Sequence alignment heuristics

Lecture 8

Optimal global alignment

- Given 2 input strings S1 and S2, and **the scoring matrix**, find an alignment with the maximum possible score
  - The scoring matrix includes also the gap scoring scheme
  - The optimality of an alignment heavily depends on the scoring matrix

Example. Scoring matrix 1

+1 – for match
-1 – for mismatch
-2 – for indel

S1	**A**	**G**	**C**	**A**
S2	**A**	**C**	**T**	**A**

S1	**A**	**G**	**C**	**A**
S2	**A**	**C**	**T**	**A**
	+1	-1	-1	+1

Total best score: 0

Example. Scoring matrix 2

+1 – for match
-1 – for mismatch
-1 – for indel

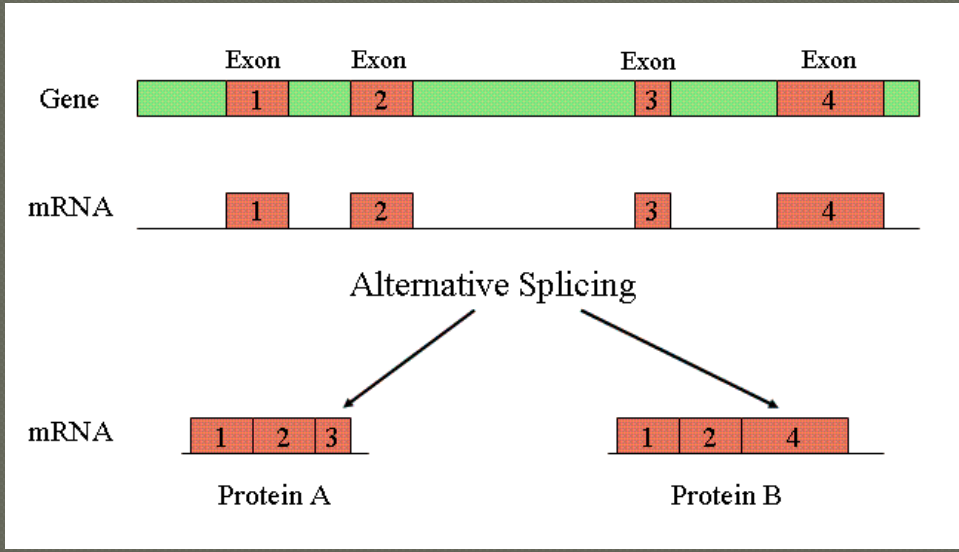
S1	**A**	**G**	**C**	**A**
S2	**A**	**C**	**T**	**A**

S1	**A**	**G**	**C**	**----**	**A**
S2	**A**	**----**	**C**	**T**	**A**
	+1	-1	+1	-1	+1

Total best score: 1

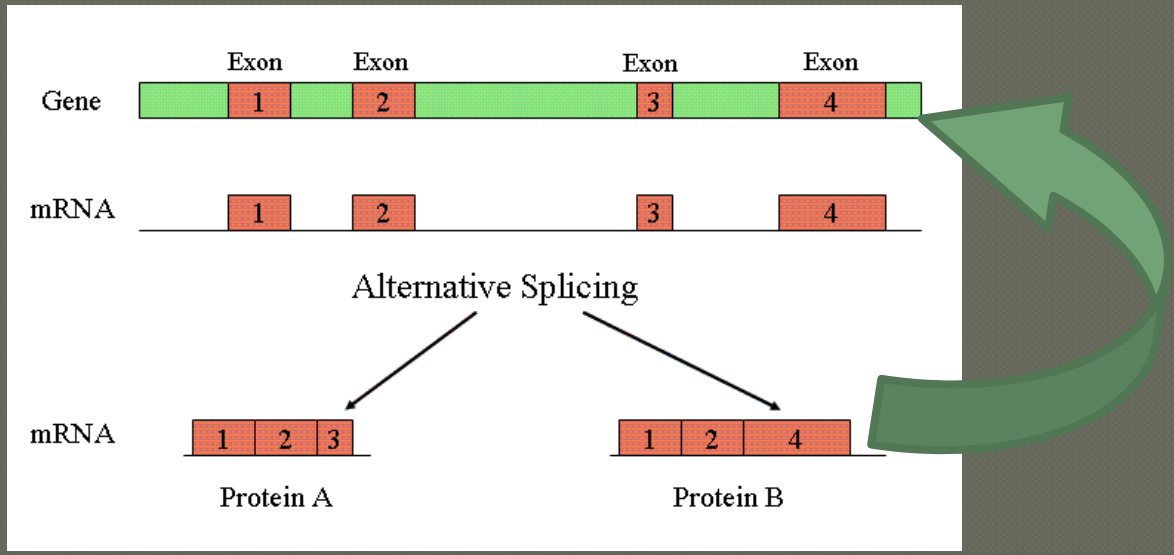
Choosing the scoring scheme

- C-DNA example
  - In Eukaryotes, a protein-coding gene is made of alternating exons (expressed sequences) and introns (intervening sequences), which do not code for a protein
  - The number of exons is generally modest (4-20), but the lengths of introns can be huge comparing to the length of introns



Choosing the scoring scheme

- C-DNA example
  - If we extract m-RNA from the working cell at any stage of its working cycle, and then we transcribe it backwards into DNA by using viral reverse transcriptase, we obtain c-DNA
  - Now we want to align the obtained c-DNA with the region of genome in order to locate the gene and to study the mechanism of splicing



Choosing the scoring scheme for c-DNA alignment

- If the spaces are penalized on the unit bases, that would align c-DNA substrings close together rather than allowing large gaps corresponding to introns
- The number of mismatches in the aligned regions should not be large, since these regions are just a transcript of the original genomic DNA

Scores for c-DNA alignment

- The following scoring scheme solves the problem:
  - Constant gap weight
  - Heavy penalty for mismatches
- The optimal alignment can be induced to cut up c-DNA to match the exons on the DNA sequence they have originated from

Diagonal method for finding similar sequences

- If we only want to find the sequences in the database, which are highly similar to a new sequence, we can enforce an additional constraint – the number of insertions, deletions and substitutions not larger than some threshold value k
- In this case, we can compute the values of the dynamic programming only in a 2k+1 strip around the main diagonal
- The Miller-Myers algorithm could work well (the edit distance is small)

When to do the global alignment

- Since the mutational events include multiple genome rearrangements (insertions, deletions, inversions), in addition to mutations, a global alignment of distantly related genomes is of a little biological value

- However, if we would be able to align 2 genomes of the same species (almost identical), we could reveal the sites of polymorphism

Aligning complete genomes. Challenges

- In order to align 2 genomes, 1,000,000,000 nucleotides each, by a traditional dynamic programming, it will require 10 days of computation on a modern computer system
- The dynamic programming is not easily parallelizable, since we cannot start to compute next value until 3 required previous values have been computed

Heuristic Algorithms

- A *heuristic method* is an algorithm that gives only approximate solution to a given problem.
- Sometimes we are not able to formally prove that this solution actually solves the problem, but heuristic methods are commonly used because they are much faster than exact algorithms.

The MUMMer algorithm
by Delcher et al., 1999

- 3 main algorithmic tools
  - Suffix tree
  - Longest increasing subsequence
  - Dynamic programming
- 3 steps
  - Find maximal unique matches – MUMs
  - Find the longest sequence of MUMs
  - Fill the gaps between MUMs using dynamic programming alignment

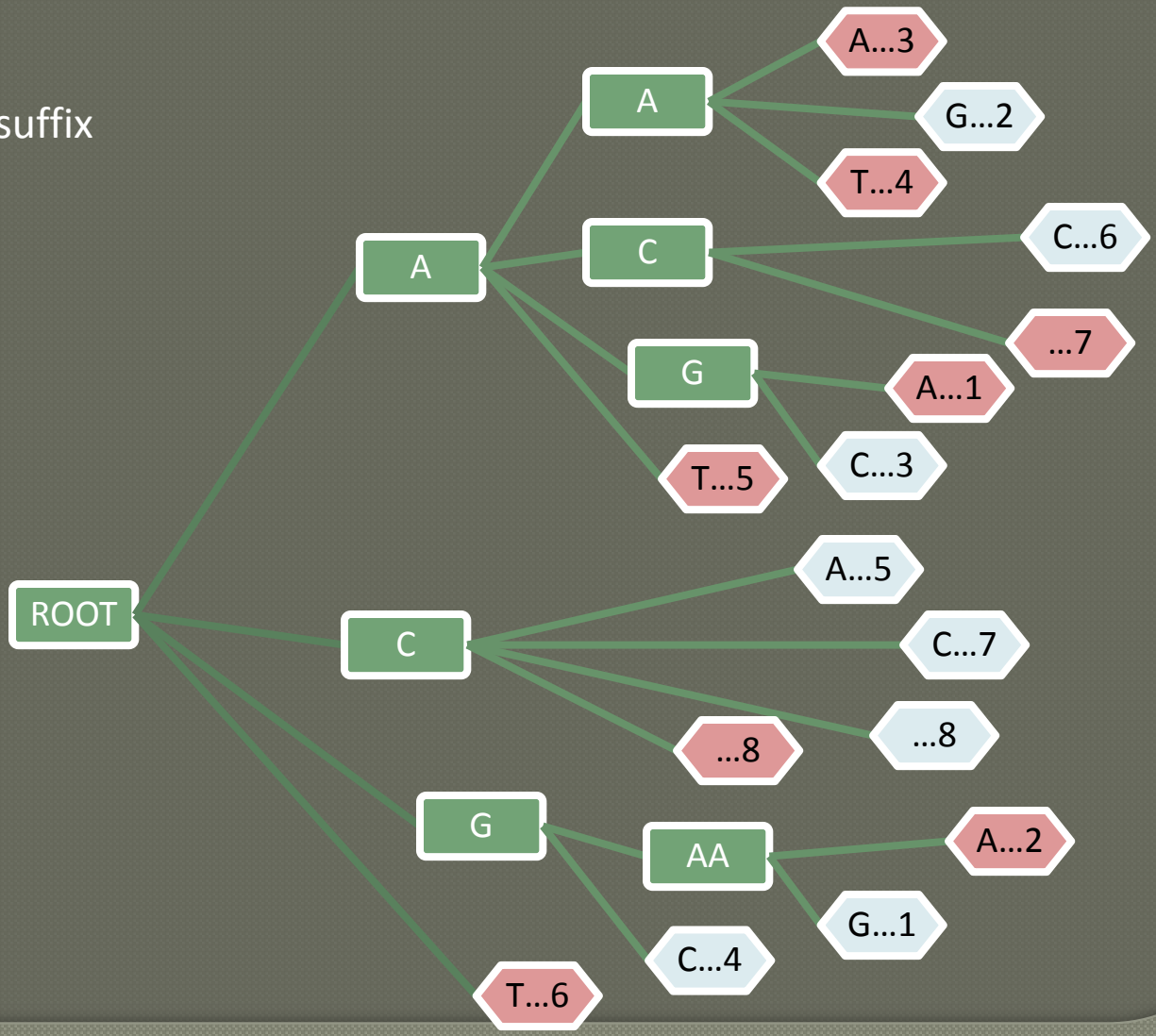
The MUMmer Algorithm. Step 1

- Find maximal unique matches – MUMs
  - A *MUM* is a substring which occurs exactly once in S1 and once in S2, and is not contained in any other such substring
  - We can find maximal repeating substrings from a suffix tree
  - Among these, take only the nodes which have exactly 2 children – 1 representing some suffix of S1, and 1 representing some suffix of S2
  - Output MUMs in form of pairs of start positions (position in S1, position in S2)

	1	2	3	4	5	6	7	8
S1	g	a	a	g	c	a	c	c
S2	a	g	a	a	a	t	a	c

MUMs example

Build a generalized suffix tree for S1 and S2

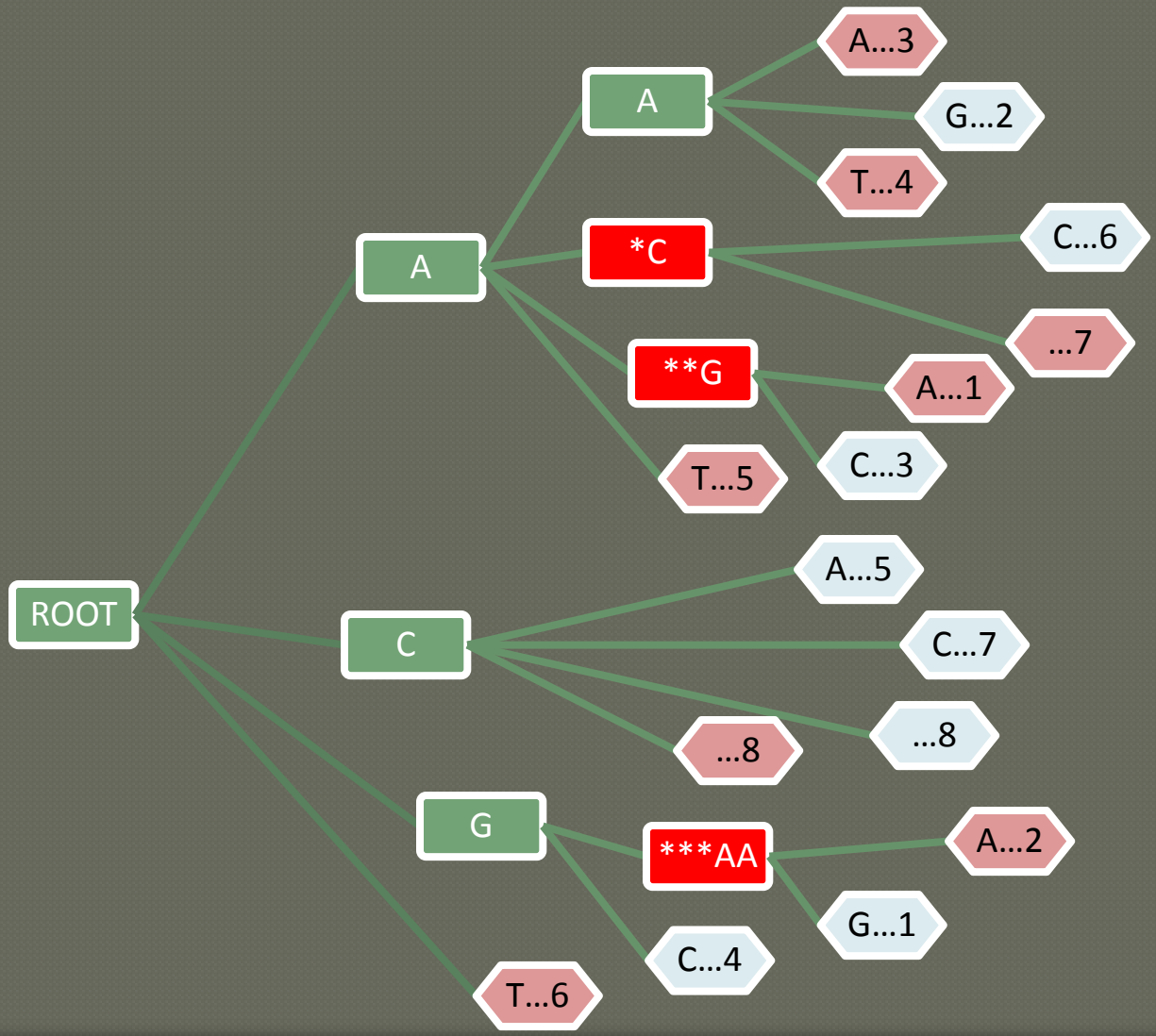


MUMs example

	1	2	3	4	5	6	7	8
S1	g	a	a	g	c	a	c	c
S2	a	g	a	a	a	t	a	c

Maximal unique matches:

- *C
- **G
- ***AA

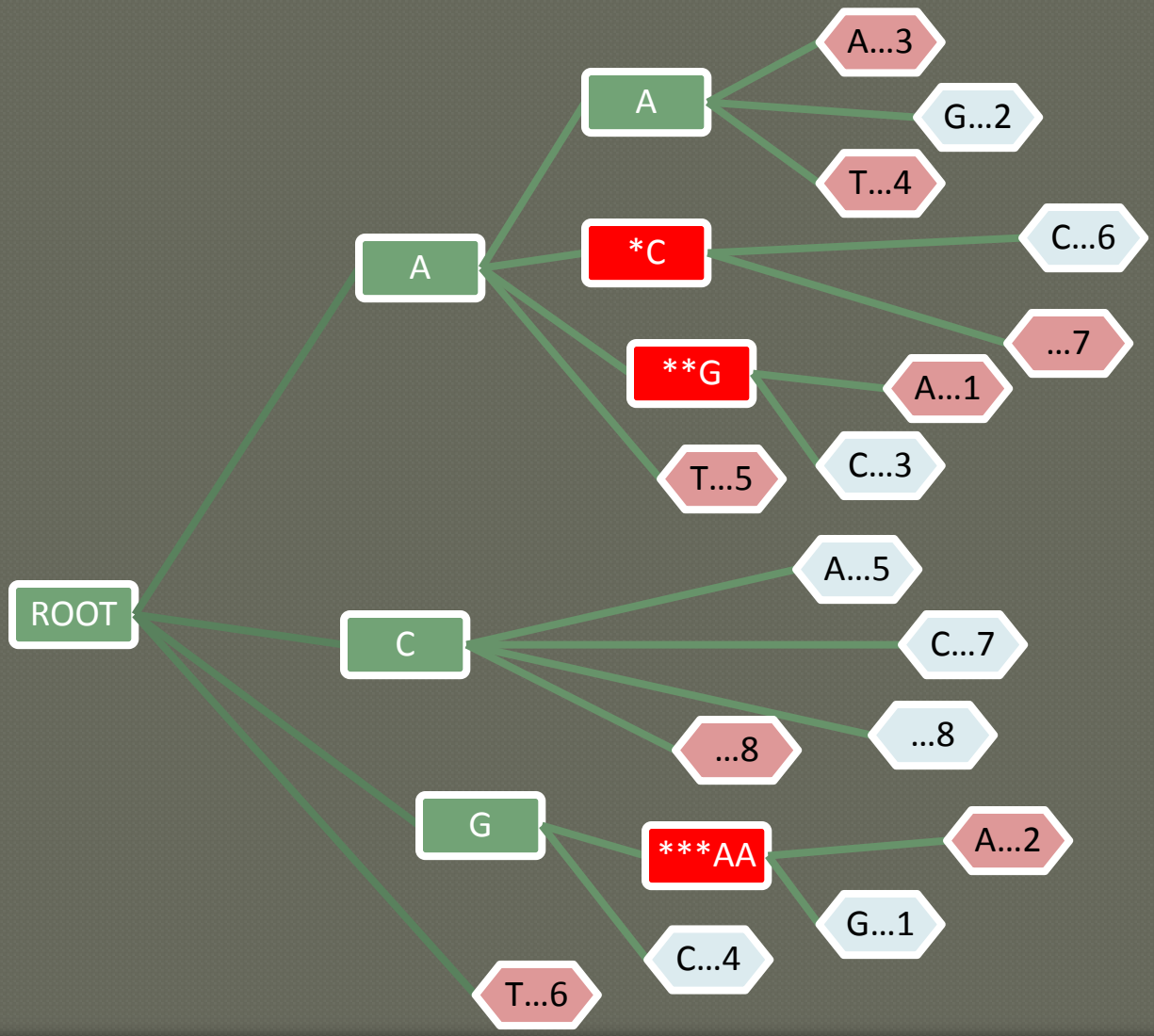


MUMs example

	1	2	3	4	5	6	7	8
S1	g	a	a	g	c	a	c	c
S2	a	g	a	a	a	t	a	c

Maximal unique matches:

- *C
- **G
- ***AA



Output of step 1:
(6,7)
(3,1)
(1,2)

The MUMmer Algorithm. Step 2

- Find a longest sequence of MUMs, such that positions in both S1 and S2 are increasing and the MUMs are not overlapping
  - If we sort MUMs by positions of S1, in order to find a longest sequence of ordered positions in MUMmers of both input strings, we can solve the *Longest Increasing Subsequence* problem for positions of S2
  - The LIS problem can be solved in time K log K, where K is a number of MUMs
  - Even by a routine Dynamic programming, for finding LIS for a sequence of K positions we need $O(K^2)$ operations, and K is much smaller than N – the length of the compared genomes

Example of DP computation for the Longest Increasing Subsequence

- Suppose the sequence of MUMs is represented by the following pairs:

  (1,7) (3,3) (4,8) (5,2) (7,6) (8,9)

  Then finding LIS of positions in S2 is the same as finding a longest common subsequence between (2,3,6,7,8,9) and (7,3,8,2,6,9)

		7	3	8	2	6	9
	0	0	0	0	0	0	0
2	0	0	0	0	1	1	1
3	0	0	1	1	1	1	1
6	0	0	1	1	1	2	2
7	0	1	1	1	1	2	2
8	0	1	1	2	2	2	2
9	0	1	1	2	2	2	3

The longest increasing subsequence (not unique) is (3,8,9)

And the resulting longest set of MUMs is
(3,3) (4,8) (8,9)

The MUMmer Algorithm. Step 3

- Align the MUMs as exact matches, and fill in the remaining positions by the locally applied dynamic programming

  For the above example with S1=gaagcacc and S2=agaaatac the resulting MUMs are (1,2) (6,7)

	1	2	3	4	5	6	7	8
S1	g	a	a	g	c	a	c	c
S2	a	g	a	a	a	t	a	c

The final alignment

S1	-	g	a	a	g	c	a	c	c
S2	a	g	a	a	a	t	a	c	-

Local Alignment

- More often, we want to find the local regions of high similarity, rather than the overall sequence scores
- The time is quadratic, and the result is highly influenced by the scoring scheme

Simple Exclusion method. Baeza-Yates, Perelberg (BYP)

- Suppose P matches a substring T1 of T with at most k errors (insertions, deletions, substitutions). Then T1 must contain at least 1 interval of length r=M/(k+1) that exactly matches one of the r-length substrings of P.
  - Proof. If we partition P into consecutive r-length regions, and align P to T1, then there would be k+1 sub-alignments. If each of these sub-alignments were to contain at least 1 error, then there would be more than k errors in total.

Alignment using BYP

- Find a local alignment of P to T with an optimal score, but with an additional constraint that there would not be more than k errors between P and the aligned region T1 of T.

1. Partition P into k+1 consecutive substrings
2. Find the set of the possible locations of alignment P to the part of T, by exactly matching each of the k+1 substrings to T
3. Extend each found match from both ends to the full length of P using dynamic programming (computing 2k+1 – strip around the main diagonal). If the resulting alignment has up to k errors, report it

BYP expected running time

- Proven that:
  - Algorithm BYP runs in O(N) time for $k<=O(M/\log_{\sigma}M)$, where σ is the size of the alphabet
  - For a DNA sequence (σ =4) of length 64, k can be as high as 64/4=16 or 25%
  - For a protein sequence (σ=20) of length 400, k can be as high as 400/2=200 or 50%
- In practice, r=M/(k+1) should be at least 9 for DNA and at least 5 for proteins to be efficient. This is because the asymptotic $O(M/\log_{\sigma}M)$ contains an unknown constant
  - For DNA of length 100 – no more than 9 errors, or 9%

Myers exclusion method (outline)

- If pattern P matches some substring T1 of T with α (for example 30%) of identity, and if we partition P=P'P'' and T1=T'T'', then either P' is 30% identical with T' or P'' is 30% identical with T''
  - Accept without the proof (the proof is complex)

Myers exclusion method. Steps

1. Partition P into r consecutive intervals

2. For each interval (the length is small) produce *α-neighborhoods* of it, meaning produce the set of all different substrings which match each interval *I* with α % identity.

   For example if *I*=aba and α=60, then the α-neighborhood of *I* should include 2 matches and at most 1 error: [bba, aaa, abb, aaba, abaa, baba, abba, abab, ba, aa, ab]

Myers exclusion method. Steps

3. Produce *a condensed α-neighborhood* by removing all substrings which are the prefix of some other substring in the neighborhood set:
   - For the previous example:
     - from [bba, aaa, abb, aaba, abaa, baba, abba, abab, ba, aa, ab]
     - to [bba, aaa, aaba, abaa, baba, abba, abab]

4. Find all locations of substrings in T which *exactly* match the substrings from a condensed neighborhood

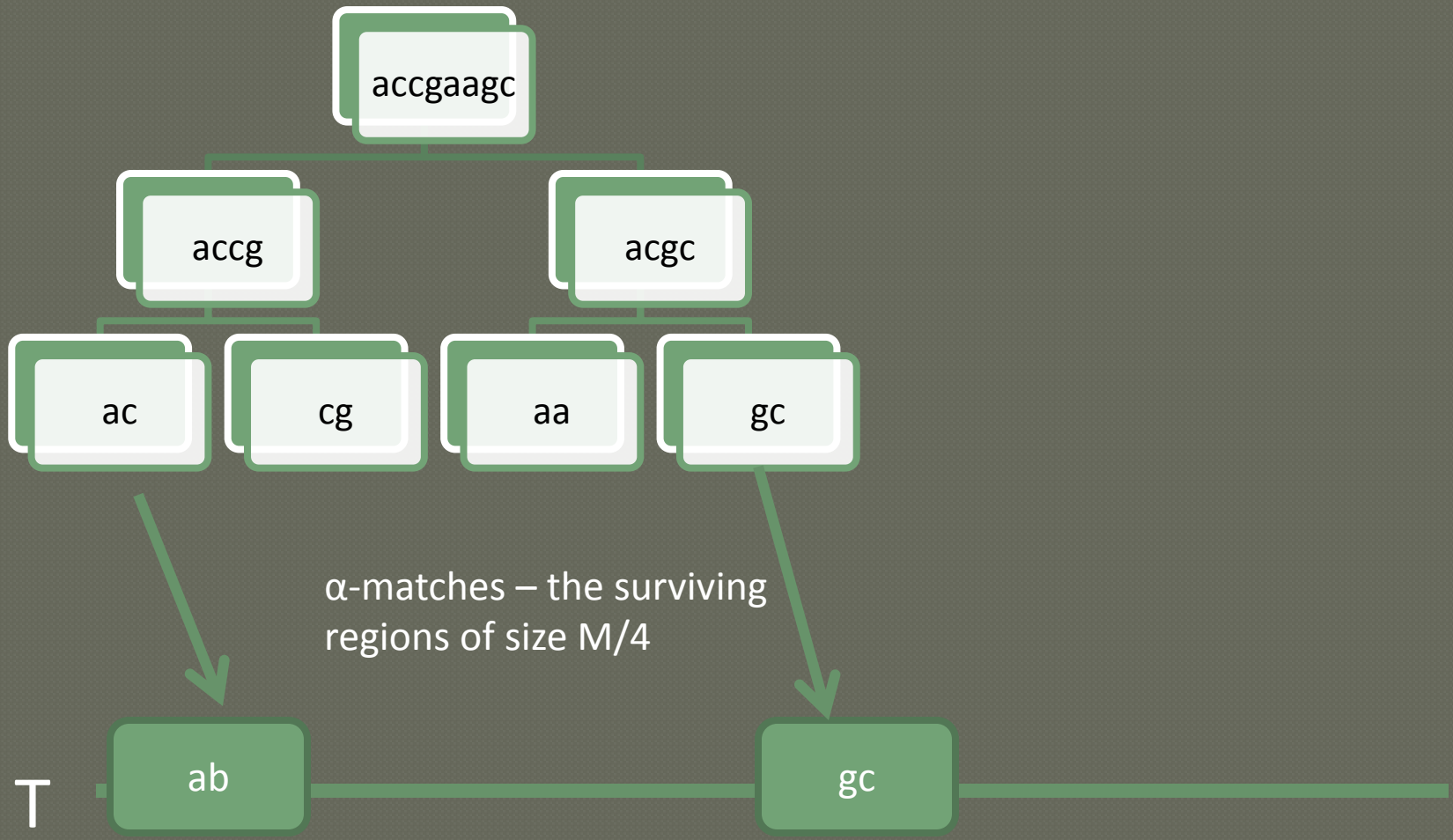
   Myers shows that this step can be implemented to run in time sub-linear in N for α>=30% (DNA) or for α>=44% (Protein)

Myers exclusion method. Steps

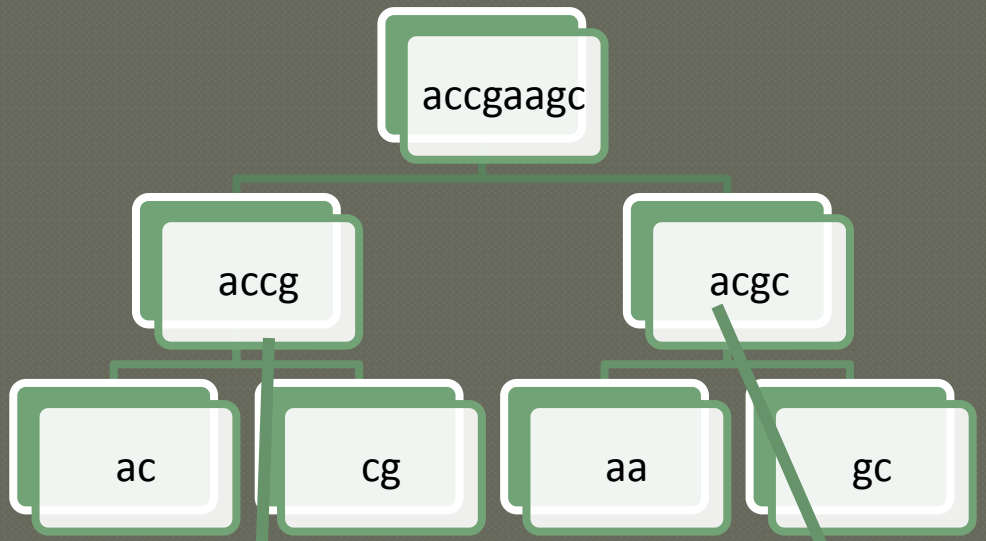
5. The matches obtained in step 4 form a set of surviving matches. Each time, we double in length the region of the surviving match trying to extend it to the right or to the left and checking if there is still 30% identity with the corresponding interval of P

6. All of the checking runs in a sublinear time for the above values of α (α>=30% (DNA) or for α>=44% (Protein)).

Myers Algorithm

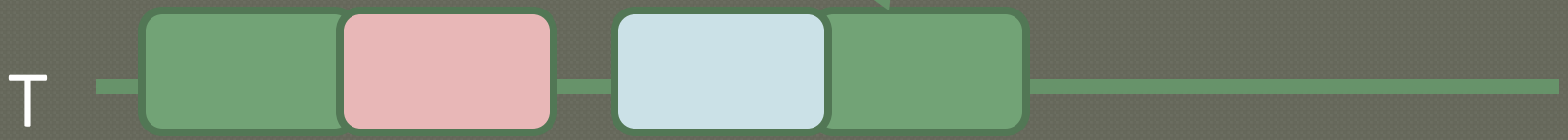


Myers Algorithm



Extend to the right

Extend to the left



The number of surviving regions drops faster than the exponential increase in the size of these regions

BLAST. Definitions.

Altschul, Gish, Miller, Myers and Lipman, 1990

- Given two strings S1 and S2*, a segment pair* is a pair of equal length substrings of S1 and S2, aligned without gaps.
- *A locally maximal segment* is a segment whose alignment score (without gaps) cannot be improved by extending it or shortening it.
- A *maximum segment pair (MSP)* in S1 and S2 is a segment pair with the maximum score over all segment pairs in S1, S2.

BLAST. Outline

- When comparing all the sequences in the database against the query, *BLAST attempts* to find all the database sequences that when paired with the query contain an *MSP above* some cutoff score *S. We call such a pair, a hi-scoring pair (HSP).*
- *We choose S such that it* is unlikely to find a random sequence in the database that achieves a score higher than *S* when compared with the query sequence.

BLAST. Step 1

- Given a length parameter *w and a threshold parameter t, BLAST finds all the w-length* substrings (called words) of database sequences that align with words from the query with an alignment score higher than *t. Each such hot spot is called a hit in BLAST.*
- Instead of requiring words to match exactly, *BLAST declares that a word hit has been* made if the word taken from the database has a score of at least *t when a substitution* matrix is used to compare the word from the query. This strategy allows the word size (*w) to be kept high (for speed), without sacrificing sensitivity.*
- *It is usually* recommended to set the parameter *w to values of 3 to 5 for amino acids, and ~ 12 for* nucleotides. Thus, *t becomes the critical parameter determining speed and sensitivity,* and *w is rarely varied.*
- *If the value of t is increased; the number of background word* hits will go down and the program will run faster. Reducing *t allows more distant* relationships to be found.

BLAST. Step 2

- In the next step, each *hit is extended to a locally maximal segment and if its score is above S, i.e. if* this sequences pair is *HSP, we report these segment*
- *Since pair score matrices typically include negative values,* extension of the initial *w-mer hit may increase or decrease the score.*
- *Accordingly,* the extension of a *hit can be terminated when the reduction in score (relative to the* maximum value encountered) exceeds certain score drop-off threshold.

Improved BLAST

1. When considering the dynamic programming matrix to align two strings, we search along each diagonal for two *w-length words such that the distance between them is ≤ A and their score is ≥ T. T can be lower than in the previous algorithm.*

Future expansion is done only to such pairs of hits.

2. In the second stage we want to allow local alignments with indels. We allow two local alignments from different diagonals to merge into a new local alignment composed of the first local alignment followed by some indels and then the second local alignment. This local alignment is essentially a path in the dynamic programming matrix, composed of two diagonal sections and a path connecting them which may contain gaps. We allow local alignments from different diagonals to merge as long as the resulting alignment has a score above some threshold.

The improved version of BLAST is about 3 times faster than the original algorithm due to much less expansions made (only two-hit words are expanded).

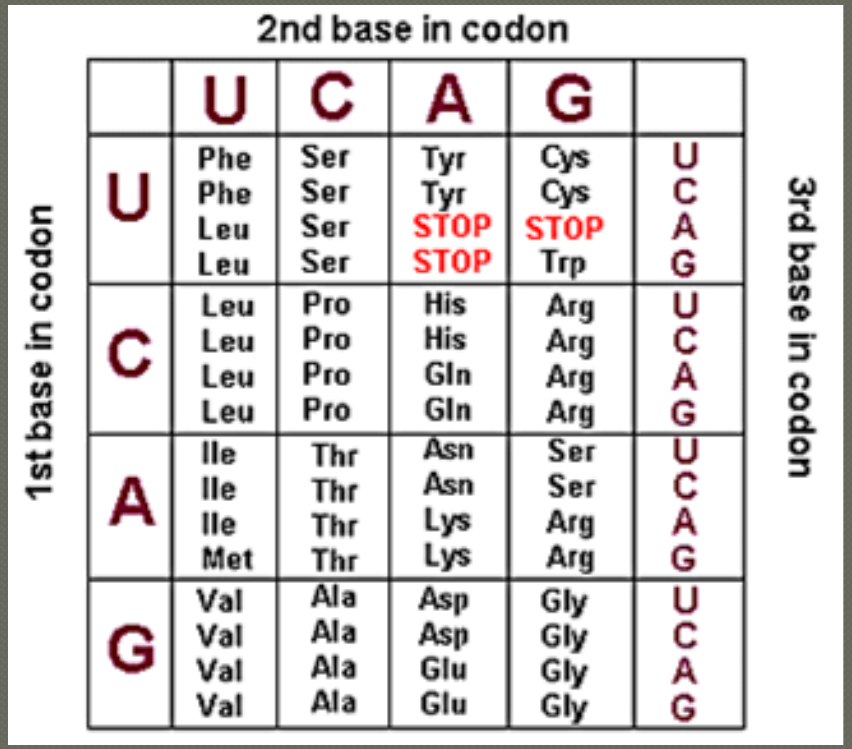
Alignments of proteins

- The mutations happen at the level of DNA
- The selection works at the level of proteins
  - Aminoacid leucine can be coded by 6 different codons: UUA, UUG, CUU, CUC, CUG, CUA

  So if more than 50 percent of nucleotides mutate (UUA->CUG) then this codon still encodes leucine

Aminoacid substitution matrix

- The unit matrix
- The genetic code matrix:
  - the entry equals the number of minimal base substitutions needed to convert a codon of amino acid *i to a codon of amino acid j. We disregard here* the importance of chemical properties of the amino acids, that evidently influence the chances for their successful substitution, like their hydrophobicity, charge or size.



For example,
distance(Phe, Leu)=1
distance(Phe, Gly)=3
distance(Phe,Phe)=0

Log-odds substitution matrices

- $q_{ij}=p_i*p_j*e^{\lambda score(i,j)}$

q_{ij} - the probability of aminoacid *i* to be replaced with aminoacid *j*

p_i*p_j - the probability of aminoacid *i* to be replaced by aminoacid *j* by chance

e^{const} - reflects how the random probability changes with the time (each time the DNA duplicates, the population of sequences growths exponentially)

How often the aminoacid *i* is replaced by aminoacid *j* during this exponential growth, depends on the intrinsic properties of the pair of aminoacids, which are reflected in the score(i,j) value

Log-odds substitution matrices

- $q_{ij}=p_i*p_j*e^{\lambda score(i,j)}$

From here:

$score(i,j)=1/\lambda * \ln [q_{ij}/ (p_i*p_j)]$

where $1 / \lambda$ is some scaling factor

If the probability of aminoacid *i* to be replaced by aminoacid *j* is the same as the random probability, the score will be 0

If it is less than the randomly expected, the score will be negative

If it is much more than expected by chance, the score will be a large positive value

Scoring matrix for proteins I. PAM- Point Accepted Mutations

	A	R	N	D	C
A	9867	2	9	10	3
R	1	9913	1	0	1
N	4	1	9822	36	0
D	6	0	42	9859	0
C	1	1	0	0	9973

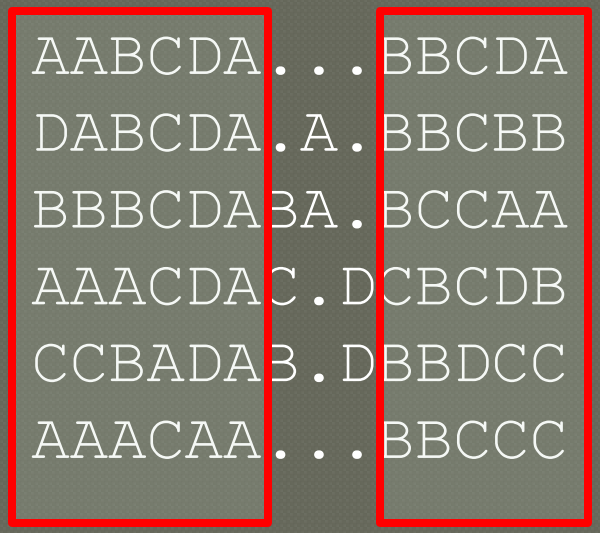
Based on the number of the substitutions from the pairwise alignment of the closely related proteins, which are not more than 1% different

This is called PAM-1 substitution matrix

The values for PAM-5 are obtained by multiplying the values in PAM-1 5 times by the same matrix

By this we extrapolate the frequency of substitutions in a closely related proteins to the distanly related, which does not really work in practice

Scoring matrix for proteins II. BLOSUM (BLOck Substitution Matrix)



The frequency of the substitutions in the conserved blocks of distantly related proteins, put into a multiple alignment

BLOSUM-65 is the matrix built on the set of proteins which are no more than 65% similar

BLOSUM-50 is for more *similar* proteins than BLOSUM-65
PAM-250 is for more *distant* proteins than PAM-50

The scores are log scores for convenience

- Frequency:
  - f(i,j)=count(i,j)/[count(i)*count(j)]
- The entry of the matrix:
  - Score(i,j)=log f(i,j)