Finding an optimal alignment

Recall Variant 2: not only $sim(s,t)$, but also an optimal alignment.

Backtrace in DP-table

- possibility 1: find correct path, redoing computation (more time)
- possibility 2: compute backtracing table during main algorithm (more space)

Analysis

- poss. 1: time: up to 3 operations per column of alignment computed, so $O(\text{length of alignment}) = O(n+m)$, or $O(n)$ if $n = m$; space: only additional space for the output alignment: $O(n+m)$
- poss. 2: time: one operation per column of alignment, so $O(n+m)$; space: additional $O(n \cdot m)$ space for matrix containing traceback pointers

Finding an optimal alignment

N.B.

1. Typically we want only **one** optimal alignment
2. Order of computation matters for output!

Re 1:
There could be an exponential number of optimal alignments, see $s = AAAA \cdots AAA = A^{2n}$, $t = A^n$, then every alignment of length $2n$ (i.e. aligning each character of t with some character of s, and aligning the remaining n characters of s with gaps) is optimal. But there are $\binom{2n}{n} \geq 2^n$ such alignments.

Algorithm *Backtracing in DP-table (without traceback pointers)*
Input: strings s, t with $|s| = n, |t| = m$; scoring function f; DP-table
Output: an optimal alignment $\mathcal{A}$ of $sim(s,t)$

$i \leftarrow n$; $j \leftarrow m$; $\mathcal{A} \leftarrow$ empty alignment;
while $(i > 0$ and $j > 0)$
  do if $D(i,j) = D(i-1,j) + g$
    then $\mathcal{A} \leftarrow \binom{s_i}{-}\mathcal{A}$;
      $i \leftarrow i-1$;
    else if $D(i,j) = D(i-1,j-1) + f(s_i,t_j)$
      then $\mathcal{A} \leftarrow \binom{s_i}{t_j}\mathcal{A}$;
        $i \leftarrow i-1; j \leftarrow j-1$;
      else $\mathcal{A} \leftarrow \binom{-}{t_j}\mathcal{A}$;
        $j \leftarrow j-1$;
if $i > 0$ **then** $\mathcal{A} \leftarrow \binom{s_1 \ldots s_i}{- \ldots -}\mathcal{A}$;
if $j > 0$ **then** $\mathcal{A} \leftarrow \binom{- \ldots -}{t_1 \ldots t_j}\mathcal{A}$;
return $\mathcal{A}$;

Space-saving variant

- For computing row i, we only need row $i-1$
- after having finished computing row i, we never need row $i-1$ again
- so we can overwrite row $i-1$ after having finished row i
- Altogether, at any given time, we only need the current row and the previous row.
- The same could be done with two columns instead of two rows.
- Space: $O(\min(n,m))$, for $n = m$: $O(n)$
- Time: $O(nm)$ (resp. $O(n^2)$), since we still need to compute all $(n+1)(m+1)$ entries
- This variant does not allow to compute an optimal alignment! (i.e. does not solve variant 2 of the problem)

Local alignment

Local alignment

- Often what we are interested in are so-called regions of high similarity in the two input strings, i.e. substrings which are similar, and not how similar the entire two strings are.
- So we want to find substrings s' of s, and t' of t s.t.

$$sim(s',t') = \max\{sim(u,v) \;:\; u \text{ substring of } s, v \text{ substring of } t\}.$$

- Typically here we also want to know all such pairs of substrings themselves and their alignment, not only their similarity value.

Smith-Waterman DP algorithm for local alignment

- Smith-Waterman DP-algorithm (1981).
- Algorithm similar to NW-algorithm for global alignment.
- Crucial points:
  1. for each pair of indices i,j, compute the highest score of an alignment of any substring u ending in position i of s with any substring v ending in position j of t
  2. the empty string is always a substring (in every position), and score of empty alignment $= 0$
  3. so all entries ≥ 0
  4. for the final output: find the maximum over all entries of the matrix
- Now we maximize like before and over 0:
  $L(i,j) = \max\{L(i-1,j)+g, L(i-1,j-1)+f(s_i,t_j), L(i,j-1)+g, 0\}$

Smith-Waterman DP algorithm for local alignment

Algorithm *DP algorithm for local alignment*
Input: strings s, t, with $|s| = n, |t| = m$; scoring function f
Output: value max

1. **for** $j = 0$ to m **do** $L(0, j) \leftarrow 0$;
2. **for** $i = 1$ to n **do** $L(i, 0) \leftarrow 0$;
3. **for** $i = 1$ to n **do**
4. **for** $j = 1$ to m **do**
5. $$L(i,j) \leftarrow \max \begin{cases} L(i-1,j) + g \\ L(i-1,j-1) + f(s_i, t_j) \\ L(i,j-1) + g \\ 0 \end{cases}$$
6. **return** $max = \max\{L(i,j) : 0 \leq i \leq n, 0 \leq j \leq m\}$;

Question: How do we compute *max* in line 6.?

Smith-Waterman DP algorithm for local alignment

Finding all optimal local alignments

- Find all occurrences of $\max\{L(i,j) : 0 \leq i \leq n, 0 \leq j \leq m\}$
- from each, backtrace until reaching a 0

Analysis

- $O(nm)$ time and space for computing matrix L
- $O(K)$ time for finding all optimal local alignments, where $K = \sum_{\mathcal{A} \text{ opt. local al.}} |\mathcal{A}|$ is the sum of the lengths of the optimal local alignments, i.e. the output size.