Last modified: December 2019
Last reviewed: December 2019

Pooled Screening Deconvolution Using PoolQ3

Introduction

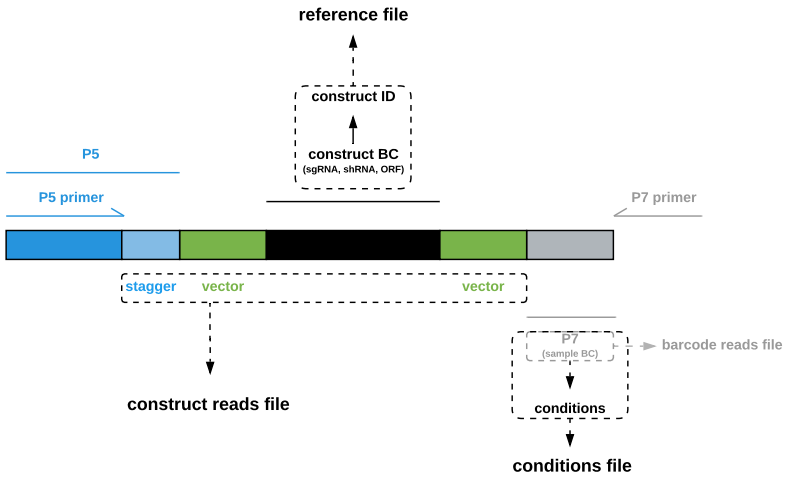
In pooled screens, a population of cells is infected with a lentiviral pool containing a mixture of constructs with the objective of integrating one perturbagen per cell, then separated into test and control populations. At the conclusion of the screen, test and control cell populations are harvested, genomic DNA is purified, and the integrated perturbagens containing a short DNA sequence (barcode) are amplified by PCR. Sequencing then determines the abundance of each perturbagen in each sample. To maximize utilization of sequencing lanes and reduce cost, multiple samples can be sequenced simultaneously. PoolQ deconvolutes these sequencing files and quantitates perturbagen barcodes in each sample. There are now two versions of the PoolQ software: PoolQ2 and PoolQ3. This document covers PoolQ3.

Definitions

- **construct barcode**: DNA barcode sequence identifying the sgRNA, shRNA or ORF perturbagen within the DNA sequences
- **construct ID**: every sgRNA, shRNA or ORF has been given an ID (Example: ID BRDN0000561048 currently corresponds to a sgRNA for NF1 gene).
- **sample barcode**: sequence in the P7 primer that gets incorporated in the PCR product. Given that a different P7 primer has been aliquoted in each well of a PCR plate, during deconvolution that sample barcode will be associated with every construct from the same sample
- **stagger**: regions of varying length in the P5 primers, inserted to prevent mono-template reads of the vector
- **fuzzy matching**: a method that attempts to find a likely approximate match when an exact match is not found; allows 1 base mismatch, but does not consider indels
- **search prefix:** Short DNA sequence that should precede all the construct barcodes in the reads file.

The PoolQ software supports a wide variety of screening and sequencing possibilities. This protocol describes the use of PoolQ for deconvolution and quantitation in the most common scenario. In this case, PCR amplification has introduced a variable-length DNA sequence via a P5 primer that will occur at the beginning of each read.

The sequencing machines split reads into two components, which are written to separate FASTQ files. One file contains the stagger followed by a portion of the vector, followed by the construct barcode. This file is called the construct reads file. The other file contains the sample barcode only and is called the sample reads file.



The inclusion of the variable length stagger region means that the construct barcode location varies from read to read. In order to find the construct barcode within a read, PoolQ makes use of the small portion of vector that exists between the stagger and the construct barcode. As long as all constructs in the sequencing lane use a similar vector, the construct barcodes may be found by locating the short DNA sequence from the vector that immediately precedes each construct barcode. We call this sequence the construct search prefix. Vectors may vary widely, but as long as the few bases before the construct barcode are the same, they can be pooled and deconvoluted together.

Prerequisites

Running PoolQ for this scenario requires the following input files:

- Construct reads file
- Barcode reads file
- Reference file
- Conditions file
- Platform reference file (optional)

You will also need to know the following information:

- Construct barcode search prefix
- Sample barcode start index (usually 0)

In addition, you will need a Java Runtime Environment (JRE) for Java 8 or later.

➢ Construct Reads File

The construct reads file (.fastq, .txt) contain the sequencing reads including the construct barcodes. PoolQ calls this the row reads file since the construct barcodes become the rows in the counts file.

➢ Barcode Reads File

The barcode file (.fastq) contains the sample barcode reads. PoolQ calls this the column reads file, since the sample names are used to form the columns in the scores file.

➢ Reference File

The reference file matches construct barcode to construct ID. PoolQ calls this the row-reference file since it contains the barcodes that will make up the rows of the counts file.

- Comma-separated (.csv) or tab-separated (.tsv or .txt)
- Use proper quoting if fields contain delimiters (commas or tabs)
- No column headers
- Column 1: construct barcode
- Column 2: construct ID
- Extra columns are allowed but ignored
- Every construct barcode must have the same length and must contain only A, T, C or G
- A construct ID cannot occur more than once in the file

➢ Conditions File

Each P7 primer corresponds to a particular PCR well. Therefore, the conditions file has a column with the sample barcode, and a column with the conditions description. PoolQ calls this the column reference file, since it contains the conditions that form the columns of the counts file.

- Comma-separated (.csv) or tab-separated (.tsv or .txt)
- Use proper quoting if fields contain delimiters (commas or tabs)
- No column headers

- Column 1: sample barcode
- Column 2: sample ID or condition
- Every barcode must have the same length and must contain only A, T, C or G
- A sample barcode cannot occur more than once in the file
- Multiple sample barcodes may map to the same condition. For example, the same cell pellet may have been amplified across multiple PCR wells

➢ Platform Reference File

Optional input file containing master list of known constructs sequences and their constructs IDs, provides constructs IDs for barcodes encountered during the PoolQ run that were not expected to occur.

➢ Barcode Policy

PoolQ uses barcode policies to locate sample and construct barcodes within the reads. It currently supports three varieties of barcode policies: fixed, prefix, and template. In this protocol, we describe a scenario that uses a fixed policy to locate sample barcodes and a prefix policy to locate a construct barcode. These are passed to PoolQ using the arguments --col-barcode-policy and --row-barcode-policy, respectively.

Fixed Barcode Policy

A fixed barcode policy indicates to PoolQ that barcodes occur at a fixed, known location within reads. Fixed barcode policies are often used for the sample barcodes. A fixed barcode policy starting at zero-based index N is written FIXED:N, so to locate barcodes at the very beginning (index 0), provide FIXED:0.

Prefix Barcode Policy

The prefix barcode policy indicates that a short DNA sequence from the vector that immediately precedes the construct barcodes. The following table describes the standard search prefix for most Broad constructs:

Construct Type	**Search Prefix**
sgRNA	CACCG
shRNA	ACCGG
ORF	GACGA

Prefix search may optionally be limited to certain regions within the read; to express this, provide the zero-based index of the first and/or last base in the read where the *prefix* may be found. Here are some examples:

- PREFIX:CACCG - searches the whole read for a CACCG prefix
- PREFIX:ACCGG@18 - searches the read starting at the 19th base for an ACCGG prefix
- PREFIX:GACGA@-49 - searches the read up to the 50th base for a GACGA prefix
- PREFIX:CACCG@18-49 - searches the 19th through 50th base for a CACCG prefix

Example Data

The PoolQ distribution includes sample data files for a number of common scenarios. The data for this scenario is in test-data/scenario4. Here is the first FASTQ record from the file scenario4.barcode_1.fastq, which contains the sample barcodes:

@HWUSI-EAS100R:6:23:398:3989#1
AACTCACG
+
4<<8-767

Here are two example FASTQ records from scenario4.1.fastq, which contains construct barcodes:

@HWUSI-EAS100R:6:23:398:3989#1
ATTACATATTAATGGGACAGGCGGCCACCGCCATAATACTAGGTGACAGA
+
2059-@0/7798;98:3<7;02=7356A.34;:965D1798=:<75<E55
@HWUSI-EAS100R:6:23:398:3989#2
CTCATTAATGGGACAGGCGGCCACCGCCTCCGTTCTGATACTCACAATTA
+
:3<,=;:;@8,2?865?79<<:=::8;:77?6;780(5252;3B139545

First, notice that the first line of the sample barcode FASTQ record and the first line of the first construct barcode FASTQ record are identical. The read IDs are used to coordinate reads between the two files. The sample barcode (highlighted green) is an 8mer. The search prefix CACCG is highlighted in orange. The construct barcodes (highlighted blue) are 20-mers. The variable-length "stagger" region located at the beginning of the construct barcode reads is highlighted in red.

Running PoolQ

The PoolQ distribution contains sample input files in the test-data directory. If you have GNU Make installed, you can test this scenario by running:

```
% make test-scenario-4
```

This will print the full command line, run PoolQ on the appropriate input files, and verify that the scores match a known scores file.

We run PoolQ using the following command line:

```
poolq3.sh \
  --compat \
  --col-reference Conditions.csv \
  --row-reference Reference.csv \
  --row-reads scenario4.1.fastq \
```

```
--col-reads scenario4.barcode_1.fastq \
--row-barcode-policy PREFIX:CACCG@18 \
--col-barcode-policy FIXED:0
```

PoolQ Output Files

➢ Counts File

The counts file is a text file that contains a simple matrix of the read counts. The columns represent the experimental conditions, and the rows correspond the construct barcode sequences. The individual values in each row are separated by tabs. If you plan on loading the scores file into a spreadsheet application such as Excel, then we recommend using a file extension such as .txt, that your spreadsheet application will recognize as being a text file. When opening the file in Excel, you will probably be prompted with a dialog asking you to describe the structure of the file. In the section about separator options, be sure that the check box for "Tab" is selected.

➢ Normalized Counts File

The normalized counts file contains a log-normalized view of the counts file. The format of the file is the same, but the counts are normalized by the following formula:

1. Take the raw read count for the construct ID and the condition
2. Divide by the total number of reads for that condition that matched a construct barcode found in a reference file
3. Multiply by a constant factor of 1 million
4. Add one
5. Take the log (base 2)

➢ The Unexpected Sequence File

The unexpected sequence file contains a report that describes briefly the collection of sequences found in the position where a construct barcode was expected during the run.

➢ Quality File

In addition to the scores file, PoolQ writes a quality file containing a variety of useful metrics to help users understand the quality of their sequencing data.

➢ The Correlation File

The correlation file contains a pairwise correlation matrix comparing the log-normalized per-construct counts for each experimental condition. The correlation metric is the Pearson product-moment correlation.

FAQ

Q: Does PoolQ handle paired-end sequencing?
A: No. As of now, there are no screen deconvolution scenarios that require paired-end reads.

Q: Does PoolQ support variable-length sample barcodes?
A: No, all sample barcodes must be of the same length.

Q: Do we support variable-length construct barcodes?
A: Not directly. If construct barcodes vary in length, the best case solution is to "pad" the shorter construct barcodes with bases from the vector backbone to achieve a reference file where all construct barcodes are the same length.

Q: How does PoolQ handle ambiguous matches?
A: Exact matches always take precedence over fuzzy matches. Even when configured to support fuzzy matches, while processing a read, if PoolQ finds an exact match it will tally the match and continue with the next read. It will begin considering fuzzy matches only if no exact match is found. If multiple construct barcodes in the reference file are 1-base mismatches to the construct barcode found in the sequencing data, PoolQ has two modes of conflict resolution. In the strictest mode (the default), it simply throws out the read. Alternately, PoolQ can be configured to count the read as matching to all possible 1-base mismatches.

Q: Should I use fuzzy matching?
A: Fuzzy matching maximizes PoolQ's ability to extract construct barcodes from the sequencing data. However, it also comes at a performance penalty. It is sometimes helpful at the start of an experiment to run data through with fuzzy matching turned off, as a check of the data. However, once the experimental set-up has been verified, under most circumstances, the best results will come from fuzzy matching.

Q: Does fuzzy matching support indels?
A: No, only 1-base substitutions or Ns