Aalto University
School of Science

CS-E5865 Computational genomics

Autumn 2020, Lecture 3: Sequence alignment

Lecturer: Pekka Marttinen

Assistants: Alejandro Ponce de León, Zeinab Yousefi, Onur Poyraz

Sequence Alignment

Given two sequences $x = x_1x_2...x_n$, $y = y_1y_2…y_m$, an alignment is an assignment of gaps in the 2 sequences so that we line up each letter in one sequence with either a letter or a gap in the other sequence

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC
```

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

Sequence alignment

- The purposes of sequence alignment
  - to measure the similarity of two sequences
  - to reveal which parts of the sequences match and which do not
- Commonly used way to visualize pairwise alignments on the right:

  "|" denote matching pair of symbols

  "-" denotes a gap symbol inserted in the sequence to improve alignment

```
GAATTCAG        GAATTCAG
|  |  ||  |     |  ||  |  |
GGA-TC-G        GCAT-C-G

GAATTC-A        GAATTC-A
|  |  ||  |     |  ||  |  |
GGA-TCGA        GCAT-CGA
```

Global and local alignment

- Two types of alignment:
- Global alignment aims to maximize the alignment quality over the whole sequences
  - leaving gaps typically penalized
- Local alignment tries to match sub-regions of the sequences
  - gaps typically not penalized

```
Global  FTFTALILLAVAV
        F--TAL-LLA-AV

Local   FTFTALILL-AVAV
        --FTAL-LLAAV--
```

Global alignment scoring functions

- By inserting gaps in different places, we get different alignments
- We wish to find the best one
- We define a scoring function σ(x,y) for any pair of symbols in the alignment
- The alignment score is the sum

$$M = \sum_{i=1}^{c} \sigma(x_i, y_i)$$

where i indexes the positions in the alignment

```
GAATTCAG      GAATTCAG
|  |  ||  |   |  ||  |  |
GGA-TC-G      GCAT-C-G

GAATTC-A      GAATTC-A
|  |  ||  |   |  ||  |  |
GGA-TCGA      GCAT-CGA
```

Global alignment scoring functions: example

- Scoring Function:

  Match: +m

  Mismatch: -s

  Gap: -d

- Score

$$M = (\#\text{ matches}) \times m + (\#\text{ mismatches}) \times (-s) + (\#\text{gaps}) \times (-d)$$

Substitution matrices

- We can collect the scores of the scoring function σ into a matrix (on the right for our example)
- Matrix S containing the σ values is called the substitution matrix
- For DNA simple scoring schemas are typically used
- For amino acids richer substitution matrices are used
  - PAM
  - BLOSUM

	a	b	$-$
a	$+1$	-1	-1
b	-1	$+1$	-1

$$S = \begin{array}{c|ccccc} & a_1 & a_2 & \dots & a_l & - \\ \hline a_1 & \sigma(a_1,a_1) & \sigma(a_1,a_2) & \dots & \sigma(a_1,a_l) & \sigma(a_1,-) \\ a_2 & \sigma(a_2,a_1) & \sigma(a_2,a_2) & \dots & \sigma(a_2,a_l) & \sigma(a_2,-) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_l & \sigma(a_l,a_1) & \sigma(a_l,a_2) & \dots & \sigma(a_l,a_l) & \sigma(a_l,-) \\ - & \sigma(-,a_1) & \sigma(-,a_2) & \dots & \sigma(-,a_l) & \sigma(-,-) \end{array}$$

Substitution matrices: Example

- In general, the scores can depend on the pair of symbols
- Consider the following substitution matrix

	A	G	C	T	-
A	10	-1	-3	-4	-5
G	-1	7	-5	-3	-5
C	-3	-5	9	0	-5
T	-4	-3	0	8	-5

- Then the alignment:
  AGACTAGTTAC
  CGA - - -GACGT
  would have the following score:

  S(A,C) + S (G,G) + S(A,A) + 3*(-5) + S(G,G) + S(T,A) + S(T,C) + S(A,G) + S(C,T)
  = -3 +7 + 10 -3*5 +7 -4 +0 -1 +0 = 1

Optimal global alignment

- The optimal alignment A* between two sequences x and y is the alignment A(s,t) that maximizes the alignment score M over all possible alignments.
- There are $\binom{2n}{n}$ possible alignments between two sequences of length n, so brute-force enumeration of all of them is not feasible
- Can be solved efficiently by using the Needleman-Wunsch algorithm, which is based on dynamic programming (we take a closer look in the following)
  - Basic idea: solve the problem prefixes of length 1,2,...,n incrementally making use of the optimal solutions for the prefixes

Dynamic programming to the rescue

- General recipe for solving complex optimization problems where there is internal subset structure
  - e.g. subsequences of a larger sequence

- Iterate the following for k=1,...,n-1
  - Solve the smaller subproblems of size k
    - e.g. optimal alignments of subsequences of length k
  - Extend the optimal solutions for size k problems to optimal solutions of size k+1 problems

Example

- Let us find the optimal global alignment for two sequences

```
s = ATTCGT
t = CTTAGCT
```

- Let us assume the simple substitution matrix with score:
  - +1 for matching a symbol with itself,
  - -1 for matching symbol with a different symbol or a gap
- The matrix M stores the intermediate alignment scores:
  - M(i,j) stores the optimal alignment score of $s_1...s_{i-1}$ and $t_1...t_{j-1}$

Example

- First consider aligning the beginning of the two sequences
- We have three choices
  1. match s_1 against a gap before t_1 :
     $M(2,1) = \sigma(\text{'A','-'}) = -1$
  2. match t_1 against a gap before s_1 :
     $M(1,2) = \sigma(\text{'-','C'}) = -1$
  3. match the first symbols with each other: $M(2,2) = \sigma(\text{'A','C'}) = -1$
- For now, all the three choices give us the same score

$M(1,2) = M(2,1) = M(2,2) = -1$

```
s = ATTCGT
t = CTTAGCT

s = ATTCGT
     |
t = -CTTAGCT

s = -ATTCGT
     |
t = CTTAGCT

s = ATTCGT
     |
t = CTTAGCT
```

M	-	C	T	T	A	G	C	T
-	0	-1						
A	-1	-1						
T								
T								
C								
G								
T								

Example

- To extend the alignment M(2,1) we have again three choices
  1. match s_2 against a gap before t_1, score M(3,1) = M(2,1)+ σ('T','-') = -2,
  2. match t_1 against a gap before s_2 , score M(2,1)+ σ('-','C') = -2 < M(2,2)
  3. match s_2 against t_1, score: M(3,2) = M(2,1)+ σ('T','C') = -2
- Notice that the second choice gives an alignment for s_1 and t_1 with a score inferior to what we have already found and stored in M(2,2) – we ignore this choice.

```
s = ATTCGT
    ||
t = --CTTAGCT

s = A-TTCGT
    ||
t = -CTTAGCT

s = ATTCGT
    ||
t = -CTTAGCT
```

M	-	C	T	T	A	G	C	T
-	0	-1						
A	-1	-1						
T	-2	-2						
T								
C								
G								
T								

Example

- To extend the alignment M(1,2) the three choices are
  1. match s_1 against a gap before t_2, score M(1,2)+ σ('A','-') = -2 < M(2,2)
  2. match t_2 against a gap before s_1 , score M(1,3) = M(1,2)+ σ('-','T') = -2
  3. match s_1 against t_2 with, score: M(2,3) = M(1,2)+ σ('A','T') = -2
- The first choice gives yet another alignment for s_1 and t_1 with inferior score to what we have already found and stored in M(2,2) –> no update.

```
s = -ATTCGT
    ||
t = C-TTAGCT

s = --ATTCGT
    ||
t = CTTAGCT

s = -ATTCGT
    ||
t = CTTAGCT
```

M	-	C	T	T	A	G	C	T
-	0	-1	-2					
A	-1	-1	-2					
T	-2	-2						
T								
C								
G								
T								

Example

- To extend the alignment M(2,2) the three choices are
  1. match s_2 against a gap before t_2, score M(2,2)+ σ('T','-') = -2 = M(3,2)
  2. match t_2 against a gap before s_2 , score M(2,2)+ σ('-','T') = -2 = M(2,3)
  3. match s_2 against t_2 with, score: M(3,3) = M(2,2)+ σ('T','T') = 0
- The two first choices give us alignment scores that match the best scores found so far – these correspond to alternative optimal alignments

```
s = ATTCGT
    ||
t = C-TTAGCT

s = A-TTCGT
    ||
t = CTTAGCT

s = ATTCGT
    ||
t = CTTAGCT
```

M	-	C	T	T	A	G	C	T
-	0	-1	-2					
A	-1	-1	-2					
T	-2	-2	0					
T								
C								
G								
T								

Example

- We can continue in the same way:
  - consider three possible options to extend an alignment M(i,j) to M(i+1,j), M(i,j+1) and M(i+1,j+1)
  - check if we have found a better alignment before
- Iterating the process we eventually fill in the matrix M
- From the bottom right corner we find the optimal global alignment score for the two sequences

M	-	C	T	T	A	G	C	T
-	0	-1	-2	-3	-4	-5	-6	-7
A	-1	-1	-2	-3	-2	-3	-4	-5
T	-2	-2	0	-1	-2	-3	-4	-3
T	-3	-3	-1	1	0	-1	-2	-3
C	-4	-2	-2	0	0	-1	0	-1
G	-5	-3	-3	-1	-1	1	0	-1
T	-6	-4	-2	-2	-2	0	0	1

Alternative look

- Alternatively, we could fill in the matrix by considering the 3 different ways how the optimal alignment M(i,j) can arise via three different paths:
  - Extending optimal alignment between $x_1 \ldots x_{i-2}$ & $y_1 \ldots y_{j-2}$ by aligning x_{i-1} with y_{j-1}
  - Extending optimal alignment between $x_1 \ldots x_{i-2}$ & $y_1 \ldots y_{j-1}$ by aligning x_{i-1} with a gap
  - Extending optimal alignment between $x_1 \ldots x_{i-1}$ & $y_1 \ldots y_{j-2}$ by aligning y_{j-1} with a gap
- Optimal alignment score is then given by

$$M(i, j) = \max \begin{cases} M(i-1,j-1) + s(x_{i-1}, y_{j-1}) \\ M(i-1, j) + s(x_{i-1}, \text{'-'}) \\ M(i, j-1) + s(\text{'-'}, y_{j-1}) \end{cases}$$

M	-	C	T	T	A	G	C	T
-	0	-1	-2					
A	-1	-1	-2					
T	-2	-2	0					
T								
C								
G								
T								

Components of dynamic programming

- The dynamic programming approach has 3 essential components:

1. Recurrence relation: How can we compute M(i,j) knowing only the values M(i',j') with i'≤i and j'≤j?

2. Tabular computation: How to store efficiently the computed values in order to avoid computing them over and over again?

3. Traceback: How to find the actual alignment of the 2 sequences after we have computed the similarity values?

The recurrence relation

- We need to establish a recursive relationship between the value M(i,j) with i,j≥1 (i.e., the similarity between $x_1x_2...x_{i-1}$, and $y_1y_2...y_{j-1}$) and values of M with index pairs smaller than i,j.
- Base conditions:
  a. M(1, 1) = 0
  b. M(1, j) = - j × d
  c. M(i, 1) = - i × d

  where –d is the score of a gap
- The recurrence relation for M(i,j) with i, j > 1 based on the principle of optimality:

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(x_{i-1}, y_{j-1}) \\ M(i-1, j) - d \\ M(i, j-1) - d \end{cases}$$

Tabular computation

- We fill in the table M(i,j), with $0 \le i \le n$ and $0 \le j \le m$, in an increasing order of pairs (i,j).

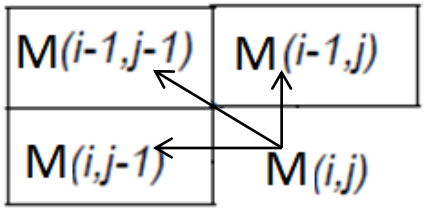
- First, we initialize first row and column according to the base cases of the recurrence relation:
  a. $M(1, 1) = 0$
  b. $M(1, j) = -j \times d$
  c. $M(i, 1) = -i \times d$

Tabular computation

- The values of the inner cells M(i,j) (i,j > 0) can be computed in any order as long as the three values required by the recurrence relation have been computed:

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(x_{i-1}, y_{j-1}) & \text{[case 1]} \\ M(i-1,j) - d & \text{[case 2]} \\ M(i,j-1) - d & \text{[case 3]} \end{cases}$$

$$TB(i,j) = \begin{cases} \text{DIAG}, & \text{if [case 1]} \\ \text{UP}, & \text{if [case 2]} \\ \text{LEFT}, & \text{if [case 3]} \end{cases}$$



Needleman-Wunsch global alignment algorithm

Input: x,y = sequences to align and sigma, gapsigma = alignment scores for non-gaps and gaps

Output:

- M = dynamic programming matrix of optimal alignment scores
- TB = matrix storing the traceback path

```
n = length(x); m = length(y);
```

Initialization:

```
M(1,1)=0
for j = 2:m+1    M(1,j) = gapsigma* j; % penalty of gaps preceding s
for i = 2:n+1    M(i,1) = gapsigma* i;  % penalty of gaps preceding t
```

Main iteration: Filling-in partial alignments

For each i = 2……n+1

For each j = 2……m+1

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(x_{i-1}, y_{j-1}) & \text{[case 1]} \\ M(i-1, j) + s(x_{i-1}, \text{‘-’}) & \text{[case 2]} \\ M(i, j-1) + s(\text{‘-’}, y_{j-1}) & \text{[case 3]} \end{cases}$$

$$TB(i,j) = \begin{cases} \text{DIAG,} & \text{if [case 1]} \\ \text{UP,} & \text{if [case 2]} \\ \text{LEFT,} & \text{if [case 3]} \end{cases}$$

Traceback: recovering the alignment

- Outputting the alignment corresponding to the optimal score requires parsing back the path that we took when computing the value for the cell

```
s =  ATTCGT   s = ATTCG-T
         |          |||||
t = CTTAGCT   t = CTTAGCT

s = ATTCG-T   s = ATTCG-T
        ||         ||||||
t = CTTAGCT   t = CTTAGCT

s = ATTCG-T   s = ATTCG-T
        |||        |||||||
t = CTTAGCT   t = CTTAGCT

s = ATTCG-T
        ||||
t = CTTAGCT
```

M	-	C	T	T	A	G	C	T
-	0	-1	-2	-3	-4	-5	-6	-7
A	-1	-1	-2	-3	-2	-3	-4	-5
T	-2	-2	0	-1	-2	-3	-4	-3
T	-3	-3	-1	1	0	-1	-2	-3
C	-4	-2	-2	0	0	-1	0	-1
G	-5	-3	-3	-1	-1	1	0	-1
T	-6	-4	-2	-2	-2	0	0	1

When does dynamic programming work?

- Key property: optimal solution for the whole problem can be decomposed into optimal solutions for subproblems
- In our case: optimal alignment of the whole sequence is composed of
  - optimal alignment of prefixes of two strings
  - optimal alignment of the last symbols of the strings
- Our score function decomposes
  - the symbols outside the subset do not affect the optimality of the alignment
  - this would not be the case if we allowed the the alignments of the symbols to cross arbitrarily

```
s = -ATTCGT
     | X X |
t = CTTAGCT
```

Local alignment

- A local alignment of two sequences s and t is a global alignment $s_{(i:j)}$ and $t_{(k:l)}$ for some choice of (i,j) and (k,l)
- The optimal local alignment A is given by the choice of (i,j) and (k,l) that maximize the alignment score

  $$M(A(s_{(i:j)}, t_{(k:l)}))$$
- Optimal local alignments can be found by a dynamic programming algorithm called Smith-Waterman that is only a minor modification of the Needleman-Wunch global alignment algorithm

Smith-Waterman local alignment

- Simple modification to the global alignment
- An additional update condition preventing the score from getting negative values

$$M(i, j) = \max \begin{cases} M(i-1,j-1) + \sigma(s_{i-1}, t_{j-1}) \\ M(i-1, j) + \sigma(s_{i-1}, \text{‘-’}) \\ M(i, j-1) + \sigma(\text{‘-’}, t_{j-1}) \\ 0 \end{cases}$$

- Interpretation: if extending the current global alignment yields a negative score, better score is obtained by starting a new alignment region
  - Ignore badly aligning regions

Smith-Waterman local alignment

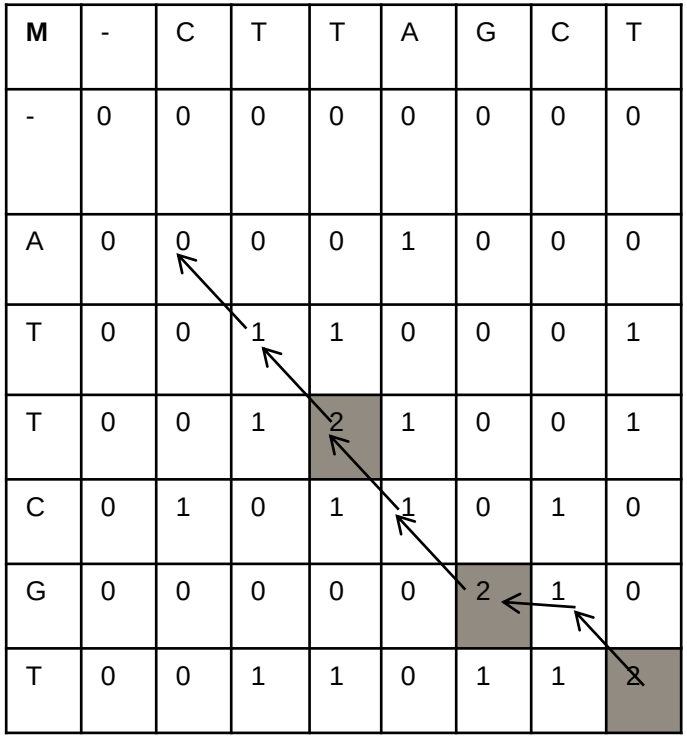
- The value in M(i,j) denotes the score of local alignments that end at the symbols s_{i-1} and t_{j-1}
- The largest values in the matrix denote the optimal local alignment end points
- In our example, we have three possible end points, corresponding to three different local alignments

M	-	C	T	T	A	G	C	T
-	0	0	0	0	0	0	0	0
A	0	0	0	0	1	0	0	0
T	0	0	1	1	0	0	0	1
T	0	0	1	2	1	0	0	1
C	0	1	0	1	1	0	1	0
G	0	0	0	0	0	2	1	0
T	0	0	1	1	0	1	1	2

```
s = ATTCG-T      s = ATTCGT       s = ATTCGT
     ||||  |          ||||             ||
t = CTTAGCT      t = CTTAGCT      t = CTTAGCT
```

Traceback for finding the local alignment

1. Start with the largest value in the matrix
   - corresponds to the last position in the alignment region
2. Trace back until a zero is found
3. Here we have multiple maximum values
   - each one corresponds to a different, equally good local alignment
   - to break ties, picking the longest one might be a good policy



M	-	C	T	T	A	G	C	T
-	0	0	0	0	0	0	0	0
A	0	0	0	0	1	0	0	0
T	0	0	1	1	0	0	0	1
T	0	0	1	2	1	0	0	1
C	0	1	0	1	1	0	1	0
G	0	0	0	0	0	2	1	0
T	0	0	1	1	0	1	1	2

Smith-Waterman local alignment algorithm

Input: s,t = sequences to align and sigma, gapsigma = alignment scores for non-gaps and gaps

Output:

- M = dynamic programming matrix of optimal alignment scores
- TB = matrix storing the traceback path

TB(i,j):

- 1 denotes match (s(i-1),t(j-1)),
- 2 denotes match (s(i-1),'-'),
- 3 denotes match ('-',t(j-1)),

```
n = length(s); m = length(t);
Initialization:
M = zeros(n+1,m+1); % initialize with zeros
TB = zeros(n+1,m+1);
Main iteration:
```

For each i = 2……n+1

For each j = 2……m+1

$$M(i,j) = \max \begin{cases} M(i-1,j-1) + s(s_{i-1}, t_{j-1}) & \text{[case 1]} \\ M(i-1, j) + s(s_{i-1}, \text{'-'}) & \text{[case 2]} \\ M(i, j-1) + s(\text{'-'}, t_{j-1}) & \text{[case 3]} \\ 0 & \text{[case 4]} \end{cases}$$

$$TB(i,j) = \begin{cases} 1, & \text{if [case 1]} \\ 2, & \text{if [case 2]} \\ 3, & \text{if [case 3]} \\ 0 & \text{if [case 4]} \end{cases}$$

Gap penalty schemes

- So far we have used a simple gap penalty scheme, where each gap symbol incurs a constant penalty
  - We may over-penalize gaps that are several symbols long
- In practice, an affine gap penalty scheme is frequently used
- Affine gap penalty is composed of
  - gap opening penalty: paid by the first gap in a sequence of gaps
  - gap extension penalty: paid by the following gaps
- Dynamic programming based algorithms can be adapted to these gap penalty schemes

Statistical significance of sequence alignments via randomization

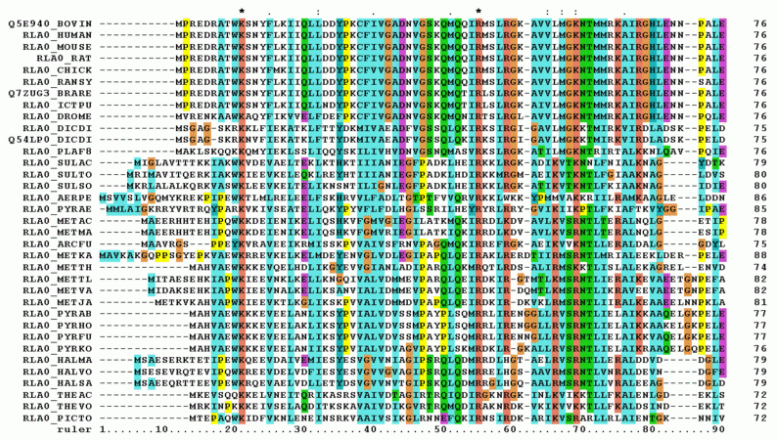
- Good alignment scores may also happen by chance, so we need to consider the statistical significance of alignment scores
- If a known probability distribution for the null model is available, we can use that to compute p-values
- If not, randomization can be used here as a tool:
  1. Generate a large set R of randomized versions s' of sequence s.
  2. Align the sequence t against the randomized sequences s'
  3. Compute the distribution of observed alignment scores
  4. The fraction of randomized alignment scores M(s',t) that have score greater than or equal to the score M(s,t) gives the P-value

$$P\{score \geq M\} = \frac{|\{s' \in R | M(s',t) \geq M\}|}{|R|}$$

Multiple sequence alignment

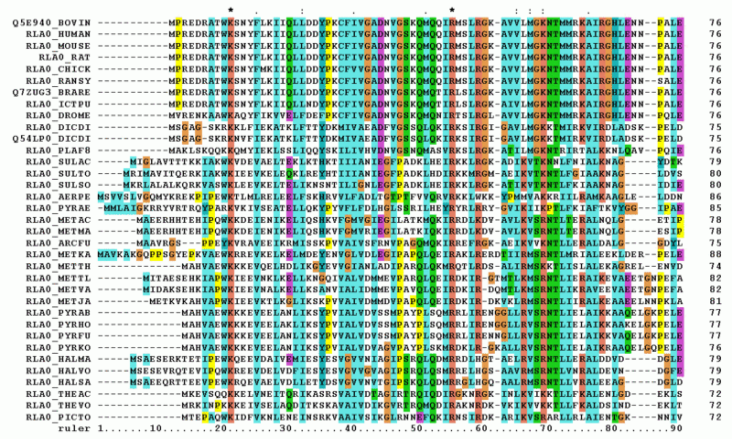
- Multiple sequence alignment is a generalization of a pairwise alignment
  - aim to align a group of sequences with a high alignment score

- Useful for finding regions of sequence that were conserved in evolution
  - e.g. functional protein domains



Multiple sequence alignment (MSA)

- Computationally harder than pairwise alignment
  - CPU-time scales exponentially w.r.t. the number of sequences aligned
  - NP-hard: little hope of finding an efficient optimal algorithm
- Heuristic methods such as CLUSTALW, MUSCLE, MAUVE use pairwise alignments as a tool to construct MSA
- A commonly used technique nowadays to align whole genomes is to align all genomes against a single reference genome



BLAST

Sequence retrieval from large databases

- The running time of Needleman-Wunsch and Smith-Waterman algorithms both scale proportionally to the size of the matrix M, which is quadratic in the length of the sequences
- On modern huge sequence databases, this is too much
  - Also wasteful, since the majority of sequences are not expected to have significant similarity to the query sequence
- In practice, the goal of finding the optimal alignment need is sacrificed for speed

BLAST (Basic Local Alignment and Search Tool)

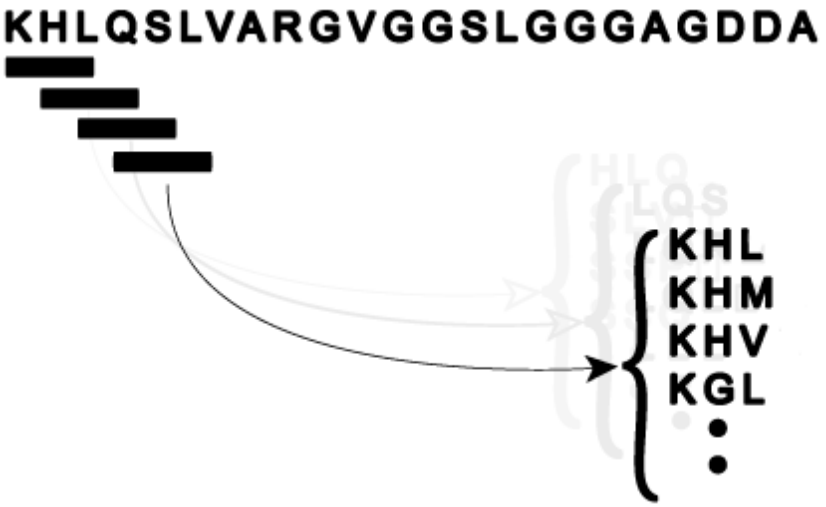
- BLAST is the most widely used fast, non-optimal alignment tool
- “blasting” is a synonym for aligning sequences and finding matches from large sequence databases
- Here we assume a setting where we have one *query sequence* and a large database (e.g. Genbank) and we want to find the most similar sequences from the database.

Basic local alignment search tool
SF Altschul, W Gish, W Miller, EW Myers... - Journal of molecular ..., 1990 - Elsevier
A new approach to rapid sequence comparison, **basic local alignment search tool** (BLAST), directly approximates alignments that optimize a measure of **local** similarity, the maximal segment pair (MSP) score. Recent matImmatical results on the stochastic properties of **...**
Cited by 55641 Related articles All 103 versions Cite Save

BLAST working principle

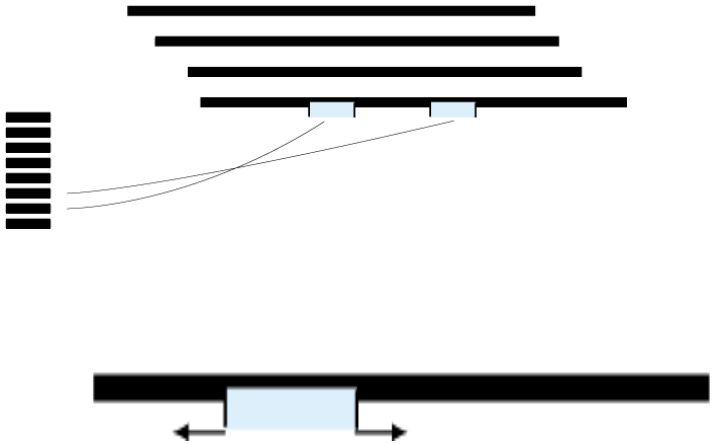
- BLAST relies on finding matching short substrings in the query sequence and a database sequence

1. First, all length-k substrings of the query sequence, called the *query words*, are extracted
2. By using a substitution matrix (e.g. BLOSUM62), the set of substrings is expanded to a set of high-scoring substrings
   – those that have alignment score with the original substrings higher than a fixed threshold



BLAST working principle

3. The high-scoring substrings are searched in a database, and matching sequenced are retrieved
4. Each matched substring is extended to right and left until the alignment score starts to decrease. The result is called a Maximal Segment Pair (MSP)
5. The resulting MSPs with score above a given threshold are tested for statistical significance
6. Several MSPs that hit the same database sequence are combined into an alignment with gaps



BLAST Bitscore and p-value

- BLAST computes several statistics of the aligned sequences
- Bitscore is a normalized version of the alignment score,

$$S' = \frac{\lambda S - \ln(K)}{\ln(2)}$$

- K and λ are constants depending on the gap penalties and the substitution matrix used (found by fitting to a Gumbel Extreme Value distribution)
- Bitscore estimates the magnitute of the search space we have to look through before we expect to find just by chance a score as good as or better than the one we have:
  - expected $2^{S'}$ alignments need to be examined to find a bitscore of S' by chance.
- Expressed as a p-value $P(score \geq S') = 2^{-S'}$

BLAST E-value

- When searching for a best match for the query sequence in a large database, we are preforming a large number of statistical significance tests.
- P-values get inflated due to multiple testing
- E-value is a correction applied to the BLAST p-value:

$$E = nN \cdot P(score \geq S') = nN \cdot 2^{-S'}$$

- n is the length of the query sequence, N is the total length of sequences in the database
- nN is the approximate number of potential alignment locations (ca. n substrings per query sequence, ca. N locations to align to in the database)