Homework 1

Computational Biology course
Handed out: 28.09.2020
Due date: 12.10.2020

Theory questions

We encourage you to answer these questions only after you have completed the programming task. Please keep your answers short, most of the questions can be answered in one or two sentences. Submit your answers in a pdf file named following the format Lastname_Firstname_HW1.pdf.

1. Why are gaps generally more penalized than mismatches?
2. Why does it make sense to disallow gaps at the start and end of a local alignment, considering the -2 score for a gap?
3. What is a potential problem that could arise when trying to locally align a very short sequence to a much longer one?
4. There are $\sum_{k=0}^{n} \binom{m+k}{k}\binom{m}{m+k-n}$ possible alignments for two sequences of lengths m and n, $m \geq n$. For sequences of lengths $m = n = 100$, this amounts to 2.05×10^{75} possible alignments. Give a rough estimate — order of magnitude only — for the number of steps needed to align two sequences of this length using the Needleman-Wunsch algorithm. (Hint: consider the maximum number of iterations of the loops in the pseudocode.) For sequences of lengths $m = n = 100$, roughly how many times less than exhaustive search is this?
5. How could you extend the Needleman-Wunsch algorithm to k sequences, $k > 2$?

Programming task

Submit your solution in a R file named following the format Lastname_Firstname.R.

Task description

In this homework you should write a script that performs pairwise sequence alignment. Your script should be able to switch between the Smith-Waterman algorithm for local alignment and the Needleman-Wunsch algorithm for global alignment. As input your code should take:

1. `seqA` and `seqB`: two sequences of *arbitrary* (not necessarily the same) length, e.g. `"TCACACTAC"` and `"AGCACAC"`;
2. `local`: a switch indicator, showing which of the local/global alignment algorithms you should run (when `local = TRUE` use Smith-Waterman and when `local = FALSE` use Needleman-Wunsch);
3. the alignment scoring scheme:
   (a) `score_gap`: score value for a gap, e.g. -2;
   (b) `score_match`: score value for a character match, e.g. +3;
   (c) `score_mismatch`: score value for a character mismatch, e.g. -1.

As output the script should return the best (local or global) alignment and its score. Bear in mind that in some cases there may be several optimal alignments, we ask you to return *any* one of the possible best alignments.

Your script should follow the structure that we have laid out in the skeleton script called `CB_HW1_SequenceAlignment_skeleton.R`. This skeleton splits the assignment into smaller functions that are necessary to compute the final result. Each function is listed with the input and output that it requires, as well as a short description of its task. You have to fill in the missing code (marked with `# ???`) in the body of the functions, which should perform the necessary computations.

> Please do not change the structure of the code and the inputs and outputs of functions. The code will be tested and graded automatically and any structural changes will result in your code failing the tests.

To help you understand the skeleton structure here we provide a cross-reference table showing which functions in the skeleton should use which others. You can see it in Table 1.

To test your code, you can use the same sequences and score values that you used for the calculations in the tutorial, as well as the examples provided in the lecture notes. We will also provide you with a test suite which is very similar to the one we will use to grade the homework. You can use that to verify that your code is performing as expected. Bear in mind that the suite used to grade your code will have a similar structure but will contain different parameters and may include more test conditions.

Function name	Uses

`align`	`init_score_matrix`
	`init_path_matrix`
	`fill_matrices`
	`get_best_alignment`

`get_best_alignment`	`get_best_move`
`get_best_move`	-
`fill_matrices`	`get_best_score_and_path`
`get_best_score_and_path`	-
`init_path_matrix`	-
`init_score_matrix`	-

Table 1: Function cross-reference table

Algorithm pseudocode

Both algorithms essentially operate in a very similar manner. They are examples of dynamic programming. Both algorithms start by initializing a score and path matrix, which are then filled out cell by cell, building the matrices of scores for partial alignments. Finally, the score for the optimal alignment is read out and the alignment itself is reconstructed. The main difference between the two methods lies in the scoring for specific partial alignments. While the global alignment algorithm allows negative scores, as all characters in both sequences have to be aligned to something (to a gap at the very least), this is not allowed in a local alignment and low-scoring gaps at either end of alignment are discarded. However, all the structures necessary for implementing the two algorithms are actually the same, as you will need the score and the path matrix for both of them.

This section provides the schematic description of the two algorithms (pseudocode), which you can use as a guide for your programming.

Smith-Waterman algorithm pseudocode The Smith-Waterman algorithm allows us to get the optimal local alignment of two given sequences.

Data: Sequences seq_A and seq_B of lengths n_A and n_B
Data: Gap score w, match score m and mismatch score k
Result: Optimal local alignment ali_A and ali_B

Initialize an empty $n_A + 1$ by $n_B + 1$ *score_matrix*;
$score\_matrix[i, 1] \leftarrow 0$, where $1 \leq i \leq n_A + 1$;
$score\_matrix[1, j] \leftarrow 0$, where $1 \leq j \leq n_B + 1$;

Initialize an empty $n_A + 1$ by $n_B + 1$ *path_matrix*;
for $i = 2$ **to** $n_A + 1$ **do**
  for $j = 2$ **to** $n_B + 1$ **do**
    $$score\_matrix[i, j] \leftarrow \max \begin{cases} score\_matrix[i-1, j-1] + \text{score}(seq_A[i-1], seq_B[j-1]) & \text{(case 1)} \\ score\_matrix[i, j-1] + w & \text{(case 2)} \\ score\_matrix[i-1, j] + w & \text{(case 3)} \\ 0 & \text{(case 4)} \end{cases};$$

    $$path\_matrix[i, j] \leftarrow \begin{cases} diag & \text{(case 1)} \\ left & \text{(case 2)} \\ up & \text{(case 3)} \\ - & \text{(case 4)} \end{cases};$$

  end
end
Initialize empty alignments ali_A and ali_B;
Initialize current position (p_A, p_B) such that
$score\_matrix[p_A, p_B] = \max(score\_matrix)$;
while $score\_matrix[p_A, p_B] > 0$ **do**
  if $path\_matrix[p_A, p_B] = diag$ **then**
    Prepend character $seq_A[p_A - 1]$ to ali_A;
    Prepend character $seq_B[p_B - 1]$ to ali_B;
    $(p_A, p_B) \leftarrow (p_A - 1, p_B - 1)$;
  else if $path\_matrix[p_A, p_B] = left$ **then**
    Prepend a gap to ali_A;
    Prepend character $seq_B[p_B - 1]$ to ali_B;
    $(p_A, p_B) \leftarrow (p_A, p_B - 1)$;
  else if $path\_matrix[p_A, p_B] = up$ **then**
    Prepend character $seq_A[p_A - 1]$ to ali_A;
    Prepend a gap to ali_B;
    $(p_A, p_B) \leftarrow (p_A - 1, p_B)$;
end

Needleman-Wunsch algorithm pseudocode The Needleman-Wunsch algorithm allows us to get the optimal global alignment of two given sequences.

Data: Sequences seq_A and seq_B of lengths n_A and n_B
Data: Gap score w, match score m and mismatch score k
Result: Optimal global alignment ali_A and ali_B

Initialize an empty $n_A + 1$ by $n_B + 1$ *score_matrix*;
$score\_matrix[1, 1] \leftarrow 0$;
$score\_matrix[i, 1] \leftarrow (i - 1) \times w$, where $1 < i \leq n_A + 1$;
$score\_matrix[1, j] \leftarrow (j - 1) \times w$, where $1 < j \leq n_B + 1$;

Initialize an empty $n_A + 1$ by $n_B + 1$ *path_matrix*;
$path\_matrix[i, 1] \leftarrow up$, where $1 < i \leq n_A + 1$;
$path\_matrix[1, j] \leftarrow left$, where $1 < j \leq n_B + 1$;
for $i = 2$ **to** $n_A + 1$ **do**
  for $j = 2$ **to** $n_B + 1$ **do**
    $score\_matrix[i, j] =$

$$\max \begin{cases} score\_matrix[i-1, j-1] + \text{score}(seq_A[i-1], seq_B[j-1]) & \text{(case 1)} \\ score\_matrix[i, j-1] + w & \text{(case 2)}; \\ score\_matrix[i-1, j] + w & \text{(case 3)} \end{cases}$$

$$path\_matrix[i, j] = \begin{cases} diag & \text{(case 1)} \\ left & \text{(case 2)}; \\ up & \text{(case 3)} \end{cases}$$

  end
end
Initialize empty alignments ali_A and ali_B;
Initialize the current positions: $p_A = n_A + 1$ and $p_B = n_B + 1$;
while $p_A > 1$ or $p_B > 1$ **do**
  if $path\_matrix[p_A, p_B] = diag$ **then**
    Prepend character $seq_A[p_A - 1]$ to ali_A;
    Prepend character $seq_B[p_B - 1]$ to ali_B;
    $(p_A, p_B) \leftarrow (p_A - 1, p_B - 1)$;
  else if $path\_matrix[p_A, p_B] = left$ **then**
    Prepend a gap to ali_A;
    Prepend character $seq_B[p_B - 1]$ to ali_B;
    $(p_A, p_B) \leftarrow (p_A, p_B - 1)$;
  else if $path\_matrix[p_A, p_B] = up$ **then**
    Prepend character $seq_A[p_A - 1]$ to ali_A;
    Prepend a gap to ali_B;
    $(p_A, p_B) \leftarrow (p_A - 1, p_B)$;
end

Additional material and help

Required packages

This programming assignment does not require any additional R packages.

Useful functions

We provide here a list of R functions that you may find useful when writing your code (Table 2).

Function name	Function description
`seq(from, to, by)`	Get a vector of values, starting from value `from`, finishing at maximum value of`to`, using step `by` for increments. E.g., command `seq(0, 7, 2)` will return `0 2 4 6`.
`matrix(value, nrow, ncol)`	Create an `nrow` by `ncol` matrix filled up with the given value `value`. E.g. command `matrix(0, 2, 4)` will return a 2 by 4 matrix of 0.
`rep(value, n)`	Create an array of the given `value`, replicated `n` times. E.g. `rep("left", 3)` will return `"left" "left" "left"`.
`substr(string, start, end)`	Extract a substring from a sequence of characters `string`, starting at position `start` and finishing at position `end` (including). E.g. the command `substr("TCACACTAC", 1, 4)` will yield `"TCAC"`. If the `start` parameter is equal to the `end` parameter, a single character at that position is returned. E.g. the command `substr("AGCACAC", 2, 2)` will yield `"G"`.
`nrow(matrix)`	Returns the number of rows in a matrix.
`ncol(matrix)`	Returns the number of columns in a matrix.
`max(matrix)`	Returns the maximal value in a given matrix.
`which(matrix == value, arr.ind = TRUE)[1,]`	Return the row and column of the first element in the given `matrix` that is equal to the provided `value`.
`nchar(string)`	Returns the number of characters in a string.
`paste0(string1, string2)`	Concatenate the two given strings without any separating characters.

Table 2: Useful R functions

Troubleshooting tips

If your code fails with an error, the error message should contain details on what went wrong and why the programme failed. However, it can also occur that your code runs without reporting an error but produces incorrect results. In this exercise you

can use the score matrix to locate the issue quickly. If the script's score matrix does not match the one you computed by hand, the error in your code is in calculation of the matrix, and if the matrices match, the error most likely is in the reconstruction of the alignment from the score matrix.

Furthermore, you need to be especially careful about the indices of positions in the score matrix: the two most likely errors are (i) mixing up column indexes and row indexes, and (ii) off-by-one errors (i.e errors where the index is off by one, for instance using `i` instead of `i-1` or `i+1` instead of `i`). Be careful to remember which sequence is on the rows of the matrix and which is on the columns, and remember that the sequences start on the second row and column of the matrix, not the first. To avoid indexing errors we also advise you to draw your planned matrices on paper before writing your code.

Using the testing suite

The `RUnit` and `R.utils` packages are necessary to run the provided test functions. To install the packages, simply enter the following command in the R command line (you only need to do this once per clean installation of R):

```
install.packages(c("RUnit", "R.utils"))
```

The packages are loaded in the test suite, so you do not need to load them manually.

The test suite checks your code on a per-function basis. This means that each function will be tested separately using various input values to ensure that each individual part of your code is working correctly. You can use this to debug your code, as you will be able to see which function is causing trouble. We recommend that you start with the simplest, low-level functions, ensure that they are working correctly, and then move on to more high-level functions.

For the test suite to work, you have to source your written code, such that the R console can access the functions that you define. Once you have sourced the code, open the provided script `run_tests.R` and set `tests_dir` variable to the correct path to the test files, e.g.:

```
tests_dir = "/Users/username/CB-course-2018/Homeworks/
CB_HW1-SequenceAlignment/code/student_test_suite/"
```

Once you have set the correct path, source the `run_tests.R` script to get the evaluation of your code.

Failures in the tests indicate cases when the output of your code is not what is expected in the test. I.e. a failure will be reported if you return a score of 4 for an alignment whereas the true score is 5. Errors in the tests indicate cases when an R error arises, e.g. in the form of a syntax error. That means that your code cannot be executed properly.

Example output of successful testing may look like this:

```
Executing test function test.fill_matrices  ...  done successfully.
Executing test function test.init_path_matrix  ...  done successfully.
Executing test function test.init_score_matrix  ...  done successfully.

RUNIT TEST PROTOCOL -- Wed Sep 19 19:14:29 2018
*********************************************
Number of test functions: 3
Number of errors: 0
Number of failures: 0

1 Test Suite :
HW - 3 test functions, 0 errors, 0 failures

Details
**************************
Test Suite: HW
Test function regexp: ^test.+
Test file regexp: ^runit.+\.[rR]$
Involved directory:
/Users/username/CB-course-2018/Homeworks/CB_HW1-SequenceAlignment/code/
student_test_suite/
---------------------------
Test file: /Users/username/CB-course-2018/Homeworks/
CB_HW1-SequenceAlignment/code/student_test_suite//runit.fill_matrices.R
test.fill_matrices: (155 checks) ... OK (0.02 seconds)
---------------------------
Test file: /Users/username/CB-course-2018/Homeworks/
CB_HW1-SequenceAlignment/code/student_test_suite//runit.get_init_matrices.R
test.init_path_matrix: (3 checks) ... OK (0 seconds)
test.init_score_matrix: (5 checks) ... OK (0 seconds)
```

Example output of a failed testing:

```
Executing test function test.fill_matrices  ...  done successfully.
Executing test function test.init_path_matrix  ...  done successfully.
Executing test function test.init_score_matrix  ... Timing stopped at:
0.001 0 0.001
Error in checkEquals(seq(0, score_gap * 11, score_gap), test_mat[1, ],  :
Mean relative difference: 4
Wrong initial values in the upper row for local=F
done successfully.

RUNIT TEST PROTOCOL -- Wed Sep 19 19:43:14 2018
*********************************************
Number of test functions: 3
Number of errors: 0
Number of failures: 1
```

```
1 Test Suite :
HW - 3 test functions, 0 errors, 1 failure
FAILURE in test.init_score_matrix: Error in
checkEquals(seq(0, score_gap * 11, score_gap), test_mat[1, ],  :
Mean relative difference: 4
Wrong initial values in the upper row for local=F

Details
**************************
Test Suite: HW
Test function regexp: ^test.+
Test file regexp: ^runit.+\.[rR]$
Involved directory:
/Users/username/CB-course-2018/Homeworks/CB_HW1-SequenceAlignment/code/
student_test_suite/
---------------------------
Test file: /Users/username/CB-course-2018/Homeworks/
CB_HW1-SequenceAlignment/code/student_test_suite//runit.fill_matrices.R
test.fill_matrices: (155 checks) ... OK (0.04 seconds)
---------------------------
Test file: /Users/username/CB-course-2018/Homeworks/
CB_HW1-SequenceAlignment/code/student_test_suite//runit.get_init_matrices.R
test.init_path_matrix: (3 checks) ... OK (0 seconds)
test.init_score_matrix: FAILURE !! (check number 2)
Error in checkEquals(seq(0, score_gap * 11, score_gap), test_mat[1, ],  :
Mean relative difference: 4
Wrong initial values in the upper row for local=F
```

In this specific case the error is in the `init_score_matrix` function, and one of the values in the upper row of the score matrix in the case of local alignment differs from the expected value by 4.