Combinatorial Methods in Bioinformatics

Multiple Sequence Alignment

Wing-Kin Sung, Ken 宋永健
ksung@comp.nus.edu.sg

Multiple Sequence Alignment (MSA)

- Given k sequences S = {S_1, S_2, …, S_k}.
- A multiple alignment of S is a set of k equal-length sequences {S'_1, S'_2, …, S'_k}.
    - where S'_i is obtained by inserting gaps in to S_i.
- The multiple sequence alignment problem aims to
    - find a multiple alignment which optimizes certain score.

Example: multiple alignment of 4 sequences

- S'_1 = `ACG--GAGA`
- S'_2 = `-CGTTGACA`
- S'_3 = `AC-T-GA-A`
- S'_4 = `CCGTTCAC-`

Applications of multiple sequence alignment

- Align the domains of proteins
- Align the same genes/proteins from multiple species
- Help predicting protein structure

Sum-of-Pair (SP) Score

- Consider the multiple alignment S’ of S.
- SP-score(a_1, …, a_k) = $\Sigma_{1 \le p < q \le k}\ \delta(a_p, a_q)$
  - where a_p can be any character or a space.
  - $\delta(-,-)=0$
- The SP-score of S’ is
  - $\sum_{i=1}^{|S'|} \mathrm{SP{-}score}(S'_1[i], S'_2[i], \ldots, S'_k[i])$

	1	2	…	i	…	|S|
S'_1				$S'_1[i]$		
S'_2				$S'_2[i]$		
…						
S'_k				$S'_k[i]$		

Example: multiple alignment of 4 sequences

- S_1 = `ACG--GAGA`
- S_2 = `-CGTTGACA`
- S_3 = `AC-T-GA-A`
- S_4 = `CCGTTCAC-`
- Assume score of
  - match and mismatch/insert/delete are 2 and -2, respectively.
- For position 1,
  - SP-score(A,-,A,C) = 2δ(A,-) + 2δ(A,C) + δ(A,A) + δ(C,-) = -8
- SP-score= -8+12+0+0–6+0+12–10+0 = 0

Sum-of-Pair (SP) distance

- Equivalently, we have SP-dist.

- Consider the multiple alignment S’ of S.
- SP-dist(a_1, …, a_k) = $\Sigma_{1 \le p < q \le k}\, \delta(a_p, a_q)$
  - where a_p can be any character or a space.
  - δ(-,-)=0
- The SP-dist of S’ is
  - $\sum_{i=1}^{|S'|} \mathrm{SP{-}dist}(S'_1[i], S'_2[i], \ldots, S'_k[i])$.

The problem of multiple sequence alignment (MSA)

- Given a set of sequences S_1, S_2, …, S_k.

- Similarity problem:
  - Find an MSA (S'_1, S'_2, …, S'_k) that maximizes SP-score

- Distance problem:
  - Find an MSA (S'_1, S'_2, …, S'_k) that minimizes SP-dist

- Question: Are these two problems the same?

Agenda

- Exact result
  - Dynamic Programming
- Approximation algorithm
  - Center star method
- Heuristics
  - ClustalW --- Progressive alignment
  - MUSCLE --- Iterative method
  - Partial Order Alignment (POA)

Dynamic Programming for aligning two sequences

- Recall that the optimal alignment for two sequences can be found as follows.
- Let V(i_1, i_2) be the score of the optimal alignment between S_1[1..i_1] and S_2[1..i_2].

$$V(i_1, i_2) = \max \begin{cases} V(i_1 - 1, i_2 - 1) + \delta(S_1[i_1], S_2[i_2]) & \text{Match/mismatch} \\ V(i_1 - 1, i_2) + \delta(S_1[i_1], \_) & \text{Delete} \\ V(i_1, i_2 - 1) + \delta(\_, S_2[i_2]) & \text{Insert} \end{cases}$$

- The equation can be rephrased as

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

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

- (Assume that S_j[0] = ‘_’.)

Dynamic Programming for aligning three sequences

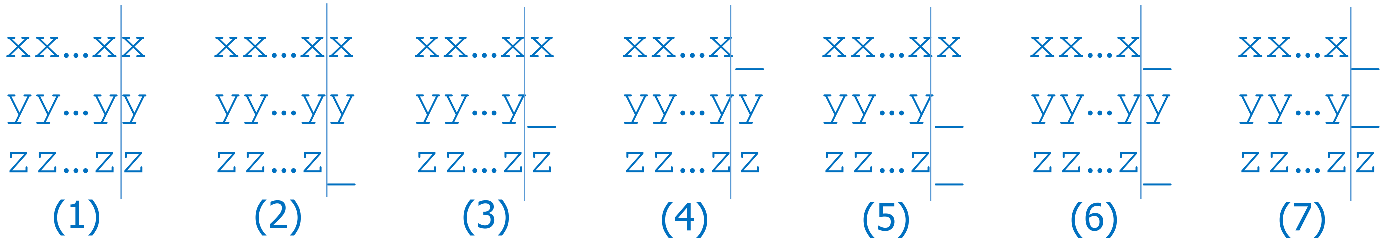
- Recall that the optimal alignment for two sequences can be found as follows.
- Let V(i_1, i_2, i_3) be the score of the optimal alignment between S_1[1..i_1], S_2[1..i_2] and S_3[1..i_3].

$$V(i_1, i_2) = \max \begin{cases} V(i_1-1, i_2-1, i_3-1) + \text{SP}-\text{score}(S_1[i_1], S_2[i_2], S_3[i_3]) & (1) \\ V(i_1-1, i_2-1, i_3-0) + \text{SP}-\text{score}(S_1[i_1], S_2[i_2], \_) & (2) \\ V(i_1-1, i_2-0, i_3-1) + \text{SP}-\text{score}(S_1[i_1], \_, S_3[i_3]) & (3) \\ V(i_1-0, i_2-1, i_3-1) + \text{SP}-\text{score}(\_, S_2[i_2], S_3[i_3]) & (4) \\ V(i_1-1, i_2-0, i_3-0) + \text{SP}-\text{score}(S_1[i_1], \_, \_) & (5) \\ V(i_1-0, i_2-1, i_3-0) + \text{SP}-\text{score}(\_, S_2[i_2], \_) & (6) \\ V(i_1-0, i_2-0, i_3-1) + \text{SP}-\text{score}(\_, \_, S_3[i_3]) & (7) \end{cases}$$

- The equation can be rephrased as

$$V(i_1, i_2, i_3) = \max_{(b_1,b_2,b_3) \in \{0,1\}^3 - \{(0,0,0)\}} \{V(i_1-b_1, i_2-b_2, i_3-b_3) + \text{SP}-\text{score}(S_1[i_1 b_1], S_2[i_2 b_2], S_2[i_3 b_3])\}$$

- (Assume that S_j[0] = ‘_’.)



```
xx…x|x   xx…x|x   xx…x|x   xx…x|_   xx…x|x   xx…x|_   xx…x|_
yy…y|y   yy…y|y   yy…y|_   yy…y|y   yy…y|_   yy…y|y   yy…y|_
zz…z|z   zz…z|_   zz…z|z   zz…z|z   zz…z|_   zz…z|_   zz…z|z
 (1)      (2)      (3)      (4)      (5)      (6)      (7)
```

Dynamic programming for aligning k sequences (I)

- Let V(i_1, i_2, …, i_k) = the SP-score of the optimal alignment of $S_1[1..i_1]$, $S_2[1..i_2]$, …, $S_k[1..i_k]$.

$$V(i_1, \ldots, i_k) = \max_{(b_1,\ldots,b_k)\in\{0,1\}^k-\{(0,\ldots,0)\}} \left\{ \begin{matrix} V(i_1 - b_1, \ldots, i_k - b_k) + \\ \mathrm{SP{-}score}(S_1[b_1 i_1], \ldots, S_k[b_k i_k]) \end{matrix} \right\}$$

- The SP-score of the optimal multiple alignment of S={S_1, S_2, …, S_k} is V(n_1, n_2, …, n_k)
  - where n_i is the length of S_i.

Dynamic Programming for aligning k sequences (II)

- By filling-in the dynamic programming table,
  - We compute V(n_1, n_2, …, n_k).
- By back-tracing,
  - We recover the multiple alignment.

Complexity

- Time:
  - The table V has $n_1 n_2 \ldots n_k$ entries.
  - Filling in one entry takes $2^k k^2$ time.
  - Total running time is $O(2^k k^2 \, n_1 n_2 \ldots n_k)$.

- Space:
  - $O(n_1 n_2 \ldots n_k)$ space to store the table V.

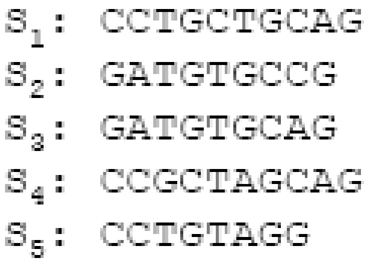
- Dynamic programming is expensive in both time and space. It is rarely used to align more than 3 or 4 sequences.

Center star method

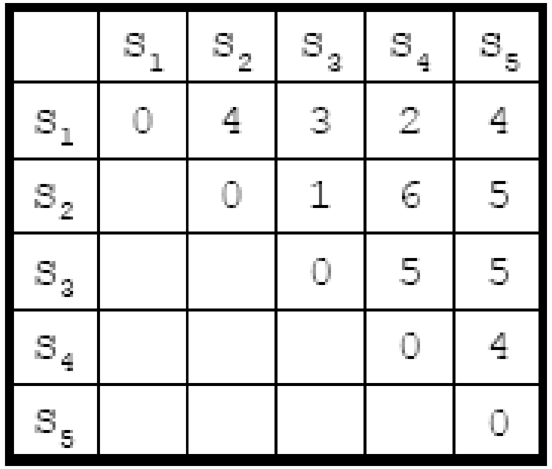
- Computing optimal multiple alignment takes exponential time.
- Can we find a good approximation using polynomial time?
- We introduce Center star method, which minimizes Sum-of-Pair distance.

Idea

- Find a string S_c.
- Align all other strings with respect to S_c.
- Illustrate by an example:

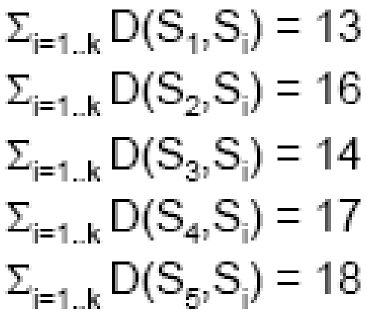


```
S1: CCTGCTGCAG
S2: GATGTGCCG
S3: GATGTGCAG
S4: CCGCTAGCAG
S5: CCTGTAGG
```



	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

S_1 has the smallest distance to all other strings.



$\Sigma_{i=1..k} D(S_1,S_i) = 13$

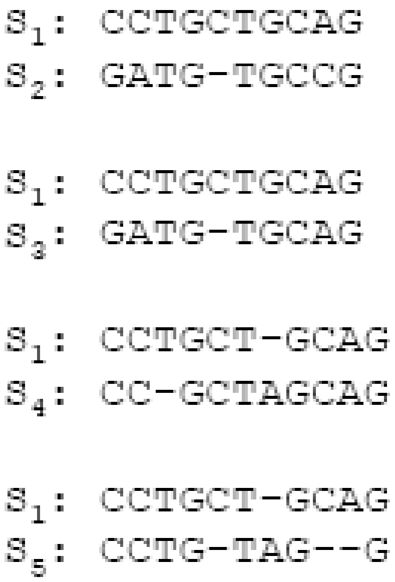
$\Sigma_{i=1..k} D(S_2,S_i) = 16$

$\Sigma_{i=1..k} D(S_3,S_i) = 14$

$\Sigma_{i=1..k} D(S_4,S_i) = 17$

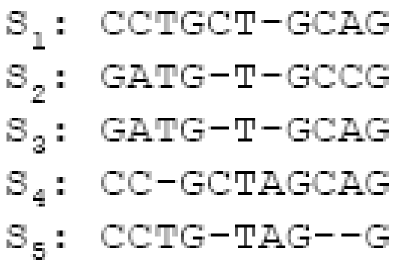
$\Sigma_{i=1..k} D(S_5,S_i) = 18$

Align all strings to S_1



```
S1: CCTGCTGCAG
S2: GATG-TGCCG

S1: CCTGCTGCAG
S3: GATG-TGCAG

S1: CCTGCT-GCAG
S4: CC-GCTAGCAG

S1: CCTGCT-GCAG
S5: CCTG-TAG--G
```

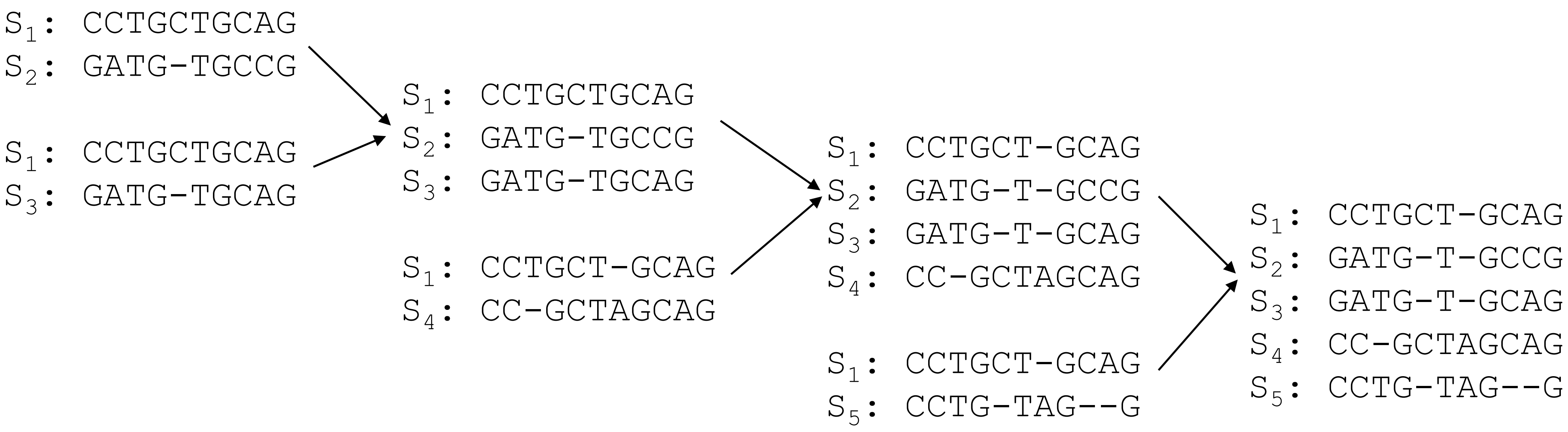
Convert pairwise alignments to multiple sequence alignment



```
S1: CCTGCT-GCAG
S2: GATG-T-GCCG
S3: GATG-T-GCAG
S4: CC-GCTAGCAG
S5: CCTG-TAG--G
```

Converting pair-wise alignment to multiple alignment

- Introduce spaces to S_c to generate the multiple alignment



S_1: CCTGCTGCAG
S_2: GATG-TGCCG

S_1: CCTGCTGCAG
S_3: GATG-TGCAG

S_1: CCTGCTGCAG
S_2: GATG-TGCCG
S_3: GATG-TGCAG

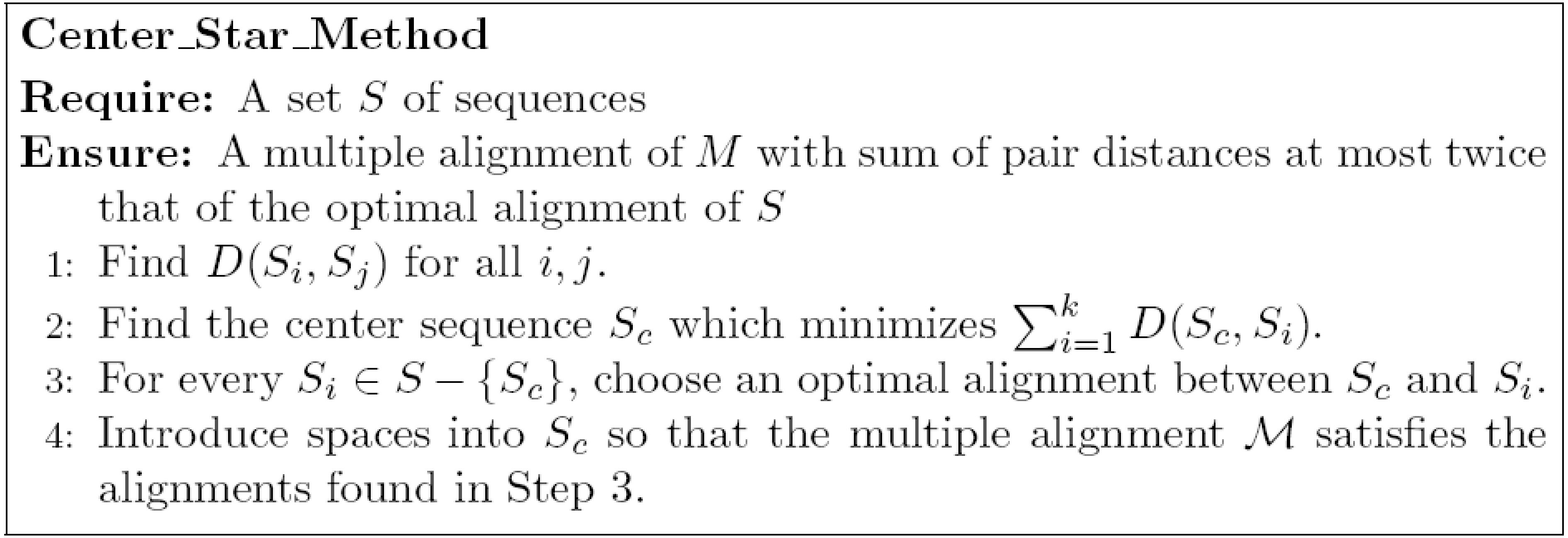
S_1: CCTGCT-GCAG
S_4: CC-GCTAGCAG

S_1: CCTGCT-GCAG
S_2: GATG-T-GCCG
S_3: GATG-T-GCAG
S_4: CC-GCTAGCAG

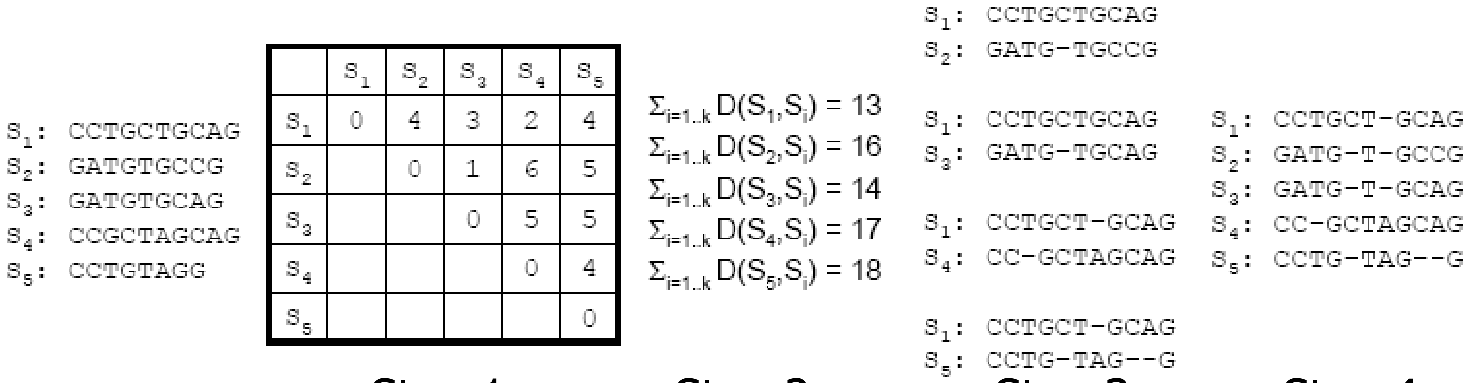
S_1: CCTGCT-GCAG
S_5: CCTG-TAG--G

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

Detail algorithm for center star method



```
Center_Star_Method
Require: A set $S$ of sequences
Ensure: A multiple alignment of $M$ with sum of pair distances at most twice
    that of the optimal alignment of $S$
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 $S_i \in S - \{S_c\}$, 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.
```



```
S1: CCTGCTGCAG
S2: GATGTGCCG
S3: GATGTGCAG
S4: CCGCTAGCAG
S5: CCTGTAGG
```

	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

$\Sigma_{i=1..k} D(S_1,S_i) = 13$

$\Sigma_{i=1..k} D(S_2,S_i) = 16$

$\Sigma_{i=1..k} D(S_3,S_i) = 14$

$\Sigma_{i=1..k} D(S_4,S_i) = 17$

$\Sigma_{i=1..k} D(S_5,S_i) = 18$

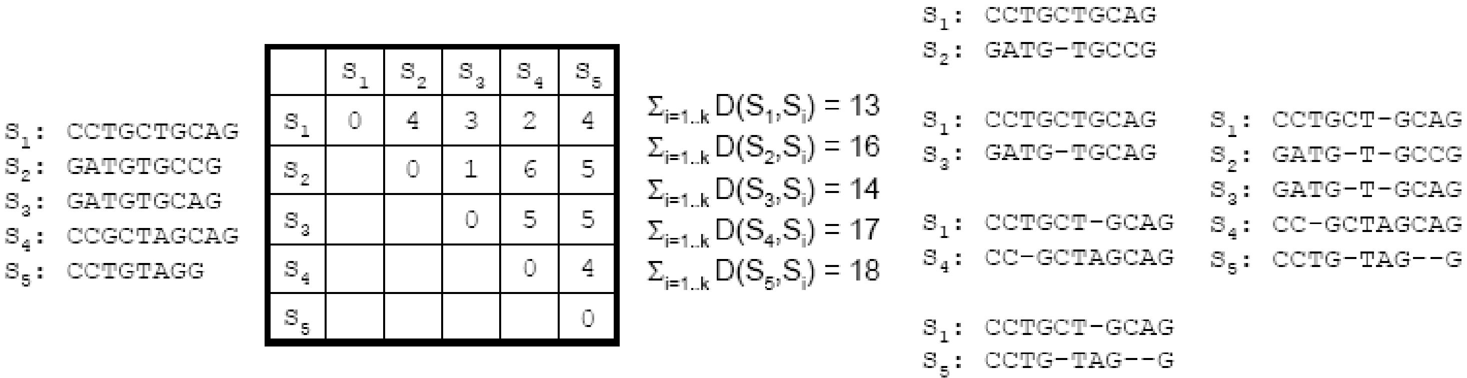
```
S1: CCTGCTGCAG
S2: GATG-TGCCG

S1: CCTGCTGCAG
S3: GATG-TGCAG

S1: CCTGCT-GCAG
S4: CC-GCTAGCAG

S1: CCTGCT-GCAG
S5: CCTG-TAG--G
```

```
S1: CCTGCT-GCAG
S2: GATG-T-GCCG
S3: GATG-T-GCAG
S4: CC-GCTAGCAG
S5: CCTG-TAG--G
```

Step 1 Step 2 Step 3 Step 4

Running time of center star method

- Assume all k sequences are of length n.
- Step 1 takes O(k^2n^2) time.
- Step 2 takes O(k^2) time to find the center string S_c.
- Step 3 takes O(kn^2) time to compute the alignment between S_c and S_i for all i.
- Step 4 introduces space into the multiple alignment, which takes O(k^2n) time.
- **In total, the running time is O(k^2n^2)**.



```
S1: CCTGCTGCAG
S2: GATGTGCCG
S3: GATGTGCAG
S4: CCGCTAGCAG
S5: CCTGTAGG
```

	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

$\Sigma_{i=1..k} D(S_1,S_i) = 13$

$\Sigma_{i=1..k} D(S_2,S_i) = 16$

$\Sigma_{i=1..k} D(S_3,S_i) = 14$

$\Sigma_{i=1..k} D(S_4,S_i) = 17$

$\Sigma_{i=1..k} D(S_5,S_i) = 18$

```
S1: CCTGCTGCAG
S2: GATG-TGCCG

S1: CCTGCTGCAG
S3: GATG-TGCAG

S1: CCTGCT-GCAG
S4: CC-GCTAGCAG

S1: CCTGCT-GCAG
S5: CCTG-TAG--G
```

```
S1: CCTGCT-GCAG
S2: GATG-T-GCCG
S3: GATG-T-GCAG
S4: CC-GCTAGCAG
S5: CCTG-TAG--G
```

Step 1 Step 2 Step 3 Step 4

Why center star method is good? (I)

- Let M* be the optimal alignment.
- The SP-dist of M*

$$
\begin{aligned}
&= \sum_{1\le i<j\le k} d_{\mathcal{M}^*}(i,j) \\
&\ge \sum_{1\le i<j\le k} D(S_i,S_j) \\
&= \tfrac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k} D(S_i,S_j) \\
&\ge \tfrac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k} D(S_c,S_j) \\
&= \tfrac{k}{2}\sum_{j=1}^{k} D(S_c,S_j)
\end{aligned}
$$

Why center star method is good? (II)

- The SP-dist of M

$$\begin{aligned}
&= \sum_{1\le i<j\le k} d_{\mathcal{M}}(i,j) \\
&= \tfrac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k} d_{\mathcal{M}}(i,j) \\
&\le \tfrac{1}{2}\sum_{i=1}^{k}\sum_{j=1}^{k}[D(S_c,S_i)+D(S_c,S_j)] \\
&= \tfrac{k}{2}\sum_{i=1}^{k} D(S_c,S_i)+\tfrac{k}{2}\sum_{j=1}^{k} D(S_c,S_j) \\
&= k\sum_{j} D(S_c,S_j)
\end{aligned}$$

- The SP-dist of M is at most twice of that of M* (the optimal alignment).

Progress alignment

- Progress alignment is first proposed by Feng and Doolittle (1987).

- It is a heuristics to get a good multiple alignment.
- Basic idea:
  - Align the two most closest sequences
  - Progressive align the most closest related sequences until all sequences are aligned.

- Examples of Progress alignment method include:
  - ClustalW, T-coffee, Probcons
- Probcons is currently the most accurate MSA algorithm.
- ClustalW is the most popular software.

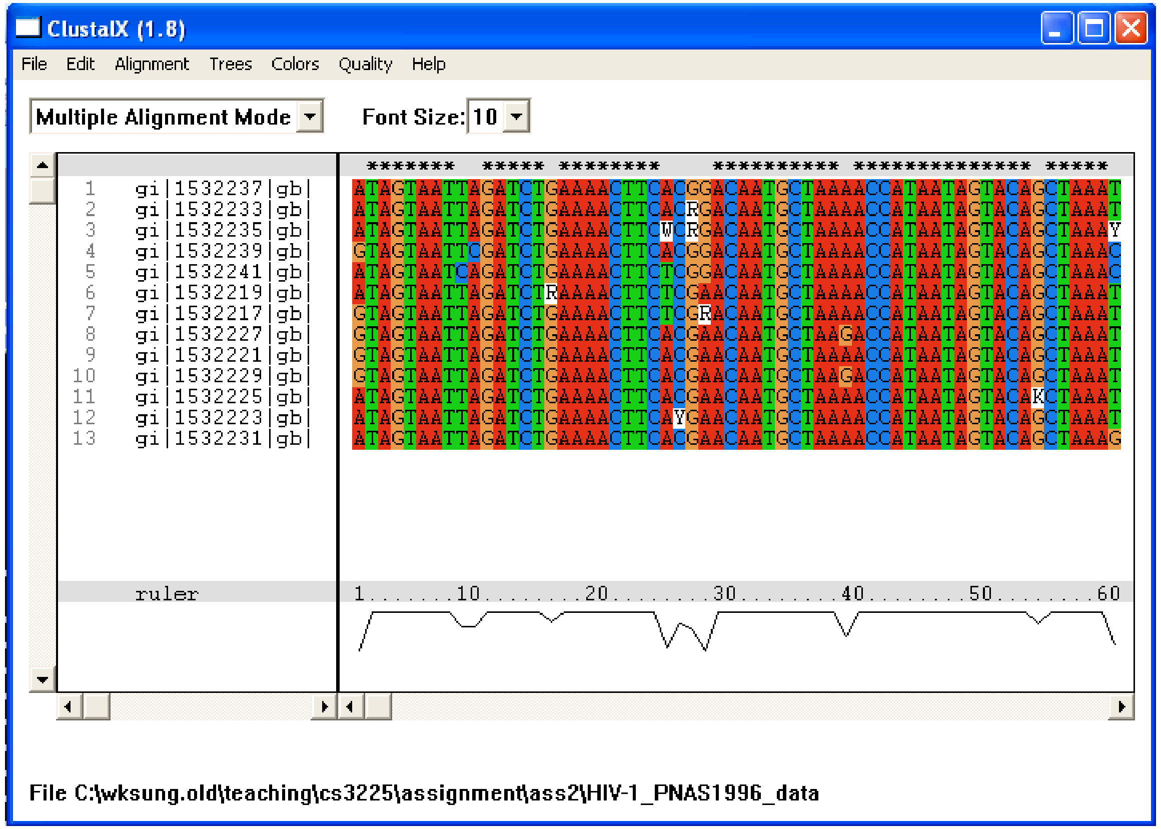
Basic algorithm

1. Computing pairwise distance scores for all pairs of sequences
2. Generate the guide tree which ensures similar sequences are nearer in the tree
3. Aligning the sequences one by one according to the guide tree

ClustalW

- A popular progressive alignment method to globally align a set of sequences.

- Input: a set of sequences
- Output: the multiple alignment of these sequences



Multiple Alignment Mode

Font Size: 10

```
                              *******  ***** ********    ********** ************** *****
 1   gi|1532237|gb|          ATAGTAATTAGATCTGAAAACTTCACGGACAATGCTAAAACCATAATAGTACAGCTAAAT
 2   gi|1532233|gb|          ATAGTAATTAGATCTGAAAACTTCACRGACAATGCTAAAACCATAATAGTACAGCTAAAT
 3   gi|1532235|gb|          ATAGTAATTAGATCTGAAAACTTCWCRGACAATGCTAAAACCATAATAGTACAGCTAAAY
 4   gi|1532239|gb|          GTAGTAATTCGATCTGAAAACTTCACGGACAATGCTAAAACCATAATAGTACAGCTAAAC
 5   gi|1532241|gb|          ATAGTAATCAGATCTGAAAACTTCTCGGACAATGCTAAAACCATAATAGTACAGCTAAAC
 6   gi|1532219|gb|          ATAGTAATTAGATCTRAAAACTTCTCGAACAATGCTAAAACCATAATAGTACAGCTAAAT
 7   gi|1532217|gb|          GTAGTAATTAGATCTGAAAACTTCTCGRACAATGCTAAAACCATAATAGTACAGCTAAAT
 8   gi|1532227|gb|          GTAGTAATTAGATCTGAAAACTTCACGAACAATGCTAAGACCATAATAGTACAGCTAAAT
 9   gi|1532221|gb|          GTAGTAATTAGATCTGAAAACTTCACGAACAATGCTAAAACCATAATAGTACAGCTAAAT
10   gi|1532229|gb|          GTAGTAATTAGATCTGAAAACTTCACGAACAATGCTAAGACCATAATAGTACAGCTAAAT
11   gi|1532225|gb|          ATAGTAATTAGATCTGAAAACTTCACGAACAATGCTAAAACCATAATAGTACAKCTAAAT
12   gi|1532223|gb|          ATAGTAATTAGATCTGAAAACTTCAYGAACAATGCTAAAACCATAATAGTACAGCTAAAT
13   gi|1532231|gb|          ATAGTAATTAGATCTGAAAACTTCACGAACAATGCTAAAACCATAATAGTACAGCTAAAG

     ruler                   1........10........20........30........40........50........60
```

File C:\wksung.old\teaching\cs3225\assignment\ass2\HIV-1_PNAS1996_data

Here, we always use BLOSUM62 to align two sequences.
In ClustalW, it uses different similarity matrix to align two sequences depending on their distance.

Step 1: pairwise distance scores

S_1=PP-GVKSDCAS S_2=PADGVK-DCAS	S_1=PP-GVKSDCAS S_3=PPDG-KSD--S	S_1=-PPGVKSDCAS S_4=GADG-K-DCCS	S_1=-PPGVKSDCAS S_5=GADG-K-DCAS
	S_2=PADGVKDCAS S_3=PPDGKSD--S	S_2=PADGVKDCAS S_4=GADG-KDCCS	S_2=PADGVKDCAS S_5=GADG-KDCAS
		S_3=PPDGKSD--S S_4=GADGK-DCCS	S_3=PPDGKSD--S S_5=GADGK-DCAS
			S_4=GADGKDCCS S_5=GADGKDCAS

- Example:
- For S_1 and S_2, the global alignment is
  - S_1=PP-GVKSDCAS
  - S_2=PADGVK-DCAS
- There are 8 match positions. min($|S_1|$,$|S_2|$)=10.
- The distance is 1 – 8/10 = 0.2

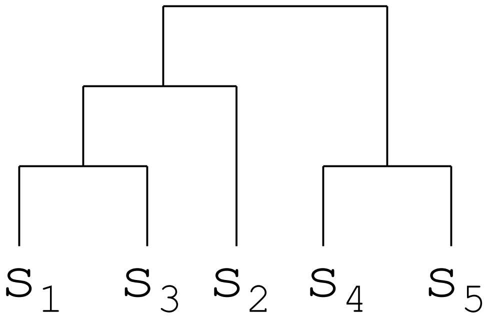
S_1: PPGVKSDCAS
S_2: PADGVKDCAS
S_3: PPDGKSDS
S_4: GADGKDCCS
S_5: GADGKDCAS

	S_1	S_2	S_3	S_4	S_5
S_1	0	0.2	0.125	0.444	0.333
S_2		0	0.375	0.333	0.111
S_3			0	0.5	0.5
S_4				0	0.111
S_5					0

Step 2: generate guide tree

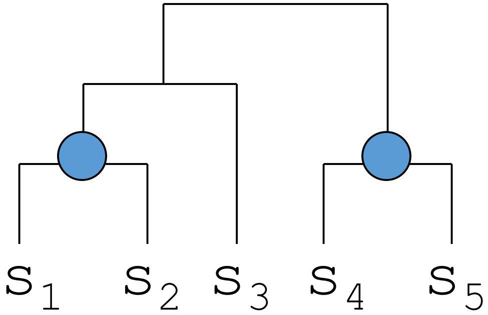
- By neighbor-joining, generate the guide tree.

	S_1	S_2	S_3	S_4	S_5
S_1	0	0.2	0.125	0.444	0.333
S_2		0	0.375	0.333	0.111
S_3			0	0.5	0.5
S_4				0	0.111
S_5					0

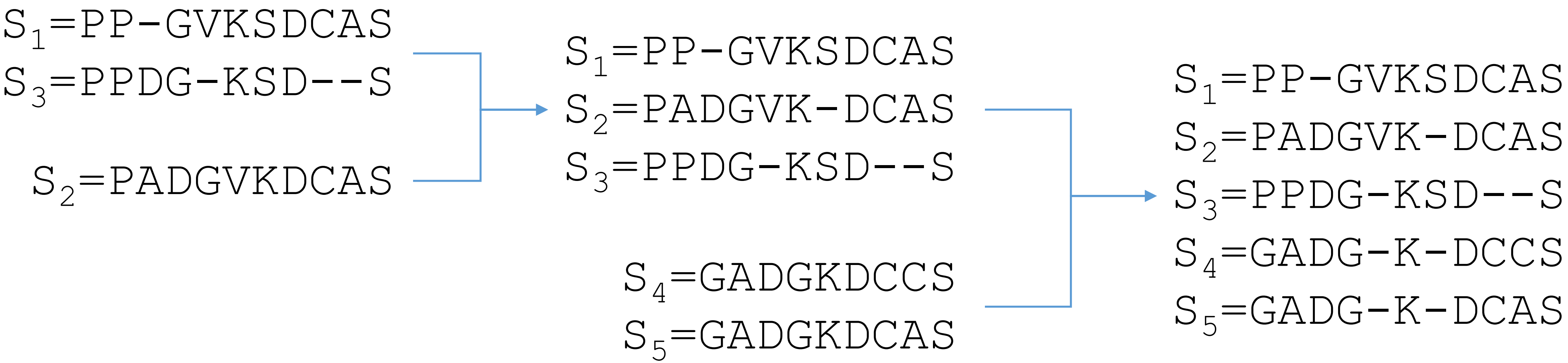


Step 3: align the sequences according to the guide tree (I)

- Aligning S_1 and S_2, we get
    - S_1=PP-GVKSDCAS
    - S_3=PPDG-KSD--S
- Aligning S_4 and S_5, we get
    - S_4=GADGKDCCS
    - S_5=GADGKDCAS



Step 3: align the sequences according to the guide tree (II)



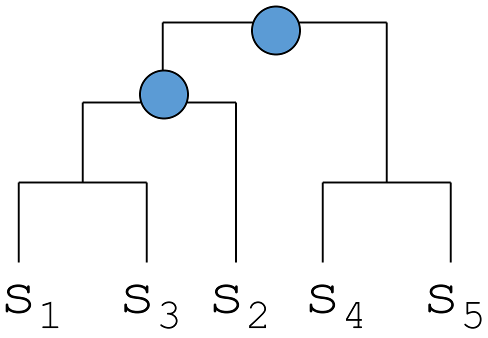
S_1=PP-GVKSDCAS
S_3=PPDG-KSD--S

S_2=PADGVKDCAS

S_1=PP-GVKSDCAS
S_2=PADGVK-DCAS
S_3=PPDG-KSD--S

S_4=GADGKDCCS
S_5=GADGKDCAS

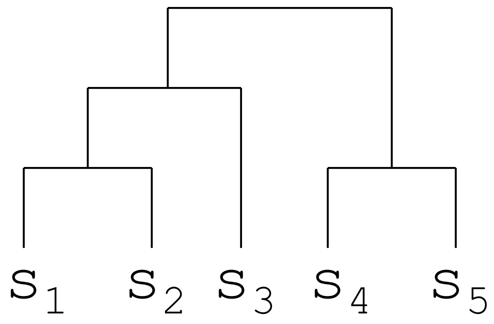
S_1=PP-GVKSDCAS
S_2=PADGVK-DCAS
S_3=PPDG-KSD--S
S_4=GADG-K-DCCS
S_5=GADG-K-DCAS



Summary

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

	S_1	S_2	S_3	S_4	S_5
S_1	0	0.2	0.125	0.444	0.333
S_2		0	0.375	0.333	0.111
S_3			0	0.5	0.5
S_4				0	0.111
S_5					0



S_1: PP-GVKSDCAS
S_2: PADGVK-DCAS
S_3: PPDG-KSD--S
S_4: GADG-K-DCCS
S_5: GADG-K-DCAS

Detail of Profile-Profile alignment (I)

- Given two sets of aligned sequences A_1 and A_2.
- Example:
  - A_1 is a length-11 alignment of three sequences S_1, S_2, S_3
    - S_1=PP-GVKSDCAS
    - S_2=PADGVK-DCAS
    - S_3=PPDG-KSD--S
  - A_2 is a length-9 alignment of two sequences S_4, S_5
    - S_4=GADGKDCCS
    - S_5=GADGKDCAS
- Similar to the sequence alignment,
  - the profile-profile alignment introduces gaps to A_1 and A_2 so that both of them have the same length.

Basic framework for profile-profile alignment

- Two profiles are aligned by a DP algorithm similar to Needleman-Wunsch algorithm.

- The score δ() is replaced by PSP().

			S_4=		G	A	D	G	K	D	C	C	S
			S_5=		G	A	D	G	K	D	C	A	S
S_1=	S_2=	S_3=		0	1	2	3	4	5	6	7	8	9
			0										
P	P	P	1										
P	A	P	2										
-	D	D	3										
G	G	G	4										
V	V	-	5										
K	K	K	6										
S	-	S	7										
D	D	D	8										
C	C	-	9										
A	A	-	10										
S	S	S	11										

PSP(A_1[i], A_2[j])

- Assume A_1[1..n_1] is an alignment of k_1 sequences, A_2[1..n_2] is an alignment of k_2 sequences.
- To determine the alignment, we need a scoring function PSP(A_1[i], A_2[j]).
- In clustalW, the score is defined as follows.
  - $PSP(A_1[i], A_2[j]) = \frac{1}{k_1}\frac{1}{k_2}\sum_{x\in A_1[i], y\in A_2[j]} \delta(x, y)$
- This is a natural scoring for maximizing the SP-score.

- $PSP(A_1[i], A_2[j])$ can be computed in O($|A_1[i]| \cdot |A_2[j]|$) time.

PSSM of an aligment A[1..n]

- Given an alignment $A_1[1..n_1]$ of k_1 sequences, its PSSM can be computed in $O(k_1n_1)$ time.
- Example:
    - A_1 is a length-11 alignment of three DNA sequences S_1, S_2, S_3
        - S_1=G-TCATAGCAT
        - S_2=GACCAT-GCAT
        - S_3=GGTG-TAG--A
- The Position-Specific Scoring Matrix (PSSM) is

	1	2	3	4	5	6	7	8	9	10	11
A	0	1/3	0	0	2/3	0	2/3	0	0	2/3	1/3
C	0	0	1/3	2/3	0	0	0	0	2/3	0	0
G	1	1/3	0	1/3	0	0	0	1	0	0	0
T	0	0	2/3	0	0	1	0	0	0	0	2/3
–	0	1/3	0	0	1/3	0	1/3	0	1/3	1/3	0

$g_C^{A_1[9]}$

PSP(A_1[i], A_2[j])

- $PSP(A_1[i], A_2[j]) = \sum_{x \in A_1[i], y \in A_2[j]} \delta(x, y)$

- To speedup, the score is modified as follows.
  - $PSP(A_1[i], A_2[j]) = \sum_{x,y \in \Lambda \cup \{-\}} g_x^{A_1[i]} g_y^{A_2[j]} \delta(x, y)$

  where Λ is the set of amino acids and $g_x^{A_1[i]}$ is the frequency of the amino acid x in column A_1[i].

- Then, PSP() can be computed in O($|\Lambda|^2$) time, independent of k_1 and k_2.

Detail of
Profile-Profile Alignment (II)

- Our aim is to find an alignment between A_1 and A_2 that maximizes the PSP score.

- Note: Our discussion assumes linear gap penalty. We did not discuss affline gap penalty.

Dynamic Programming

- Assume $A_1[1..n_1]$ is an alignment of k_1 sequences, $A_2[1..n_2]$ is an alignment of k_2 sequences.
- Let V(i,j) = the score of the best alignment between $A_1[1..i]$ and $A_2[1..j]$.
- We have:

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

- By fill-in the dynamic programming table, we can find the optimal alignment.
- Time complexity: $O(k_1n_1+k_2n_2+|\Lambda|^2n_1n_2)$ time.

Example

- Assume BLOSUM62 and the penalty of a space is -4.
  - A_1[1..11] is the alignment of S_1, S_2, S_3
    - S_1=P-PGVKSDCAS
    - S_2=PADGVK-DCAS
    - S_3=PPDG-KSD--S
  - A_2[1..9] is the alignment of S_4, S_5
    - S_4=GADGKDCCS
    - S_5=GADGKDCAS

By profile-profile alignment, we have

S_1=PP-GVKSDCAS
S_2=PADGVK-DCAS
S_3=PPDG-KSD--S
S_4=GADG-K-DCCS
S_5=GADG-K-DCAS

	0	1	2	3	4	5	6	7	8	9
0	0.00	-4.00	-8.00	-12.00	-16.00	-20.00	-24.00	-28.00	-32.00	-36.00
1	-4.00	-2.00	-5.00	-9.00	-13.00	-17.00	-21.00	-25.00	-29.00	-33.00
2	-8.00	-5.33	-1.33	-5.33	-9.33	-13.33	-17.33	-21.33	-25.33	-29.33
3	-10.67	-8.00	-4.00	1.33	-2.67	-6.67	-10.67	-14.67	-18.67	-22.67
4	-14.67	-4.67	-8.00	-2.67	7.33	3.33	-0.67	-4.67	-8.67	-12.67
5	-17.33	-7.33	-6.00	-5.33	4.67	4.67	0.67	-2.67	-6.33	-10.33
6	-21.33	-11.33	-8.33	-7.00	0.67	9.67	5.67	1.67	-2.33	-6.33
7	-24.00	-14.00	-11.00	-9.67	-2.00	7.00	8.33	4.33	0.33	-1.00
8	-28.00	-18.00	-15.00	-5.00	-6.00	3.00	13.00	9.00	5.00	1.00
9	-30.67	-20.67	-17.67	-7.67	-8.33	0.33	10.33	17.67	13.67	9.67
10	-33.33	-23.33	-19.33	-10.33	-9.00	-2.33	7.67	15.00	17.67	13.67
11	-37.33	-27.33	-22.33	-14.33	-10.33	-6.33	3.67	11.00	15.00	21.67

Complexity

- Step 1 performs k^2 global alignments, which takes $O(k^2n^2)$ time.
- Step 2 performs neighbor-joining, which takes $O(k^3)$ time.
- Step 3 performs at most k profile-profile alignments, each takes $O(kn+n^2)$ time. Thus, Step 3 takes $O(k^2n+kn^2)$ time.

- Hence, ClustalW takes $O(k^2n^2+k^3)$ time.

Limitation of progressive alignment method

- Progressive alignment methods have the property of “once a gap, always a gap”. It will not realign the sequences.
  - Hence, the final alignment is bad if we have a poor initial alignment.
- Progressive alignment method does not guaranteed to converge to the global optimal.

- As pointed out by Gibson and co-workers (Thompson et al., 1994), progressive alignment is greedy by nature and may find a local minimum either because the guide tree is not correct or because alignment errors that happen early on in the process of building the MSA.

Iterative method

- To reduce the error in progress alignment, iterative methods are introduced.

- Iterative methods are also heuristics.
- Basic idea:
    - Generate an initial multiple alignment based on methods like progress alignment.
    - Iteratively improve the multiple alignment.

- Examples of iterative method include:
    - PRRP, MAFFT, MUSCLE
- We discuss the detail of MUSCLE.

Multiple sequence comparison by log-expectation (MUSCLE)

- Idea 1:
  - Try to construct a draft multiple alignment as fast as possible; then, MUSCLE iteratively improves the alignment.
- Idea 2:
  - Introduce the log-expectation score for profile-profile alignment

3 Stages of MUSCLE

1. Draft progressive
   - Generate an initial alignment based on some progressive alignment method
2. Improved progressive
   - Based on the alignment generated, compute a more accurate pairwise distance
   - An improved multiple alignment is generated by using a progressive alignment method
3. Refinement
   - An optional tree-based iteration step is included to further improve the alignment.

Stage 1: Draft progressive

- The steps are similar to ClustalW.

1. Pairwise distance matrix
   - To improve efficiency, we first compute the q-mer similarity, which is the fraction F of q-mers shared by two sequences. Then, the distance is 1-F.
2. Build guide tree
   - Instead of using neighbor joining, we use UPGMA, which is more efficient.
3. Profile-profile alignment
   - When performing profile-profile alignment, default uses log-expectation score.

Complexity of Stage 1

- Step 1 performs k^2 q-mer distance computation, which takes $O(k^2n)$ time.
- Step 2 performs UPGMA, which takes $O(k^2)$ time.
- Step 3 performs at most k profile-profile alignments, each takes $O(kn+n^2)$ time. Thus, Step 3 takes $O(k^2n+kn^2)$ time.

- Hence, Stage 1 takes $O(k^2n+kn^2)$ time.

q-mer distance

- Consider two sequence X and Y.
- $F = \frac{\#shared\ q{-}mers}{\min\{|X|-q+1, |Y|-q+1\}}$. The q-mer distance is 1 – F.
  - |X|-q+1 is the number of q-mers in X

- Example: Suppose q=3.
  - X=ACTGACTCAGT
  - Y=ACTACTCAGT
- Shared q-mers = { 2 x ACT, CTC, TCA, CAG, AGT }
- $F = \frac{\#shared\ q{-}mers}{\min\{|X|-q+1, |Y|-q+1\}} = \frac{6}{\min\{9,8\}} = 0.75.$
- The 3-mer distance is 1 – 0.75 = 0.25.

Stage 2: Improved progressive

- The steps are similar to ClustalW.

1. Pairwise distance matrix
   - We first find the fraction D of identical bases shared by two aligned sequences. Then, the distance is $-\log_e(1\text{-}D\text{-}D^2/5)$.
2. Build guide tree
   - The guide tree is built using UPGMA.
3. Profile-profile alignment
   - When performing profile-profile alignment, default uses log-expectation score.
   - Only perform re-alignment when there are changes relative to the original guide tree.

Complexity of Stage 2

- Step 1 performs k^2 distance computation, which takes $O(k^2n)$ time.
- Step 2 performs UPGMA, which takes $O(k^2)$ time.
- Step 3 performs at most k profile-profile alignments, each takes $O(kn+n^2)$ time. Thus, Step 3 takes $O(k^2n+kn^2)$ time.

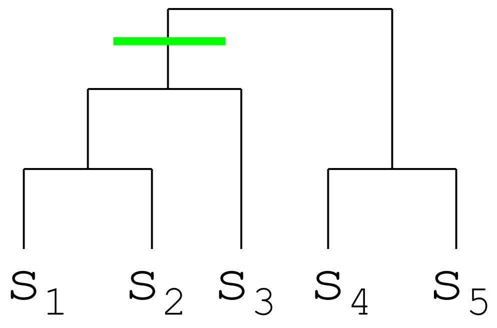
- Hence, Stage 2 takes $O(k^2n+kn^2)$ time.

Stage 3: Refinement

- This stage is optional. It refines the multiple sequence alignment to maximizes the SP-score.

A. Visit the edges e in decreasing distance from the root,
   1. Partition the alignment into two sets by deleting the edge e from the guide tree.
   2. The two sets are realigned using profile-profile alignment, default uses log expectation score.
   3. Compute the SP-score for the new alignment.
   4. If the SP-score is improved, we keep the new alignment.

B. Iterate Step A until there is no improvement in SP-score or a user defined maximum number of iterations.



Complexity of Stage 3

- Step A.1 takes $O(1)$ time.
- Step A.2 takes $O(kn+n^2)$ time.
- Step A.3 takes $O(k^2n)$ time.

- Step A iterates k times. So, Step A takes $O(k^3n+kn^2)$ time.
- Suppose we perform x refinements. This stage takes $O(xk^3n+xkn^2)$ time.

Total running time of MUSCLE

- Stage 1: O(k^2n+kn^2) time
- Stage 2: O(k^2n+kn^2) time
- Stage 3: O(xk^3n+xkn^2) time

- Total time: O($xk^3n +xkn^2$) time.

- Assuming x=O(1), we have
  - Runing time: O(k^3n+kn^2) time.

- Note: The time complexity we got is a bit different from MUSCLE analysis since MUSCLE assumes the length of the alignment is (k+n) instead of n.

PSP score and LE score alignment

- For clustalW, we use the PSP score
  - $\mathrm{PSP}(A_1[i], A_2[j]) = \sum_{x,y} g_x^{A_1[i]} g_y^{A_2[j]} \log_\lambda \left(\frac{p_{xy}}{p_x p_y}\right) = \sum_{x,y} g_x^{A_1[i]} g_y^{A_2[j]} \delta(x, y)$, where
    - $g_x^{A_1[i]}$ is the observed frequency of amino acid x in column i.
    - p_x is the background proportion of amino acid x.
    - p_{xy} is the probability that x aligns with y
    - Note: $\frac{p_{xy}}{p_x p_y} = \lambda^{\delta(x,y)}$

- For MUSCLE, the log-expectation (LE) score is used.
  - Let $g^{A_1[i]} = \sum_{x \in \Lambda} g_x^{A_1[i]}$, which is fraction of non-gap bases.
  - $\mathrm{LE}(A_1[i], A_2[j]) = g^{A_1[i]} g^{A_2[j]} \log_\lambda \left(\sum_{x,y} \frac{g_x^{A_1[i]}}{g^{A_1[i]}} \frac{g_y^{A_2[j]}}{g^{A_2[j]}} \frac{p_{xy}}{p_x p_y} \right)$
    - Note: In MUSCLE, p_x and p_{xy} are derived from the 240 PAM VTML matrix.
    - We did not discuss affline gap penalty

Profile-profile alignment

- For clustalW, we use the PSP score
  - $\mathrm{PSP}(A_1[i], A_2[j]) = \sum_{x,y} g_x^{A_1[i]} g_y^{A_2[j]} \log_\lambda \left(\frac{p_{xy}}{p_x p_y} \right) = \sum_{x,y} g_x^{A_1[i]} g_y^{A_2[j]} \delta(x, y)$, where
    - $g_x^{A_1[i]}$ is the observed frequency of amino acid x in column i.
    - p_x is the background proportion of amino acid x.
    - p_{xy} is the probability that x aligns with y
    - Note: $\frac{p_{xy}}{p_x p_y} = \lambda^{\delta(x,y)}$

- For MUSCLE, the log-expectation (LE) score is used.
  - Let $g^{A_1[i]} = \sum_{x \in \Lambda} g_x^{A_1[i]}$, which is fraction of non-gap bases.
  - $\mathrm{LE}(A_1[i], A_2[j]) = g^{A_1[i]} g^{A_2[j]} \log_\lambda \left(\sum_{x,y} \frac{g_x^{A_1[i]}}{g^{A_1[i]}} \frac{g_y^{A_2[j]}}{g^{A_2[j]}} \frac{p_{xy}}{p_x p_y} \right)$
    - Note: In MUSCLE, p_x and p_{xy} are derived from the 240 PAM VTML matrix.
    - We did not discuss affline gap penalty

Why MUSCLE uses LE() instead of PSP()?

- The issue is that the PSP score does not compute the log likelihood ratio between two profiles.
- Consider the special case of finding the likelihood ratio of transition from an amino acid profile (g_1, …, g_{20}) to the y^th amino acid.
- The likelihood ratio is $\sum_{x=1}^{20} g_x \frac{p_{xy}}{p_x p_y}$. Hence, the log likelihood ratio is $\log\left(\sum_{x=1}^{20} g_x \frac{p_{xy}}{p_x p_y}\right)$.
- The PSP score is $\mathrm{PSP}(A_1[i], y) = \sum_{x=1}^{20} g_x \log\left(\frac{p_{xy}}{p_x p_y}\right)$.
- The log likelihood ratio is different from the PSP score.

Sum-of-Pair (SP) Score: Extension to affine gap penalty

- Consider the multiple alignment S’ of S.
- $\mathrm{SP{-}score}\left(S'_p, S'_q\right)$ is the alignment score after removing columns where both sequences have space.
- $\mathrm{SP{-}score}(S'_1, S'_2, \ldots, S'_k) = \sum_{1 \le p < q \le k} \mathrm{SP{-}score}\left(S'_p, S'_q\right)$.

Example: multiple alignment of 3 sequences

- S_1’ = ACG--GAGA
- S_2’ = -CGTTGACA
- S_3’ = AC-T-GA-A
- Assume score of
  - match and mismatch/insert/delete are 2 and -2, respectively. Inital gap=-1, extend gap=-1
- S_1’ and S_2’ have 5 matches, 1 mistmach, 1 len-1 gap and 1 len-2 gap, $\text{SP}-\text{score}(S'_1, S'_2) = 5*2+1*(-2)+1*(-2)+1*(-3) = 3.$

S_1’ = ACG--GAGA
S_2’ = -CGTTGACA

- S_1’ and S_3’ have 5 matches, 3 len-1 gap, $\text{SP}-\text{score}(S'_1, S'_3) = 5*2+3*(-2) = 4.$

S_1’ = ACG-GAGA
S_3’ = AC-TGA-A

- S_2’ and S_3’ have 5 matches, 4 len-1 gap, $\text{SP}-\text{score}(S'_1, S'_3) = 5*2+4*(-2) = 2.$

S_2’ = -CGTTGACA
S_3’ = AC-T-GA-A

- In total, $\text{SP}-\text{score}(S'_1, S'_2, S'_3) = 3+4+2 = 9.$

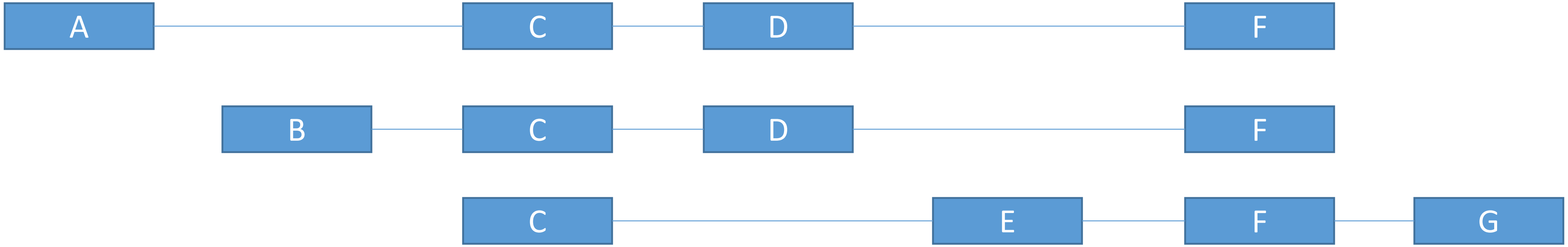
Extension to affine gap penalty

- ClustalW and MUSCLE can be extended to handle affine gap penalty.
- The detail is not covered in the course.

Partial order graph

Motivation

- A gene may have multiple splice variants.



- To align the three splice variants, the alignment will look like:

S_1=AAA---CCCDDD---FFF---

S_2=---BBBCCCDDD---FFF---

S_3=------CCC---EEEFFFGGG

Issue of the multiple sequence alignment

- In 1D alignment, some gaps are artificial.
  - For the previous example, the gaps aligned with AAA and BBB can be interchanged. Similarly, the gaps aligned with DDD and EEE can interchange.
  - They are not real gaps. (AAA, BBB) and (DDD, EEE) are unaligned portions of the MSA.
  - Our previous MSA representation put them in the same linear representation, which is not appropriate.

S_1=AAA---CCCDDD---FFF---

S_2=---BBBCCCDDD---FFF---

S_3=------CCC---EEEFFFGGG

↗

S_1=---AAACCCDDD---FFF---

S_2=BBB---CCCDDD---FFF---

S_3=------CCC---EEEFFFGGG

↘

S_1=AAA---CCC---DDDFFF---

S_2=---BBBCCC---DDDFFF---

S_3=------CCCEEE---FFFGGG

Issues of previous MSA methods (I)

- Previous methods represent MSA as a consensus sequence or a profile (also called Position-Specific Scoring Matrix (PSSM)) during the progressive alignment.
- However, information is lost when MSA is converted into a consensus sequence or a profile.
- In particular, the information of the gap is lost.
- Example: Although the profile shows that positions 4-10 have gaps, the profile cannot tell that they form two different gaps: positions 4-7 and positions 8-10.

S_1 = CAAACAC---AGAAC
S_2 = CACACAC---AGAAC
S_3 = CACACAC---AG-AC
S_4 = CAC----GTTAGAAC
S_5 = CCC----GTTAGAAC
Consensus = CACACAC---AGAAC

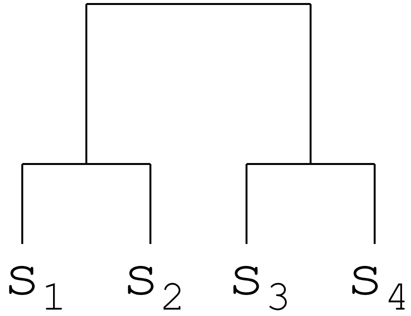
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	0	4/5	1/5	3/5	0	3/5	0	0	0	0	1	0	4/5	1	0
C	1	1/5	4/5	0	3/5	0	3/5	0	0	0	0	0	0	0	1
G	0	0	0	0	0	0	0	2/3	0	0	0	1	0	0	0
T	0	0	0	0	0	0	0	0	2/5	2/5	0	0	0	0	0
–	0	0	0	2/5	2/5	2/5	2/5	3/5	3/5	3/5	0	0	1/5	0	0

Issue of previous MSA methods (II)

- The information loss may create trouble.
- For example, consider 4 sequences:

S_1=AAACCCDDDFFFGGG
S_2=AAACCCEEEFFFGGG
S_3=BBBCCCEEEFFFHHH
S_4=BBBCCCDDDFFFHHH

- The guide tree is:



Issue of previous MSA methods (III)

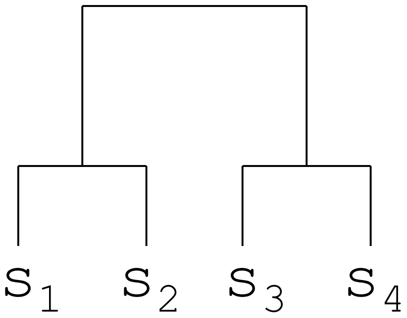
S_1=AAACCCDDD---FFFGGG
S_2=AAACCC---EEEFFFGGG
Consensus =AAACCCDDDEEEFFFGGG

S_3=BBBCCCEEE---FFFHHH
S_4=BBBCCC---DDDFFFHHH
Consensus =BBBCCCEEEDDDFFFHHH

S_1=AAA---CCCDDD------FFFGGG---
S_2=AAA---CCC---EEE---FFFGGG---
S_3=---BBBCCC---EEE---FFF---HHH
S_4=---BBBCCC------DDDFFF---HHH
Consensus =AAABBBCCCDDDEEEDDDFFFGGGHHH

Issue: DDD appears twice!

S_1=AAACCCDDDFFFGGG
S_2=AAACCCEEEFFFGGG
S_3=BBBCCCEEEFFFHHH
S_4=BBBCCCDDDFFFHHH

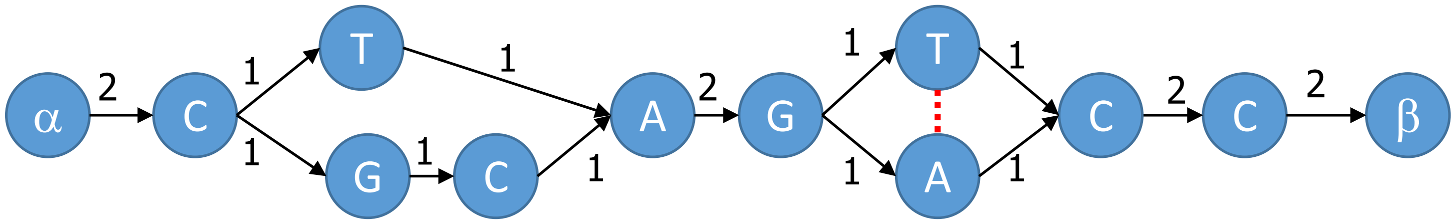


Partial Order Multiple Sequence Alignment (PO-MSA)

- Previous discussion shows that the linear representation of MSA is not good.
- If we relax this constraint, an MAS of a set $\{S_1, S_2, \ldots, S_k\}$ can be represented as a PO-MSA.
- A PO-MSA is a weighted directed acyclic graph where
  1. Each node is labeled by a character (representing a nucleotide or an amino acid residue).
  2. Each node has directed edges pointing to nodes labeled by **distinct** characters.
  3. Every S_i corresponds to a unique directed path in the graph starting from α and ending at β.
  4. The weight of each edge is the number of sequences whose paths passing through the edge.
  5. For characters that are aligned, their corresponding nodes are connected by a dotted line.

S_1=CTAGTCC

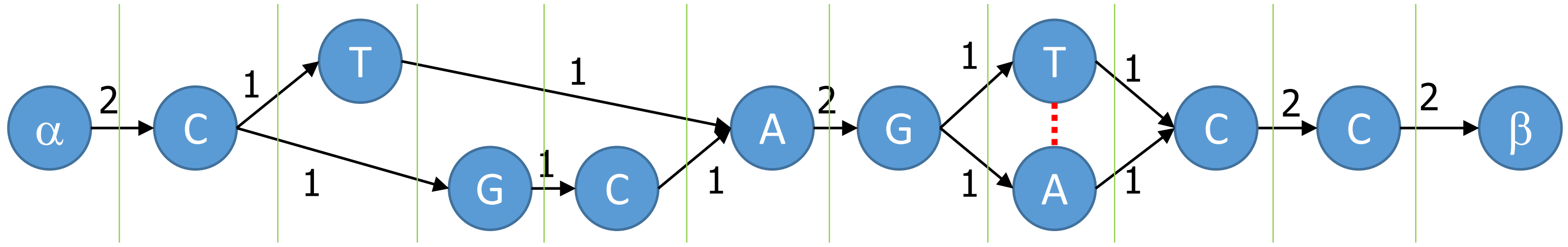
S_2=CGCAGACC



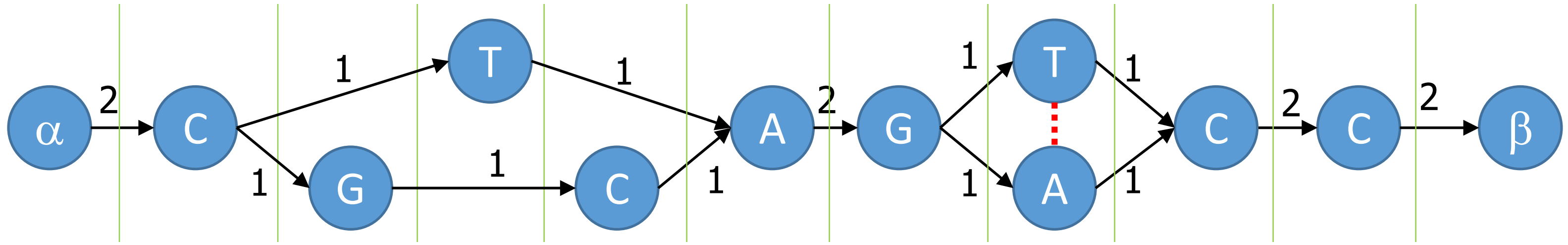
Constructing PO-MSA from the linear representation of MSA

- From the linear MSA, we can construct the corresponding PO-MSA.

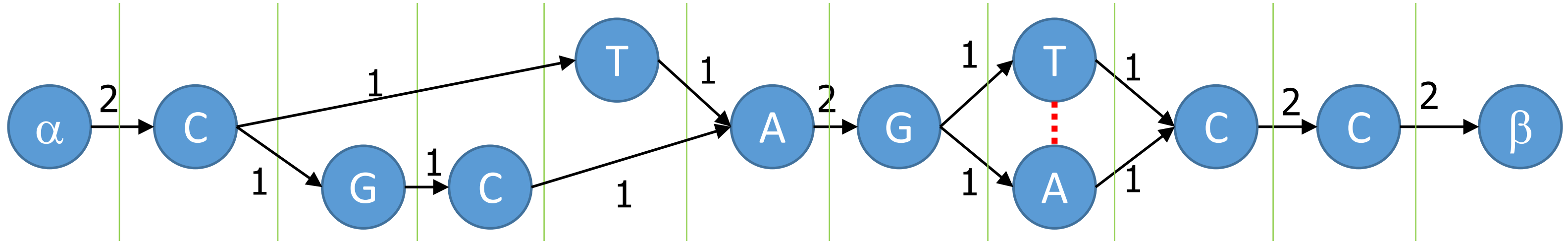
S_1=CT--AGTCC
S_2=C-GCAGACC
Consensus=CTGCAG[TA]CC



S_1=C-T-AGTCC
S_2=CG-CAGACC
Consensus=CGTCAG[TA]CC



S_1=C--TAGTCC
S_2=CGC-AGACC
Consensus=CGCTAG[TA]CC

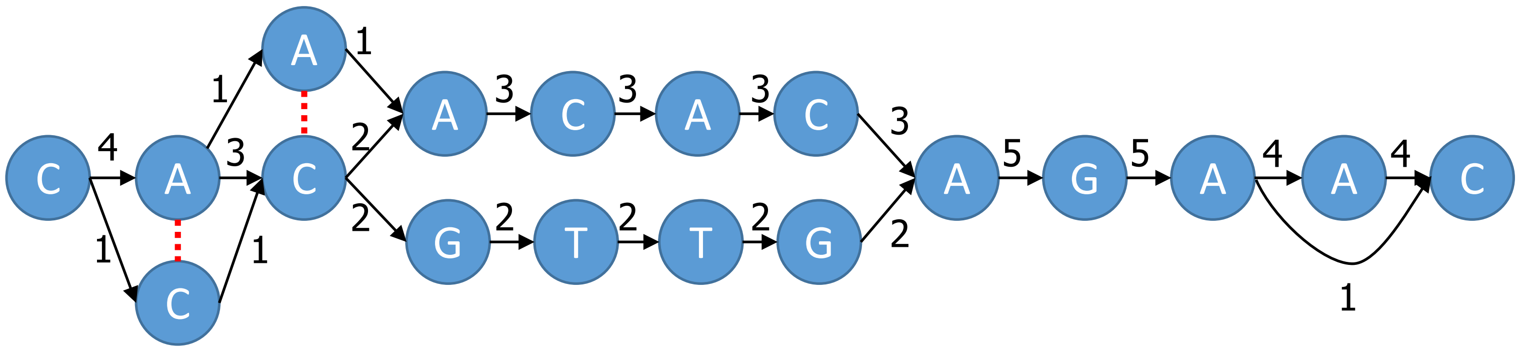


(T, GC) is an unaligned pair. PO-MSA avoids the artificial gap!

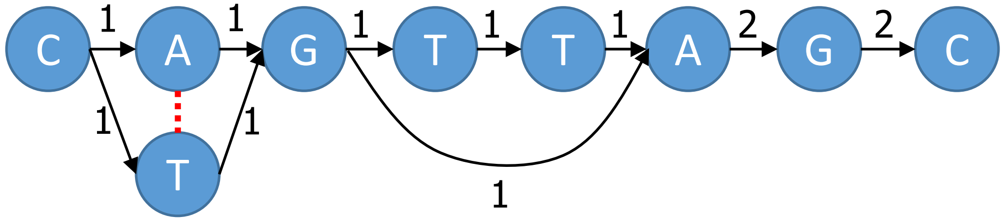
Example

- Two more examples for constructing a PO-MSA from an MSA.

```
S1 = CAAACAC----AGAAC
S2 = CACACAC----AGAAC
S3 = CACACAC----AG-AC
S4 = CAC----GTTGAGAAC
S5 = CCC----GTTGAGAAC
```

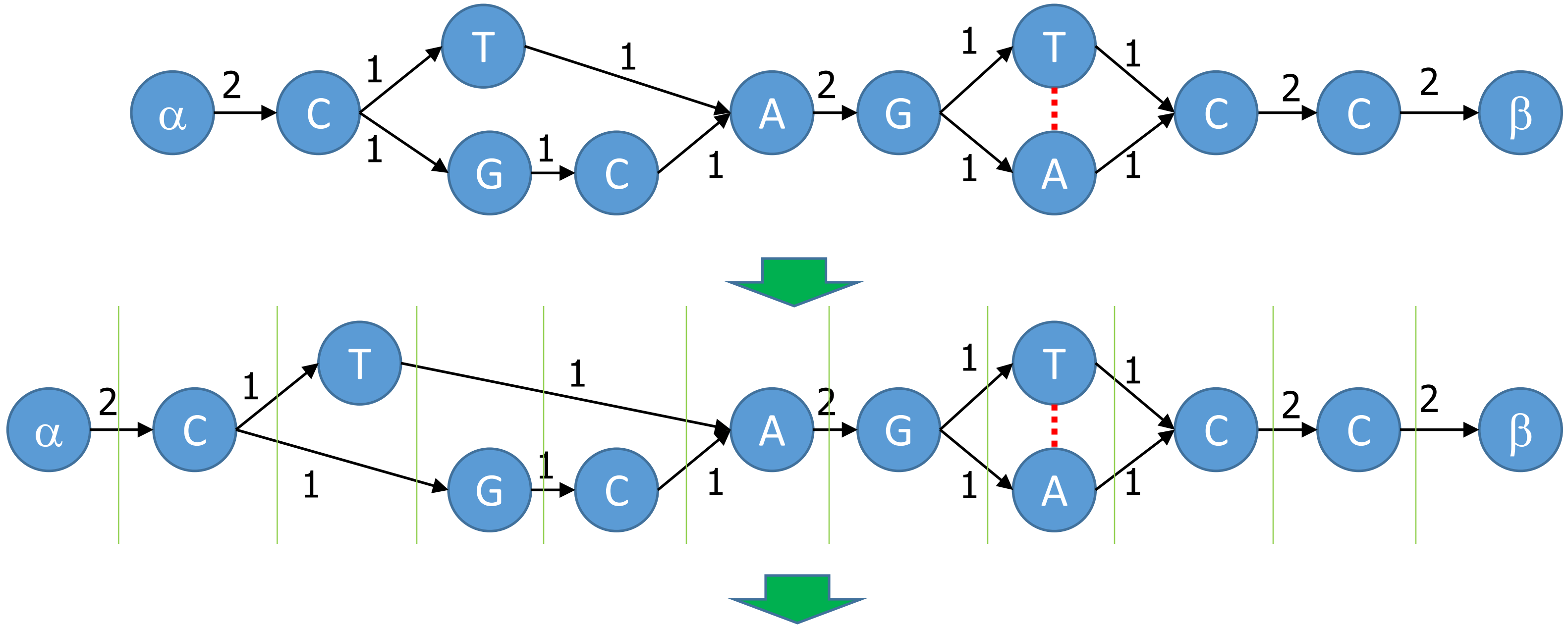


```
S1 = CAG--AGC
S2 = CTGTTAGC
```



Generating linear MSAs from PO-MSA

- Given a PO-MSA, we can generate the linear representation of MSA.



Arrange the nodes in multiple layers. Each layer represents an aligned position.

S_1=CT--AGTCC

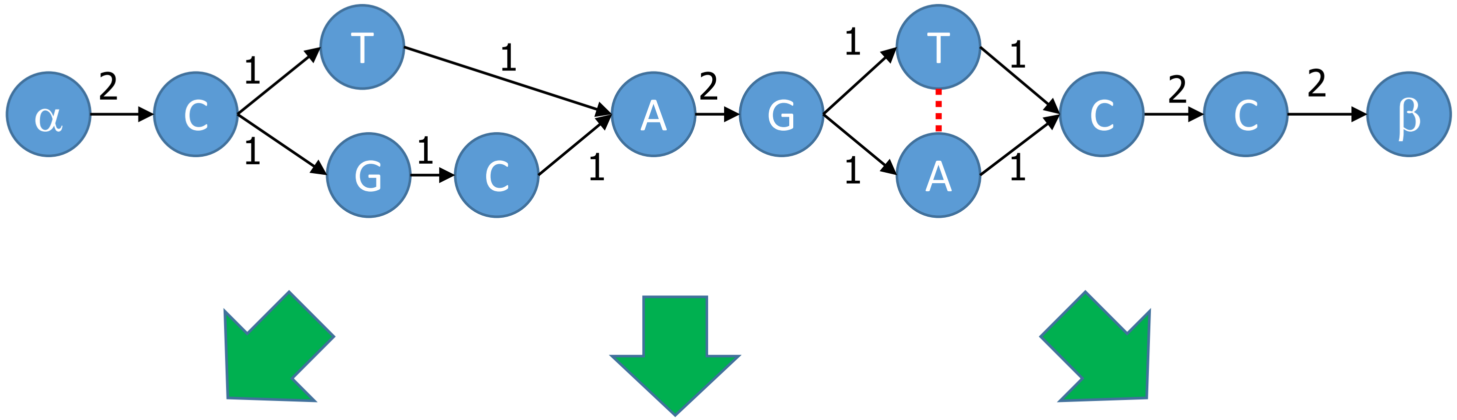
S_2=C-GCAGACC

Consensus=CTGCAG[TA]CC

Generating linear MSAs from PO-MSA

- Given a PO-MSA, we can generate the linear representation of MSA.

In this example, there are 3 ways to arrange the nodes in multiple layers.
We obtain 3 different linear representations of MSA.



S_1=CT--AGTCC
S_2=C-GCAGACC
Consensus=CTGCAG[TA]CC

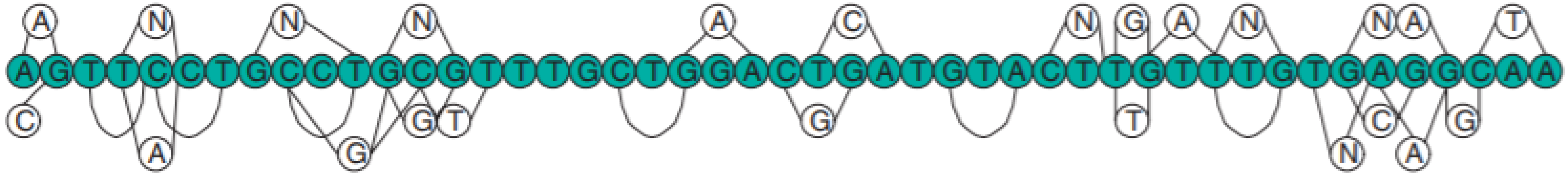
S_1=C-T-AGTCC
S_2=CG-CAGACC
Consensus=CGTCAG[TA]CC

S_1=C--TAGTCC
S_2=CGC-AGACC
Consensus=CGCTAG[TA]CC

An example

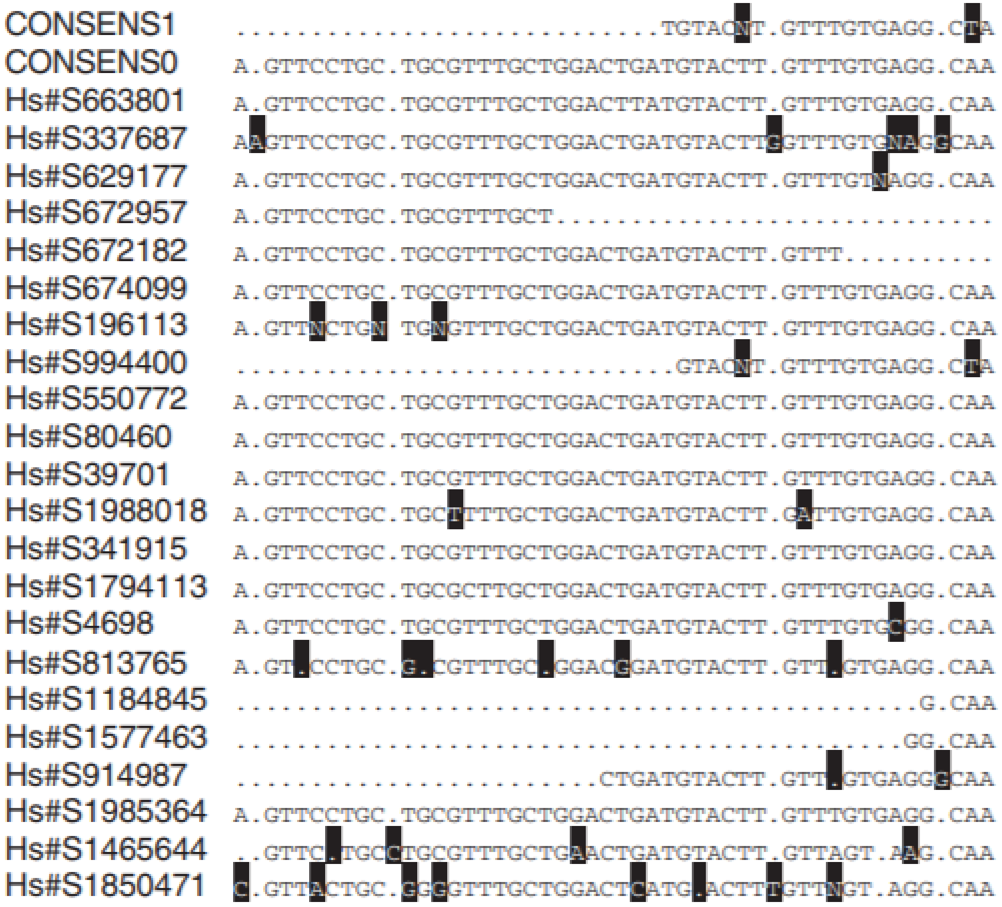
- This is an example for amino acids

(a)



AGTTCCTGCCTGCGTTTGCTGGACTGATGTACTTGTTTGTGAGGCAA
A N N N A C N G A N N A T
C A G G T G T N C A G N A

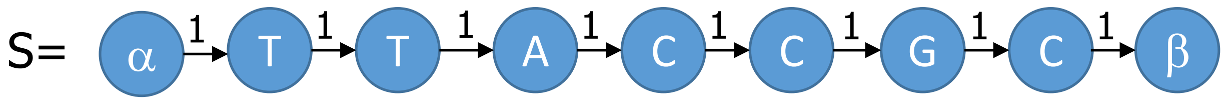
(b)



```
CONSENS1     ............................TGTACNT.GTTTGTGAGG.CTA
CONSENS0     A.GTTCCTGC.TGCGTTTGCTGGACTGATGTACTT.GTTTGTGAGG.CAA
Hs#S663801   A.GTTCCTGC.TGCGTTTGCTGGACTTATGTACTT.GTTTGTGAGG.CAA
Hs#S337687   AAGTTCCTGC.TGCGTTTGCTGGACTGATGTACTTGGTTTGTGNAGGCAA
Hs#S629177   A.GTTCCTGC.TGCGTTTGCTGGACTGATGTACTT.GTTTGTNAGG.CAA
Hs#S672957   A.GTTCCTGC.TGCGTTTGCT.............................
Hs#S672182   A.GTTCCTGC.TGCGTTTGCTGGACTGATGTACTT.GTTT..........
Hs#S674099   A.GTTCCTGC.TGCGTTTGCTGGACTGATGTACTT.GTTTGTGAGG.CAA
Hs#S196113   A.GTTNCTGN TGNGTTTGCTGGACTGATGTACTT.GTTTGTGAGG.CAA
Hs#S994400   .............................GTACNT.GTTTGTGAGG.CTA
Hs#S550772   A.GTTCCTGC.TGCGTTTGCTGGACTGATGTACTT.GTTTGTGAGG.CAA
Hs#S80460    A.GTTCCTGC.TGCGTTTGCTGGACTGATGTACTT.GTTTGTGAGG.CAA
Hs#S39701    A.GTTCCTGC.TGCGTTTGCTGGACTGATGTACTT.GTTTGTGAGG.CAA
Hs#S1988018  A.GTTCCTGC.TGCTTTTGCTGGACTGATGTACTT.GATTGTGAGG.CAA
Hs#S341915   A.GTTCCTGC.TGCGTTTGCTGGACTGATGTACTT.GTTTGTGAGG.CAA
Hs#S1794113  A.GTTCCTGC.TGCGCTTGCTGGACTGATGTACTT.GTTTGTGAGG.CAA
Hs#S4698     A.GTTCCTGC.TGCGTTTGCTGGACTGATGTACTT.GTTTGTGCGG.CAA
Hs#S813765   A.GT.CCTGC.G.CGTTTGC.GGACGGATGTACTT.GTT.GTGAGG.CAA
Hs#S1184845  .............................................G.CAA
Hs#S1577463  ............................................GG.CAA
Hs#S914987   ........................CTGATGTACTT.GTT.GTGAGGGCAA
Hs#S1985364  A.GTTCCTGC.TGCGTTTGCTGGACTGATGTACTT.GTTTGTGAGG.CAA
Hs#S1465644  ..GTTC.TGCCTGCGTTTGCTGAACTGATGTACTT.GTTAGT.AAG.CAA
Hs#S1850471  C.GTTACTGC.GGGGTTTGCTGGACTCATG.ACTTTGTTNGT.AGG.CAA
```

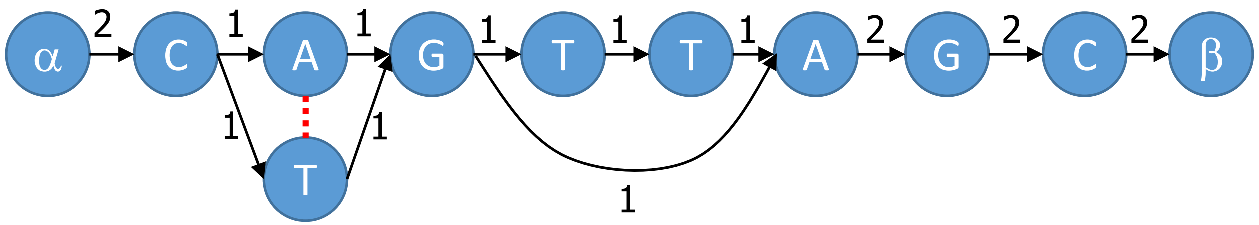
Given a set of sequences, can we build a partial order graph directly?

- One sequence can be represented as a linear graph.
- Example: TTACCGC



- For two sequences, we can construct the partial order graph from the pairwise alignment.
- Example:

```
S1 = CAG--AGC
S2 = CTGTTAGC
```



PO-MSA for 3 or more sequences

- Let G_k be the PO-MSA for S_1, S_2, …, S_k.

- Set G_1 be the PO-MSA of S_1;
- For i = 2 to k
  - Compute G_i by aligning S_i with G_{i-1};

- Below, we describe how to align a sequence S_i with a PO-MSA G_{i-1}.

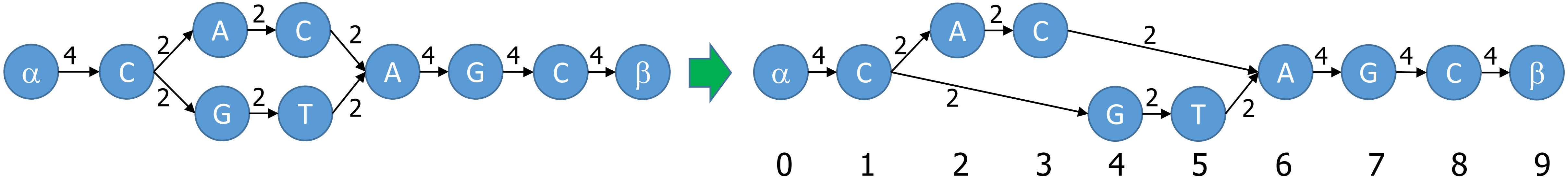
Aligning a sequence and a PO-MSA

- Consider a sequence S[1..m] and a PO-MSA G.

- The alignment involves two steps:
- 1. Find the best path in G that aligns well with S[1..m].
- 2. Incorporate S into G.

Topological sort

- Consider a PO-MSA G with n nodes. We label the nodes in G from 0 to n-1 according to the topological order.



Dynamic programming

Let $\delta()$ be the substation matrix

- Let V(i, v) be the maximum alignment score between any suffix of S[1..i] and any path in G ends at v.
- The solution is similar to Smith-Waterman algorithm.
- Base case (either $i = 0$ or $v = 0$):
  - $V(i, 0) = 0$
  - $V(0, v) = 0$
- Recursive case:
  - $$V(i, v) = \max \begin{cases} 0 & \\ \max\limits_{(u,v)\in G} V(i-1, u) + \delta(S[i], G[v]) & \text{Match/mismatch} \\ V(i-1, v) + \delta(S[i], \_) & \text{Delete} \\ \max\limits_{(u,v)\in G} V(i, u) + \delta(\_, G[v]) & \text{Insert} \end{cases}$$

Algorithm

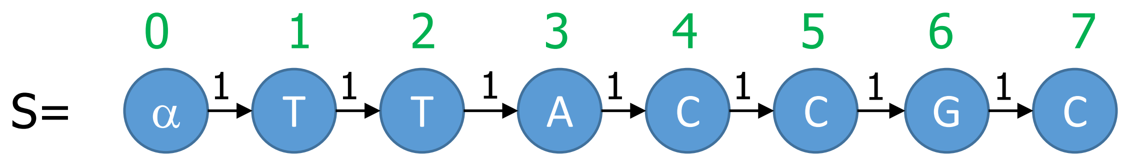
- Input: A sequence S[1..m] and a directed acyclic graph G.
- V(i,0)=0 for i= 0 to m
- V(0,v)=0 for v= 0 to n
- For i = 1 to m
    - For v = 1 to n
        - $V(i,v) = \max \begin{cases} 0 \\ \max\limits_{(u,v)\in G} V(i-1,u) + \delta(S[i], G[v]) \\ V(i-1,v) + \delta(S[i], \_) \\ \max\limits_{(u,v)\in G} V(i,u) + \delta(\_, G[v]) \end{cases}$

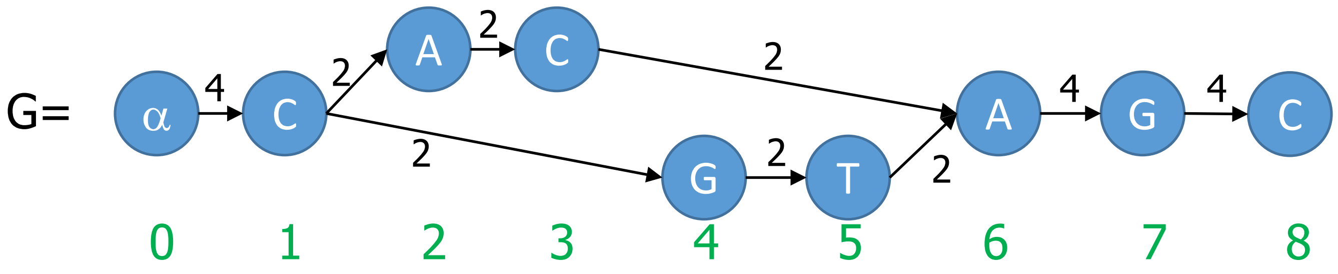
Example

- Assume match=1, mismatch= -1, gap=-2

- S=TTACCGC

- $V(5,6) = \max \begin{cases} 0 \\ V(4,3) + \delta(C,A) \\ V(4,5) + \delta(C,A) \\ V(4,6) + \delta(C,\_) \\ V(5,3) + \delta(\_,A) \\ V(5,5) + \delta(\_,A) \end{cases}$

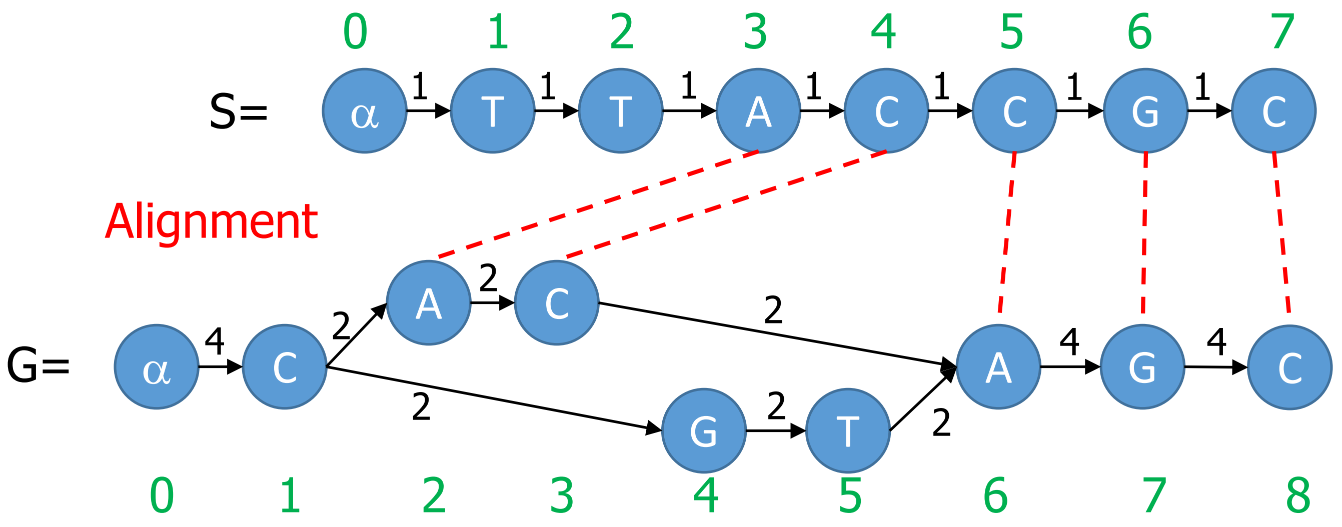




		0	1	2	3	4	5	6	7	8
		_	C	A	C	G	T	A	G	C
0	_	0	0	0	0	0	0	0	0	0
1	T	0	0	0	0	0	1	0	0	0
2	T	0	0	0	0	0	1	0	0	0
3	A	0	0	1	0	0	0	2	0	0
4	C	0	1	0	2	0	0	0	1	1
5	C	0	1	0	0	1	0	**1**		
6	G	0								
7	C	0								

Example

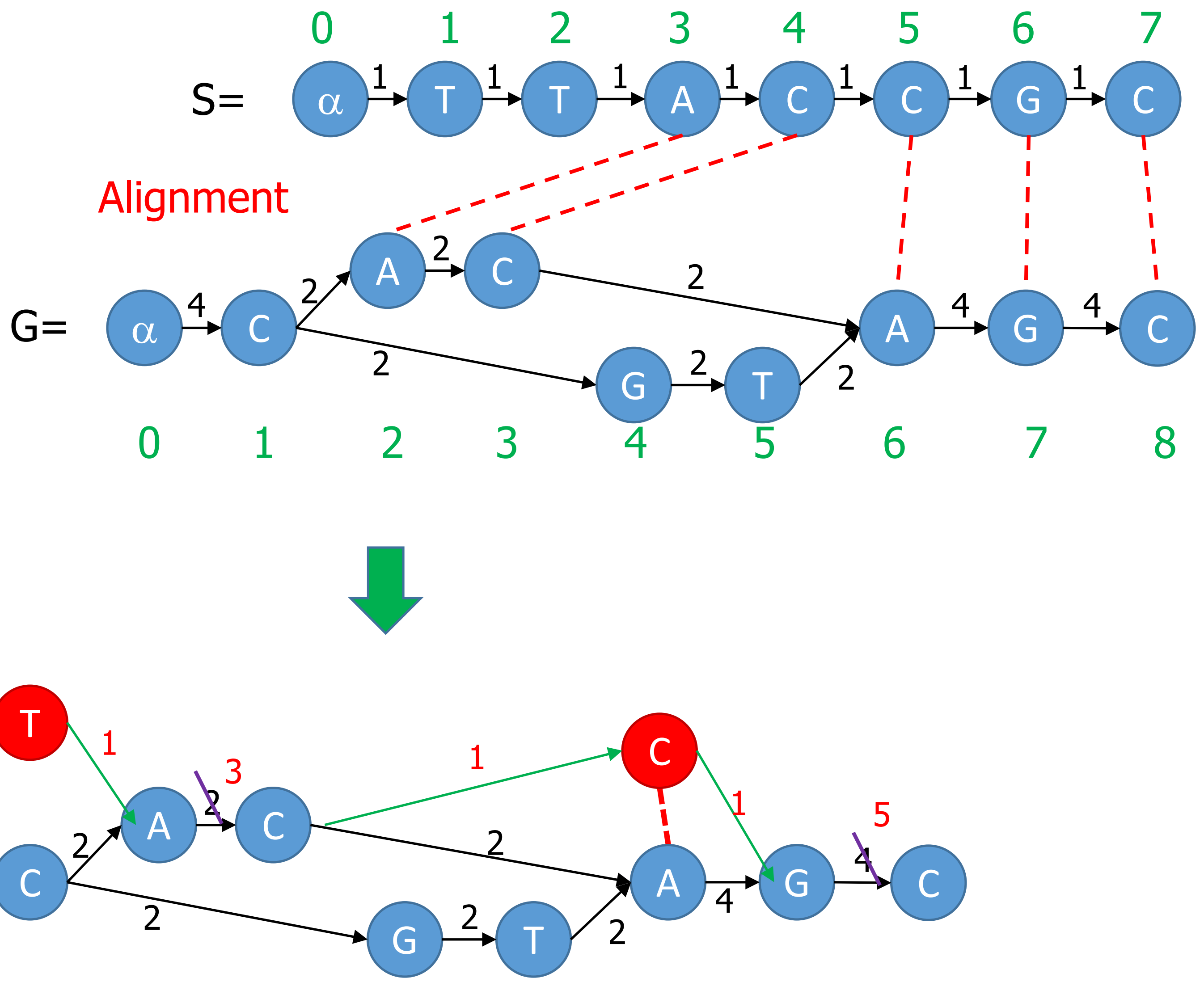
- Assume match=1, mismatch= -1, gap=-2
- S=TTACCGC
- The best alignment score is 3.



		0	1	2	3	4	5	6	7	8
		_	C	A	C	G	T	A	G	C
0	_	0	0	0	0	0	0	0	0	0
1	T	0	0	0	0	0	1	0	0	0
2	T	0	0	0	0	0	1	0	0	0
3	A	0	0	1	0	0	0	2	0	0
4	C	0	1	0	2	0	0	0	1	1
5	C	0	1	0	0	1	0	1	0	2
6	G	0	0	0	0	2	0	0	2	0
7	C	0	1	0	1	0	1	0	0	3

Incorporate S into G

- From the alignment, we merge S and G.



Running time

- Let n be the length of S and m be the number of nodes in G.
- Let $\bar{d}$ be the average in-degree of the nodes in the partial order graph G.
    - Note: $\bar{d} \leq 4$ for DNA and $\bar{d} \leq 20$ for amino acid. For most cases, $\bar{d} \approx 1$.
- The time to align S and G is $\mathrm{O}\left(\bar{d}nm\right)$.
- To incorporate S into G, the back-tracing, it takes $O(\min\{n, m\})$ time.

The basic algorithm is slow due to low locality

$$V(i,v) = \max\begin{cases} 0 \\ \max\limits_{(u,v)\in G} V(i-1,u) + \delta(S[i],G[v]) \\ V(i-1,v) + \delta(S[i],\_) \\ \max\limits_{(u,v)\in G} V(i,u) + \delta(\_,G[v]) \end{cases}$$

- Input: A sequence S[1..m] and a directed acyclic graph G.
- V(i,0)=0 for i= 0 to m
- V(0,v)=0 for v= 0 to n
- For i = 1 to m
  - For v = 1 to n
    - $V(i,v) = \max\{0, V(i-1,v) + \delta(S[i],\_)\}$
    - For every u such that (u, v)∈G,
      - $V(i,v) = \max\begin{cases} V(i,v) \\ V(i-1,u) + \delta(S[i],G[v]) \\ V(i,u) + \delta(\_,G[v]) \end{cases}$

We need to access V(.,u) for (u,v)∈G, which are distributed in different columns in the table V()

Low locality!

Speedup (Improve locality!)

- Input: A sequence S[1..m] and a directed acyclic graph G.
- $V(i, 0) = 0$ for i= 0 to m
- For v = 1 to n
    - $V(i, v) = 0$ for i = 0 to m
    - For every u such that (u, v)$\in$G,
        - For i = 1 to m
            - $V(i, v) = \max \begin{cases} V(i, v) \\ V(i-1, u) + \delta(S[i], G[v]) \\ V(i, u) + \delta(\_, G[v]) \end{cases}$
    - For i = 1 to m
        - $V(i, v) = \max \begin{cases} V(i, v) \\ V(i-1, v) + \delta(S[i], \_) \end{cases}$

Affine gap penalty, global alignment, semi-global alignment

- We illustrated local alignment between a sequence and a partial order graph.
- A number of changes are possible:
  - 1. Use affine gap penalty
  - 2. Perform global alignment instead of local alignment
  - 3. Perform semi-global alignment instead of local alignment

Iterative partial order alignment

- Input: S_1, S_2, S_3, …, S_k.

- Set G = S_1.
- For i = 2 to k
  - Find the alignment of S_i and G
  - Incorporate S_i into G
- Report G

- Let L be the number of nodes in the finally partial order alignment graph and $\bar{d}$ be its average in-degree.
- Let n be the length of the sequence.
- The running time is $\mathrm{O}\left(\mathrm{k}\bar{d}nL\right)$.

Issues with Iterative POA

- Iterative POA is sensitive to the order in which sequences are aligned.

- ClustalW uses guide tree to determine the alignment order.

- Can POA be improved by aligning sequences progressively in the order of a guide tree?

Progressive POA

- Input:
  - A set of sequences S_1, S_2, …, S_n.
  - The pairwise similarity scores between these sequences.

1. Guide tree construction
2. Progressive alignment following the guide tree

Guide tree construction

- Given the pairwise similarity between the sequences, we use distance based method like neighbor joining to build the guide tree.

- This step is similar to the guide tree construction step of ClustalW.

The PO-PO-alignment algorithm

Let $\delta()$ be the substation matrix

- Consider two partial order graphs G_1 and G_2.
- Let V(u, v) be the maximum global alignment score between any path in G_1 ends at u and any path in G_2 ends at v.
- The solution is similar to Needleman–Wunsch algorithm.
- Base case (either $i = 0$ or $v = 0$):
  - $V(0,0) = 0$
  - $V(u,0) = \max_{(p,u)\in G_1} V(p,0) + \delta(G_1[u],\_)$
  - $V(0,v) = \max_{(q,v)\in G_2} V(0,q) + \delta(\_,G_2[v])$
- Recursive case:
  - $V(u,v) = \max \begin{cases} \max_{\substack{(p,u)\in G_1 \\ (q,v)\in G_2}} V(p,q) + \delta(G_1[u], G_2[v]) & \text{Match/mismatch} \\ \max_{(p,u)\in G_1} V(p,v) + \delta(G_1[u],\_) & \text{Delete} \\ \max_{(q,v)\in G_2} V(u,q) + \delta(\_,G_2[v]) & \text{Insert} \end{cases}$

Global alignment

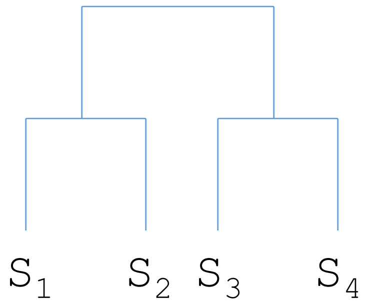
δ	_	A	C	G	T
_		-2	-2	-2	-2
A	-2	1	-1	-1	-1
C	-2	-1	1	-1	-1
G	-2	-1	-1	1	-1
T	-2	-1	-1	-1	1

Example

- Consider the following set of 4 sequences
  - S_1 = ACTAGCT
  - S_2 = ACTAACT
  - S_3 = GTTAGCAG
  - S_4 = GTTAGCAC

M	S_1	S_2	S_3	S_4
S_1		5	-1	-1
S_2			-3	-3
S_3				6
S_4				

- M is the similarity matrix.
- The corresponding neighbor-joining tree is



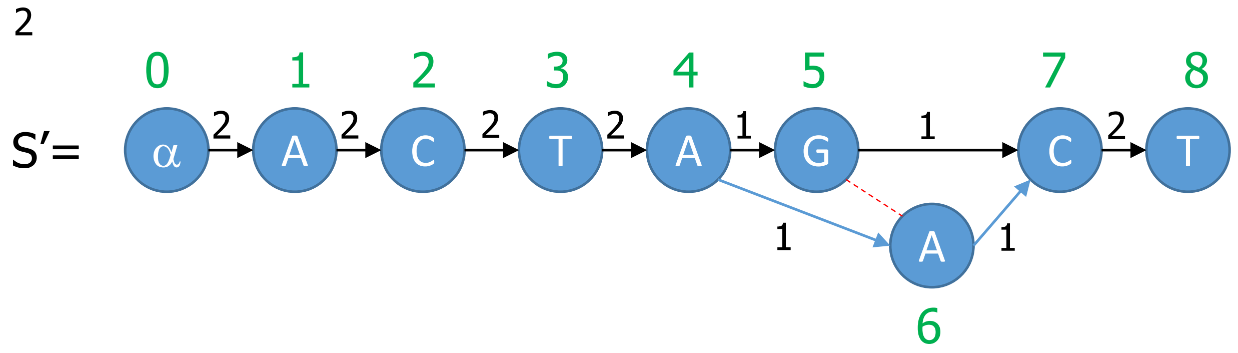
Global alignment

δ	_	A	C	G	T
_		-2	-2	-2	-2
A	-2	1	-1	-1	-1
C	-2	-1	1	-1	-1
G	-2	-1	-1	1	-1
T	-2	-1	-1	-1	1

Aligning reads according to guide-tree

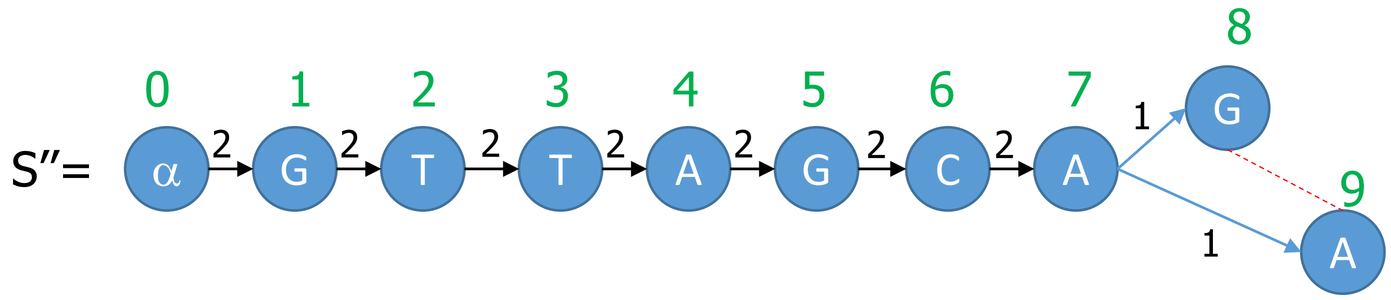
- Aligning S_1 and S_2:

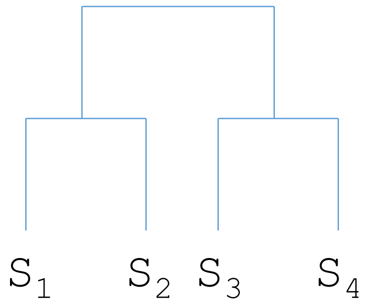
S_1 = ACTAGCT
S_2 = ACTAACT



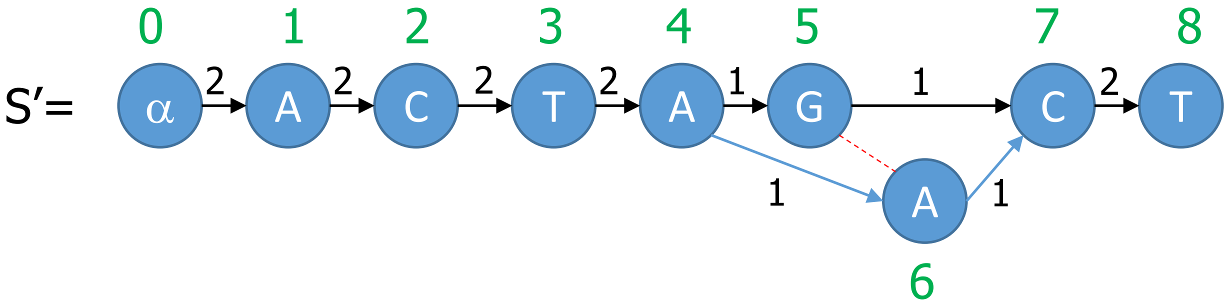
- Aligning S_3 and S_4:

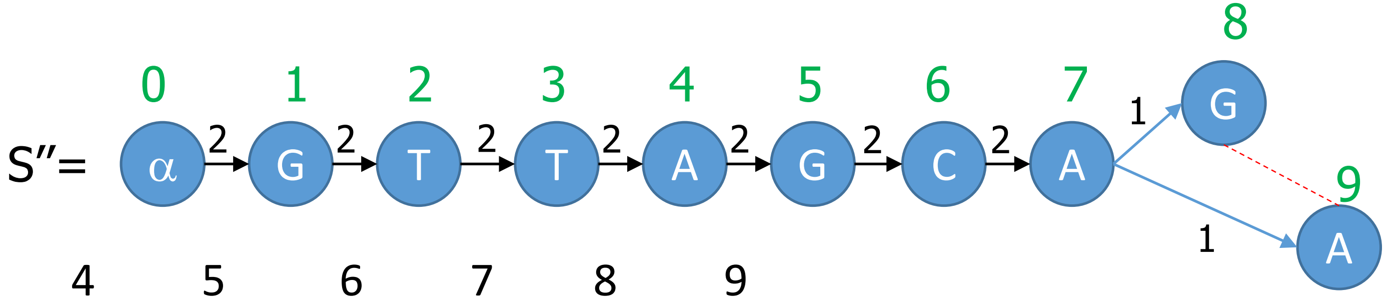
S_3 = GTTAGCAG
S_4 = GTTAGCAC





Aligning S’ and S’’



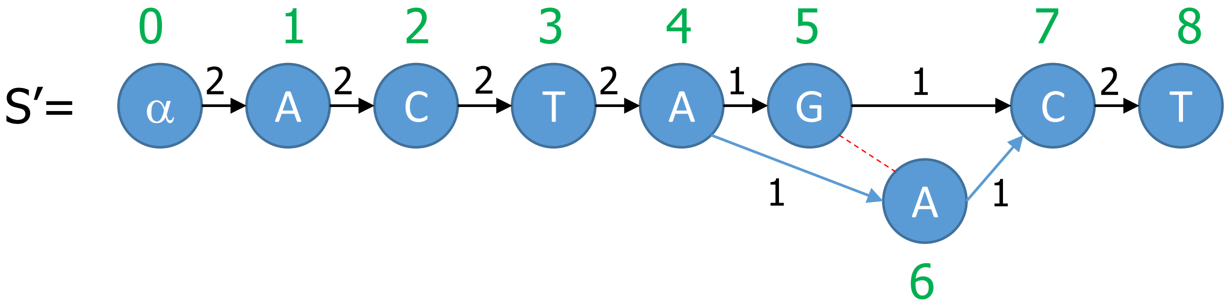


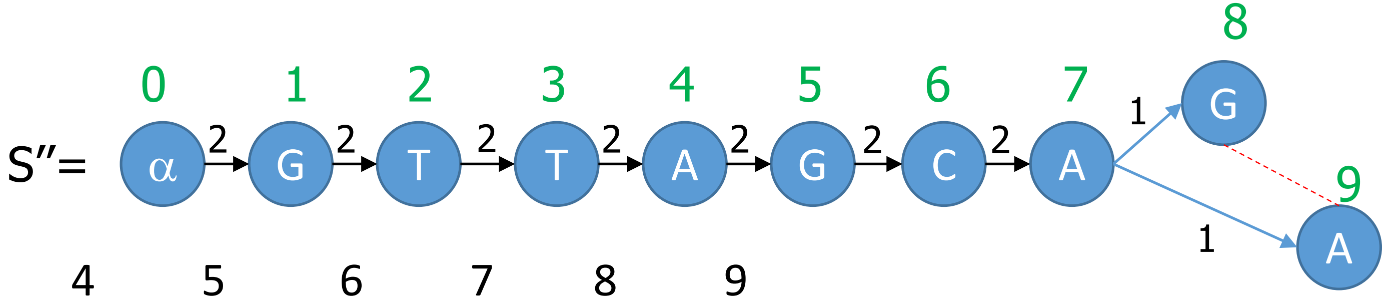
δ	_	A	C	G	T
_		-2	-2	-2	-2
A	-2	1	-1	-1	-1
C	-2	-1	1	-1	-1
G	-2	-1	-1	1	-1
T	-2	-1	-1	-1	1

		0	1	2	3	4	5	6	7	8	9
		_	G	T	T	A	G	C	A	G	A
0	_	0	-2	-4	-6	-8	-10	-12	-14	-16	-16
1	A	-2									
2	C	-4									
3	T	-6									
4	A	-8									
5	G	-10									
6	A	-10									
7	C	-12									
8	T	-14									

Initialization of base cases

Aligning S’ and S’’



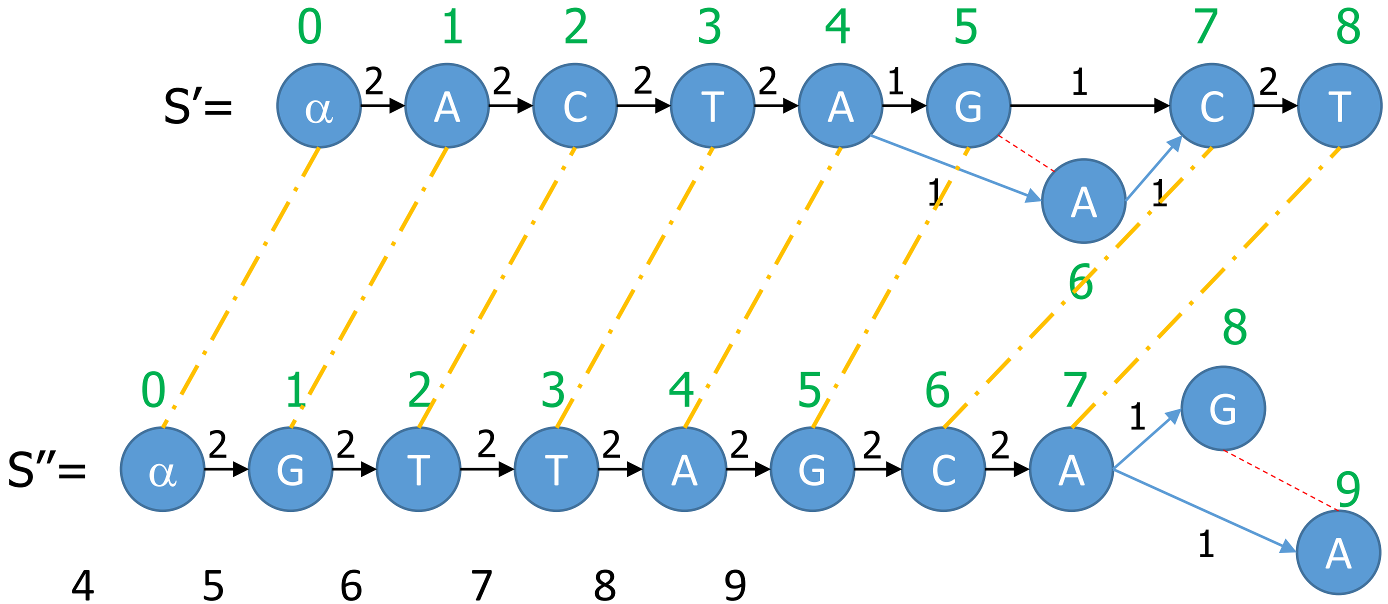


δ	_	A	C	G	T
_		-2	-2	-2	-2
A	-2	1	-1	-1	-1
C	-2	-1	1	-1	-1
G	-2	-1	-1	1	-1
T	-2	-1	-1	-1	1

Recursive cases

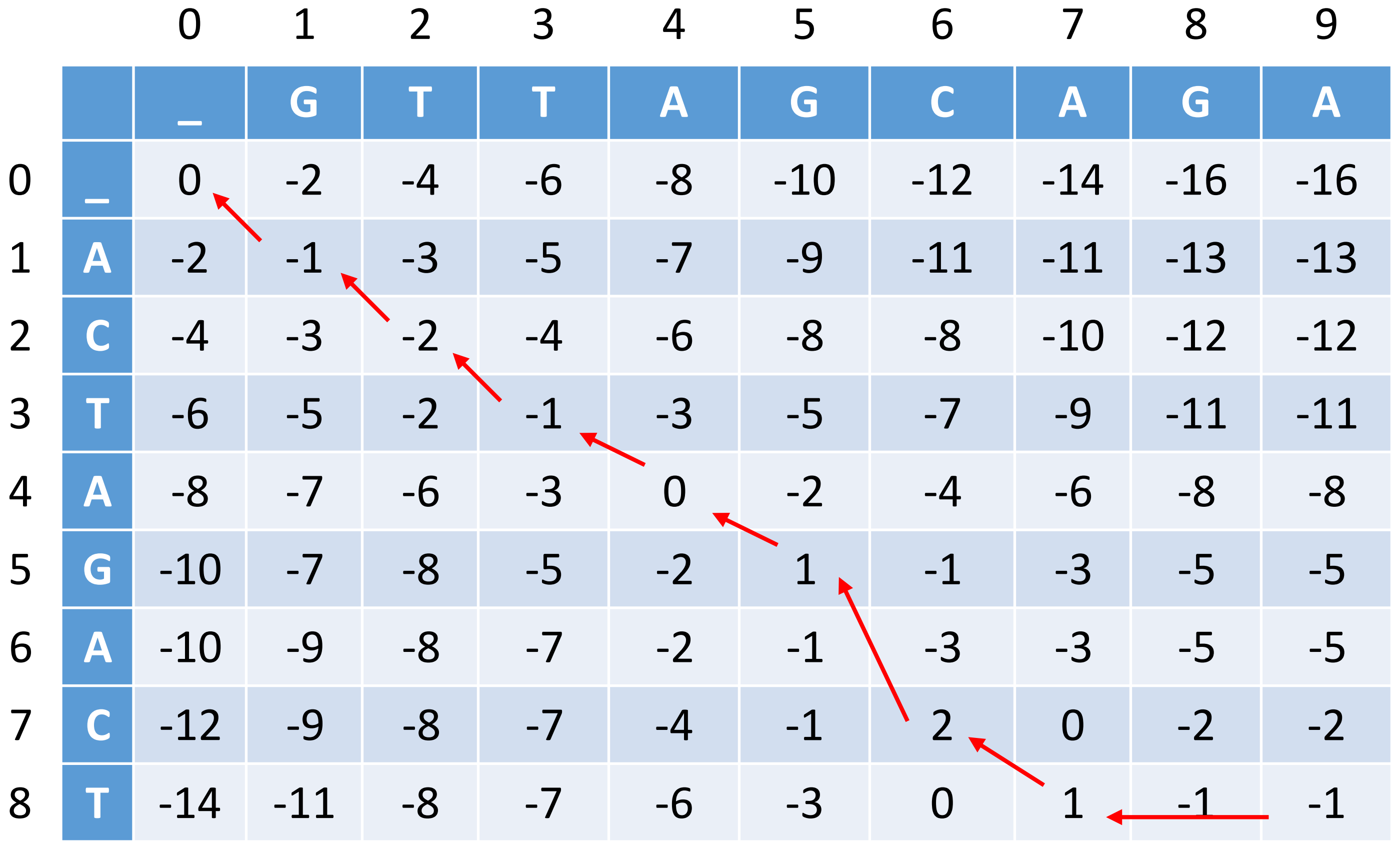
		0	1	2	3	4	5	6	7	8	9
		_	G	T	T	A	G	C	A	G	A
0	_	0	-2	-4	-6	-8	-10	-12	-14	-16	-16
1	A	-2	-1	-3	-5	-7	-9	-11	-11	-13	-13
2	C	-4	-3	-2	-4	-6	-8	-8	-10	-12	-12
3	T	-6	-5	-2	-1	-3	-5	-7	-9	-11	-11
4	A	-8	-7	-6	-3	0	-2	-4	-6	-8	-8
5	G	-10	-7	-8	-5	-2	1	-1	-3	-5	-5
6	A	-10	-9	-8	-7	-2	-1	-3	-3	-5	-5
7	C	-12	-9	-8	-7	-4	-1	2			
8	T										

Aligning S’ and S’’

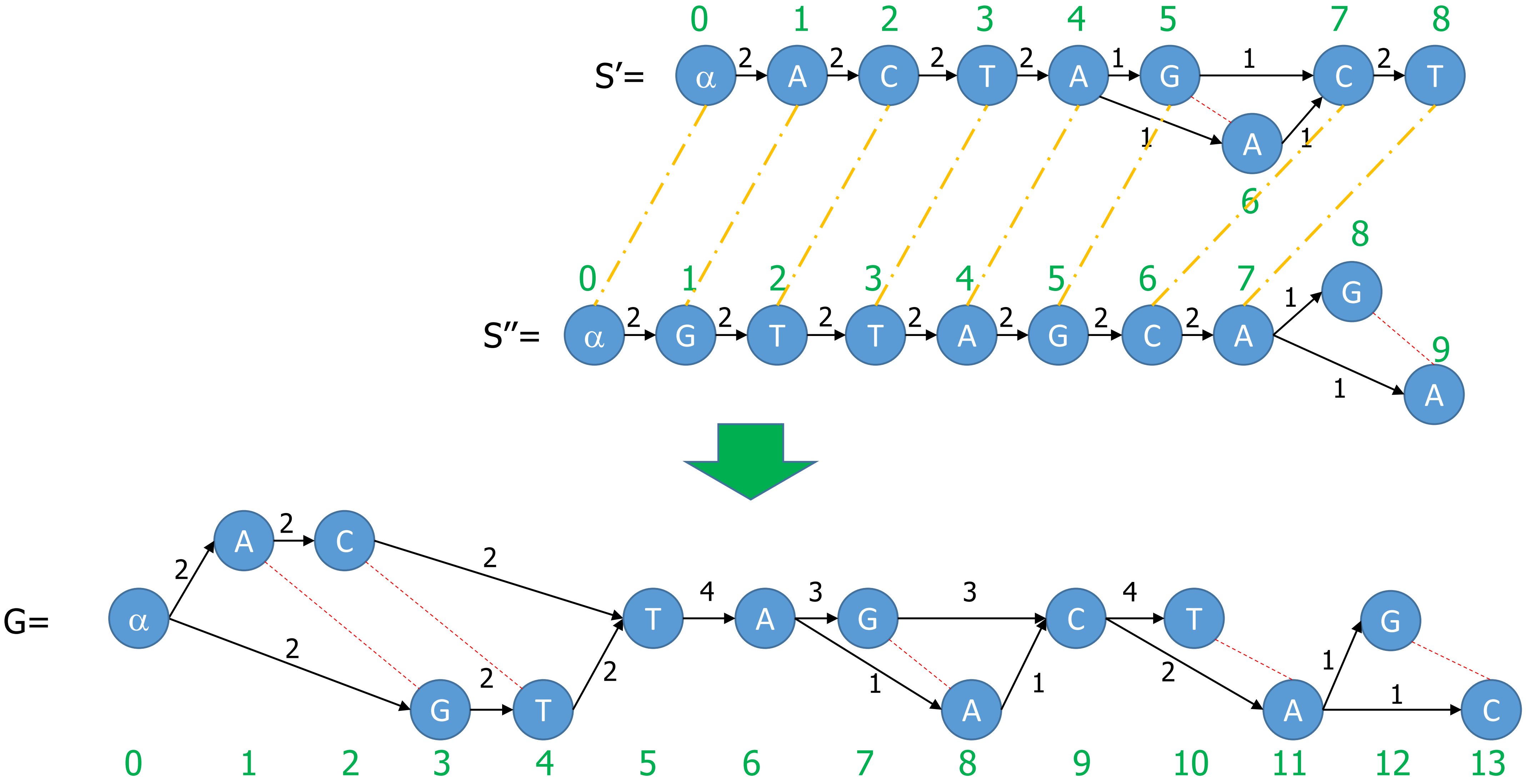


δ	_	A	C	G	T
_		-2	-2	-2	-2
A	-2	1	-1	-1	-1
C	-2	-1	1	-1	-1
G	-2	-1	-1	1	-1
T	-2	-1	-1	-1	1

Recursive cases



		0	1	2	3	4	5	6	7	8	9
		_	G	T	T	A	G	C	A	G	A
0	_	0	-2	-4	-6	-8	-10	-12	-14	-16	-16
1	A	-2	-1	-3	-5	-7	-9	-11	-11	-13	-13
2	C	-4	-3	-2	-4	-6	-8	-8	-10	-12	-12
3	T	-6	-5	-2	-1	-3	-5	-7	-9	-11	-11
4	A	-8	-7	-6	-3	0	-2	-4	-6	-8	-8
5	G	-10	-7	-8	-5	-2	1	-1	-3	-5	-5
6	A	-10	-9	-8	-7	-2	-1	-3	-3	-5	-5
7	C	-12	-9	-8	-7	-4	-1	2	0	-2	-2
8	T	-14	-11	-8	-7	-6	-3	0	1	-1	-1



S_1 = ACTAGCT-

S_2 = ACTAACT-

S_3 = GTTAGCAG

S_4 = GTTAGCAC

Affine gap penalty, local alignment, semi-global alignment

- We illustrated global alignment between a sequence and a partial order graph.
- A number of changes are possible:
  - 1. Use affine gap penalty
  - 2. Perform local alignment instead
  - 3. Perform semi-global alignment instead

Consensus sequence

- Given the multiple sequence alignment, how can we obtain the consensus sequence?

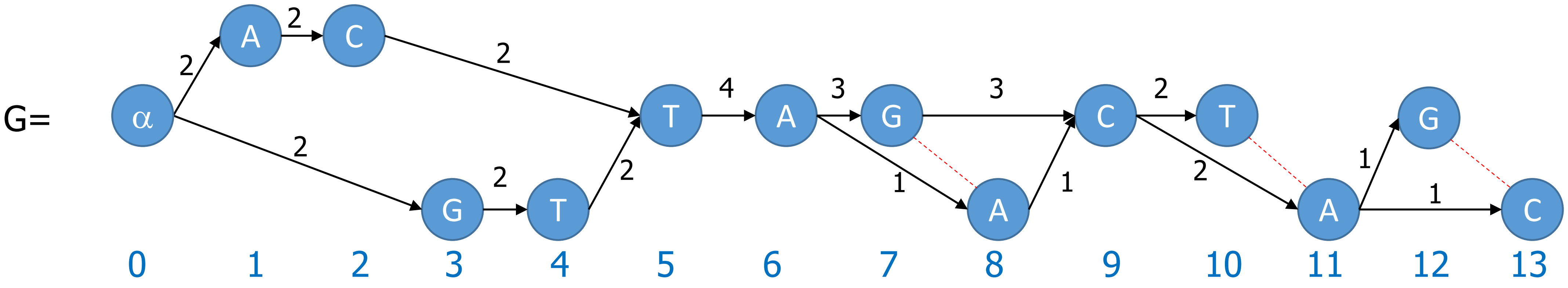
S_1 = AC--TAGCT-
S_2 = AC--TAACT-
S_3 = --GTTAGCAG
S_4 = --GTTAGCAC

- A simple solution: Find the majority base for every aligned column.
  - ACGTTAGCTG

- This sequence is not good since it is not similar to S_1, S_2, S_3, S_4.

Generate consensus sequence from the partial order graph

```
S1 = AC--TAGCT-
S2 = AC--TAACT-
S3 = --GTTAGCAG
S4 = --GTTAGCAC
```



- Given the partial order graph, how can we generate a consensus sequence?
- Precisely, we want to find the path that maximizes the probability.

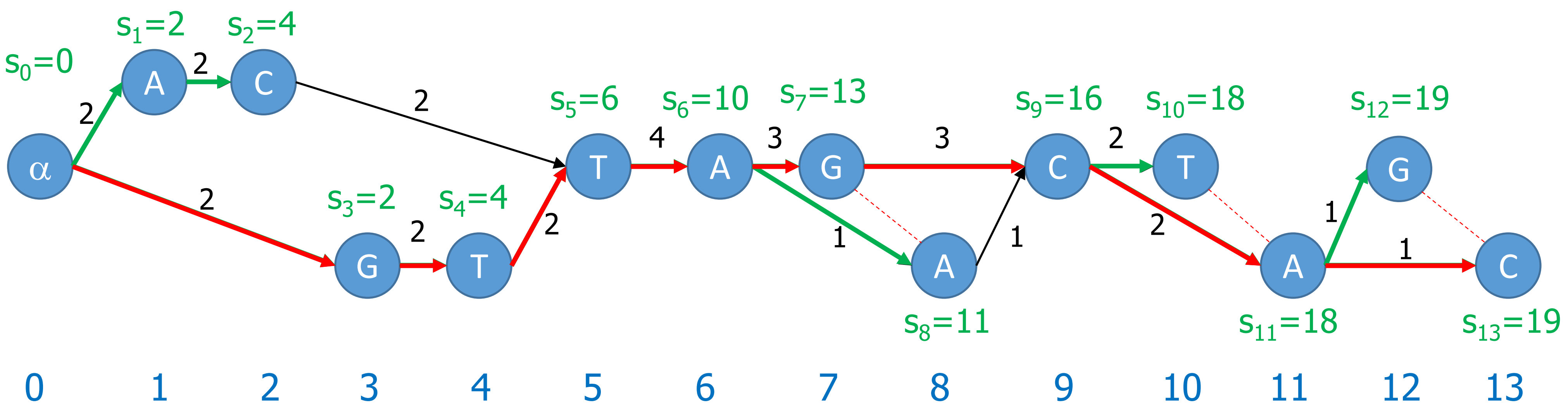
Algorithm for finding consensus sequence

- Assume the nodes in G are ordered from 1 to n in topological order.
- For every edge (i,j) in G, let its weight be w_{ij}, which is the number of sequences passing through (i, j).

- For every node i, denote s_i be the maximum weight of the path from α to node i.
- We have $s_i = \max_{(p,i)\in G, w_{pi}=\delta_i} (s_p + w_{pi})$, where $\delta_i = \max_{(q,i)\in G} w_{qi}$

- For i = 1 to n
  - s_i=0
- For i = 1 to n
  - $\delta_i = \max_{(q,i)\in G} w_{qi}.$
  - $s_i = \max_{(p,i)\in G, w_{pi}=\delta_i} (s_p + w_{pi})$

Example



1. Compute s_i for i = 0, 1, …, 13.
2. For every node i, determine the parent p_i.
   - Note: for node 5, it is a tie, we can select either node 2 or node 4. Here, we select node 4.
3. Node 13 has the maximum score s_{13}=19.
4. By back-tracing, we get the path α→3→4→5→6→7→9→11→13.
   - Hence, the consensus sequence is `GTTAGCAC`.

Additional consensus sequences?

- Previous slides describe a way to obtain a consensus sequence C from G.
- This consensus sequence C may not represent all sequences.
- If necessary, we want to obtain additional consensus sequences.

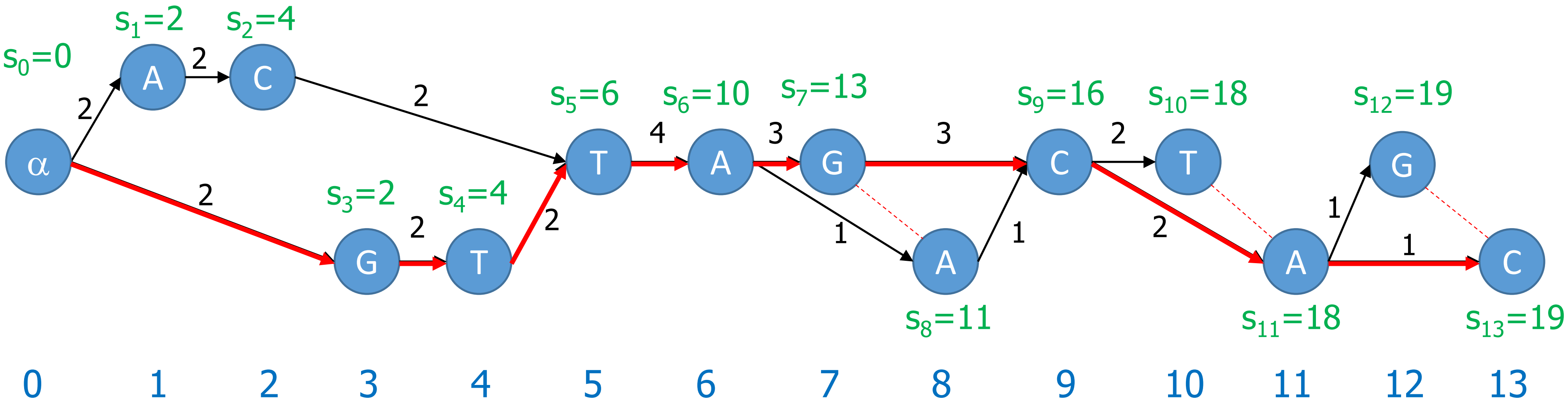
- The steps are as follows.
- 1. Identify sequences that are supported by C
- 2. Remove those sequences that are supported by C from G
- 3. Generate another consensus sequence from G.
- 4. We repeat this process until all sequences are represented by some consensus sequence.

Identify sequences that are supported by the consensus C

- Given a consensus sequence C, a sequence S_i is said to be represented by C if
  - p% of S_i can match C. (Default, p%=50%)
- Example: consensus sequence C = `GTTAGCAC`.
  - S_1 has 3 supports. (less than 50% support)
  - S_2 has 1 supports. (less than 50% support)
  - S_3 has 7 supports.
  - S_4 has 8 supports.

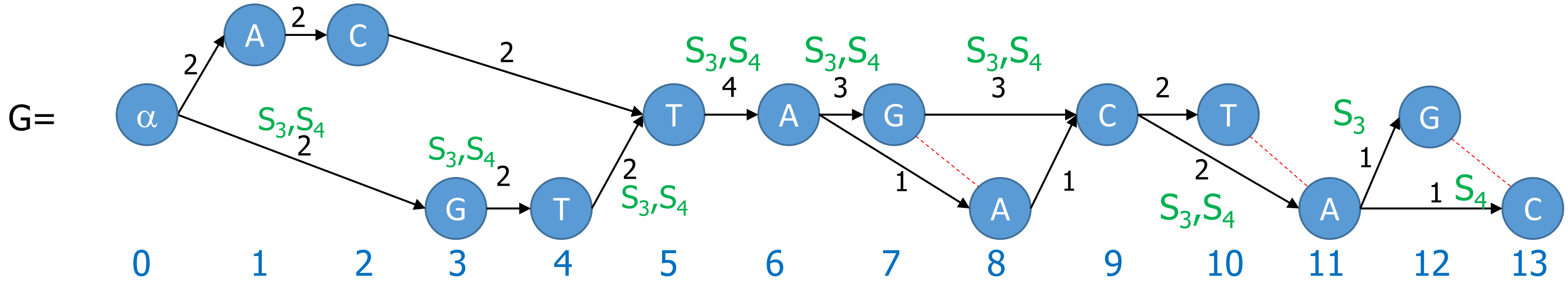
```
S1 = AC--TAGCT-
S2 = AC--TAACT-
S3 = --GTTAGCAG
S4 = --GTTAGCAC
```

G=

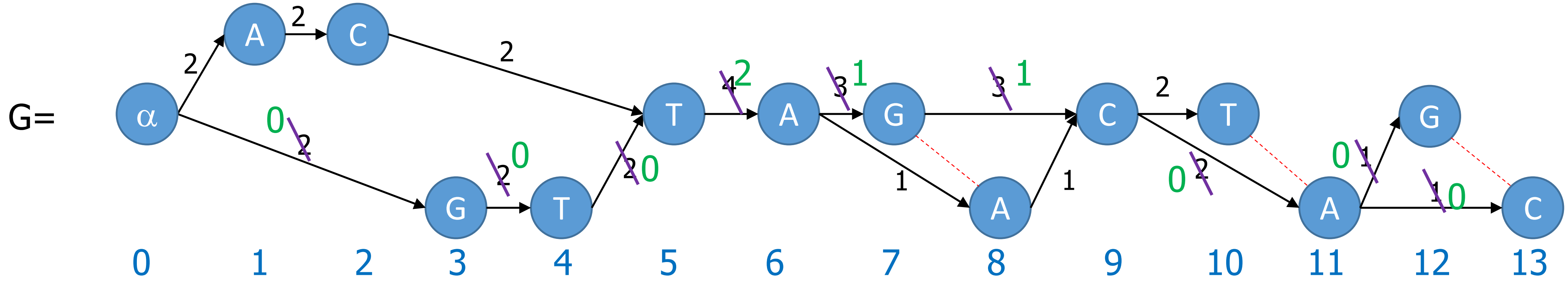


Remove those sequences that are supported by C from G (I)

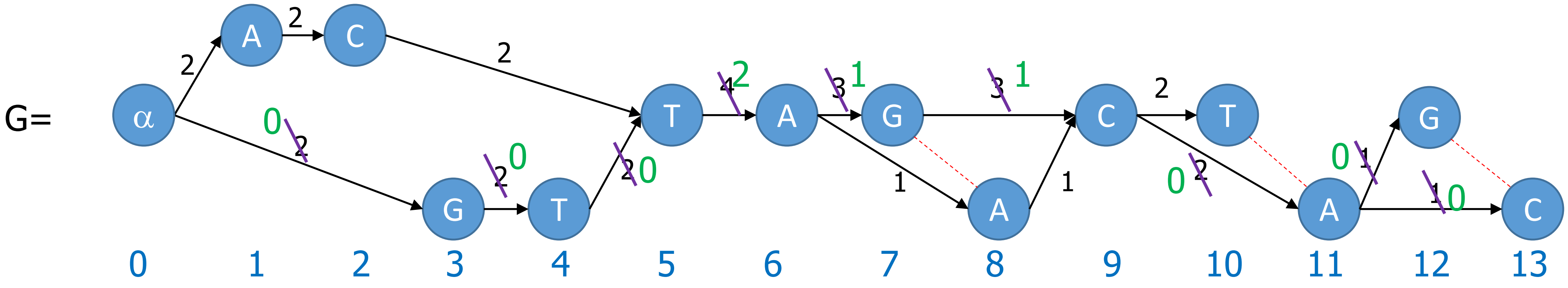
- Mark edges that support S_3 and S_4.



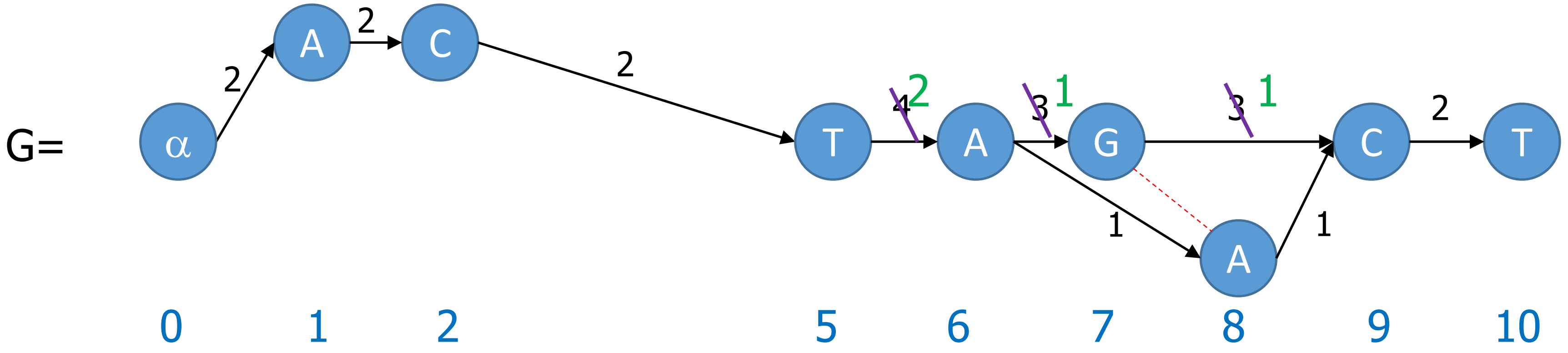
- Update the support counts.



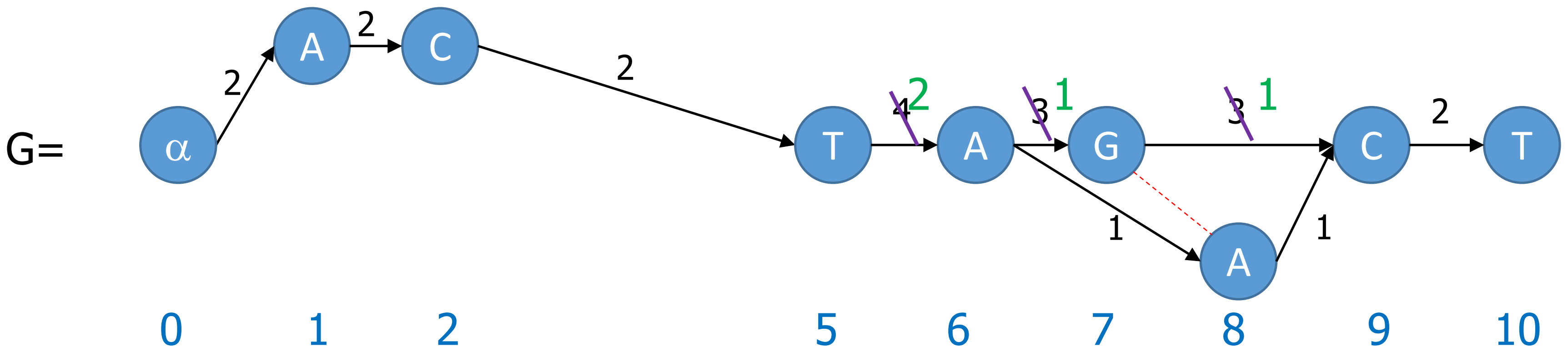
Remove those sequences that are supported by C from G (II)



- Remove edges with count 0.



Generate another consensus sequence



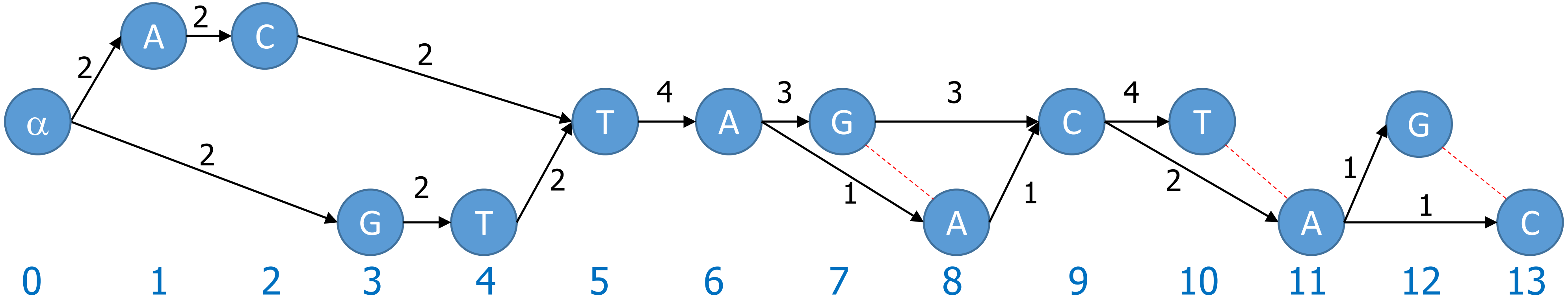
- From this partial order graph, we can obtain another consensus sequence ACTAGCT.

This set of sequences have two consensus sequences

- C_1= GTTAGCAC.
- C_2= ACTAGCT.

```
S1 = AC--TAGCT-
S2 = AC--TAACT-
S3 = --GTTAGCAG
S4 = --GTTAGCAC
```

G=



Sequence alignment graph (SAG)

- POA is an alignment of a set of sequences.

- When a reference is provided, we want to align a set of reads on the reference.
- Sequence alignment graph (SAG) is a way to summarize all alignments from the reads to the reference genome.

- We can use SAG to generate a consensus sequence of all reads.

Reference

- D. F. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. Journal of Mol Evol, 25:351-360, 1987.
- D. G. Higgins and P. M. Sharp. CLUSTAL: a package for performing multiple sequence alignment on a microcomputer. Gene, 73(1):237-244, 1988.
- J. D. Thompson and D. G. Higgins and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. Nucleic Acids Research, 22:4673-4680, 1994.
- R. C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Research, 32:1792-1797, 2004.
- R. C. Edgar. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. BMC Bioinformatics, 5:113, 2004.
- Christopher Lee, Catherine Grasso and Mark F. Sharlow. Multiple sequence alignment using partial order graphs. Bioinformatics, Volume 18, Issue 3, March 2002, Pages 452–464.
- Christopher Lee. Generating consensus sequences from partial order multiple sequence alignment graphs. Bioinformatics, Volume 19, Issue 8, 22 May 2003, Pages 999–1008.
- Catherine Grasso and Christopher Lee. Combining partial order alignment and progressive multiple sequence alignment increases alignment speed and scalability to very large alignment problems. Bioinformatics, Volume 20, Issue 10, 1 July 2004, Pages 1546–1556.