

Class 08: Gene Alignments (3)

Held: Tuesday, 22 September 2009

Summary: We consider the Needleman-Wunsch algorithm for optimal alignments.

Related Pages:

- EBoard.
- Due: On your own project (2.6).

Notes:

- It sounds like there was some confusion about the on-your-own project. It was given in section 2.6 of the text. We'll talk about it for a moment.
- We'll be working on Project 3.5 for much of today's class.

Overview:

- Plan of Action.
- Needleman-Wunsch.
- An Example.
- Lab.

Approximate Plan of Action

So, where are we going with the topics for chapter 3? Here's our approximate plan of action.

- Today:
 - Go over the Needleman-Wunsch algorithm
 - Discuss the next "On your own" assignment (based on this algorithm and on the BLAST stuff we did last week)
 - Start the Needleman-Wunsch lab
- Thursday:
 - Continue working on the Needleman-Wunsch lab
 - Start working on the "On your own" assignment
- Next Tuesday:
 - "On your own" assignment due
 - Start chapter 4
- Next Thursday:
 - Discuss a paper based on the "On your own" assignment

The Needleman-Wunsch Algorithm

- Problem domain: Finding the best match between two strings by aligning individual components.
 - With particular costs for insertion, deletion, and mutation
 - Basic value function:
 - Insertion has a value of -1
 - Deletion has a value of -1
 - Mutation has a value of 0
 - Exact match has a value of 1
 - We can use others
- Input to algorithm
 - Sequence 1 (which we might call the *database sequence*)
 - Sequence 2 (which we might call the *search sequence*)
 - The value function (which we'll leave implicit)
- Basic idea: Recursion
 - Given two sequences, simplify the sequences, find the best matches of the simplified sequences, and extend to the current sequences.
 - We'll work from right to left.
- Very general example (not worked all the way through)
 - Database: CACGTAT
 - Search: CGCA
- Three ways to align
 - We can pair the last value in each sequence and then align the rest of each sequence.
 - Pair T with A
 - Align CACGTA to CGC.
 - We can drop the last value in sequence 1 (corresponding to a deletion).
 - Accumlate a penalty for the deletion
 - Align CACGTA to CGCA.
 - We can drop the last value in sequence 2 (corresponding to an insertion).
 - Accumlate a penalty for the insertion
 - Align CACGTAT to CGC
- Which do we choose?
 - *Whichever gives us the highest value alignment!*
- In pseudo code:

```
def valueOfBestAlignment(seq1, seq2):  
    valMat = valueOfMatch(last(seq1), last(seq2)) +  
             valueOfBestAlignment(dropLast(seq1), dropLast(seq2))  
    valDel = deletionCost +  
             valueOfBestAlignment(dropLast(seq1), seq2)  
    valIns = insertionCost +  
             valueOfBestAlignment(seq, dropLast(seq2))  
    return max(valMat, valDel, valIns)
```

- Of course, we should handle the cases in which one of the sequences is empty.

```

if seq1.isEmpty()
    return len(seq2) * values.insertion
elif seq2.isEmpty()
    return len(seq1) * values.deletion
else
    ...

```

- But that much recursion is expensive!

Dynamic Programming

- Solution: Build a table of the values
 - The value at column c , row r is the value of the best match between the first c values in sequence 1 and the first r values in sequence 2
 - The idea of putting all the values for intermediate computations in the table is called *dynamic programming*
- Row 0 of the table shows the cost of matching the empty string against the database string.
- Column 0 of the table shows the cost of matching the search string against the empty string.

Beyond Scoring

- But we care about more than the score of an alignment.
- We also care about the way we get that alignment.
- The book suggests that we build a table that describes how to get the alignment after we've built the scoring table.
- Traditionally, we build the alignment table at the same time as we build the scoring table.
- We then read the alignment strategy off of the alignment table, starting at the lower-right corner.

An Example

- We'll work on an example together.
- We'll use the same example as St. Clair and Visick, partially because it makes it easier to check our work.
- Sequence 1 = CACGTAT
- Sequence 2 = CGCA

Lab

- Project 3.5

Copyright © 2009 Vida Praitis and Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.