Marcus Bamberger

Bio 131 Final Project

The purpose of this project was to construct a function to determine global alignment with affine gaps. When comparing a raw section of DNA to one that has been spliced to remove introns and exons, it is necessary to recognize that the correct alignment minimizes the number of gaps, not the number of individual deletions. This program incorporates that prioritization into its alignment-finding.

This program, gapAlign, used toy sequences constructed by hand and data obtained from the UCSC genome browser of human chromosome 1. In order to test the influence of the affine gap prioritization, I tested this program against a more basic alignment-finding function from lab six with the same sequences. The toy sequences used were generally a repeated motif or motifs, such as GCAT, repeated with no intervening base pairs in one string – the "short sequence" and junk, random base pairs in the second – the "full sequence". In this way, I tested whether the alignment would minimize gaps and match motif to motif or align with no regard for the number of gaps. The UCSC genome sequences used were trimmed to a usable length, as aligning an entire chromosome was extremely impractical. One sequence, the "full sequence", was prepared by capitalizing all letters in the sequence, eliminating the distinction between introns and exons. The second sequence, the "short sequence", was prepared by removing the introns entirely, creating the same setup as the toy sequences with authentic data.

gapAlign takes as input two strings, to compute an alignment between, and several values, which determine how it scores the alignments. The values are for opening a gap, continuing a gap, a mismatch, and a match. Different values provide different optimal

alignments. For the purposes of affine gaps, I found setting gapContinue to -0.1, gapStart to -1, match to 1 and mismatch to -1 provided desirable results.

gapAlign begins by constructing six empty tables. Three will contain scores and three will contain backtrack instructions. The central table concerns itself with matches, mismatches, and opening gaps. The upper and lower tables concern themselves with continuing and closing gaps in the first and second strings. The tables are then filled in. Each coordinate is filled for all six tables before the next coordinate is started, working in top-bottom right-left diagonals, to prevent any referencing of unfilled table coordinates.

Once the backtrack tables are complete, the alignment is constructed by working backwards through the backtrack tables, building the alignment in reverse as it goes. Finally, the completed alignment is returned and the score of the alignment is given.

For the toy sequences GCATGCATGCATGCAT and

CTCGAGTCTAGAGCATCTCGAGTCTAGAGCATCTCGAGTCTAGAGCATCTCGAGTCTAGAGCATCTCGAGTCTAGA

my function returned the following alignment:

CTCGAGTCTAGAGCATTCGAGTCTAGAAGCATTCGAGTCTAGAAGCATTCGAGTCTAGAAGCATTCGAGTCTAGAA
----------GCAT----------GCAT----------GCAT----------GCAT----------

(Note that the number of indels in the second sequence does not match the number of junk characters in the first sequence; in order to more accurately represent the alignment visually, the lengths of the gaps have been extended to show which base pairs were matched.)

When the same two sequences were given to the program I constructed for lab 6.1, the following alignment was returned:

CTCGAGTCTAGAGCATTCGAGTCTAGAAGCATTCGAGTCTAGAAGCATTCGAGTCTAGAAGCATTCGAGTCTAGAA

----------G---------C---A------T------G----------C---A------T------G---------C---A------T------G----------C--A-------T-----

This indicates that my program is a more accurate tool for determining alignments between two sequences, as it works to minimize the number of gaps opened and finds clustered

alignments. This is valuable for locating smaller repeated motifs concealed within larger sequences.

The scoring system used to find this particular alignment was as follows: Opening a gap and mismatching two base pairs were both penalized by 1 point. Matching two base pairs rewarded one point. Continuing an already open gap penalized the score by -0.1 points. The exact values used here are likely to be dependent on the size of the alignment that is being computed and the length of the junk sequences separating the motifs. Past a certain length of intron, the program will determine that it is better to find a closer alignment with less gaps, even if this increases the number of mismatches. Consider this alignment of the same toy sequence, increasing the penalty of continuing a gap to -1, and opening a gap to -5:

CTCGAGTCTAGAGCATTCGAGTCTAGAAGCATTCGAGTCTAGAAGCATTCGAGTCTAGAAGCATTCGAGTCTAGAA

----------GCATGCATGCATGCAT

Here, despite the mismatch penalty, the gap penalty is too unfavorable, and the program opts for an alignment without any gaps at all. A similar effect will occur if the introns are extended to several times their current length. When computing alignments with real data, the length of the introns required smaller penalties for continuing the sequence to prevent the grouping problem described above - -0.01 or even -0.001. Setting the penalty for continuing a gap to zero would solve this problem, but it would also cause the program to create infinitely long gaps at no penalty, potentially crossing much of the chromosome to find a better match. The optimal gap-continuing penalty will vary based on the length of the sequences being tested and the length of the introns or junk base pairs between motifs, and there is no one good fit that will find accurate alignments for all sequences.