# Homework 2

**Exercises:**

These are ungraded, short questions that are representative of the types of problems you might expect to see on exams. You do not need to submit written answers to these, but be prepared to discuss them in class.

1. What is an affine gap penalty? What are the advantages of using one in scoring sequence alignments? What are the disadvantages?
2. What do scoring matrices like PAM and BLOSUM contribute to the process of finding alignments that constant scoring schemes (with set values for $M, m$, and $g$) do not?
3. Two sequences separated by 1-PAM differ in about 1% of amino acids. Do sequences separated by 30-PAM differ on average in 30% of amino acids? Why?
4. What does it mean when a PAM log-odds substitution matrix entry between two different amino acids, say D and N, is greater than zero?
5. Which represents a greater degree of divergence, BLOSUM-45 or BLOSUM-62? Why?
6. In database searching, what is an alignment E-value? How does it depend on the size of the database? What implications does this have for choosing which database to search?

**Part 1:**

Answer the questions below. Some questions rely on content from the assigned reading on the syllabus (but not the optional reading); most are based on material discussed in class as well.

1. Compute the local alignment of the following two sequences: HOTSHOT and SHORTEST. Use a scoring scheme where match (M) = +2, mismatch (m) = -1, and gap (g) = -2.

   1.1 (10 points) Show the alignment matrix with scores and traceback pointers. You may use the alignment template at http://www.cs.tufts.edu/comp/167/private/alignment.template.pdf if you like.

1.2 (10 points) What is the score of the optimal local alignment? How many optimal local alignments are there? Show them.

2. Neither global nor local

   (12 points) Suppose you wanted to compare a short sequence with a long one. For example, you might want to compare a novel chimpanzee gene to the human genome, or a short functional site to a larger regulatory region. Thus, you'd like an alignment algorithm that penalizes gaps at the ends of the long sequence, but that doesn't penalize gaps at the ends of the short sequence. That is,

   ```
   NEWSPAPER
   ___SPA___
   ```

   would not penalize gaps, but the gaps aligned with CH below would be penalized.

   ```
   NEWSPAPER__
   ______PERCH
   ```

   Describe how to modify the Needleman-Wunsch global alignment algorithm to do this. Write the equation for $A(i, j)$ in the alignment matrix, and explain how to find an optimal alignment and its score.

3. Suboptimal local alignment.

   3.1 (12 points) Suppose you wanted to find a suboptimal local alignment that does not match any of the pairs of letters, or more specifically, the aligned subsequences of the original words, matched in the optimal local alignment(s).

   Note: there can still be letters from one sequence *or* the other that are part of both alignments - we are just requiring that they align with different letters in the other sequence. They can also be the same letters of the alphabet - they just cannot be from the same positions in the sequence. E.g., if substring CATT from positions 3-6 in sequence x aligns with CATT in position 10 in sequence y, the suboptimal alignment might have CATT in positions 3-6 in sequence x aligned with the substring CAGT in position 18 in sequence y.

   Describe how you could modify the Smith-Waterman algorithm to find the best-scoring such suboptimal alignments.

   3.2 (8 points) Use the approach you described in 2.1 to find the best such suboptimal local alignment(s) of the sequences in question 1 that do not share any pairs of aligned letters with the optimal local alignment(s) you found in question 1.

   What is the score of the suboptimal alignment(s) you find this way? Show it (them).

4. In this problem, you will consider a number of possible scoring schemes for sequence alignments.

   4.1 (8 points) What restrictions on the scoring function make it suitable for finding local alignments using the Smith-Waterman algorithm?

   4.2 (7 points) Does the BLOSUM-62 matrix (p. 83 in your text) appear to satisfy these criteria? Explain why, or why not, in one or two sentences. (Note: you can answer this question by just looking at the BLOSUM-62 scoring matrix. You are not expected to do any exact mathematical calculations. State explicitly any assumptions you make.)

   4.3 (3 points each; total of 12) Consider the three proposed scoring schemes below for pairwise sequence alignment, where $M$ designates a match, $m$ designates a mismatch, and $g$ designates a gap. For each scoring scheme, state whether it is suitable for global alignment only, local alignment only, both local and global alignment, or neither. In each case, justify your answer with one or two sentences.

      i. $M = 0, m = 3, g = 2$.
      ii. $M = 2, m = -3, g = -1$.
      iii. $M = 3, m = -1, g = -3$.
      iv. $M = -2, m = -3, g = -4$.

5. Database searching with protein sequences

   5.1 (7 points) If you were going to compare different hemoglobin protein sequences in the human genome, would PAM-30 or BLOSUM-62 be more suitable? Why? (Think about those sequences and how the scoring matrices were constructed to help answer this question.)

   5.2 (7 points) Briefly explain the difference between the database search problem and the pairwise local alignment problem. What is the key difference in the *problems* (not the solutions) that makes algorithms other than Smith-Waterman local alignment preferable for database search?

   5.3 (7 points) What do banded dynamic programming and X-drop have in common that helps improve the efficiency of database search methods?

**Part 2:**

Write a Python program (using Python3) that implements global DNA sequence alignment using a simple similarity scoring scheme with a linear gap penalty: $M$ for each matched pair in the alignment, $m$ for each mismatch, and $g$ for each gap.

The program should be called "**align.py**". It should accept a FASTA-formatted file as input (see the sample input files mentioned below; the web site also has a link to a description of the FASTA file format) and should find an optimal alignment for the two sequences in the file.

Overview

In your program you will implement the class `GlobalSeqAlignment`. This class should include any code that implements the global alignment algorithm for two sequences using a simple similarity scoring scheme with a linear gap penalty. We will be using "-" (the hyphen) to represent gaps, not the underscore character ("_") as was used in class. This is just for consistency with some of the current grading code.

The constructor of the class should take in the two sequences, `x` and `y`, match score `M`, mismatch score `m`, and gap score `g`. This class must include the class function `get_optimal_alignment()` that returns the tuple `(x', y')` that correspond to the optimal alignment found for the given two sequences. You are welcome to (in fact, encouraged to) write any additional helper functions within the class to help implement the algorithm.

You should be able to execute the following code:

```
align = GlobalSeqAlignment(x, y, M, m, g)
x_prime, y_prime = align.get_optimal_alignment()
```

so that `x_prime` and `y_prime` contain the optimal alignments found by your program for the two sequences `x` and `y`.

The default values for the scoring scheme should be: Match (`M`): +4, mismatch (`m`): -2, gap (`g`): -2. However, you should design your program so that it is easy for you to try different values for `M`, `m`, and `g` (e.g. passing them in as arguments).

**Important:** In your implementation of the traceback algorithm, we ask that you resolve ties for a given matrix element (i.e., where there are two or more pointers from a given matrix square) by favoring the match/mismatch case first; next, the case aligning $x_i$ with a gap; and last, the case aligning $y_j$ with a gap.

In other words, if your matrix has $x$ going down the x axis and y going across the y axis, then you should greedily favor a *diagonal traceback*, then an *upward traceback*, and lastly a *left traceback* when performing your alignment. This is simply so that your

alignment results can be compared easily to the provided outputs (see the sample output files mentioned below).

In your program you must also implement a separate function `parse_FASTA_file(fasta_file)` that takes in a *streamed* FASTA-formatted file and returns the tuple `(x, y)` that correspond to the 2 input sequences that you will pass into your global sequence alignment. Overall, you should be able to execute the following code

```
x, y = parse_FASTA_file(sys.stdin)
```

such that `x` and `y` are the string input sequences and `sys.stdin` is the FASTA-formatted file streamed as input (see more below).

Testing

For help in formatting your output and testing your code, there are files on the course web page provided called "**aligntest.input{1,2,3}**" and "**aligntest.output{1,2,3}**". The input files are all FASTA-formatted files that include exactly two input sequences. The first sequence in the file should be saved as `x` and the second sequence in the file should be saved as `y`, where `x` and `y` are the input sequences to your program.

The output file should contain the optimal alignment for those inputs using the default scoring scheme and the indicated traceback order. For example, if you run your program by typing

```
python3 align.py < aligntest.input1 > my.output1
```

and then run

```
diff aligntest.output1 my.output1
```

using the default scoring parameters listed above, you should see no differences between the two files. Note that your program should accept an input FASTA-formatted file as a **streaming input** (i.e. through `sys.stdin`), as indicated by the "<" symbol. Similarly, the ">" symbol indicates that your optimal alignment should be **streamed output** (i.e. the aligned sequences should be **printed out**).

Submission & Autograder

**All submissions should include a file called README.**

Note that the README file should just be called that, because the autograder will check for this. It should not be named README.txt or README.pdf. If you are doing your

work on a Linux machine this will not be a problem, but if you create the file on a different operating system (such as on many laptops), there are often unexpected file extensions. Make sure you have removed these before submitting your solutions.

The README file should contain instructions on how to run your code. Please be sure to document in the README file how to change the scoring scheme for your program. Command line arguments are encouraged.

You should also naturally submit all source code. Ensure that your program is called exactly align.py, as the autograder will check for this.

The autograder will mostly check your code's overall validity and test individual components. Not all of the test cases will be visible ahead of time, but upon submission you should be able to see some of the test cases of the autograder to guide you. Resubmissions up to the due date are allowed.

Manual Evaluation Criteria: We will mostly be checking to see that your code works correctly and shows understanding of the algorithm. However, particularly unreadable coding style (all one big function; meaningless variable names; no comments) may result in deductions.

**How to submit your assignment using Gradescope:**

Go to www.gradescope.com and find the Tufts CS 167 course for Spring 2025 (or find via the course website).

Save your solutions to the written parts of the assignment as a single pdf file and upload it to the assignment titled "**Homework 2 (part 1)**".

You may resubmit as often as you want, but Gradescope only keeps the latest submission and the timestamp for the latest submission. Please keep the class token policy in mind if you are resubmitting after the deadline.

For part 2, go to the assignment titled "**Homework 2 (part 2)**" and upload whatever files we need to run and test your program. You will lose points if you do not include a README file describing how to run your program and set the scoring scheme.