CS 262 Lecture Notes

Ilan Goodman

January 8, 2015

1 Lecture 2 (January 8, 2015)

1.1 Evolution

- By 2020, the professor thinks the majority of the population will have their genome sequenced—yay data!
- Humans are roughly $98 - 99\%$ identical to chimpanzees, but there are certain genes where humans are closer to gorillas than chimpanzees or chimpanzees are closer to gorillas than humans
- Different lengths of lines on cladograms refer to substitutions per site
- On the order of 100 mutations from your parents (out of 3 billion)
- If we go back in time far enough, there was an individual who had DNA that split to eventually yield the DNA for each pair of individuals (farther back the less related two species are)
- Mean substitutions per site is influenced by frequency of generations and population size
- Because we have a long generation period and we used to have a small population size, humans are very similar to each other
- Most common type of mutation is point mutation—a single letter changes
- Ten times rarer is deletion or insertion of a few laters (frequency of such events decreases exponentially with the length of the insertion/deletion)
- Sometimes there are larger-scale rearrangements: inversions, translocation, and duplication is all possible—these are much, much rarer, usually on the scale of less than one per generation
- Some mutations (very few) are advantageous, many are neutral, and some are deleterious
- Between species, exons are much more similar than the rest of it
- As we move closer and closer in evolutionary distance, many of the other parts (the regulatory parts) stay more similar

1.2 Sequence Alignment

Definition Given two strings $x = x_1x_2 \dots x_M, y = y_1y_2 \dots y_N$, an *alignment* is an assignment of gaps to positions $0, \dots, M$ in x and $0, \dots, N$ in y so as to line up each letter in one sequence with either a letter or a gap in the other sequence.

- There are often multiple ways to align two sequences
  - Which is better: 6 matches, 3 matches, 1 gap, 7, 1, and 3, or 7, 0, and 5 (respectively)?
  - We'll determine a scoring function based on sequence edits (mutations, insertions, and deletions)

- Matches get a reward $+m$, mismatches get a penalty $-s$, and gaps get a penalty $-d$
- (Score) $F = m \cdot (\#\text{ matches}) - s \cdot (\#\text{ mismatches}) - d \cdot (\#\text{ gaps})$
- Minimum edit distance

- There are too many possible alignments to try everying out, so we use dynamic programming to search through all the alignments efficiently

- Alignment is additive
  - The score of aligning $x_1 \dots x_M$ and $y_1 \dots y_N$ is additive
  - Say that $x_1 \dots x_i$ aligns to $y_1 \dots y_j$ and $x_{i+1} \dots x_M$ aligns to $y_{j+1} \dots y_N$
  - Then the two scores add up: $F(x[1:M], y[1:N]) = F(x[1:i], y[1:j]) + F(x[i{+}1:M], y[j{+}1:N])$

- Dynamic programming
  - There are only a polynomial number of subproblems (align $x_1 \dots x_i$ to $y_1 \dots y_j$)
  - The original problem is one of the subproblems (align $x_1 \dots x_M$ to $y_1 \dots y_N$)
  - Each subproblem is easily solved from smaller subproblems
  - If the above conditions are satisfied, then we can use dynamic programming
  - F is the DP matrix or table (similar to memoization)

- Let $F(i, j)$ be the optimal score of aligning $x_1 \dots x_i$ and $y_1 \dots y_j$. Notice that there are three possible cases, if we have already computed the smaller subscores:

  1. x_i aligns to y_j. Then $F(i, j) = F(i-1, j-1) + \begin{cases} m, & x_i = y_j \\ -s, & \text{otherwise} \end{cases}$.
  2. x_i aligns to a gap. Then $F(i, j) = F(i-1, j) - d$.
  3. y_j aligns to a gap. Then $F(i, j) = F(i, j-1) - d$.

- The only problem now is to figure out which case is correct

- Our *inductive assumption* is that $F(i, j-1), F(i-1, j), F(i-1, j-1)$ are all optimal. Then

$$F(i, j) = \max\{F(i-1, j-1) + s(x_i, y_j), F(i-1, j) - d, F(i, j-1) - d\},$$

  where $s(x_i, y_j) = m$ if $x_i = y_j$ and $s(x_i, y_j) = -s$ otherwise.

- It is easy to fill in the first row and first column of the matrix once we set m, s, and d (usually take $m = s = d = 1$). From there, we follow our DP algorithm to fill in the rest of the matrix as desired. We also put in backpointers to the cell that leads to the optimal alignment. At the end, to output the optimal alignment, we simply follow the backpointers from the bottom right square to the top:
  - When the backpointer is diagonal, we output x_i, y_j (these two letters are aligned)
  - When the backpointer is up, we output y_j (we have a gap in x)
  - When the backpointer is to the left, we output x_i (we have a gap in y)

- The Needleman-Wunsch Matrix: every nondecreasing path from $(0, 0)$ to (M, N) corresponds to an alignment of the two sequences and an optimal alignment is composed of optimal subalignments

- This algorithm takes $O(NM)$ time and $O(NM)$ space

- Note that it is easy to parallelize filling in the matrix diagonally (a line from upper right to lower left all at the same time) because each cell only depends on cells up and to the left of it

- There are several variants of this algorithm that come in handy:

- Perhaps we would like to allow an unlimited number of gaps in the beginning and the end of the sequence—then we shouldn't penalize gaps on either end (overlap detection)
  - Example: two overlapping "reads" from a sequencing project
  - Example: search for a mouse gene within a human chromosome
  - To initialize we set $F(i,0) = 0$ and $F(0,j) = 0$ for all i, j and we terminate by taking the max along every square on the bottom and the right side of the table
- The local alignment problem: given two strings $x = x_1 \dots x_M, y = y_1 \dots y_N$, find substrings x', y' whose similarity (optimal global alignment value) is maximum
  - Intuitively, this means we can find a path from anywhere in the matrix to anywhere in the matrix
  - We care about this because genes are shuffled between genomes, so this allows us to look for genes similar across species
  - The Smith-Waterman algorithm: initialize the top and left sides to 0 and in the max (the update step), also max with 0, so no square can be negative (because we want to be able to start at any place)
  - At the end, we look for the maximum value in the entire matrix and trace back until we find a place where the score is 0
  - If we want all local alignments scoring $> t$, we often run into problems with overlapping local alignments: Waterman-Eggert '87 finds all non-overlapping local alignments with minimal recalculation of the DP matrix
- Scoring the gaps more accurately
  - Currently, a gap of length n incurs a penalty of nd
  - However, a gap of length 2 is more likely than two gaps of length 1 and a gap of length 3 is more likely than three gaps of length 1, but our current score function scores these identically
  - Since gaps usually occur in bunches, we can define a concave penalty function $\gamma(n)$ (i.e., convex $-\gamma(n)$): for all n, $\gamma(n+1) - \gamma(n) \leq \gamma(n) - \gamma(n-1)$
  - Now our iteration step of our DP algorithm becomes $F(i,j) = \max\{F(i-1,j-1) + s(x_i, y_j), \max_{k=0,\dots,i-1} F(k,j) - \gamma(i-k), \max_{k=0,\dots,j-1} F(i,k) - \gamma(j-k)\}$
  - Running time is now $O(N^2M)$ (if $N \geq M$), while space is $O(NM)$
  - In practice, this algorithm is never used