CSE 584A Class 21

Jeremy Buhler

April 11, 2016

1 Eliminating Redundant Work in Tree Alignment

When can we stop downward DP in a tree?

- Neither local nor semi-local alignments are *a priori* anchored at the first row of the DP matrix (i.e. top of tree).
- Hence, it seems that we need to continue DP all the way to the bottom of the tree on every branch to find all alignments.
- (This would cost $\Theta(|P||T|^2)$ – blech!)
- But wait – a full computation repeats a lot of work!
- Example:

- An alignment starting at $T[i]$ will be found i times, one for each suffix of T starting at or before position i.

The following argument is key to removing redundancy.

- Suppose we are aligning P to the path in τ labeled with text suffix $T[h..n]$.
- Now suppose that an optimal alignment skips over some nonempty prefix $T[h..j]$ of this suffix before using any characters in P.
- The same alignment, at the same location in T, will also be discovered when we perform DP on the path in τ labeled with $T[j+1..n]$.
- Hence, WLOG, *we can always anchor* our alignments to the top of the tree, since paths for all suffixes of T begin there.

- Above argument applies to either local or semi-local alignment w/r to P.

How do we apply this observation in practice?

- *First step*: prevent skipping of any nonempty prefix of T before starting to align P.
- Initialize first column of DP matrix with $-\infty$ after first (initialization) row.
- This prevents us from finding the same alignment on several branches, but it doesn't seem to help us stop DP before the bottom of the tree.
- To win big, we need a stronger rule, as follows.
- Suppose we have found an DP alignment path Π that uses at least one character of T.
- Suppose further that Π can be replaced by another path Π', ending at the same cell as Π, that does *not* use any of T and has at least the same score as Π.
- Surely, any alignment path $\Pi \cdot \Phi$ can be replaced by $\Pi' \cdot \Phi$ with at least the same score, so Π' dominates Π.
- (something trivial to check for affine gap case)
- Moreover, Π' will be found by alignment on another branch of the tree that starts later in T.
- Conclude that we may *cut off extension of* Π because some other branch will find a better or equal alignment path through the same part of P and a later suffix of T.

We now have an X-drop-like rule for cutting off DP.

- Let's first consider fully local case.
- Suppose that, after the initialization row, we set a cell (i, j) in the DP matrix to 0.
- By our Smith-Waterman recurrence, any nonempty path ending at (i, j) (which must use some of T) is no better than the empty path (which does not use any of T).
- Hence, by our reasoning above, we can cut off extension from (i, j).
- As with X-dropping, we cut off extension by replacing the 0 in cell (i, j) with $-\infty$.
- The same hacks as for cutting off BLAST extension now tell us when we can safely stop aligning!
- A detailed version of the affine Smith-Waterman recurrence with this form of cutting off is used in the BWT-SW program, described in "Compressed indexing and local alignment of DNA," by Lam et al. (*Bioinformatics* 2008).

But what about the semi-local case?

- In semi-local alignment, it is not free to skip a prefix of P.

- In particular, an alignment that starts in column j of the DP matrix incurs a gap cost $c(j) = -g_o - j \cdot g_e$ (for the affine case).
- Now suppose that the score of a cell (i, j), after the initialization row, is $\sigma \leq c(j)$.
- We can replace any alignment path ending at this cell by a path of equal or better score that skips the first j characters of P, at cost $c(j)$, and uses none of T.
- By our argument above, some other path in the tree will find this better alignment.
- Hence, *in column j, we can replace any score $\leq c(j)$ after the initialization row by $-\infty$.*
- Again, usual cutoff tricks apply.

2 Further Pruning of Tree Alignment

So far, we have one provably correct criterion for cutting off alignments. Can we do even better?

- Suppose we seek alignments to a pattern $P[1..m]$ above some score threshold θ.
- Now suppose that, after using up characters $P[1..j]$ (or some suffix thereof for local alignment), we have a partial alignment path ending at cell (i, j) with score $\sigma < \theta$.
- Is it possible to reach score θ by aligning the rest of P to the subsequent chars of T?
- If not, we might as well cut off this path by setting (i, j) to $-\infty$.
- But how can we prove that we cannot reach θ without actually doing the rest of the alignment?
- *Idea*: if we can prove that no alignment of $P[j + 1..m]$ to the subsequent characters of T can exceed some score ψ, and $\sigma + \psi < \theta$, then we can never reach score θ.
- So, any heuristic that can overestimate (but *not* underestimate) the true remaining alignment score can be used for cutting off.
- (like the "admissible heuristic" criterion in state-space search)

Can we come up with some heuristics that don't require as much work as just completing the alignment?

- Here's a trivial one: compute the score $\rho(j)$ of aligning each character of $P[j..m]$ against itself.
- If the scoring system always scores matches higher than mismatches or gaps, then no alignment of $P[j..m]$ to T can score better than $\rho(j)$.
- This seems like a pretty weak heuristic.
- BWA uses a better approximation for the *special case* that the scoring function
  - assigns all matches the same bonus a,

  - assigns all edits (mismatches or gaps) the same linear penalty b

- *Idea*: compute the *matching statistics* of P vs T, and use them to upper-bound cost of matching P against an arbitrary substring of T.
- **Defn**: the *matching statistic* $\mu(j)$ is the length of the longest prefix match between $P[j..m]$ and T.
- We can compute $\mu(j)$ in time proportional to its value using a top-down tree walk to match $P[j..m]$ vs T.
- Now consider trying to match $P[j..j+\mu(j)]$ against T.
- By definition, this substring, which is one longer than $\mu(j)$, does not occur anywhere in T, so we must incur at least one edit in *any* alignment of it to T.
- Can recursively compute this cost for the rest of P.
- Here's BWA's edit-distance bounding algorithm.

```
EditBound(P, T)
  j ← 1
  d ← 0
  while j ≤ m do
    if j + μ(j) ≤ m
      d ← d + 1
    j ← j + μ(j) + 1
  return d
```

- To convert d to a cost bound, just compute $(m-d) \cdot a - d \cdot b$.
- We can precompute the cost bound for every suffix of P. For each suffix, we may do $O(m)$ work to compute matching stats.
- Hence, total cost is $O(m^2)$.
- However, if we do the equivalent computation starting from the *end* of P, we still get valid bounds (not necessarily the same ones), but we can reuse the matching stats computed for $P[k..m]$, $k > j$, when computing d for $P[j-1..m]$.
- Total cost for all suffixes of P is then only $O(m)$.
- BWA actually does the computation the other way around, i.e. bounding the number of edits to align each *prefix* of P, because it computes alignments of P to T starting from the end of P, using the BWT to simulate traversal backwards in T.

Coming up with better-than-trivial heuristics for cutting off DP with arbitrary match, substitution, and gap costs (as for protein alignment) is (AFAIK) an open problem.