Algorithms in Bioinformatics: A Practical Introduction

Multiple Sequence Alignment
Multiple Sequence Alignment

- Given k sequences $S = \{S_1, S_2, \ldots, S_k\}$.
- A multiple alignment of $S$ is a set of k equal-length sequences $\{S'_1, S'_2, \ldots, S'_k\}$.
  - where $S'_i$ is obtained by inserting gaps into $S_i$.
- The multiple sequence alignment problem aims to
  - find a multiple alignment which optimize certain score.
Example: multiple alignment of 4 sequences

- $S_1 = \text{ACG--GAGA}$
- $S_2 = \text{-CGTTGACA}$
- $S_3 = \text{AC-T-GA-A}$
- $S_4 = \text{CCGTTTCAC-}$
Applications of multiple sequence alignment

- Align the domains of proteins
- Align the same genes/proteins from multiple species
- Help predicting protein structure
Consider the multiple alignment $S'$ of $S$.

$\text{SP-score}(a_1, \ldots, a_k) = \sum_{1 \leq i < j \leq k} \delta(a_i, a_j)$

where $a_i$ can be any character or a space.

The SP-score of $S'$ is

$\sum_x \text{SP-score}(S'_1[x], \ldots, S'_k[x])$. 
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,
- $SP-score(A,-,A,C) = 2\delta(A,-) + 2\delta(A,C) + \delta(A,A) + \delta(C,-) = -8$
- $SP-score= -8+12+0+0–6+0+12–10+0 = 0$
Sum-of-Pair (SP) distance

- Equivalently, we have SP-dist.

- Consider the multiple alignment $S'$ of $S$.

  - $\text{SP-dist}(a_1, \ldots, a_k) = \sum_{1 \leq i < j \leq k} \delta(a_i, a_j)$
    - where $a_i$ can be any character or a space.

- The SP-dist of $S'$ is

  - $\sum_x \text{SP-dist}(S'_1[x], \ldots, S'_k[x])$. 
Agenda

- Exact result
  - Dynamic Programming
- Approximation algorithm
  - Center star method
- Heuristics
  - ClustalW --- Progressive alignment
  - MUSCLE --- Iterative method
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 $S_2[1..i_2]$.

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

- The equation can be rephrased as

$$
V(i_1, i_2) = \max_{(b_1,b_2) \in \{0,1\}^2 - \{(0,0)\}} \left\{ V(i_1 - b_1, i_2 - b_2) + \delta(S_1[i_1b_1], S_2[i_2b_2]) \right\}
$$
Dynamic Programming for aligning k sequences (I)

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

- Observation: The last column of the optimal alignment should be either $S_j[i_j]$ or ‘-’.

- Hence, the score for the last column should be $\text{SP-score}(S_1[b_1i_1], S_2[b_2i_2], \ldots, S_k[b_ki_k])$
  - For $(b_1, b_2, \ldots, b_k) \in \{0,1\}^k$.
  - (Assume that $S_j[0] = ‘-’$.)
Dynamic programming for aligning k sequences (II)

Based on the observation, we have

\[ V(i_1, i_2, \ldots, i_k) = \max_{(b_1, b_2, \ldots, b_k) \in \{0,1\}^k} \{ V(i_1-b_1, \ldots, i_k-b_k) + SP\text{-score}(S_1[b_1], \ldots, S_k[b_k]) \} \]

The SP-score of the optimal multiple alignment of \(S=\{S_1, S_2, \ldots, S_k\}\) is

\[ V(n_1, n_2, \ldots, n_k) \]

where \(n_i\) is the length of \(S_i\).
Dynamic Programming for aligning $k$ sequences (III)

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

- **Time:**
  - The table $V$ has $n_1n_2\cdots n_k$ entries.
  - Filling in one entry takes $2^k k^2$ time.
  - Total running time is $O(2^k k^2 n_1n_2\cdots n_k)$.

- **Space:**
  - $O(n_1n_2\cdots n_k)$ space to store the table $V$.

- Dynamic programming is expensive in both time and space. It is rarely used for aligning 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:
Converting pair-wise alignment to multiple alignment

\[ S_1: \text{CCTGCTGCAG} \]
\[ S_2: \text{GATG-TGCCG} \]
\[ S_3: \text{GATG-TGCAG} \]

\[ S_1: \text{CCTGCTGCAG} \]
\[ S_2: \text{GATG-TGCCG} \]
\[ S_3: \text{GATG-TGCAG} \]

\[ S_1: \text{CCTGCTGCAG} \]
\[ S_2: \text{GATG-TGCCG} \]
\[ S_3: \text{GATG-TGCAG} \]
\[ S_4: \text{CC-GCTAGCAG} \]

\[ S_1: \text{CCTGCTGCAG} \]
\[ S_2: \text{GATG-TGCCG} \]
\[ S_3: \text{GATG-TGCAG} \]
\[ S_4: \text{CC-GCTAGCAG} \]
\[ S_5: \text{CCTG-TAG--G} \]
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 $M$ satisfies the alignments found in Step 3.

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$: CCTGCTGCAG</td>
<td>$S_2$: GATG-TGCCG</td>
<td>$S_3$: GATG-TGCCG</td>
<td>$S_4$: CCTGCTGCAG</td>
</tr>
<tr>
<td>$S_2$: GATG-TGCCG</td>
<td>$S_3$: GATG-TGCCG</td>
<td>$S_4$: CCTGCTGCAG</td>
<td>$S_5$: CCTG-TAG--G</td>
</tr>
<tr>
<td>$S_3$: GATG-TGCCG</td>
<td>$S_4$: CCTGCTGCAG</td>
<td>$S_5$: CCTG-TAG--G</td>
<td>$S_1$: CCTGCTGCAG</td>
</tr>
<tr>
<td>$S_4$: CCTGCTGCAG</td>
<td>$S_5$: CCTG-TAG--G</td>
<td>$S_1$: CCTGCTGCAG</td>
<td>$S_2$: GATG-TGCCG</td>
</tr>
<tr>
<td>$S_5$: CCTG-TAG--G</td>
<td>$S_1$: CCTGCTGCAG</td>
<td>$S_2$: GATG-TGCCG</td>
<td>$S_3$: GATG-TGCCG</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S_1$:</th>
<th>$S_2$:</th>
<th>$S_3$:</th>
<th>$S_4$:</th>
<th>$S_5$:</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$:</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$S_2$:</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>$S_3$:</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>$S_4$:</td>
<td>0</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_5$:</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
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 $O(k^2n^2)$.
Why center star method is good? (1)

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

\[
\begin{align*}
\quad &= \sum_{1 \leq i < j \leq k} d_{M^*}(i, j) \\
&\geq \sum_{1 \leq i < j \leq k} D(S_i, S_j) \\
&= \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1}^{k} D(S_i, S_j) \\
&\geq \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1}^{k} D(S_c, S_j) \\
&= \frac{k}{2} \sum_{j=1}^{k} D(S_c, S_j)
\end{align*}
\]
Why center star method is good? (II)

- The SP-dist of $M$
  
  \[
  \sum_{1 \leq i < j \leq k} d_M(i, j) \\
  = \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1}^{k} d_M(i, j) \\
  \leq \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1}^{k} [D(S_c, S_i) + D(S_c, S_j)] \\
  = \frac{k}{2} \sum_{i=1}^{k} D(S_c, S_i) + \frac{k}{2} \sum_{j=1}^{k} D(S_c, S_j) \\
  = k \sum_{j} D(S_c, S_j)
  \]

- 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 heuristic 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
Step 1: pairwise distance scores

- Example: For $S_1$ and $S_2$, the global alignment is
  - $S_1 = \text{PPGVKS\ldots}$
  - $S_2 = \text{PADGVK\ldots}$

- There are 9 non-gap positions and 8 match positions.
- The distance is $1 - \frac{8}{9} = 0.111$

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>0</td>
<td>0.111</td>
<td>0.25</td>
<td>0.555</td>
<td>0.444</td>
</tr>
<tr>
<td>$S_2$</td>
<td>0</td>
<td>0.375</td>
<td>0.222</td>
<td>0.111</td>
<td></td>
</tr>
<tr>
<td>$S_3$</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>$S_4$</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td>0.111</td>
<td></td>
</tr>
<tr>
<td>$S_5$</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
## Step 2: generate guide tree

- By neighbor-joining, generate the guide tree.

<table>
<thead>
<tr>
<th></th>
<th>S₁</th>
<th>S₂</th>
<th>S₃</th>
<th>S₄</th>
<th>S₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>0</td>
<td>0.111</td>
<td>0.25</td>
<td>0.555</td>
<td>0.444</td>
</tr>
<tr>
<td>S₂</td>
<td>0</td>
<td>0.375</td>
<td>0.222</td>
<td>0.111</td>
<td></td>
</tr>
<tr>
<td>S₃</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>S₄</td>
<td>0</td>
<td>0</td>
<td>0.111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S₅</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Step 3: align the sequences according to the guide tree (I)

- Aligning $S_1$ and $S_2$, we get
  - $S_1 = P-PGVKSDCAS$
  - $S_2 = PADGVK-DCAS$

- 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)

- Aligning (S1, S2) with S3, we get
  - S₁ = P-PGVKSDCAS
  - S₂ = PADGVK-DCAS
  - S₃ = PPDG-KSD--S

- Aligning (S1, S2, S3) with (S4, S5), we get
  - S₁ = P-PGVKSDCAS
  - S₂ = PADGVK-DCAS
  - S₃ = PPDG-KSD--S
  - S₄ = GADG-K-DCCS
  - S₅ = GADG-K-DCAS
Summary

<table>
<thead>
<tr>
<th></th>
<th>S₁</th>
<th>S₂</th>
<th>S₃</th>
<th>S₄</th>
<th>S₅</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>0</td>
<td>0.111</td>
<td>0.25</td>
<td>0.555</td>
<td>0.444</td>
</tr>
<tr>
<td>S₂</td>
<td>0</td>
<td>0.375</td>
<td>0.222</td>
<td>0.111</td>
<td></td>
</tr>
<tr>
<td>S₃</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S₄</td>
<td>0</td>
<td>0.111</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S₅</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

S₁: PPGVKSDCAS
S₂: PADGVKDCAS
S₃: PPDGKSDS
S₄: GADGKDCCS
S₅: GADGKDCAS
Detail of Profile-Profile alignment (1)

- Given two aligned sets of sequences $A_1$ and $A_2$.
- Example:
  - $A_1$ is a length-11 alignment of $S_1$, $S_2$, $S_3$
    - $S_1$ = P-PGVKSDCAS
    - $S_2$ = PADGVK-DCAS
    - $S_3$ = PPDG-KSD--S
  - $A_2$ is a length-9 alignment of $S_4$, $S_5$
    - $S_4$ = 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.
To determine the alignment, we need a scoring function \( \text{PSP}(A_1[i], A_2[j]) \).

In clustalW, the score is defined as follows.

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

where \( g_x^i \) is the observed frequency of amino acid \( x \) in column \( i \).

This is a natural scoring for maximizing the SP-score.

Our aim is to find an alignment between \( A_1 \) and \( A_2 \) to maximize the PSP score.
Example

- $A_1[1..11]$ is the alignment of $S_1$, $S_2$, $S_3$
  - $S_1 = \text{P-PGVKSDCAS}$
  - $S_2 = \text{PADGVK-DCAS}$
  - $S_3 = \text{PPDG-KSD--S}$

- $A_2[1..9]$ is the alignment of $S_4$, $S_5$
  - $S_4 = \text{GADGKDCCS}$
  - $S_5 = \text{GADGKDCAS}$

- $\text{PSP}(A_1[3], A_2[3]) = 1 \times 2 \times \delta(P,D) + 2 \times 2 \times \delta(D,D)$
- $\text{PSP}(A_1[9], A_2[8]) = 2 \delta(C,C) + 2 \delta(C,A) + \delta(-,C) + \delta(-,A)$
Dynamic Programming

- Let $V(i,j) = \text{the score of the best alignment between } A_1[1..i] \text{ and } A_2[1..j]$.
- We have $V(i,j) = \text{maximum of}$
  - $V(i-1,j-1)+\text{PSP}(A_1[i],A_2[j])$
  - $V(i-1,j)+\text{PSP}(A_1[i],-)$
  - $V(i,j-1)+\text{PSP}(-,A_2[j])$

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

- By profile-profile alignment, we have
  - \( S_1 = P-PGVKSDCAS \)
  - \( S_2 = PADGVK-DCAS \)
  - \( S_3 = PPDG-KSD--S \)
  - \( S_4 = GADG-K-DCCS \)
  - \( S_5 = GADG-K-DCAS \)
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 method will not realign the sequence
  - 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.
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 improves the alignment.

- Idea 2:
  - Introduce the log-expectation score for profile-profile alignment
Profile-profile alignment

For clustalW, we use the PSP score

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

where \( g_x^i \) is the observed frequency of amino acid \( x \) in column \( i \).

PSP score may favor more gaps. So, we use the log-expectation (LE) score.

\[ \text{LE}(A_1[i], A_2[j]) = \left(1-f_i^G\right)\left(1-f_j^G\right) \log \left( \sum_{x,y} f_i^x f_j^y \frac{p_{xy}}{p_x p_y} \right) \]

\( f_i^G \) is the proportion of gaps in \( A_1 \)
\( f_i^x \) is the proportion of amino acid \( x \) in \( A_1 \)
\( p_x \) is the background proportion of amino acid \( x \)
\( p_{xy} \) is the probability that \( x \) aligns with \( y \)

Note: \( p_{xy}/(p_x p_y) = e^{\delta(x,y)} \)
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

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, we use 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.
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 $-\log_e(1-D-D^2/5)$.

2. Build guide tree
   - The guide tree is built using UPGMA.

3. Profile-profile alignment
   - When performing profile-profile alignment, we use 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.
   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
- Running 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 $n$. 
Reference


