Lecture 19: Edit Distance

Agenda:

- Scoring schemes in sequence comparison
- Edit distance
- Affine gap penalty scoring scheme

Reading:

- No textbook pages

LCS problem review:

Definitions:

- Sequence or String:
  `dynamicprogramming` is a sequence over the English alphabet
- Base/letter/character
- Subsequence:
  the given sequence with zero or more bases left out
  e.g., `dog` is a subsequence of `dynamicprogramming`
  WARNing: bases appear in the same order, but not necessarily consecutive
- Common subsequence
- LCS problem: given two sequences $X = x_1x_2 \ldots x_n$ and $Y = y_1y_2 \ldots y_m$, find a maximum-length common subsequence of them.

- The LCS problem has the "optimal substructure" ...
  - if x_n is NOT in the LCS (to be computed), then we only need to compute an LCS of $x_1x_2 \ldots x_{n-1}$ and $y_1y_2 \ldots y_m$ ...
  - similarly, if y_m is NOT in the LCS (to be computed), then we only need to compute an LCS of $x_1x_2 \ldots x_n$ and $y_1y_2 \ldots y_{m-1}$ ...
  - if x_n and y_m are both in the LCS (to be computed), then $x_n = y_m$ and we need to compute an LCS of $x_1x_2 \ldots x_{n-1}$ and $y_1y_2 \ldots y_{m-1}$;
    and then adding x_n to the end to form an LCS for the original problem

Sequence Alignment:

Definition: An *alignment* of two sequences S_1 and S_2 is obtained by first inserting spaces, either into or at the ends of S_1 and S_2, and then placing the two resultant sequences one above the other so that every character or space in either sequence is opposite a unique character or a unique space in the other sequence.

- An example, $S_1 =$ `rests`, $S_2 =$ `stress`

```
- - r e s t - s

s t r e s - s s
```

Note: space - is not allowed to be opposite to space -!!!

- Scoring scheme:

  For every pair of characters in $\Sigma \cup \{-\}$, say a and b, define a score $s(a,b)$ for them to be aligned in one column of the alignment.

- An example scoring scheme — LCS:

  $s(a,a) = 1$, for all $a \in \Sigma$; otherwise $s(a,\cdot) = 0$

- Another notion: distance — how much it costs if a is replaced by b?

- A distance measure (metric) must satisfy 3 conditions:

  1. $d(a,a) = 0$;
  2. $d(a,b) = d(b,a)$;
  3. $d(a,b) \le d(a,c) + d(b,c)$.

Edit Distance:

- A distance metric which specifies how much it costs to replace letter a by letter b — $d(a,b)$.

- Goal: compute an edit transcript which minimizes the overall cost.

- Again, Edit Distance possesses the *optimal substructure* ...

  Explain how ???

  Letting $Edit[i,j]$ to denote the minimum cost of editing $S_1[1..i]$ into $S_2[1..j]$, then we have the following recurrence:

$$Edit[i,j] = \min \begin{cases} Edit[i-1,j] + d(S_1[i], -), \\ Edit[i,j-1] + d(-, S_2[j]), \\ Edit[i-1,j-1] + d(S_1[i], S_2[j]) \end{cases}$$

- Base cases ???

Edit Distance:

- Pseudocode to implement the above recurrence
- Correctness
- Can return an associated Edit Transcript ... trace back
- Running time: $\Theta(n \times m)$
  There are $n \times m$ entries each takes constant time to compute.
- Space requirement ... $\Theta(n \times m)$

  Can be reduced to $\Theta(\min\{n, m\})$

Scoring Schemes:

- Edit distance:
  1. letter dependent scoring scheme;
  2. letter independent scoring scheme: match, mismatch, insertion/deletion (indel)
- An edit transcript $\Longleftrightarrow$ an alignment

  Score/Cost of the alignment is the sum of scores/costs of columns ...
- Now ask: from `rests` to `stress`,

  Are `r` and `e` deleted separately, or they are deleted at the same time?

  If deleted at the same time, how do we assign a cost for it?
- Or, consecutive spaces should be counted as a *gap* ...
- Affine gap penalty scoring schemes:

  penalties for a gap: gap opening d_o and gap extension d_e
- Now how do we compute an optimal edit transcript?

  Consider three cases ...

Edit Distance with Affine Gap Penalty Scoring Scheme:

- It still possesses the *optimal substructure* ...

  Letting $Edit_M[i,j]$ to denote the minimum cost of editing $S_1[1..i]$ into $S_2[1..j]$ where the last operation is either a match or a mismatch;

  Letting $Edit_I[i,j]$ to denote the minimum cost of editing $S_1[1..i]$ into $S_2[1..j]$ where the last operation is an insertion;

  Letting $Edit_D[i,j]$ to denote the minimum cost of editing $S_1[1..i]$ into $S_2[1..j]$ where the last operation is a deletion.

- Recurrence:

  $\widetilde{Edit}[i,j] = \min\{Edit_M[i,j], Edit_I[i,j], Edit_D[i,j]\}$

$$Edit_M[i,j] = \widetilde{Edit}[i-1,j-1] + d(S_1[i], S_2[j]);$$

$$Edit_I[i,j] = \min \begin{cases} Edit_M[i,j-1] + d_o + d_e, \\ Edit_I[i,j-1] + d_e, \\ Edit_D[i,j-1] + d_o + d_e \end{cases}$$

$$Edit_D[i,j] = \min \begin{cases} Edit_M[i-1,j] + d_o + d_e, \\ Edit_I[i-1,j] + d_o + d_e, \\ Edit_D[i-1,j] + d_e \end{cases}$$

  Output $\widetilde{Edit}[n,m]$!

- Base cases ???

- Running time? Space complexity?

Have you understood the lecture contents?

well	ok	not-at-all	topic
☐	☐	☐	sequence alignment
☐	☐	☐	edit distance
☐	☐	☐	DP for edit distance
☐	☐	☐	affine gap penalty scoring scheme
☐	☐	☐	edit distance with AGPSS
☐	☐	☐	DP for edit distance with AGPSS