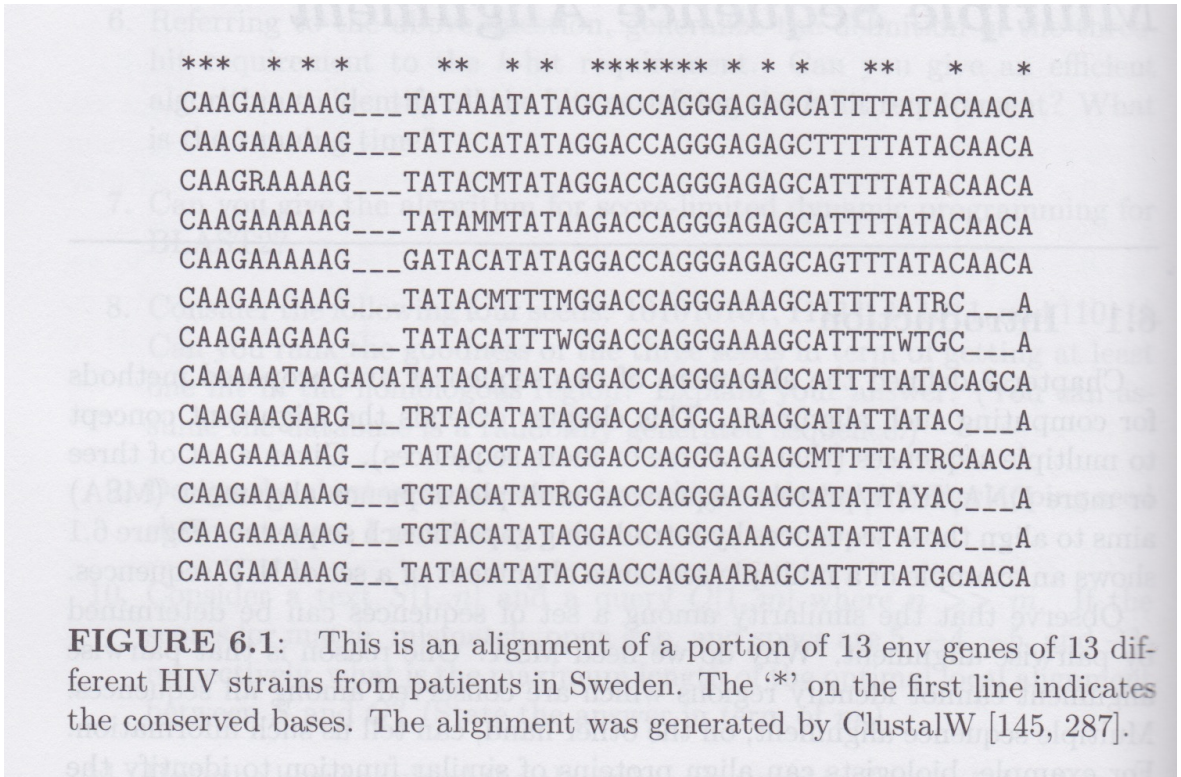
Algorithms in Bioinformatics: Lecture 12-13: Multiple Sequence Alignment

Lucia Moura

Fall 2010

Multiple Sequence Alignment (MSA)

- Given a set of 3 or more DNA/protein sequences, align the sequences by introducing gaps.
- This allows us to discover regions that are conserved among all sequences.



```
***  * * *    **  * *  ******* * * *   **   *   *
CAAGAAAAAG___TATAAATATAGGACCAGGGAGAGCATTTTATACAACA
CAAGAAAAAG___TATACATATAGGACCAGGGAGAGCTTTTTATACAACA
CAAGRAAAAG___TATACMTATAGGACCAGGGAGAGCATTTTATACAACA
CAAGAAAAAG___TATAMMTATAAGACCAGGGAGAGCATTTTATACAACA
CAAGAAAAAG___GATACATATAGGACCAGGGAGAGCAGTTTATACAACA
CAAGAAGAAG___TATACMTTTMGGACCAGGGAAAGCATTTTATRC___A
CAAGAAGAAG___TATACATTTWGGACCAGGGAAAGCATTTTWTGC___A
CAAAAATAAGACATATACATATAGGACCAGGGAGACCATTTTATACAGCA
CAAGAAGARG___TRTACATATAGGACCAGGGARAGCATATTATAC___A
CAAGAAAAAG___TATACCTATAGGACCAGGGAGAGCMTTTTATRCAACA
CAAGAAAAAG___TGTACATATRGGACCAGGGAGAGCATATTAYAC___A
CAAGAAAAAG___TGTACATATAGGACCAGGGAAAGCATATTATAC___A
CAAGAAAAAG___TATACATATAGGACCAGGAARAGCATTTTATGCAACA
```

FIGURE 6.1: This is an alignment of a portion of 13 env genes of 13 different HIV strains from patients in Sweden. The '*' on the first line indicates the conserved bases. The alignment was generated by ClustalW [145, 287].

Multiple Genome Alignment Problem

Let $\mathcal{S} = \{S_1, \ldots, S_k\}$ be a set of k DNA/protein sequences.
A multiple alignment $\mathcal{M}$ of $\mathcal{S}$ is a set of k equal-length sequences $\mathcal{M} = \{S'_1, \ldots, S'_k\}$, where each S'_i is a sequence obtained by inserting spaces into S_i.

Multiple Sequence Alignment Problem: Find a multiple sequence alignment of $\mathcal{S}$ that maximizes a similarity function or minimize a distance function. Popular similarity/distance function:

- sum-of-pair (SP) score:

$$\sum_{1 \leq i < j \leq k} align\_score_{\mathcal{M}}(S'_i, S'_j)$$

- sum-of-pair (SP) distance:

$$\sum_{1 \leq i < j \leq k} d_{\mathcal{M}}(S'_i, S'_j)$$

where $align\_score_{\mathcal{M}}(S'_i, S'_j)$/ $d_{\mathcal{M}}(S'_i, S'_j)$ is a score/distance function for 2 seq's.

Methods for solving the MSA problem

- Global optimization (dynamic programming, exponential time)
- Approximation algorithms (approximation with performance guarantee, polytime)
- Heuristic methods (no performance guarantee but effective in practice, polytime)
- Probabilistic methods (different framework as other 3; assume a model for mutation indel/probability & evolution process, try to learn the model parameters, then find alignment that fits the model).

Dynamic programming

Generalizing the dynamic programming from two to more sequences.
For aligning $S_1[1..n_1]$ and $S_2[1..n_2]$, we use:

$$V(i_1, i_2) = \max\{V(i_1 - 1, i_2 - 1) + \delta(S_1[i_1], S_2[i_2]),$$
$$V(i_1 - 1, i_2) + \delta(S_1[i_1], -), V(i_1, i_2 - 1)\delta(-, S_2[i_2])\}$$

Let us rewrite this. Let b_1, b_2 be indicator variables regarding properties of the optimal alignment for S_1, S_2, respectively. So
$(b_1, b_2) = (1, 1)$ if the last column of the alignment is match/mismatch
$(b_1, b_2) = (1, 0)$ if the last column of the alignment is a delete
$(b_1, b_2) = (0, 1)$ if the last column of the alignment is an insert.
Define $S_1[0] = S_2[0] =$ " $-$ ".

$$V(i_1, i_2) = \max_{(b_1,b_2)\in\{0,1\}^2\setminus\{(0,0)\}} \{V(i_1 - b_1, i_2 - b_2) + \delta(S_1[i_1 b_1], S_2[i_2 b_2])\}$$

Generalizing Dynamic Programming for Multiple Sequences

Consider a set of sequences $S = \{S_1[1..n_1], S_2[1..n_2], \ldots, S_k[1..n_k]\}$; add a convention that $S_j[0] =$ " $-$ " for all j.
Then the optimal alignment over all sequences is given by
$V(0, \ldots, 0) = 0$

$$V(i_1, i_2, \ldots, i_n) = \max_{(b_1,\ldots,b_k)\in\{0,1\}^k\setminus\{0^k\}} V(i_1-b_1, \ldots, i_k-b_k)+\text{SPscore}(i_1 b_1, \ldots, i_k b_k)$$

where $\text{SPscore}(i_1, i_2, \ldots, i_k) = \sum_{1\le p<q\le k} \delta(S[i_p], S[i_q])$.

To find the score of the optimal alignment we calculate $V(n_1, n_2, \ldots, n_k)$, and to find the alignment we backtrace.

Time complexity:
$n_1 n_2 \cdots n_k$ positions to fill, each entry takes $O(k^2 2^k)$ so the running time is $O(k^2 2^k n_1 n_2 \cdots n_k)$, which is exponential.

Center Star Method

In approximation algorithms, the running time is polynomial but we do not find an optimal solution. Instead, we get (for a minimization problem):

$$\text{(objective value of the solution found)} \leq \alpha \text{ (optimal value of solution)}$$

$\alpha \geq 1$ is the approximation ratio; the closest to 1 the better.

The Center Star Method is an approximation algorithm with ratio 2.

Let $D(X, Y)$ be the pairwise optimal alignment distance between X, Y.

$S_c \in \mathcal{S}$ is said to be the center string of $\mathcal{S}$ if it minimizes: $\sum_{i=1}^{k} D(S_c, S_i)$

The center star method works as follows:

1. Identify the center string S_c of $\mathcal{S}$.
2. Uses the alignments of S_c with each S_i to create a multiple alignment.

Center Star Algorithm

Center_Star_Method

Require: A set $\mathcal{S}$ of sequences

Ensure: A multiple alignment of M with sum of pair distances at most twice that of the optimal alignment of $\mathcal{S}$

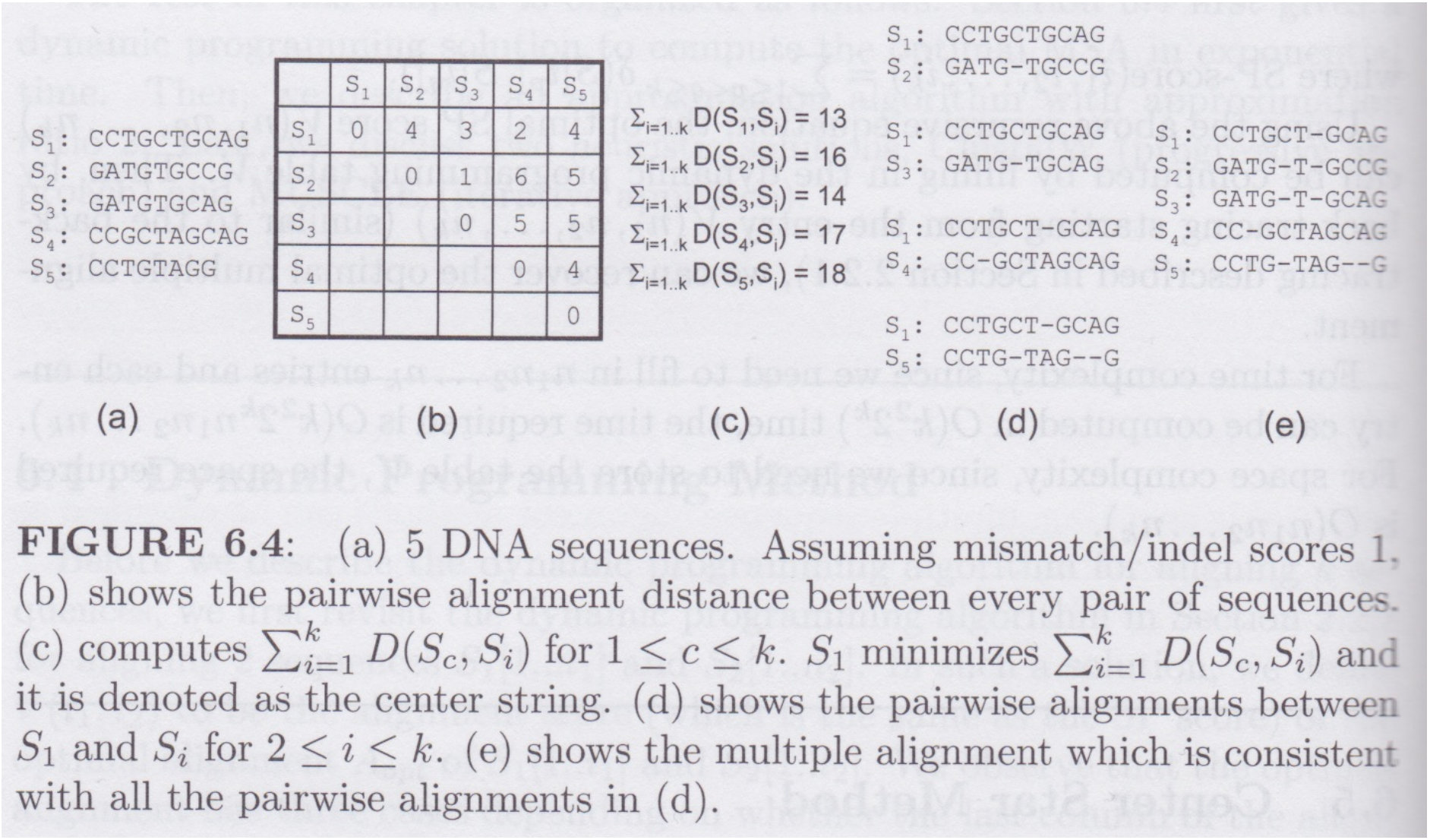
1: Find $D(S_i, S_j)$ for all i, j.

2: Find the center sequence S_c which minimizes $\sum_{i=1}^{k} D(S_c, S_i)$.

3: For every $S_i \in \mathcal{S} - \{S_c\}$, choose an optimal alignment between S_c and S_i.

4: Introduce spaces into S_c so that the multiple alignment $\mathcal{M}$ satisfies the alignments found in Step 3.

Center Star Method Example



	S_1	S_2	S_3	S_4	S_5
S_1	0	4	3	2	4
S_2		0	1	6	5
S_3			0	5	5
S_4				0	4
S_5					0

(b)

$\Sigma_{i=1..k} D(S_1,S_i) = 13$
$\Sigma_{i=1..k} D(S_2,S_i) = 16$
$\Sigma_{i=1..k} D(S_3,S_i) = 14$
$\Sigma_{i=1..k} D(S_4,S_i) = 17$
$\Sigma_{i=1..k} D(S_5,S_i) = 18$

(c)

S_1: CCTGCTGCAG
S_2: GATG-TGCCG

S_1: CCTGCTGCAG
S_3: GATG-TGCAG

S_1: CCTGCT-GCAG
S_4: CC-GCTAGCAG

S_1: CCTGCT-GCAG
S_5: CCTG-TAG--G

(d)

S_1: CCTGCT-GCAG
S_2: GATG-T-GCCG
S_3: GATG-T-GCAG
S_4: CC-GCTAGCAG
S_5: CCTG-TAG--G

(e)

FIGURE 6.4: (a) 5 DNA sequences. Assuming mismatch/indel scores 1, (b) shows the pairwise alignment distance between every pair of sequences. (c) computes $\sum_{i=1}^{k} D(S_c, S_i)$ for $1 \leq c \leq k$. S_1 minimizes $\sum_{i=1}^{k} D(S_c, S_i)$ and it is denoted as the center string. (d) shows the pairwise alignments between S_1 and S_i for $2 \leq i \leq k$. (e) shows the multiple alignment which is consistent with all the pairwise alignments in (d).

Center Star Method: details on step 4

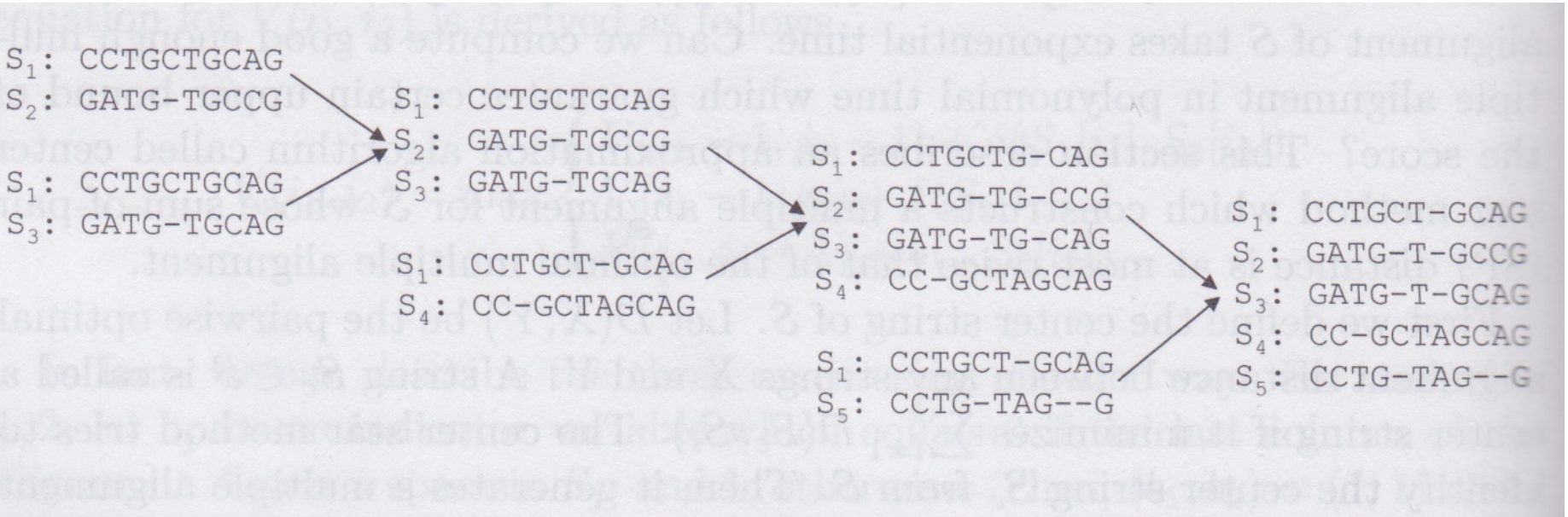


FIGURE 6.5: This figure demonstrates how to convert the pairwise alignments in Figure 6.4(d) to the multiple alignment in Figure 6.4(e).

Approximation ratio

Theorem

Let $\mathcal{M}$ be the multiple alignment of $\mathcal{S} = \{S_1, \ldots, S_k\}$ computed by the center method. The sum of pair distance of $\mathcal{M}$ is at most twice that of the optimal alignment.

(proof will be derived during class)

Running time analysis:

Suppose all sequences in $\mathcal{S}$ are of length $O(n)$.

- step 1 - computing the optimal distance $D(S_i, S_j)$ for all i, j: $O(k^2n^2)$.
- step 2 - computing the center string S_c: $O(k^2)$.
- step 3 - generating the k pairwise alignments with S_c: $O(kn^2)$
- step 4 - insert spaces into S_c in order to satisfy all multiple alignments of step 3 simultaneously: $O(k^2n)$

Total running time: $O(k^2n^2)$.

Heuristics: Progressive Alignment Method

This is a heuristic method for multiple sequence alignment.
Start by aligning the two closest sequences, and then add the next most closely related sequences, until all sequences are aligned.

3 steps:

1. Compute pairwise distance scores for all pairs of sequences.
2. Generate a guide tree, which ensures that similar sequences are near in the tree.
3. Align sequences one by one according to the guide tree.

We will look at one method of progressive alignment: ClustalW.

ClustalW method

Given $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$, each of length $O(n)$.
Steps:

1. computes a global optimal alignment for every S_i, S_j;
   then set distance score between S_i and S_j to: $1 - \frac{y}{x}$, where x is the number of non-gap positions and y in the number of identical positions in their alignment.
   $O(k^2n^2)$.
2. Build guide tree from the distance matrix, using the neighbour joining algorithm (to be seen in next chapter) $O(k^3)$
3. Perform several alignments according to the guide tree.
   Do at most k **profile-profile alignments**, each taking $O(kn + n^2)$, so total $O(k^2n + kn^2)$.

total running time: $O(k^2n^2 + k^3)$.

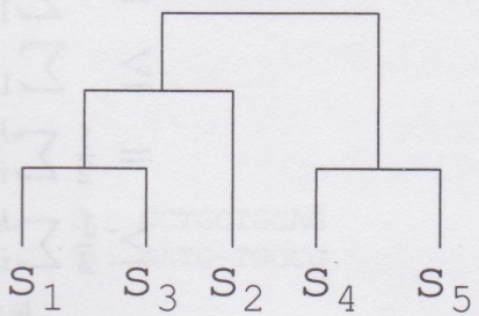
ClustalW method

S_1: PPGVKSDCAS
S_2: PADGVKDCAS
S_3: PPDGKSDS
S_4: GADGKDCCS
S_5: GADGKDCAS

(a)

	S_1	S_2	S_3	S_4	S_5
S_1	0	0.111	0.25	0.555	0.444
S_2		0	0.375	0.222	0.111
S_3			0	0.5	0.5
S_4				0	0.111
S_5					0

(b)



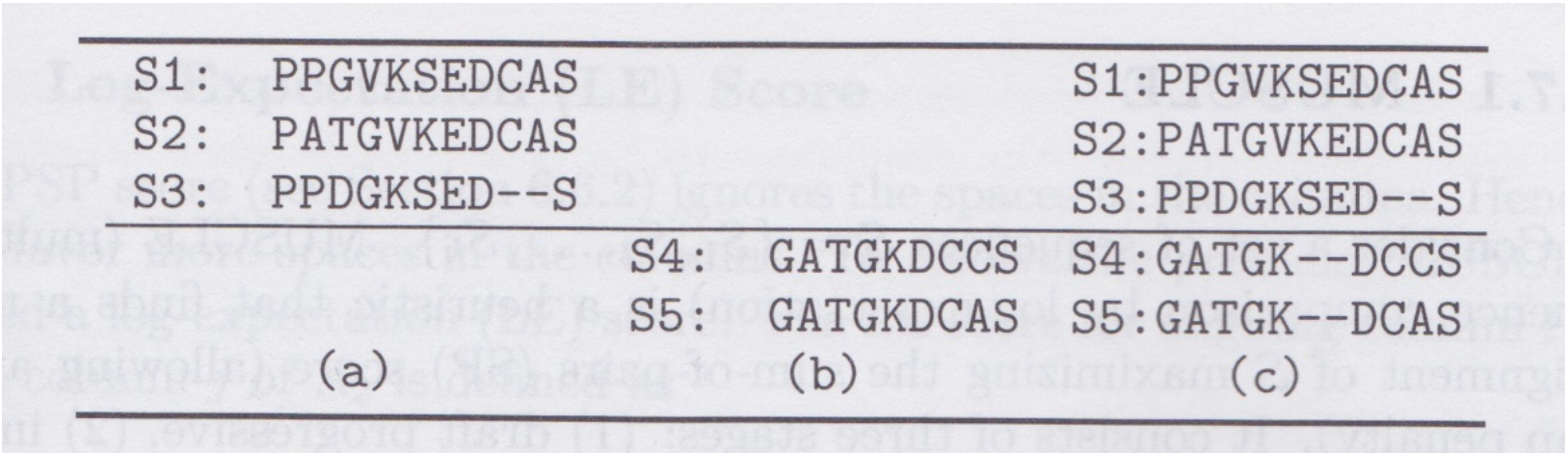
(c)

S_1: PPGVKSDCAS
S_2: PATGVKDCAS
S_3: PPTGKSD--S
S_4: GATGK-DCCS
S_5: GATGK-DCAS

(d)

Profile-profile alignments (step 3)

A profile-profile alignment is the alignment of two alignments.



Profile-profile alignments (step 3)

Input: two **sets** of sequences $\mathcal{S}_1, \mathcal{S}_2$ and their alignments A_1, A_2.
Output: an alignment of A_1 and A_2 (introduce gaps in each whole alignment)

The following is the function used for similarity between column i of alignment A_1, and column j of alignment A_2:

$$PSP(A_1[i], A_2[j]) = \sum_{x,y \in \mathcal{A}} g_x^i g_y^j \delta(x, y),$$

where $\mathcal{A}$ is the alphabet (4 symbols for DNA, 20 for protein),
g_x^i is the observed frequency of character x in position i.

To align two alignments...

- gap penalty: penalty for opening and for extending the gap;
- each aligned position is scored using PSP function above;
- use dynamic programming like in global alignment.

Dynamic programming for profile-profile alignment

Aligning two profiles A_1 and A_2.

With no gap opening penalty, the dynamic programming algorithm calculates:

$$V(i,j) = \max \quad \{ \quad V(i-1,j-1) + PSP(A_1[i], A_2[j]),$$
$$V(i,j-1) + PSP(-, A_2[j]),$$
$$V(i-1,j) + PSP(A_1[i], -) \quad \}$$

With affine gap penalty, we use a similar dynamic programming method as seen before.

Time complexity for profile-profile alignment of A_1 and A_2 each with k_1, k_2 sequences:

- computing g_x^i, g_y^j: $O(k_1 n_1 + k_2 n_2)$
- dynamic programming to fill in $O(n_1 n_2)$ entries using or not gap penalty: $O(n_1 n_2)$

Total running time: $O(k_1 n_1 + k_2 n_2 + n_1 n_2)$.

Limitations of progressive alignment

- Being a heuristic method that does not change initial decisions, the solution may be very far from optimal.
- **Iterative methods** start from an initial multiple alignment (which can be done via progressive alignment), but then apply some modifications to try to improve it. There are several iterative methods such as PRRP, MAFFT, MUSCLE.

Iterative Methods: MUSCLE

MUSCLE (MUltiple Sequence Comparison by Log Expectation), 3 steps:

1. **draft progressive:**
   - consists of a progressive sequence alignment
   - (accuracy) it uses log-expectation score instead of PPS score in profile-profile alignment;
   - (efficiency) uses k-mer distance instead of alignment score for sequence similarity (a k-mer is a substring of length k)
   - instead of neighbour joining, it uses UPGMA (see next chapter)
2. **improved progressive:**
   - use alignment to compute more accurate pairwise distance between sequences, Kimura distance: $-\ln(1 - D - \frac{D^2}{5}$, where D is the fraction of identical bases between the pair of sequences.
   - from new distance matrix, build the guide tree and a new alignment.
3. **refinement:** tries to improve alignment
   refines multiple alignment using the **tree-dependent restricted partition technique** - a process of deleting edges of guide tree, and re-combine the alignment of the disjoint trees, if better.

Log-Expectation (LE) Score

PSP ignores spaces in the columns:

$$PSP(A_1[i], A_2[j]) = \sum_{x,y \in \mathcal{A}} g_x^i g_y^j \delta(x, y),$$

where $\mathcal{A}$ is the alphabet (4 symbols for DNA, 20 for protein), g_x^i is the observed frequency of character x in position i.

Log-Expectation score tries to penalize more these spaces:

$$LE(A_1[i], A_2[j]) = (1 - f_G^i)(1 - f_G^j) \log \sum_{x,y \in \mathcal{A}} f_x^i f_y^j \delta(x, y),$$

- f_G^i is the observed frequency of gaps in column i
- f_x^i is the normalized observed frequency of character x in column i: $g_x^i / (\sum_{x \in \mathcal{A}} g_x^i)$.
- $\delta(x, y)$ for any pair of aminoacids x,y uses 240 PAM VTML matrix.

Conclusion

The multiple sequence assignments seen here are just a small sample on a vasta literature.

See textbook section 6.8 for further reading, including several surveys.