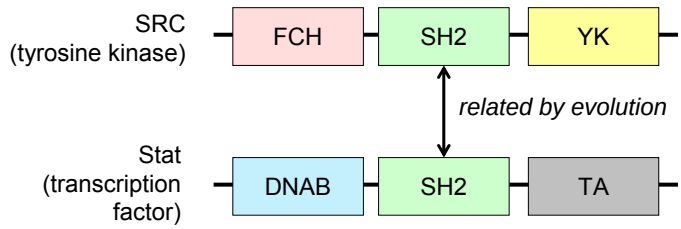
Example

- *S1*=pqr**axabcs**tuv and *S2*=xy**axbacs**ll, where α=**axabcs and** β=**axbacs.**
- With weight assigned as match=+2, mismatch=-2 and space=-1, the following gives one of the maximum value alignment:

a x a b - c s
a x - b a c s
2 2 -1 2 -1 2 2 = 8 total

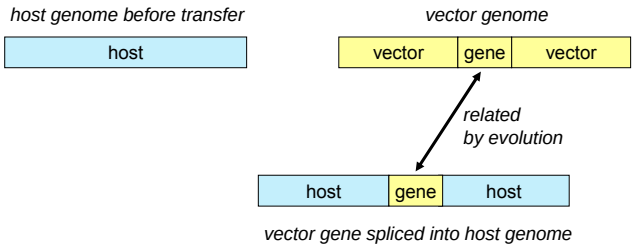
Local alignment

- Global alignment often doesn't make biological sense
- Example: protein domains



Local alignment

- Example: lateral transfer between genomes



Local alignment

- Define the problem
- Given an objective score function, strings A, B
- Find alignment of two substrings of A and B with highest score

```
THEMOTIVATIONFORALIGNMENT
   ISTOFINDTHEMUTATIONS

THEMOTIVATION
THEMUT--ATION
```

- An obvious exhaustive algorithm is to enumerate all the substrings of *S1* and *S2* and execute a dynamic programming algorithm on each pair.
  - There are $O(n^2m^2)$ such pairs.
- For one string, a substring is defined by two positions the string which can be chosen in $O(n^2)$ and $O(m^2)$ ways for *S1* and *S2*, respectively.
- For each pair, dynamic programming takes $O(nm)$ time. Thus, the complexity of such an approach is $O(n^3m^3)$.

Local alignment: the trick

Set of all substrings = set of all suffixes of all prefixes

Example. String = "ABC"

Set of suffixes = "ABC" "BC" "C"

Set of prefixes of	"ABC"	is	"A" "AB" "ABC"
Set of prefixes of	"BC"	is	"B" "BC"
Set of prefixes of	"C"	is	"C"
	All prefixes of "ABC"		All substrings of "ABC"

Local alignment, edit distance

Re-define the similarity matrix.

For global alignment:

M[i,j] = smallest edit distance between
the first i letters in A and
the first j letters in B.

For local alignment:

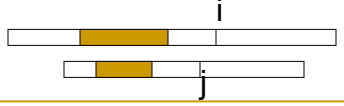
M[i,j] = smallest edit distance between
any suffix of the first i letters in A and
any suffix of the first j letters in B.

Local alignment, edit distance

In other words,

M[i,j] = smallest edit distance between
any suffix of (the prefix of A of length i)
any suffix of (the prefix of B of length j)

So smallest value of M for all i,j considers all prefixes of A and B
= smallest edit distance between any suffix of any prefix of A and any suffix of any prefix of B
= edit distance of the best local alignment of A and B.



More Restricted Version of Problem

- Definition:
  - Given two strings *S1* and *S2*, and integers *i<=n and j<=m*, the ***local suffix alignment problem*** is to find a (possibly empty) suffix α of *S1*(1……*i*) and a (possibly empty) suffix β of *S2*(1…..*j*) such that the pair of suffixes *V*= (α,β) has the maximal alignment value *v*(*i,j*) (which is greater than equal to 0 since the definition ALLOWS both α and β to be empty).

Example

1 2 3 4 5 6 7 1 2 3 4 5 6

Let *S1*= *a b c x d e x* and *S2*= *x x x c d e*

Match=+2, mismatch or space=-1

V(3,4)= (*c,c)* and *v*(3,4)=2

V(4,5)= (*cx,cd)* and *v*(4,5)=1

V(5,5)= (*xd,xcd)* and *v*(5,5)=3

V(6,6)= (*xde,xcde)* and *v*(6,6)=5 etc.

Algorithm to find value of optimal *v(i,j)*

- The algorithm is very similar to the algorithm to determine maximum similarity of two strings.
- Use again recurrence relations.
- Make reasonable assumptions about insert and delete operations as $s(-, x) \leq 0$ and $s(x,-) \leq 0$, respectively.
- Since the optimal suffix to align with an empty suffix is a string of length zero, we can write the basis as:

$$v(i,0)=0$$
$$v(0,j)=0$$

The Recurrence Realtion

- For $i>0$ and $j>0$, the recurrence relations are:

$$v(i, j) = \max[0, v(i-1, j-1) + s(S_1(i), S_2(j),$$
$$v(i-1, j) + s(S_1(i),-),$$
$$v(i, j-1) + s(-, S_2(j))]$$

Justification of the Recurrence

- **We now have an additional '0' term in the expression.**
- **The justification of the recurrence is as follows.**
- **Suppose *A* is an optimal alignment of a suffix α of S_1 (1……*i*) and a suffix β of S_2(1…..*j*) .**

Justification of the Recurrence

- There are four possible cases:
  - Both α and β are empty sequence. In this case, the value =0.
  - Assume α is not empty. Then $S_1(i)$ must have aligned with either '—' or some character in S_2. If β is not empty then $S_2(j)$ must have aligned with either '—' or some character in S_1.
  - If $S_1(i)$ and $S_1(i)$ aligned in the optimal local alignment, then these two characters contribute $s((S_1(i) , S_2(j))$ to $v(i,j)$ and the remainder of its value must come from $v(i-1,j-1)$ .Thus, in this case, we have

$$v(i,j)= v(i-1,j-1) + s((S_1(i) , S_2(j))$$

Justification of the Recurrence

- Similarly, if $S_1(i)$ is aligned with either '—' , we have $v(i,j)= v(i\text{-}1,j\text{-}1) + s((S_1(i),\text{--}))$.
- and if $S_2(j)$ is aligned with either '—' ,we have $v(i,j)= v(i\text{-}1,j\text{-}1) + s((\text{-- } , S_2(j))$.
- This covers all the cases. Conversely, for each of the four terms in the recurrence relation, there is a way to choose suffixes of S_1 $(1\ldots i)$ and $S_2(1\ldots j)$ to produce the alignment corresponding to the associated terms. Hence $v(i,j)$ must be one of the four terms and the maximum term will yield the best alignment.

Optimal Local Alignment

- So far we have discussed an optimal suffix alignment, not an optimal local alignment. Fortunately, an optimal suffix alignment yields an optimal local alignment.
- More formally, let v^* be the value of the optimal local alignment. Then,
- Theorem:

$$v^* = \max[v(i, j) : i \le n, j \le m]$$

- This is an existence theorem. To be specific, the algorithm that we just described for computing maximum $v(i,j)$ actually yields the values of i^* and j^*.

Proof

- Every optimal solution to the local suffix alignment problem is a feasible solution to the local alignment problem.
- Hence $v^* \geq \max\{v(i, j)\}$
- Conversely, Suppose *Al* is an optimal local alignment. The alignment has a last character *S1* (*i**) of *S1* and a last character *S2* (*j**) of *S2*.
- This alignment is nothing but an optimal alignment between the suffix of *S1*(1……*i**) and a (possibly empty) suffix of *S2*(1…..*j**) .
- Hence, $v^* \leq v(i^*, j^*) \leq \max\{v(i, j)\}$
- Therefore $v^* = \max[v(i, j) : i \leq n, j \leq m]$

Initialization

		G	E	N	E
	0	0	0	0	0
A	0				
P	0				
E	0				

D[0,0] = 0
(edit distance between two empty strings)

Courtesy : Bob Edgar, UC Berkeley

Example

Let S = ***ABCLDEL*** and T = ***LLLCDE***, a match score +2, and a mismatch or space score -1. The dynamic programming algorithm fills the table of *v(i,j)* as:

j i		0	1 L	2 L	3 L	4 C	5 D	6 E
0		0	0	0	0	0	0	0
1	A	0	0	0	0	0	0	0
2	B	0	0	0	0	0	0	0
3	C	0	0	0	0	2	1	0
4	L	0	2	2	2	1	1	0
5	D	0	1	1	1	1	3	2
6	E	0	0	0	0	0	2	5
7	L	0	2	2	2	1	1	4

Example

The value of optimal alignment is *V*(6,6) = 5. We can construct optimal alignments by retracing from any maximum entry to any zero entry:

j i		0	1 L	2 L	3 L	4 C	5 D	6 E
0		0	0	0	0	0	0	0
1	A	0	0	0	0	0	0	0
2	B	0	0	0	0	0	0	0
3	C	0	0	0	0	↖2	1	0
4	L	0	2	2	↖2	←↑1	1	0
5	D	0	1	1	1	1	↖3	2
6	E	0	0	0	0	0	2	↖5
7	L	0	2	2	2	1	1	4

The Optimal Local Alignment

- The optimal local alignments corresponding to these paths are:

C	L	D	E
C	-	D	E

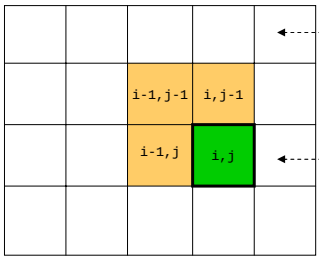
L	-	D	E
L	C	D	E

Space Complexity

- It is easy to see that the time complexity of the algorithm is ***O(nm)*** , as in the general case of dynamic programming.
- The algorithm takes ***O(nm)*** space. This is quite expensive if the sequences are large.
- If one were interested only in the value of the alignment and not obtaining a trace, this could easily be done by keeping only the last two rows of the matrix to compute the next row.
- This will need only ***O(n+m)*** space.
- Is it possible to reconstruct an alignment using only linear space?

Saving space

Compute matrix left-to-right
and top-to-bottom



This row no longer needed

Current row depends only
on previous row and current row

Need only store two rows to
compute score of best alignment
= O(L) space

(Can be done with space for one
row only).

Trace-back in O(L) space

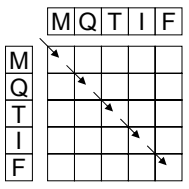
- Trace-back is harder
- Myers-Miller algorithm
  - Myers, E.W. and Miller, W. (1988) Optimal alignments in linear space. *Comput Appl Biosci* **4**(1): 11-7.
- Repeatedly divides similarity matrix in half
- About 2x slower than $O(L^2)$ algorithm

Faster speed

- Speed improvements require approximation
  - give up guarantee that an objective score is optimized
- Global alignment: k-difference
- Local and global alignment: seeds

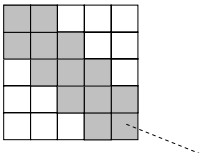
K-difference algorithm

- Global alignment of identical sequences
- Edit graph is the main diagonal



K-difference algorithm

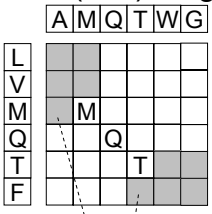
- Max k deletes or inserts, graph cannot diverge more than k cells from the main diagonal
  - k must be ≥ difference in sequence length otherwise no solution
- E.g., k=1, allow no more than 1 insert or delete



Compute only shaded region of similarity matrix(es)

Seeds

- "Seed" is a short, usually ungapped, region of high similarity
  - For example, identical sub-strings ("k-mers", "words" or "k-tuples")
- Assume seed is in the alignment
- Seed appears as a (sub)-diagonal in the edit graph



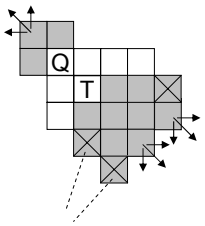
Compute only shaded region of similarity matrix(es)

Finding seeds

- Use a faster method than dynamic programming
  - so beyond the scope of this tutorial to cover this in detail
- Examples:
  - Edgar, R.C. (2004) Local homology recognition and distance measures in linear time using compressed amino acid alphabets. *Nucleic Acids Res* **32**(1): 380-5.
  - Kent, W.J. (2002) BLAT--the BLAST-like alignment tool. *Genome Res* **12**(4): 656-64.
  - Katoh, K., Misawa, K., Kuma, K. and Miyata, T. (2002) MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Res* **30**(14): 3059-66.

Extending a seed

- For local alignment, can "extend" a seed using a technique similar to k-difference algorithm for global alignment
- Explore region of similarity matrix at each end of the seed
- Stop when score drops below a threshold



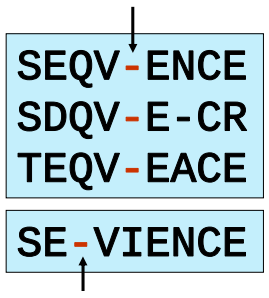
Score in these cells too low, stop

Freely available source code

- FASTA package
  - align: Myers-Miller global alignment
  - lalign: Smith-Waterman local alignment
  - fasta: fast database search by k-mer matching and d. p. extension
- BLAST (NCBI)
  - Fast database search
  - Seeds by “neighborhood” method
  - Match seeds by lookup in pre-computed index
  - Extend seeds by d. p. with score threshold

Profile alignment

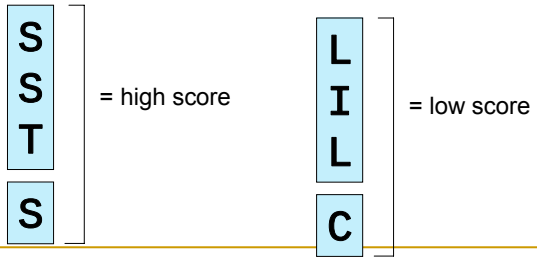
- Align an existing multiple alignment (“profile”) to a sequence
- Columns of the existing alignment kept intact



Arrows indicate gaps added to create the profile-sequence alignment.

Profile alignment

- A profile is a sequence of columns
- Apply algorithms used to align two sequences
- Replace substitution matrix for letter + letter (e.g. BLOSUM62)...
- ...by function that gives a score to column + letter
  - E.g. average BLOSUM62 score vs. all letters in the column



Example: PSI-BLAST

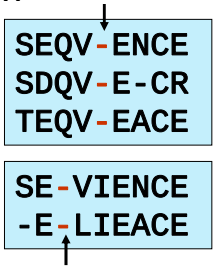
- First iteration: BLAST search of database
- Create profile (=multiple alignment) from alignment of each hit to the query sequence
- Search database with profile as a query
  - Uses modified BLAST algorithm
- Create new profile by aligning each hit to search profile
- Iterate
- Able to find more distantly related proteins than BLAST alone

Example: SAM-Txx

- Similar design to PSI-BLAST
- Uses hidden Markov model (HMMs) profile
- SAM-Txx significantly more sensitive than PSI-BLAST
- Also much slower
- http://www.soe.ucsc.edu/research/compbio/sam.html
  - Public Web server
  - License required to run locally
  - Source code not available

Profile-profile alignment

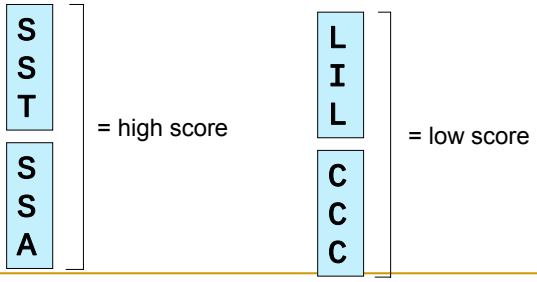
- Align two multiple sequence alignments
- Keep columns in both alignments intact
- Insert columns of gaps as needed to align them



Arrows indicate columns of gaps added to create the profile-profile alignment.

Profile-profile alignment

- A profile is a sequence of columns
- Apply algorithms used to align two sequences
- Replace substitution matrix for letter + letter (e.g. BLOSUM62)...
- ...by function that gives a score to column + column
  - E.g. average BLOSUM62 score for letters in one column vs letters in the other



Profile-profile applications

- Iterated step in multiple sequence alignment, e.g. CLUSTALW
- Distant homolog detection
  - For each sequence of known function or structure...
  - ...create a profile (e.g., by PSI-BLAST)
  - Make a database of these profiles (similar idea to PFAM)
  - Create profile of query sequence (e.g. PSI-BLAST)
  - Align query profile to profiles of all annotated sequences
  - Compute e-value
  - Works (slightly) better than profile-sequence (PSI-BLAST, SAM-Txx)
  - Works (a lot) better than BLAST

Profile-profile programs

- COMPASS
  - Sadreyev, R. and Grishin, N. (2003) COMPASS: a tool for comparison of multiple protein alignments with assessment of statistical significance. *J Mol Biol* **326**(1): 317-36.
  - Source code available (? upon request to authors).
- prof_sim
  - Yona, G. and Levitt, M. (2002) Within the twilight zone: a sensitive profile-profile comparison tool based on information theory. *J Mol Biol* **315**(5): 1257-75.

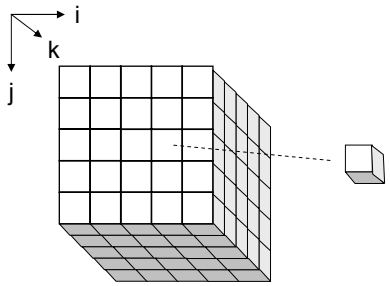
Multiple alignment

- Objective score: Sum-of-pairs (SP)
- Sum of objective score for alignment of each pair of sequences

$$SP\left(\begin{matrix}\texttt{SEQVENCE}\\ \texttt{SDQVE-CR}\\ \texttt{TEQVEACE}\end{matrix}\right) = \text{Score}\left(\begin{matrix}\texttt{SEQVENCE}\\ \texttt{SDQVE-CR}\end{matrix}\right) + \text{Score}\left(\begin{matrix}\texttt{SEQVENCE}\\ \texttt{TEQVEACE}\end{matrix}\right) + \text{Score}\left(\begin{matrix}\texttt{SDQVE-CR}\\ \texttt{TEQVEACE}\end{matrix}\right)$$

Optimize SP for N sequences

- Similarity matrices become N-dimensional
- E.g., for 3 sequences are cubes



M[i,j,k] =
score of best alignment of
first i letters in A
first j letters in B
first k letters in C

Very slow

- Time and space is $O(L^N)$
- Is NP-complete
  - Wang, L. and Jiang, T. (1994) On the complexity of multiple sequence alignment. *J Comput Biol* **1**(4): 337-48.
- Totally impractical for most biologically interesting problems
- Faster methods needed