TrimReads

Motivation

High-throughput sequencing technologies have revolutionized genomics research by generating vast amounts of genetic data. However, these sequences often contain low-quality segments, particularly at the ends of reads, which can adversely affect downstream analyses such as genome assembly, variant calling, and functional annotation. To ensure data reliability, it is crucial to remove these low-quality regions while preserving high-quality segments. This tool addresses this need by providing flexible trimming approaches (base-by-base or window-based) with configurable quality thresholds, enabling researchers to preprocess their sequencing data effectively before further analysis.

Algorithm

The tool implements two distinct trimming strategies:

1. **Base-by-Base Trimming** (`--base-threshold`):
   - **Input**: A `FASTQ` record with `Phred` quality scores.
   - **Steps**:
     1. Scan the sequence from the **left end** until encountering a base with quality ≥ the threshold.
     2. Scan from the **right end** until encountering a base with quality ≥ the threshold.
     3. Extract the subsequence between these positions.
2. **Sliding Window Trimming** (`--window-threshold`):
   - **Input**: A `FASTQ` record, window size (`--window-size`), and average quality threshold.
   - **Steps**:
     1. Slide a window from the **left end** until the window's average quality ≥ threshold.
     2. Slide a window from the **right end** until the window's average quality ≥ threshold.
     3. Retain the subsequence between these windows.

Key Features:

- **Flexibility**: Users can choose between the two methods via command-line arguments.
- **Configurable Parameters**: Thresholds, window size are adjustable.
- **FASTQ Compatibility**: Uses `Biopython's` `SeqIO` for parsing and writing `FASTQ` files.

Experiment

The provided Python script (`TrimReads.py`) is ready for deployment. Users can execute it via:

```
# Base-by-base method
python TrimReads.py input.fastq --base-threshold 25
# Sliding window method
python TrimReads.py input.fastq --window-threshold 20 --window-size 5
```

Output includes a trimmed `FASTQ` file and a summary of retained reads. The tool balances simplicity with robust functionality, making it suitable for both small-scale and batch processing workflows.

We use a sample to conduct our solution, which is downloaded from `SRA` database.`Sample.fastq` includes various sequence ,their annotation and quality encoding.

Here is one sequence information in `Sample.fastq` file:

```
@83b951be-cf09-402a-8b19-48105583c067
runid=34c547ba84ed3971a437c6252da360118a5fabd7 sampleid=1 read=68633 ch=80
start_time=2019-10-18T05:04:31Z
GATGCTTTGCGTGATTCCAGATGGGTGTTTATGGACCATATGCGCCTACCGTGACAAGAAAGTTGTCGGTGTCTTTGTGT
TTCTGTTGGTGCTGATATTGCCGAAAATCGGTAGACGCTACGGACTAAATCCGCTTCTTCCTGAAATGCGGGTTTGATCC
CTCTCACAGATAGAGCGACAGGCAAGTCGCAGACTGCGACAGCTTTCTGTC
+
(#*%$%&'%$&$%#-(&*$&#&&)(51-.,&%$'"#$&'%$%31336-798:5-(((/60.5;A1(<4:?
9::6;>/==CD@E@;=>028.*,24765:69:61%*((8,4966;;863(*%(##%*,38::.$$%(,.25+-
%*-02;6;C>;.(*$%'%(/$)-3-.91..)+488*$,-.$&%$-&&$&%$$$$'(.%&%%+1%')%&
```

Three parts in file are Basic information of sequence , Base sequence and Quality encoding. Quality encoding standard shows behind:

```
  Quality encoding: !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHI
                    |         |         |         |         |
     Quality score: 0........10........20........30........40
```

So the quality of example sequence showing above is:

```
Quality Encoding:
(#*%$%&'%$&$%#-(&*$&#&&)(51-.,&%$'"#$&'%$%31336-798:5-(((/60.5;A1(<4:?
9::6;>/==CD@E@;=>028.*,24765:69:61%*((8,4966;;863(*%(##%*,38::.$$%(,.25+-
%*-02;6;C>;.(*$%'%(/$)-3-.91..)+488*$,-.$&%$-&&$&%$$$$'(.%&%%+1%')%&
Quality score:
[7, 2, 9, 4, 3, 4, 5, 6, 4, 3, 5, 3, 4, 2, 12, 7, 5, 9, 3, 5, 2, 5, 5, 8, 7, 20,
16, 12, 13, 11, 5, 4, 3, 6, 1, 2, 3, 5, 6, 4, 3, 4, 18, 16, 18, 18, 21, 12, 22,
24, 23, 25, 20, 12, 7, 7, 7, 14, 21, 15, 13, 20, 26, 32, 16, 7, 27, 19, 25, 30,
24, 25, 25, 21, 26, 29, 14, 28, 28, 34, 35, 31, 36, 31, 26, 28, 29, 15, 17, 23,
13, 9, 11, 17, 19, 22, 21, 20, 25, 21, 24, 25, 21, 16, 4, 9, 7, 7, 23, 11, 19,
24, 21, 21, 26, 26, 23, 21, 18, 7, 9, 4, 7, 2, 2, 4, 9, 11, 18, 23, 25, 25, 13,
3, 3, 4, 7, 11, 13, 17, 20, 10, 12, 4, 9, 12, 15, 17, 26, 21, 26, 34, 29, 26, 13,
7, 9, 3, 4, 6, 4, 7, 14, 3, 8, 12, 18, 12, 13, 24, 16, 13, 13, 8, 10, 19, 23, 23,
9, 3, 11, 12, 13, 3, 5, 4, 3, 12, 5, 5, 3, 5, 4, 3, 3, 3, 3, 6, 7, 13, 4, 5, 4,
4, 10, 16, 4, 6, 8, 4, 5]
```

Base-by-base method:

```
# Base-by-base method
python TrimReads.py sample.fastq --base-threshold 20
```

The sequence after trimming will be:

```
TGTTTATGGACCATATGCGCCTACCGTGACAAGAAAGTTGTCGGTGTCTTTGTGTTTCTGTTGGTGCTGATATTGCCGAA
AATCGGTAGACGCTACGGACTAAATCCGCTTCTTCCTGAAATGCGGGTTTGATCCCTCTCACAGATAGAGCGA
```

Final output will include basic information : Sequence encoding， Original length, Trimmed length, Bases trimmed, Left trim (base), Right trim (base) of total sequence and each sequence. And of course the final `FASTQ` file of all the sequence after trimming will be given.

Output,for example,when the input just like showing above:

```
 Sequence 83b951be-cf09-402a-8b19-48105583c067:
    Original length: 211
    Trimmed length: 153
    Bases trimmed: 58
    Left trim (base): 25
    Right trim (base): 33
```

And in the `sample_trimmed.fastq` file ,we will see:

```
@83b951be-cf09-402a-8b19-48105583c067
runid=34c547ba84ed3971a437c6252da360118a5fabd7 sampleid=1 read=68633 ch=80
start_time=2019-10-18T05:04:31Z
TGTTTATGGACCATATGCGCCTACCGTGACAAGAAAGTTGTCGGTGTCTTTGTGTTTCTGTTGGTGCTGATATTGCCGAA
AATCGGTAGACGCTACGGACTAAATCCGCTTCTTCCTGAAATGCGGGTTTGATCCCTCTCACAGATAGAGCGA
+
51-.,&%$'"#$&'%$%31336-798:5-(((/60.5;A1(<4:?9::6;>/==CD@E@;=>028.*,24765:69:61%*
((8,4966;;863(*%(##%*,38::.$$%(,.25+-%*-02;6;C>;.(*$%'%(/$)-3-.91..)+488
```

Sliding window method:

```
# Sliding window method
python TrimReads.py sample.fastq --window-threshold 20 --window-size 5
```

The sequence after trimming will be:

```
TACCGTGACAAGAAAGTTGTCGGTGTCTTTGTGTTTCTGTTGGTGCTGATATTGCCGAAAATCGGTAGACGCTACGGACT
AAATCCGCTTCTTCCTGAAATGCGGGTTTG
```

Final output will include basic information : Sequence encoding， Original length, Trimmed length, Bases trimmed, Left trim (base), Right trim (base) of total sequence and each sequence. And of course the final `FASTQ` file of all the sequence after trimming will be given.

Output,for example,when the input just like showing above:

```
Sequence 83b951be-cf09-402a-8b19-48105583c067:
   Original length: 211
   Trimmed length: 110
   Bases trimmed: 101
   Left trim (window): 46
   Right trim (window): 55
```

And in the `sample_trimmed.fastq` file ,we will see:

```
@83b951be-cf09-402a-8b19-48105583c067
runid=34c547ba84ed3971a437c6252da360118a5fabd7 sampleid=1 read=68633 ch=80
start_time=2019-10-18T05:04:31Z
TACCGTGACAAGAAAGTTGTCGGTGTCTTTGTGTTTCTGTTGGTGCTGATATTGCCGAAAATCGGTAGACGCTACGGACT
AAATCCGCTTCTTCCTGAAATGCGGGTTTG
+
6-798:5-(((/60.5;A1(<4:?9::6;>/==CD@E@;=>028.*,24765:69:61%*((8,4966;;863(*%
(##%*,38::.$$%(,.25+-%*-02;6;C>;.(
```

Conclusion

This tool provides an efficient solution for quality trimming of high-throughput sequencing data. By implementing both base-level and window-based approaches, it accommodates diverse quality profiles in sequencing reads. The use of `argparse` ensures user-friendly parameter customization.

Perspective

Future enhancements could include

- The tool’s modular design also facilitates extension to additional trimming strategies or file formats, such as extending compatibility to FASTA, BAM, or CRAM formats.
- Currently, only one method (base or window) can be used at a time. Future versions could allow combined trimming (e.g. base trimming first, then window trimming).
- Additional Quality Metrics， including per-base quality plots (similar to FastQC) in the output report.