

# CSE 584A Class 14

Jeremy Buhler

March 19, 2018

## 1 Mutation Happens

So far, we haven't even considered matching problems that aren't *exact* string matching.

- Why study inexact matching?
- Sequences mutate!
- DNA damage, replication errors, etc.
- pieces jump in and out (transposons, large duplication / deletion)
- In coding sequence, mutation changes protein product.
- In the lab, sequencing machines make (many) mistakes.
- In experimental assays, approximate matches sometimes matter.
- (**Example:** PCR primers, microarray hybridization)

## 2 Sequence Identity

What does “inexact matching” mean, anyway?

- Let  $S$  be “acgt”
- Which is more similar to  $S$ : “acga” or “acgc”? (does it matter which new base replaces the old one?)
- How about “actt” (which base changed)? How about “acagt”? “act”? (indels)
- *What is a sensible notion of similarity for biosequences?*

At this point, stop reading these notes and go look at my notes from BIO4342 for a biologist's perspective on what “similarity” means in the context of biological sequences. I'll assume that material in what follows.

- A couple of items not mentioned in the other notes...
- **Defn:** number of single-character mutations (insertion, deletion, substitution) required to create an optimal alignment of  $S$  with  $T$  is called the *Levenshtein distance* or *edit distance* between  $S$  and  $T$ .

- (And yes, it's a metric when substitutions and indels are permitted.)
- In practice, we use a somewhat more general measure of similarity than raw identity / edit distance.
- Set substitution scores  $\sigma(x, y)$  for all pairs  $x, y \in \Sigma$ .
- When  $x = y$ , score is positive; otherwise, it is usually negative.
- *Idea*: less common mutations get penalized more; matches are always preferred to mismatches
- **Example**: penalize transitions less than transversions in DNA. (DNA-PAM-TT matrices)
- Indels are assigned a *gap penalty*  $-g$ .
- Optimal alignment now *maximizes* score, rather than minimizing edit distance.
- (Simple equivalence: if  $M(S, T)$  is score of alignment for  $S, T$  of common length  $n$ , and  $\sigma(x, x) = a$  for all  $x$ , then  $D(S, T) = na - M(S, T)$  is a weighted edit distance.)
- For more on probabilistic models underlying this scoring system, read e.g. the book by Durbin and Eddy.

### 3 OK, Time for Algorithms

How do we compute the similarity score / edit distance of an optimal alignment of strings  $S, T$ ?

- Classic technique uses dynamic programming.
- Let  $M_{i,j}$  be score of optimal alignment between  $S[1..i]$  and  $T[1..j]$ .
- Since score of aligning anything to a gap is  $< 0$ , it is never optimal to align a gap to a gap.
- Hence, three possibilities:

1. Alignment ends by aligning  $S[i]$  with  $T[j]$ :

$$M_{i,j} = \text{best alignment score for } S[1..i-1], T[1..j-1] + \sigma(S[i], T[j])$$

2. Alignment leaves  $S[i]$  unaligned:

$$M_{i,j} = \text{best alignment score for } S[1..i-1], T[1..j] + -g$$

3. Alignment leaves  $T[j]$  unaligned:

$$M_{i,j} = \text{best alignment score for } S[1..i], T[1..j-1] + -g$$

- **Claim**: best of these three options is score of best alignment for  $S[1..i], T[1..j]$ .

- **Pf:** Observe that score of alignment can be written as score of its last posn plus score of rest.
- Suppose we had a better alignment  $A$  than any of the three given above. Divide  $A$  into its last position and the “rest” of the alignment  $A_0$ .
- The last position matches one of the above three cases; WLOG, suppose it is case 1.
- But then  $A_0$  must be better than the optimal alignment of the rest of the sequence for that case, which is impossible! QED
- Hence, we may write recurrence:

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

- By definition,  $M_{0,0} = 0$  (align two empty strings).
- Define  $M_{i,-1} = M_{-1,j} = -\infty$  to forbid these cases. (covers matrix boundaries)
- Want to compute  $M_{|S|,|T|}$ , best score of any alignment of the complete sequences  $S$  and  $T$ .

OK, time for the dreaded DP matrix.

- To avoid redundancy, we build up computation of  $M_{i,j}$  values from smallest to largest.
- Visualize as a DP matrix of size  $|S| + 1$  by  $|T| + 1$ .
- Indices run from  $0..|S|$  and  $0..|T|$
- Cell  $i, j$  holds value  $M_{i,j}$ .
- **Picture:**

- Starting from known  $M_{0,0}$ , fill in matrix in any consistent order (by row, by column, by antidiagonal, whatever)

- When you're done,  $M_{|S|,|T|}$  is value you want!
- **[fill it in here]**

One more step: how to get alignment itself, not just score?

- When computing  $M_{i,j}$ , remember which of three cases it came from. Did you add a gap in  $S$  or  $T$  or match  $S[i]$  with  $T[j]$ ?
- Store "back edges" in alignment matrix to keep this info (or reconstruct them in  $O(1)$  time per edge).
- When matrix is full, follow path of back edges from  $M_{|S|,|T|}$  until you reach  $M_{0,0}$ .
- (Could be more than one path)
- Any such path describes an optimal alignment; each edge gives one position (diagonal for match/mismatch, rectilinear for indel.)

**Example:**