A tool for global pairwise alignment of DNA sequences using dynamic programming approach

BIOI2250 Introduction to Programming
December 17, 16

Reeta Rahkola (#516067, remarah@utu.fi)
Anna Lipsanen (#515668, annacu@utu.fi)
Niina Pitkänen (#48363, nisusi@utu.fi)

Table of contents

1. General description of the problem

Sequence alignment (DNA, RNA and protein) is one of the most commonly used procedures in Bioinformatics. Sequence alignment provides information about conserved and variable regions within the sequences, which in turn allows for identification of structural and functional motifs that could be used to identify phylogenic relationships or predict the potential function of an unknown gene based on similarities identified when its sequence is aligned with that of a reference genome.

Given rapid advancements in DNA sequencing methods and technology, the success of the human genome project and increased affordability enabling research labs to utilize genome wide sequencing technology, there has been a substantial increase in the amount of sequencing data requiring analysis. However, as sequence data grows exponentially, so has the need for computational algorithms that can examine fragments of DNA sequences and align them accordingly.

The goal of the pairwise sequence alignment algorithm is to produce the best possible alignment of two homologous sequences. In order to identify the best possible alignment, a scoring system is necessary. In DNA sequence alignment, the scoring system is defined by assigning a separate score for each aligned matching nucleotide pair and a penalization score for mismatches and gaps. Global pairwise alignment of two sequences is performed by either aligning each successive nucleotide or by introducing gaps where necessary across the length of one or both sequences in order to maximize matching positions.

2. Problem analysis

The best possible alignment is the one with the highest score. Since the number of possible alignments increases factorially with the lengths of the sequences, it is not possible to evaluate every single potential alignment. Instead, algorithms using dynamic programming are commonly used to find the optimal solution.

The basic concept of dynamic programming algorithms is that they solve a given complex problem by dividing it into small sub-problems that are fast to solve. Each solution to a sub-problem is stored and used later in order to achieve a solution for the original problem. When such algorithms are applied to sequence alignments, the complex problem of aligning the whole sequences is divided into subproblems of only aligning any two residues of the sequences. Each of the subproblems is solved by calculating which of the following moves will yield the highest score: aligning either of the residues with a gap or aligning the residues with each other. The solution of each subproblem thus contains the highest score obtained and the move made to reach the highest score. The subproblems are solved in a specified order and when combined together, can be used to find the solution for the original problem (i.e. finding the highest-scoring alignment(s) when the sequences are aligned on their whole length). Although the dynamic programming method for aligning sequences is slow, especially for long sequences, it is guaranteed to give the optimal alignment under the given scoring scheme. Hence, sequence alignment via dynamic programming is the recommended approach to take when only a few sequences need to be aligned and obtaining high accuracy is of paramount importance.

3. The structure of the solution

Overall, the solution to the alignment scoring was solved by providing the user with an opportunity to adjust the parameters by entering values for scoring matches and penalty score that is used for mismatched nucleotides or gaps inserted into the aligned sequence. Alternatively, the user may select to utilize the default parameters as set out by this program (i.e. match = 1, mismatch = -1, gap penalty = -1). Irrespective of choice (default settings or user defined settings), this program uses a 'linear gap penalty approach,' whereby the same penalty score is applied to every inserted gap. In this approach, all gaps are penalised by the same score regardless of whether they 'open' a gap or merely extend an existing gap.

The sequence alignment problem is solved using the Needleman-Wunsch algorithm, first published in 1970 [1], which is still widely used in bioinformatics. The alignment score of two sequences is computed via the application of the dynamic programming approach described previously.

To make the program code easier to follow, subprograms are clustered under the headings: 'Main program,' 'Sequences,' 'Parameters,' 'Initialization Score Matrix,' 'Global Pairwise Alignment Score Matrix' and 'Results.' These were created specifically to divide the code into sections based on function of the subprograms contained within and are only used for visualization purposes. Hence, they have no bearing on the program code and are ignored by Python when the program is executed.

Main Program: When the program is executed, a 'welcome' notation is displayed for the user before they are prompted to provide the nucleotide sequences that they wish to have aligned. A while loop is used (one for each sequence entered), which calls on the subprogram, askSeq(), listed under the heading 'Sequences'. When the conditions of the while loop are met, the user is notified of the default score settings. The default scoring system is stored within the program as a dictionary called 'defScores', where nucleotide pairs are 'keys' and the corresponding match or mismatch

score is the ‘value’. The user is prompted to respond, this time, in order to decide whether or not they wish to use the default settings as defined by the program or define their own score parameters.

Since there may be the potential for the user to make an error when responding to this prompt, a while loop was introduced to check their response against to two variables called ‘yes’ and ‘no’ which consist of all possible, acceptable responses that the user could make. In this case, as we only need to determine if the answer given by the user already exists within our lists of acceptable answers and hence, the variables ‘yes’ and ‘no’ were created as sets. Should the user not give an acceptable answer, they are notified and subsequently prompted with the same question once again until such time that an acceptable answer is provided by the user.

If the user selects to utilize default settings, the subprogram initializeScoreMatrix(seq1, seq2, gp), listed under headings ‘Initialization Score Matrix’, is called upon using the given sequences and default gap penalty as arguments. This subprogram returns a matrix that can be used to calculate the highest score for the global alignment. Then, calculateScoreMatrix(matrix, dic, gp), listed under heading ‘Global Pairwise Alignment Score Matrix’, is called upon with the matrix created in the previous step and default scores. This subprograms return the final score matrix for the user input sequences. However, should the user wish to define their own scoring parameters, then the subprogram defineSettings(dic), listed under heading ‘Parameters’ is called upon in order to create a user defined scoring system. The scoring system consists of a dictionary with user defined values for match and mismatch and a score for gap penalty. The dictionary is created by the subprogram defineSettings(dic) by assigning scores given by the user to the keys found in the ‘defScores’ dictionary. This user specified dictionary and gap penalty are then given as arguments to the initializeScoreMatrix(seq1, seq2, gp) and the calculateScoreMatrix(matrix, dic, gp) subprograms to calculate the final score matrix for the user input sequences. At the completion of the program, a text file is created using the subprogram outputResults(matrix,

seq1, seq2, gp), found under heading: ‘Results’. Following which, the user is given an opportunity to align new sequences or quit the program.

As mentioned above, this module contains the following subprograms:

1. def askSeq() – Checks the nucleotide sequences
2. def defineSettings(dic) – Returns the user defined scores and gap penalty
3. def initializeScoreMatrix(seq1, seq2, gp) – Returns the substitution score matrix
4. def calculateScoreMatrix(matrix, dic, gp) – Returns the alignment score matrix
5. def outputResults(matrix, seq1, seq2, gp) – Creates a txt file with results

4. Detailed description of subprograms

4.1 def askSeq()

This subprogram is a function that queries the user for a DNA sequence that should consist only of nucleotide characters: 'A', 'T', 'C' and 'G'. In order to ensure the input sequence is acceptable, the input sequence is tested to ensure it is at least one character long and that it does not contain any non-nucleotide characters.

The function does not have parameters.

If the sequence that the user enters passes the set criteria (i.e. it is long enough and only containing the nucleotide characters mentioned above), then the function returns the input sequence in lowercase letter format as a string. In all other cases, the function returns "True". This makes it possible to call the function inside a while loop as described in section 3 of this report. In such a case, the while loop is not broken until the sequence given by the user is in the correct format.

4.2 def defineSettings(dic)

This subprogram is a function that queries the user for substitution scores and gap penalty. It then creates a dictionary that contains the given substitution scores as value that are paired with the corresponding, correct keys.

The function takes one parameter: dic. This parameter specifies a dictionary from which the names of the keys are retrieved. For each key found in dic, a corresponding key with user specified value is stored in the new dictionary. For example, a match key found in dic (ie. Keys: 'aa', 'tt', 'cc', 'gg') are paired

with the integer value the user defined for ‘match score’, whilst all other possible nucleotide combinations (eg. Keys: ‘at’, ‘tg’, ‘cg’, ‘ga’ and so forth) are matched with the integer value the user defined for ‘mismatch score’.

The function subsequently returns the dictionary that was created and the value assigned by the user to apply to gap penalties.

4.3 def initializeScoreMatrix(seq1, seq2, gp)

This subprogram is a function for initializing the score matrix. The score matrix is a list that contains lists as its items. The length of the list equals the number of rows in the matrix and the number of items in each nested list equals the number of columns in the matrix.

The function takes three parameters: seq1, seq2 and gp. Seq1 and seq2 refer to the DNA sequences input by the user and gp refers to the chosen gap penalty. Seq1 and seq2 are strings while gap penalty is an integer or a floating point number.

The function creates a matrix whose dimensions depend on the lengths of the input DNA sequences: the matrix will have two more rows than there are characters in seq1 and two more columns than there are characters in seq2. A hyphen is stored into all cells in the matrix at this point. The first row and the first column of the matrix are then filled with characters of seq1 and seq2, respectively. Filling is started from the end of the row and from the bottom of the column.

Then, as a starting point for calculating the alignment score the second column in the second row, is filled with zero. Starting from this cell and moving towards the right, a score is calculated for each cell in the row. With each move rightwards, one gap penalty is added to the score that is stored in the previous

cell. Similarly, a score is calculated for each cell in the second column starting from the same cell as above. With each move downwards, one gap penalty is added to the score that is found in the previous cell. In each of these cells for which the score was calculated, the score and the move made to reach the score are stored as a tuple.

The function then returns the matrix.

4.4 def calculateScoreMatrix(matrix, dic, gp)

This subprogram is a function that creates the final score matrix for global pairwise alignment.

The function takes three parameters: matrix, dic and gp. In the matrix, the first two rows and the first two columns must contain items of correct type (a tuple or a number) or correct meaning (depends on the argument dic). It is advisable that the matrix used as an argument is the matrix obtained from the initializeScoreMatrix function. Dic is a dictionary that contains numbers as values. The contents of the matrix must match with the keys in the dictionary so that the data stored in the matrix can be used to obtain the keys. The gp refers to gap penalty and is an integer or a floating point number.

The function calculates the highest score for each cell in the matrix using the chosen substitution scores and gap penalty. Starting from the third row and the third column and iterating over the matrix row by row, column by column, the function calculates the scores as follows: For each cell, three scores are calculated from existing scores in the upper, left and top-left (diagonal) cells. When score is calculated from the upper cell or from the cell on the left, gap penalty is added to the existing score. When score is calculated from the diagonal cell, correct substitution score is added to the existing score. Substitution scores are read from the dictionary. The key for the dictionary is obtained by

combining the nucleotides from the first row and first column that correspond to the cell in question. For each position, the maximum score is chosen. The score and the move made to reach the score are stored as a tuple in the cell.

The function then returns the matrix.

4.5 def outputResults(matrix, seq1, seq2, gp)

This subprogram produces a text-format results-file that includes:

- Date and time the analysis was run
- The two sequences entered by the user
- The penalty score
- The alignment score of the best alignment found
- The final score matrix for global pairwise alignment

The function takes four parameters: matrix, seq1, seq2 and gp. In the main program this function is called using the matrix produced by calculateScoreMatrix-function (matrix), seq1 and seq2 (entered by the user) and either default or user given gap penalty (gp) as input. The outputResults-function selects lowest and the rightmost cell of the final score matrix and saves the value as “score” variable. This is the final score of the best alignment found by the program. Next the function creates a text file named “results” in which it writes the results.

The text file gives the user the final score matrix for global pairwise alignment of the given sequences. This matrix can be easily used to identify the best possible alignment(s) by starting from the low right corner and backtracking left/up/diagonal direction indicated by the letter in that cell (L/U/D). In cases where the cell includes more than one letter, several equally good paths to follow, and thus several equally good alignments, exist and each one should be considered.

5. References

1. Needleman, Saul B. & Wunsch, Christian D. (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". *Journal of Molecular Biology*. **48** (3): 443–53. doi:10.1016/0022-2836(70)90057-4. PMID 5420325.