Indexing and Querying Sequences Using an Alignment-Free Method

Developer Report

Alexine, Julie Buset

December 16, 2024

1 Overview

This developer report is part of a project aiming to represent and index sequencing data for rapid, alignment-free queries. The goal is to transform a set of input sequences into an SPSS (a concatenated structure representing solid k-mers), and then build an FM-index over it. The resulting index allows efficient substring queries to identify shared k-mers between the query sequences and the indexed dataset.

The source code for this project is available on GitHub at: `https://github.com/jalexine/spss_exploration_project`

The repository is organized as follows:

```
spss_exploration_project
|-- README
|-- src/
|   |-- pysuffix3/
|   |   `-- tools_karkkainen_sanders.py
|   |-- query_indexed_spss.py
|   |-- fmi.py
|   |-- sequences_to_indexed_spss.py
|   `-- timer.py
|-- benchmark
|   |-- scripts
|   |   |-- plot_threshold_05x.py
|   |   |-- plot_threshold_30x.py
|   |   |-- plot_spss.py
|   |   `-- plot_logs.py
|   |-- plots
|   `-- stats
|-- rapports
|   |-- developper_report.pdf
|   `-- scientific_report.pdf
|-- config.yaml
`-- Snakefile
```

In this report, we focus exclusively on two key modules within the `src/` directory:

- `sequences_to_indexed_spss.py`: Processes raw sequence data to produce the SPSS and then constructs an FM-index. It can also records performance metrics and validates the index.
- `fmi.py`: Provides the tools for constructing and querying the FM-index. This module handles the Burrows–Wheeler Transform (BWT), the prefix arrays, and substring search capabilities.

This report outlines the structure of the code, explains the key functions, and highlights the main technical choices made. It is intended to help developers understand, maintain, and extend the codebase.

2 The `sequences_to_indexed_spss.py` Module

2.1 Purpose and Role

The `sequences_to_indexed_spss.py` module is responsible for processing raw genomic sequence data to generate a Spectrum-preserving string sets (SPSS) and subsequently constructing an FM-index based on this SPSS. This module facilitates the transformation of sequencing data into a format that supports rapid, alignment-free substring queries. This module takes 2 mode : one for the generation of unitigs, the other one for simplitigs. Additionally, it records performance metrics and validates the integrity of the generated FM-index to ensure its reliability for subsequent querying tasks.

2.2 General Structure

- **Functions:**
  - `canonical_kmer(kmer)`: Determines the canonical form of a k-mer by selecting the lexicographically smaller between the k-mer and its reverse complement.
  - `count_kmers(fasta_file, k)`: Counts all canonical k-mers in the input FASTA file.
  - `filter_kmers(kmer_counts, threshold)`: Filters out k-mers that do not meet the solidity threshold.
  - `generate_simplitigs(kmers, k)`: Constructs maximal simplitigs from the set of solid k-mers.
  - `generate_unitigs(kmers)`: Generates unitigs from the set of k-mers.
  - `generate_spss(kmers, k, mode='simplitig')`: Creates the SPSS by concatenating simplitigs or unitigs based on the selected mode.
  - `prepare_output_dir(base_name, mode, k, t, extension='dump', parent_dir='benchmark')` : Creates and formats the output directory for saving files.
  - `save_fm_index(fm_index, fasta_file, mode, output_file=None)`: Serializes and saves the FM-index to disk.
  - `save_benchmark_results(fasta_file, mode, stats)`: Records benchmarking statistics to a CSV file.
  - `test_fm_index(fm_index, spss, filtered_kmers, sample_size=100)`: Tests the FM-index by comparing results from its 'contains' method with Python's 'in' operator on the SPSS string.
- **Main Execution Flow:**
  - Parsing command-line arguments using `argparse`.
  - Counting and filtering k-mers based on solidity threshold.
  - Constructing the de Bruijn graph (DBG) and generating the SPSS.
  - Building the FM-index using the `FmIndex` class from the `fmi.py` module.
  - Validating the FM-index to ensure its correctness.
  - Saving the FM-index and benchmarking results.

2.3 Key Steps and Functions

2.3.1 K-mer Processing

- `canonical_kmer(kmer)`: Determines the canonical form of a k-mer by selecting the lexicographically smaller string between the k-mer and its reverse complement. This ensures a unique representation for each k-mer, reducing redundancy.
- `count_kmers(fasta_file, k)`: Parses a FASTA file to count the occurrences of each canonical k-mer. For each k-mer, it also computes the reverse complement using the translate method and determines the canonical k-mer by selecting the lexicographically smaller string. Only sequence lines are processed, while header lines that begin with the $>$ character are ignored.
- `filter_kmers(kmer_counts, threshold)`: Filters out k-mers that do not meet the specified solidity threshold, retaining only those with occurrences greater than or equal to the threshold.

2.3.2 SPSS Construction

- `generate_simplitigs(kmers, k)`: Constructs maximal simplitigs from the set of solid k-mers. Simplitigs are simplified paths through the de Bruijn graph that represent concatenated k-mers without branching, optimizing the SPSS for efficient indexing.
- `generate_unitigs(kmers,k)`: Generates unitigs by dynamically traversing a graph formed by a set of solid k-mers. A unitig is a maximal non-branching path in the k-mer graph, representing contiguous regions of sequences that do not branch in the de Bruijn graph.
- `generate_spss(kmers, k, mode='simplitig')`: Creates the SPSS by concatenating simplitigs or unitigs based on the selected mode ('simplitig' or 'unitig'). The SPSS is terminated with a special character ('$') to denote the end of the sequence.

2.3.3 FM-Index Construction and Validation

- `save_fm_index(fm_index, fasta_file, mode, output_file=None)`: Serializes and saves the constructed FM-index to a file. If no output file is specified, it generates a default file name based on the input FASTA file and mode.
- `save_benchmark_results(fasta_file, mode, stats)`: Records benchmarking statistics, such as execution times and SPSS metrics, to a CSV file. It also prints these statistics to the console for immediate feedback.
- `test_fm_index(fm_index, spss, filtered_kmers, sample_size=100)`: Validates the FM-index by randomly sampling k-mers from the filtered set and comparing the FM-index's `contains` method against substring searches within the SPSS. This ensures the accuracy and reliability of the index.

2.4 Input/Output and Parameters

- **Input:**
  - `-i <sequence_file_name>`: Path to the input FASTA file containing genomic sequences.
  - `-k <kmer_size>`: Size of the k-mers to be extracted.
  - `-t <solidity_threshold>`: Threshold for k-mer solidity; only k-mers with occurrences above this threshold are retained.
  - `-m <mode>`: Mode of SPSS construction, either 'simplitig' or 'unitig'.
  - `-o <output_file_name>` (optional): Path to save the serialized FM-index. If not provided, a default naming scheme is used.
  - `-h` (optional): Displays a help message and usage instructions.
  - `-stats` (optional): Produce csv file for benchmarking
- **Output:**
  - `FM-index file (.dump)`: Serialized FM-index ready for querying.
  - `Benchmark statistics (.csv)`: CSV file containing performance metrics such as execution times and SPSS characteristics.
  - `Console logs`: Printed information detailing the processing steps and benchmarking results.

2.5 Integration Within the Project

The `sequences_to_indexed_spss.py` module interacts closely with the `fmi.py` module to build and validate the FM-index. Specifically, it utilizes the `FmIndex` class from `fmi.py` to construct the FM-index based on the generated SPSS. The module performs validation checks to ensure the FM-index accurately represents the SPSS by comparing FM-index queries with direct substring searches. Furthermore, it outputs the FM-index in a serialized format, making it readily available for use by other modules, such as `query_indexed_spss.py` (not covered in this report). This integration ensures that the FM-index is both accurate and efficient, serving as a reliable foundation for subsequent querying operations.

2.6 Technical Advantages

The `sequences_to_indexed_spss.py` module offers several technical benefits. By utilizing canonical k-mers and constructing the SPSS through simplitigs or unitigs, the module reduces redundancy and optimizes the data structure for indexing. The use of the FM-index allows the system to handle large genomic datasets efficiently, enabling rapid substring searches even as the dataset size grows. The clear separation of functions and the modular design facilitate easy maintenance and potential future enhancements, such as integrating alternative indexing methods or optimizing existing algorithms. Additionally, the built-in validation mechanism ensures that the FM-index accurately reflects the original SPSS, providing confidence in the integrity of the indexing process. Comprehensive performance metrics enable developers to monitor and optimize the processing pipeline, ensuring that the system meets required efficiency standards. Moreover, the code adheres to PEP 8 conventions, ensuring consistent style and readability.

3 The `fmi.py` Module

3.1 Purpose and Role

The `fmi.py` module provides the core mechanisms for constructing and querying an FM-index from a given sequence. The FM-index allows efficient substring searches, enabling alignment-free queries. By leveraging this data structure, the presence of a particular substring can be confirmed in time proportional to the query length, without the overhead of full alignments.

3.2 General Structure

- `FmIndex` **class**: The main class responsible for building and manipulating the FM-index.
- `load_fm_index(filename)`: A utility function to deserialize a previously saved FM-index.

3.3 FM-Index Construction Steps

Suffix Array The construction process starts by obtaining the suffix array of the input sequence, using the `pysuffix3` library. The suffix array lists all suffixes of the sequence in lexicographical order and forms the foundation of the FM-index.

Burrows–Wheeler Transform (BWT) Given the suffix array, the module computes the BWT. For each sorted suffix, it takes the character preceding it in the original sequence, producing a transformed string that is more conducive to substring searches. This transformation is carried out by the `set_bwt()` method.

***n* and *rank* Structures** The `set_n_and_ranks()` method computes two crucial auxiliary structures:

- ***n***: A dictionary mapping each character to its starting position in the sorted BWT.
- ***rank***: A dictionary of cumulative counts, allowing the number of occurrences of a given character up to any position in the BWT to be retrieved efficiently.

These structures support LF-mapping, a key FM-index operation that links positions in the BWT back to positions in the suffix array.

3.4 Key Methods and Functions

- `save(filename)`: Serializes the FM-index to disk for later reuse.
- `lf(alpha, k)`: Performs the LF-mapping to find the position in the suffix array corresponding to the k-th occurrence of character `alpha` in the BWT.
- `set_bwt(self)`: Computes the Burrows-Wheeler Transform (BWT) of the input sequence using its suffix array.
- `contains(q)`: Checks if the substring `q` occurs in the indexed sequence by performing a backward search. This method relies on the n and $rank$ arrays for efficient lookups.

3.5 Input/Output

- **Input**: A sequence (for example, the SPSS generated by the `sequences_to_indexed_spss.py` module).
- **Output**: The script does not produce any immediate output like printed results or files unless the save() method is explicitly called to serialize the FM-index. The FM-index is built and stored in memory for query operations via methods like contains().

3.6 Integration Within the Project

The `fmi.py` module is utilized by `sequences_to_indexed_spss.py` to build the FM-index on the SPSS and to validate its correctness. Although the querying module (`query_indexed_spss.py`) is not discussed in this report, the FM-index stands at the core of its quick substring lookups.

3.7 Technical Advantages

By using the FM-index, the project benefits from efficient substring searches without storing the full original sequence. The initial computations (suffix array, BWT, n, and $rank$) are performed only once, enabling subsequent queries to be resolved quickly and with minimal memory overhead.

4 Conclusion

This developer report has outlined the two key modules of the project aimed at representing and indexing DNA sequencing data for rapid, alignment-free queries.

The `sequences_to_indexed_spss.py` module is pivotal in transforming raw genomic sequences into an optimized structure, the SPSS, through the use of canonical k-mers and the generation of simplitigs or unitigs. This process minimizes redundancy and prepares the data efficiently for indexing. Additionally, the module ensures the reliability of the FM-index by recording performance metrics and validating the constructed index.

The `fmi.py` module provides the essential algorithmic framework for building and querying the FM-index. By implementing the Burrows–Wheeler Transform (BWT) and computing the auxiliary structures n and $rank$, this module facilitates substring searches in linear time relative to the query length. The seamless integration between the two modules guarantees that the FM-index is both accurate and efficient, establishing a foundation for fast and reliable search operations.

The technical advantages of this approach include a reduced memory footprint due to the FM-index, enhanced efficiency of substring searches, and a modular design that simplifies maintenance and future enhancements as demonstrated in the **scientific report**. Furthermore, the built-in validation mechanisms provide confidence in the index's integrity and accuracy, ensuring reliable query results.

In summary, the `sequences_to_indexed_spss.py` and `fmi.py` modules form a modular and efficient architecture for alignment-free DNA sequence indexing. This structure meets the project's current needs for performance and reliability.