Assignment 9 - Alignment

01DTUSM - BioQuants

Nunzio Licalzi s344860

Politecnico di Torino

December 2024
Academic year 2024/2025

Contents

1 Assignment Introduction

The assignment chosen is **Assignment #9 (Python - Alignment)**.
This assignment aims to introduce a solution to the Next-Generation Sequencing (NGS) problem.
To achieve this, I wrote a program that takes in a reference genome and a set of short reads and outputs the alignments of the reads to the reference genome.
The following functions were developed:

1. **score_alignment**: This function takes two strings as input and returns the evaluated score alignment between them.

2. **find_best_alignment**: This function takes a reference string, a query string, and the scoring function from step 1, and returns the best alignment of the query string to the reference string.

3. **align_reads**: This function takes a reference genome and a list of query strings, and returns the alignments of the query strings to the reference genome.

Additionally, I developed other functions for data input, validation, and output.
A test suite was also implemented for testing the object-oriented implementation of the program.
Finally, two versions of the application were developed: the first using a procedural programming approach (Jupyter Notebook) and the second using an object-oriented approach (Python file).

2 Content of the Submitted Folder

This section describes the contents of each file/folder:

- **Assignment9.py**: Contains the implementation of the object-oriented application.
- **s344860_Assignment9_Sol.ipynb**: Procedural implementation of the application via Jupyter Notebook (uses Python standard libraries). I chose Jupyter for the markdown support it offers.
- **referenceSequence.txt**: Example file containing a reference sequence.
- **queryData.txt**: Example file containing query sequences (one per row).
- **test folder**: Folder containing the test suite created to test the object-oriented implementation of the application.
  Each file tests a specific function, except for the first two, which test the whole class, and the `completeUsage_test.py` file that tests the complete usage of the application.
  - **align_reads_test.py**: Tests the function `align_reads`.
  - **AlignmentClass_test.py**: Tests the entire class.
  - **AlignmentClass2_test.py**: Tests the class with respect to edge cases.
  - **checkSequenceValidity_test.py**: Tests the function `checkSequenceValidity`.
  - **completeUsage_test.py**: Tests a complete usage of the application, from data input to final output.
  - **find_best_alignment_test.py**: Tests the `find_best_alignment` function.
  - **prettyprint_test.py**: Tests the `prettyprint` function, which outputs the results of `align_reads` in a formatted manner.
  - **readQueryData_test.py**: Tests the `readQueryData` function, which reads query data from a file.
  - **readSequence_test.py**: Tests the `readSequence` function, which reads the reference sequence from a file.
  - **score_alignment_test.py**: Tests the `score_alignment` function that evaluates the score between two sequences.

3 How to Use the Application

This section summarizes how to correctly use the application and describes all the functions (or methods) it includes, as well as the key differences between the two slightly different implementations.

3.1 Functions (Methods) Present

3.1.1 Data Input

For data input, the following two main functions are available (two more are present in the Jupyter implementation):

- **readSequence**: Accepts a path to a text file containing the reference sequence. The sequence is read as a single string. The function checks for errors, and if none are found, the sequence is returned; otherwise, an error is raised.
- **readQueryData**: Accepts a path to a text file containing query sequences (one per row). Each sequence is validated. If all sequences are valid, a list of sequences is returned; otherwise, an error is raised.

3.1.2 Data Validation

To validate input data, the function **checkSequenceValidity** can be used. It returns `True` if the given sequence is not empty and contains only characters from the set {A, C, G, T, X, -}.

3.1.3 Function: `score_alignment`

This function evaluates the alignment score between two strings.
Inputs:

- Two sequences (as strings).

Outputs:

- The alignment score.

The function raises an error if the two sequences have different lengths or if either sequence is an empty string.
The alignment score is computed as:

$$\sum_{i=0}^{N} 2 \cdot \left[\frac{1}{2} - \delta(seq_1[i] \neq seq_2[i]) \right]$$

Where:

- seq_1: First sequence.

- seq_2: Second sequence.
- N: Length of either sequence (they have the same length).
- $\delta(seq_1[i] \neq seq_2[i])$: Kronecker delta (1 if characters do not match, 0 otherwise).

Example: The alignment of "ACGGT" and "ACGGC" has a score of 3 (4 matching characters and 1 mismatch).

3.1.4 Function: `find_best_alignment`

This function evaluates the best possible alignment for a query sequence within a reference sequence.
Inputs:

- Reference sequence.
- Query sequence.
- Scoring function.

Outputs:

- Starting position of the best alignment within the reference sequence.
- Best alignment score.

The function raises an error if the reference sequence length is less than or equal to the query sequence length.

3.1.5 Function: `align_reads`

This function evaluates the best alignments for a sequence of queries against a reference sequence.
Inputs:

- Reference sequence.
- List (or set) of query sequences.
- Alignment function to compute scores.

Outputs:

- A list of lists, where each sub-list contains:
    0. Portion of the reference sequence best matching the query sequence.
    1. Query sequence.
    2. Starting position of the best match.
    3. Alignment score of the best match.

Note: The sub-list numbering corresponds to the item's position within the sub-list.

3.1.6 Data Output

The function `prettyprint` formats and outputs the results of `align_reads`.
Inputs:

- Results from `align_reads`.
- Reference sequence.
- Output specification (screen or file).

The `outputFilePath` parameter has three possible meanings:

- Boolean `True`: Output is printed to the screen (default).
- Non-empty string: Output is written to a file at the specified path.
- Anything else: No output is produced.

Moreover, the last parameter, *outputFilePath*, can have three different meanings based on its value:

- if a bool is passed, and it's value is True, the output is print on screen (i.e. stdout).
- if a non empty string is passed, the output is printed on a text file whose path is represented by the given string.
- anything else will result in no output being produced.

Note: the default behavior of the parameter is the one indicated by the first point.
The function returns the `results` parameter (output of `align_reads`).

3.2 Object-Oriented Implementation

3.2.1 Key Differences

In the object-oriented implementation:

- Attributes **referenceSequence** and **querySequence** were created and set as private for consistency reasons.
- Getters and setters were created for both attributes. Setters validate sequences (via the `checkSequenceValidity` function) and raise errors for invalid data.

3.2.2 `align_reads`

The `align_reads` function acts as the main method of the class.
Inputs:

- **referenceSequence**: Reference sequence provided directly.
- **pathReferenceSequence**: Path to a file containing the reference sequence.
- **querySequence**: List (or set) of query sequences.
- **pathQuerySequence**: Path to a file containing query sequences (one per row).
- **alignmentFunction**: Function for scoring alignments.
- **outputFile**: Specifies output behavior (refer to 3.1.6).

All the parameters are optional.
The function handles the input of the reference sequence (and similarly for the query sequence) as follows:

- If both parameters are set to **None** and the corresponding attribute (e.g., *referenceSequence*) is also **None**, an error is raised.
- If both parameters are set to **None** but a valid value exists for the corresponding attribute, the attribute's value is used as the reference sequence.
- If one of the two parameters is provided, the function interprets this as a request to update the data. The new input is validated, and if valid, it replaces the current value of the respective attribute. If the new input is invalid, an error is raised, and the attribute remains unchanged.
- If both parameters are provided, the function prioritizes the parameter given directly as an input over the one provided through a file.

Note: When updating both parameters, if either input is invalid, the other parameter's value remains unchanged, thereby ensuring compliance with the ACID properties.

3.3 Procedural Implementation

In the procedural implementation, there are minimal differences from the behavior described in the *How to Use the Application* section. Two additional functions were introduced for data input:

- **readSequenceFromKeyboard**: Reads a valid reference sequence from keyboard input. Keeps prompting until a valid sequence is provided. Returns the sequence as a string.
- **readQueryDataFromKeyboard**: Reads query sequences via keyboard input, either as individual strings or a comma-separated list. Stops reading when an invalid sequence is encountered and returns all valid sequences provided up to that point.

4 Testing

To evaluate the developed applications, two testing strategies were employed, aligned with the programming paradigm used.

4.1 Testing Object-Oriented Implementation

As described in the *Content of the Submitted Folder* section (2), the `test` folder contains a comprehensive test suite with 10 Python files.
The test suite:

- Verifies individual function functionality.
- Tests the entire class.
- Simulates potential user scenarios, including edge cases.

A total of **82 distinct tests** were written.

4.2 Testing Procedural Implementation

For the procedural implementation, two distinct and non-trivial example scenarios were developed to simulate end-user behavior.
This approach was chosen instead of an ad hoc test suite to reflect the application's dual-purpose design for distinct user types and use cases.