CSE 584A Class 16

Jeremy Buhler

March 23, 2016

More alignment basics

1 Cost of DP Alignment

- Note that global alignment algo between S, T with sizes m, n takes time $\Theta(mn)$.
- If $m = n$, time is $\Theta(n^2)$!
- Unfortunately, best algo known for this problem improves time to only $\Theta(n^2/\log n)$ (the "four Russians speedup"; Gusfield 12.7).
- (Aside: the "Four Russians" are Arlazarov, Dinic, Kronrod, and Faradzev. Only one is actually Russian.)
- What about space? More later!

2 Local vs Global Alignment

We've seen basic global alignment algorithm (Needleman-Wunsch).

- Global alignment is fine if it makes sense to align all of both sequences.
- However, more typical case seeks inexact *substring matches*.
- **Definition**: a *local alignment* of S with T aligns a substring from S to a substring from T.
- **Problem**: how to find optimal (highest-scoring) local alignment between two sequences?

Need modification to original DP recurrence.

- Previously, computed $M_{i,j}$, score of best alignment between sequences $S[1..i]$ and $T[1..j]$.
- Modification: compute $L_{i,j}$, score of best alignment between *any suffix* of $S[1..i]$ and *any suffix* of $T[1..j]$.
- Why? A local alignment ending at $S[i]$, $T[j]$ aligns some substring $S[i'..i]$ to some substring $T[j'..j]$. These are suffixes of $S[1..i]$, $T[1..j]$ respectively.

- How to modify recurrence? Now **four** possibilities for best alignment A ending at $S[i]$, $T[j]$.
- First three cases (A ends with gap in T, ends with gap in S, or ends by aligning $S[i]$ with $T[j]$) same as before.
- As before, last column of alignment is fixed, but can choose optimal (semi)local alignment for remaining columns.
- What is fourth case? A aligns *empty* suffixes of both $S[1..i]$ and $T[1..j]$. Score of empty alignment is 0.
- These ideas lead to new recurrence:

$$L_{i,j} = \max(L_{i-1,j-1} + \sigma(S[i], T[j]), L_{i-1,j} - g, L_{i,j-1} - g, 0).$$

- As before, $L_{0,0} = 0$, and disallow any illegal moves when filling in $L_{i,0}$ or $L_{0,j}$. (Implies these values should also be 0.)
- Want best alignment between *any* substrings, so must remember highest-scoring cell L_{i^*,j^*} over entire DP matrix.
- Optimal local alignment aligns suffix of $S[1..i^*]$ with suffix of $T[1..j^*]$.

This modified method is the *Smith-Waterman* algorithm, though some people use S-W to refer to both global and local alignment.

Example:

What about traceback?

- Can store back arrows as before, but what to do with Case 4 (empty alignment)?
- Case 4 implies that WLOG, can *cut off* backwards trace at any cell where best option is to use empty alignment.
- In this case, don't store any arrows.

3 Gaps: Making Life Difficult

From biologists' POV, our alignment algorithms have a rather serious problem.

- Consider these two alignments:

- Which alignment is evolutionarily easier to explain?
- First one requires many small indels, while second one requires only *one* multibase insertion/deletion.
- Most biologists would probably prefer the latter explanation, but our mutation model weights both alignments equally!
- Can we modify our DP algorithms to support a mutation model that includes *block* indels?
- The result of a block indel is more commonly referred to as a *gap*.
- For a gap of length k, our current scoring system assigns a penalty $-g \cdot k$, a so-called *linear gap penalty*.
- In principle, we could use *any* function $f(k)$ for gap penalties to reflect beliefs about relative likelihood of large *vs* small gaps.
- How should we choose $f(k)$?

A brief history of answers to this question...

- Most general answer: let $f(k)$ be *arbitrary* function of k. (maybe not even monotonic)
- Best DP alignment algorithm known for this scoring system requires time $\Theta(mn^2 + nm^2)$ for sequences of lengths m, n. (In other words, cubic time.)
- Easy but slow; see Gusfield 11.8.5.
- Suppose $-f(k)$ is *convex*; that is, $-\frac{d^2 f}{dk^2} < 0$. (Alternately, $-f(k)$ is "concave down").
- **Example**: $f(k) = -g - \log k$. "Gaps are bad, but marginal cost of making a long gap longer is near zero."
- Optimal alignment can be now found in time $O(nm \log \max(m, n))$.

- Algorithm is a bit hairy; see Gusfield 13. Example of a submodular optimization problem.
- Suppose $f(k)$ is *affine*; that is, $f(k) = -g_o - g_e \cdot k$. $-g_o$ is *gap opening penalty*, while $-g_e$ is *gap extension penalty*.
- Interpretation: longer gaps are still linearly worse, but there is a nonzero cost to making two small gaps instead of one big gap.
- We will show that optimal alignment with affine gap penalties can be found in time $\Theta(mn)$!

Affine gap model is standard for common sequence alignment tools like BLAST.

4 Extending Needleman-Wunsch to Support Affine Gaps

We'll look at global alignment; local version is very similar.

- How should we extend the DP recurrence for $M_{i,j}$, score of best alignment A between $S[1..i]$ and $T[1..j]$?
- As before, three cases.
  1. A ends by aligning $S[i]$ to $T[j]$. Best such alignment *still* has score $M_{i-1,j-1} + \sigma(S[i], T[j])$.
  2. A ends with a gap in sequence T.
  3. A ends with a gap in sequence S.
- **Problem**: aligning $S[i]$ or $T[j]$ to a gap character "$-$" has two possible costs, depending on whether we create a new gap or extend an existing one.

- **Solution**: let U_{ij} and V_{ij} be scores of the best alignment of $S[1..i]$ and $T[1..j]$ *that ends with a gap in S or T, respectively.*
- Given these quantities, we have

$$M_{i,j} = \max(U_{ij}, V_{ij}, M_{i-1,j-1} + \sigma(S[i], T[j])).$$

- How to compute V_{ij}? Two cases.

1. New gap at end of T is of length 1 (covers only $S[i]$). Cost is $-g_o - g_e$. Alignment prior to this gap may end with gap in either seq *or* match/mismatch. Best such alignment has score $M_{i-1,j}$.
2. Existing gap at end of T extended by 1 char. Extension costs $-g_e$. Alignment prior to last position must *also* align $S[i-1]$ to a gap in T. Best such alignment has score $V_{i-1,j}$.

- These two cases lead to recurrence for V values:

$$V_{ij} = \max(M_{i-1,j} - g_o - g_e, V_{i-1,j} - g_e).$$

- Similar recurrence holds for U values:

$$U_{ij} = \max(M_{i,j-1} - g_o - g_e, U_{i,j-1} - g_e).$$

- To initialize recurrences, $M_{0,0} = 0$, $U_{i,0} = V_{0,j} = -\infty$.
- DP computation similar to regular S-W, but we compute cells for three matrices U, V, M instead of one.

Example:

Easy to see that time cost remains $\Theta(|S||T|)$. Computing traceback is doable but slightly exciting (left for homework!).