Madeleine Fenner

Computational Biology 131

Professor Anna Ritz

14 May 2020

Modifying the Local Alignment Algorithm to Return Multiple Alignments

Comparing sequences of DNA is often done to find regions that are similar, and locating conserved subsequences is a process that is relevant for many biological questions. An example of where this could be applied is in better understanding the relationship between two organisms by identifying genetic overlaps. Global alignment is an algorithm that can align the entirety of two strings of DNA in the most optimal way. Local alignment uses similar mechanisms but can be applied to find a more specific single internal alignment between two sequences, which is useful for scenarios where well conserved areas may be embedded in sections of sequence that are very different. Although these algorithms can be useful in certain situations, often there is not just one best local alignment between strings of DNA but multiple that are conserved to varying degrees. In order to capture multiple alignments, the local alignment algorithm can be altered so that when given two strings of DNA, scores for indels, mismatches, matches, and k (number of alignments to find), it will return k local alignments within the strings and an alignment score for each.

In order to do this, I modified the algorithm so that as the alignments are being created by backtracking, the letters used in the alignment are "masked" out from the alignment table and the backtrack table by altering the corresponding row or column for the letter. For the alignment table, a low number is substituted for the numbers within the row or column so that the algorithm will not pick any location within the previously used subsequence to form the next alignment,

and in this way avoids overlaps. For the backtrack table, the corresponding row or column is filled with a different symbol so that the alignment stops when it hits the symbol, avoiding overlaps from alignments started outside of the region as well. In addition, the upper right and lower left quadrants of both tables must be masked to avoid aligning multiple subsequences that are not arranged linearly and therefore could not exist together.

The masking process is shown here for the alignment of TTTGGCGAAA with TTTACCTAAA and with score inputs of indel = -2, match = 1, mismatch = -2 (using scores that will return highly conserved alignments to show the example on a small scale). When TTT is aligned with TTT, the corresponding columns and rows are altered.

For the alignment table, the values are changed to a low number, in this case -1:

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 2, 2, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 2, 3, 1, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1]
[0, 0, 0, 0, 1, 0, 0, 0, 1, 2, 2]
[0, 0, 0, 0, 1, 0, 0, 0, 1, 2, 3]
```

```
[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
[-1, -1, -1, -1, 1, 0, 0, 0, 0, 0, 0]
[-1, -1, -1, -1, 0, 0, 0, 0, 0, 0, 0]
[-1, -1, -1, -1, 0, 1, 1, 0, 0, 0, 0]
[-1, -1, -1, -1, 0, 0, 0, 0, 0, 0, 0]
[-1, -1, -1, -1, 1, 0, 0, 0, 1, 1, 1]
[-1, -1, -1, -1, 1, 0, 0, 0, 1, 2, 2]
[-1, -1, -1, -1, 1, 0, 0, 0, 1, 2, 3]
```

For the backtracking table, the symbol ‘x’ is substituted in:

```
['-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-']
['-', 'd', 'd', 'd', '-', '-', '-', 'd', '-', '-', '-']
['-', 'd', 'd', 'd', 'e', '-', '-', 'd', '-', '-', '-']
['-', 'd', 'd', 'd', 'e', '-', '-', 'd', '-', '-', '-']
['-', '-', 's', 's', 'd', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', '-', 'd', 'd', '-', '-', '-', '-']
['-', '-', '-', '-', '-', '-', '-', '-', '-', '-', '-']
['-', '-', '-', '-', 'd', '-', '-', '-', 'd', 'd', 'd']
['-', '-', '-', '-', 'd', '-', '-', '-', 'd', 'd', 'd']
['-', '-', '-', '-', 'd', '-', '-', '-', 'd', 'd', 'd']
```

```
['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x']
['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x']
['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x']
['x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x', 'x']
['x', 'x', 'x', 'x', 'd', '-', '-', '-', '-', '-', '-']
['x', 'x', 'x', 'x', '-', '-', '-', '-', '-', '-', '-']
['x', 'x', 'x', 'x', '-', 'd', 'd', '-', '-', '-', '-']
['x', 'x', 'x', 'x', '-', '-', '-', '-', '-', '-', '-']
['x', 'x', 'x', 'x', 'd', '-', '-', '-', 'd', 'd', 'd']
['x', 'x', 'x', 'x', 'd', '-', '-', '-', 'd', 'd', 'd']
['x', 'x', 'x', 'x', 'd', '-', '-', '-', 'd', 'd', 'd']
```

Once the tables are masked accordingly, the same process can be repeated, each time using the updated tables to find the next alignment. In this way, multiple alignments of the same two strings can be found. If this loop is run indefinitely, eventually there will be no more alignments left as each section is used in a previous alignment.

With this design, the algorithm will never return alignments that overlap. For some biological questions, this may be a helpful feature. However, it must be noted that this algorithm prioritizes highest scored alignments. If there was a more biologically relevant alignment, but it had a lower alignment score, it could be masked in this process. In this scenario, modifying an algorithm to return every possible alignment within a certain score range may be more useful. Instead, this algorithm would be most relevant for preliminary comparisons of sequences that are intended for finding general areas of similarity, not any specific subsequence. Additionally, the function of this new algorithm has only been tested on a small scale and with formulated strings that do not have a directly biological basis, but would presumably still capture the function.

Replit link to algorithm: https://repl.it/join/pstudanl-maddyfenner

(I am okay with this being posted online)