Comparing Multiple Sequences

Approximate String Matching:

Input: Text \( T \), Pattern \( P \), threshold value \( v \)
Output: All substrings of \( T \) that have similarity score with \( P \) higher than \( v \).

Dynamic programming solution: \( O(nm) \)
LCP: \( O(\sqrt{n} \log m / m) \) based on filtration

Filtration: If \( d(x,y) = -1 \) for \( x \neq y \) or \( x = y ^{\text{"_"}} \)
then \( \text{if } d(S,P) \geq -v \text{ then } \exists \text{ a substring of } S \text{ of size } m/v \text{ which exactly matches the corresponding substring in } S. \)

The general approach is:
1) Find all matches of substrings of size \( l \) in both pattern and text for \( l = \left[ \frac{m}{v+1} \right] \)
2) Verify each match by extending it towards left and right.
Multiple Sequence Alignment

Finding similarity among multiple sequences.

Motivation:
Protein family: set of proteins with similar structure (3-D shape) similar function of evolutionary background. It is important to detect the family of newly sequenced proteins. This may be done through aligning the new sequence with representatives from each family.

Formal definition: Given strings $S_1, ..., S_k$ a multiple global alignment maps them to strings $S'_1, S'_2, ..., S'_k$ that may contain blanks. Here: $1S, 1 = 1S'1 = 1S'1 = 0$ and removal of blanks from $S'_i$ leaves $S_i$.

Challenge: How to score a multiple alignment...

Sum of pairs vs. majority vote. Controversial and not clear which one works best.

We'll focus on sum of pairs as a generalization of pairwise alignment score.

When we have a pair of sequences our goal was to maximize: $\sum_{i=1}^{n} S(C'E_i] J T'C_iJ)$

When we have $k$ sequences our goal will be to maximize: $\sum_{i=1}^{k} \sum_{k \neq j} \sum_{i=1}^{n} S(C'E_iJ, C_j'']$
E.g., $S_1: a ~ c ~ d ~ b ~$
$S_2: ~ c ~ a ~ d ~ b ~ d$
$S_3: a ~ b ~ c ~ d ~ a ~ d$

with $S(x, y) = 0$ if $x = y 
eq "-"$
$S(x, y) = -1$ if $x 
eq y$

Sum-of-pairs score: $-(3 + 5 + 4) = 12$

Optimal alignment: The alignment that maximizes sum-of-pairs score

Algorithm to find optimal alignment:
Dynamic programming with $k$-dimensional table with $(n+1)^k$ entries. Each entry depends on $2^{k-1}$ adjacent entries.

Running time: $O((2n)^k)$; for $n=350$ $k$ cannot be more than 8

Q: A better algorithm? No
The sum of pairs multiple alignment is NP-hard
Introduction to NP-completeness

A problem has a poly-time solution if there is an algorithm that solves the problem in \( O(n^c) \) time for some constant \( c \).

NP problems: problems whose solutions can be checked in poly-time.

NP-complete problems: problems to which any problem in NP can be reduced in poly-time.

There are many problems in NP-complete class. If one is poly-time then all NP-problems have poly-time solutions. Nobody was able to find a poly-time solution to any problem in NPC.

Ways to deal with NPC problems:
1) Sometimes problems have small inputs
2) NPC is on the worst cases - typical cases may work fast
3) Approximation algorithms get close-to-optimal solutions
4) Heuristics can work in practice
5) Practical problems may be more specialized than the theoretical model.
A poly-time approximation scheme:

Assumptions: 1. \( g(x, x) = 0 \)
2. \( g(x, y) \leq g(x, z) + g(z, y) \)

Goal: minimize penalties

Given strings \( S, T \), let \( \text{OPT}(S, T) \) the "minimum" alignment score.

Given \( S_1, S_2 \ldots S_k \) our goal is to minimize sum-of-pairs score:

\[
\sum_{i} \sum_{j \neq i} \text{OPT}(S_i, S_j)
\]

Algorithm: First find \( S_1 \) that minimizes

\[
\sum_{j \geq 2} \text{P}(S_1, S_j)
\]

\( \text{P} \) is the pairwise alignment score

by dynamic programming on \( O(k^2) \) pairs, each requiring \( O(n^2) \) time. Add other strings to this "collection" (which initially has \( S_1 \)) as follows:

At the end of \((i-1)\)th iteration \( S_1 \ldots S_{i-1} \) are in the collection

let the aligned versions of these strings be:

\( S_1', S_2' \ldots S_{i-1}' \)

To add \( S_i \), align it with \( S_1' \). Let the aligned versions of these two strings be \( S_i'' \) and \( S_i''' \). Replace \( S_i'' \) by \( S_i''' \).

Adjust \( S_2', S_3' \ldots \) by adding blank symbols to the locations which has blanks in \( S_i'' \) as a result of alignment with \( S_i \).
Time analysis: Running time is $O(n^2k^2)$

Initial step: $O(n^2k^2)$
Adding the $i$th string to the collection: $O(in.n) = O(n^2)$
Adding all strings $S_2, S_k$: $\sum_{i=2}^{k} O(in^2) = O(k^2n^2)$

Approximation Ratio:
Let $DC(S_i,S_j)$ be the alignment score between $S_i, S_j$ obtained under the method described above,

$$v = \sum_{i=1}^{k} \sum_{j=1}^{k} DC(S_i,S_j)$$

$v$ is twice the sum of pairs score obtained by the method above.
Then $DC(S_i,S_i) = PC(S_i,S_i)$ for all $i \neq 1$
as $PC(S_i',S_j') = PC(S_i,S_j)$ as $C(-,-) = 0$.

Let $v^* = \sum_{i=1}^{k} \sum_{j=1}^{k} opt(S_i,S_j)$

Then:
Claim: $v/v^* \leq 2(k-1)/k < 2$

Notice $v = \sum_{i=1}^{k} \sum_{j=1}^{k} DC(S_i,S_j) \leq \sum_{i=1}^{k} \sum_{j=1}^{k} \sum_{j \neq k}^{k} [DC(S_i,S_j) + DC(S_j,S_i)]$

$$\leq 2(k-1) \sum_{i=2}^{k} PC(S_i,S_i)$$

because each $DC(S_i,S_i) = DC(S_i,S_i)$ occurs $2(k-1)$ times.
However: \( v^* = \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1, i \neq j}^{k} \text{OPT}(S_i, S_j) \geq \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1, i \neq j}^{k} P(S_i, S_j) \geq \left( \sum_{j=2}^{k} P(S_i, S_j) \right) \) \\

\( = k \sum_{j=2}^{k} P(S_i, S_j) \) \\

\( \Rightarrow v^* \geq k \sum_{j=2}^{k} P(S_i, S_j) \) and \( v \leq 2(k-1) \sum_{i=2}^{k} P(S_i, S_j) \) \\

\( \Rightarrow \frac{v}{v^*} \leq \frac{2(k-1)}{k} \) \( v \leq \frac{2(k-1)}{k} \) approximation in \( O(n^2 k^4) \) time