NGS Quality Control

Projektmanagement im Softwarebereich - SeqAn 2013

Daniel Kersting   Antje Oldenburg

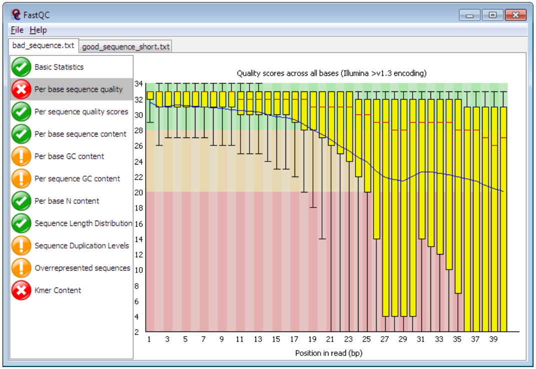
Berlin, 24. April, 2013

Background NGS Quality Control

Vorbilder: FastQC FastX, PrinSeq

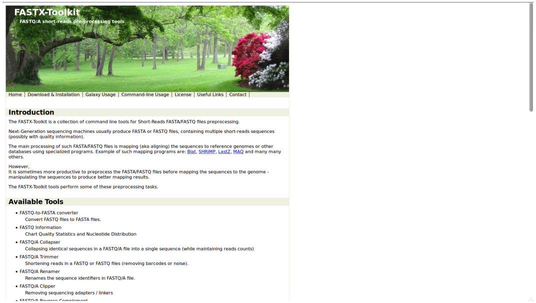
Background NGS Quality Control

Vorbilder: FastQC FastX, PrinSeq


FastQC
File Help
bad_sequence.txt
good_sequence_short.txt
Basic Statistics
Per base sequence quality
Per sequence quality scores
Per base sequence content
Per base GC content
Per sequence GC content
Per base N content
Sequence Length Distribution
Sequence Duplication Levels
Overrepresented sequences
Kmer Content
Quality scores across all bases (Illumina >v1.3 encoding)
Position in read (bp)

Background NGS Quality Control

Vorbilder: FastQC FastX, PrinSeq


FASTX-Toolkit
FASTQ/A short-reads pre-processing tools
Home | Download & Installation | Galaxy Usage | Command-line Usage | License | Useful Links | Contact

Introduction

The FASTX-Toolkit is a collection of command line tools for Short-Reads FASTA/FASTQ files preprocessing.

Next-Generation sequencing machines usually produce FASTA or FASTQ files, containing multiple short-reads sequences (possibly with quality information).

The main processing of such FASTA/FASTQ files is mapping (aka aligning) the sequences to reference genomes or other databases using specialized programs. Example of such mapping programs are: Blat, SHRiMP, LastZ, MAQ and many many others.

However,
It is sometimes more productive to preprocess the FASTA/FASTQ files before mapping the sequences to the genome - manipulating the sequences to produce better mapping results.

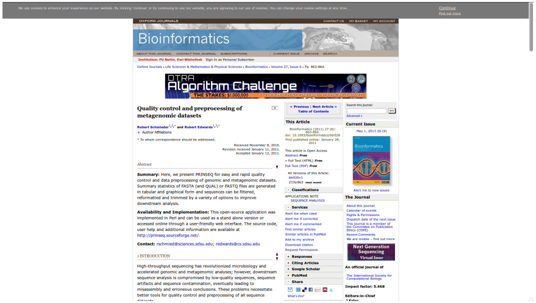
The FASTX-Toolkit tools perform some of these preprocessing tasks.

Available Tools

- FASTQ-to-FASTA converter
  Convert FASTQ files to FASTA files.
- FASTQ Information
  Chart Quality Statistics and Nucleotide Distribution
- FASTQ/A Collapser
  Collapsing identical sequences in a FASTQ/A file into a single sequence (while maintaining reads counts)
- FASTQ/A Trimmer
  Shortening reads in a FASTQ or FASTQ files (removing barcodes or noise).
- FASTQ/A Renamer
  Renames the sequence identifiers in FASTQ/A file.
- FASTQ/A Clipper
  Removing sequencing adapters / linkers
- FASTQ/A Reverse-Complement

Background NGS Quality Control

Vorbilder: FastQC FastX, PrinSeq



We use cookies to enhance your experience on our website. By clicking 'continue' or by continuing to use our website, you are agreeing to our use of cookies. You can change your cookie settings at any time.

Quality control and preprocessing of metagenomic datasets

Robert Schmieder[1,2,*] and Robert Edwards[1,3,*]

+ Author Affiliations

* To whom correspondence should be addressed.

Received November 8, 2010.
Revision received January 11, 2011.
Accepted January 12, 2011.

Abstract

Summary: Here, we present PRINSEQ for easy and rapid quality control and data preprocessing of genomic and metagenomic datasets. Summary statistics of FASTA (and QUAL) or FASTQ files are generated in tabular and graphical form and sequences can be filtered, reformatted and trimmed by a variety of options to improve downstream analysis.

Availability and Implementation: This open-source application was implemented in Perl and can be used as a stand alone version or accessed online through a user-friendly web interface. The source code, user help and additional information are available at http://prinseq.sourceforge.net/.

Contact: rschmied@sciences.sdsu.edu; redwards@cs.sdsu.edu

1 INTRODUCTION

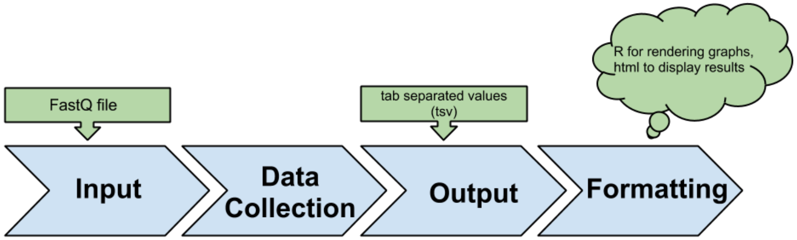
High-throughput sequencing has revolutionized microbiology and accelerated genomic and metagenomic analyses; however, downstream sequence analysis is compromised by low-quality sequences, sequence artifacts and sequence contamination, eventually leading to misassembly and erroneous conclusions. These problems necessitate better tools for quality control and preprocessing of all sequence

Bioinformatics (2011) 27 (6): 863-864.
doi: 10.1093/bioinformatics/btr026
First published online: January 28, 2011

This article is Open Access
Abstract *Free*
» Full Text (HTML) *Free*
Full Text (PDF) *Free*

All Versions of this Article:
btr026v1
27/6/863 *most recent*

Functional Overview


FastQ file
tab separated values (tsv)
R for rendering graphs, html to display results
Input
Data Collection
Output
Formatting

Statistical Information we plan to show

and how we collect the data for it

We grouped our quality metrics into groups, depending on wether it describes data

1. about the whole file
2. about all reads
3. all positions in all reads

This reflects where we collect the data:

- open seqan::SequenceStream
    - read Record
        - read nucleotide and quality and update data object

Simple statistics about all positions in all reads

and how we collect the data for it

Our data object holds:

- a 2-dimensional counter for score per position
- a 2-dimensional counter for Dna5 element per position

This will enable us to collect the following data:

- basic quality distribution data: median, mean, quantiles (10,25,75,90)
- distribution of [A,C,G,T]
- GC percent content
- N Content

Statistics about about all reads

and how we collect the data for it

Our data object holds:

- a 2-dimensional counter for score per position
- counter for encountered sequence lengths

This will enable us to collect the following data:

- mean qualities distribution
- sequence length distribution

Statistics about the whole file

and how we collect the data for it

Our data object holds:

- all program arguments, explicitly set and not
- a 2-dimensional counter for score per position
- a 2-dimensional counter for Dna5 element per position
- a counter for the number of records encountered

This will enable us to collect the following data:

- input filename and format
- which scoring system was used
- total number of sequences
- overall quality score average of all bases in all sequences
- overall GC percent
- overall N percent

K-mer content statistics

k-mer distribution

Goal: Find plentiful k-mer

Problem: memory and time contraints

Strategy: found and count

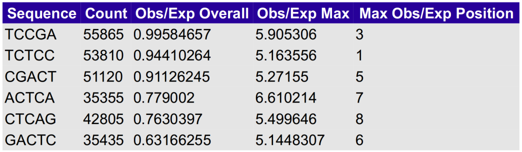
1. create k-mer index
2. count each k-mer for each position

Implementation

```
String<Dna5> genome
Index<String<Dna5>, IndexEsa<> > esaIndex(genome)
Finder<Index<String<Dna5>, IndexEsa<> > > esaFinder(esaIndex)
find(esaFinder) for each k-mer
save in Matrix (position,all k-mer,count)
```

K-mer content statistics

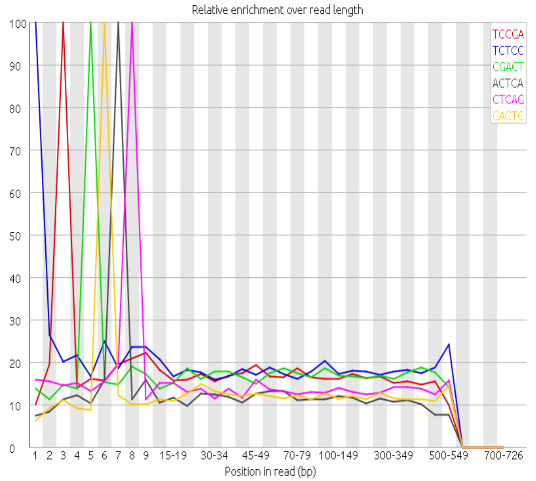
k-mer distribution



Sequence	Count	Obs/Exp Overall	Obs/Exp Max	Max Obs/Exp Position
TCCGA	55865	0.99584657	5.905306	3
TCTCC	53810	0.94410264	5.163556	1
CGACT	51120	0.91126245	5.27155	5
ACTCA	35355	0.779002	6.610214	7
CTCAG	42805	0.7630397	5.499646	8
GACTC	35435	0.63166255	5.1448307	6

K-mer content statistics

k-mer distribution



sequence duplication

and how we collect the data for it

Goal	Detect fully duplicated reads
Problem	memory and time contraints
Strategy	for n reads contained in dataset 1. collect the first k reads and create a suffix array 2. starting from k, increment a counter if exact matches occur
Implementation	▶ `<StringSet<Dna5> >` as haystack ▶ `appendValue` to haystack k times ▶ `Index` as `IndexEsa<>` ▶ `clear` and `find` on `IndexEsa<>`
Implications	If used for 200.000 reads of length 100, needed memory for this would be 20 MBytes.

Data output and formatting

Steps to create a visual summary

1. output tsv file with tabular data from our data
2. secondary app: script that does
   2.1 creates a static R script
   2.2 calls R script on tsv to create png's
   2.3 creates a static html file that displays results

NICE-TO-HAVEs

what we want to add if time allows

- read for data formats: fastq compressed, bam, sam,
- sequence complexity statistics
- KNIME integration
- Galaxy integration

Milestones

1st week	testing and implementation of a functionally minimal version that works through all steps
2nd week	testing an implementation of all basic statistics (A) and k-mer content (D)
3rd week	testing and implementation of sequence duplication (A) and output refinement (D)
4th week	buffer for surprises, testing and implementation of `NICE-TO-HAVE` features (A+D)