PoolQ

PoolQ is a counter for indexed samples from next-gen sequencing of pooled DNA.

This documentation covers PoolQ version 3.2.0 (last updated 11/19/2019).

Background

The Broad Institute Genetic Perturbation Platform (GPP) uses Illumina sequencing to tally the results of pooled screens. PoolQ can process pooled screens representing a variety of genetic perturbation techniques, including CRISPR/Cas9, RNAi, and barcoded ORFs. We perform the following steps to generate one or more files of sequencing reads (or "reads files") for the pooled screen:

1. We start with multiple genomic DNA samples; generally it is assumed that the samples share a common pool of constructs but vary by some experimental condition.
2. We apply PCR to the genomic DNA samples using primers that amplify the the barcode region of each construct in the pool and attach a fixed-length barcode uniquely identifying the DNA sample. In this way, all the amplification products will contain the inserted sample barcode.
3. We pool all the samples, normalized to equalize barcode representation, and run them in a single sequencing lane.
4. The sequencing process generates an output file for each sequencing lane.

PoolQ comes into play after the sequencing files have been generated. A single PoolQ invocation processes one sequencing run. It attempts to parse out the unique barcodes representing the sample and construct from each read. It maps sample barcodes to conditions and construct barcodes to construct IDs, and generates a matrix with the conditions as columns and the constructs as rows.

Changes in PoolQ 3.0

PoolQ was completely rewritten for version 3.0. The new code is faster and the codebase is much cleaner and more maintainable. We have taken the opportunity to make other changes to PoolQ as well.

- There are substantial changes to the command-line interface for the program.
- The default counts file format has changed slightly, although there is a command-line argument that indicates that PoolQ 3.0 should write a backwards-compatible counts file. The differences are in headers only; file parsers should be able to adapt easily.
- The quality file has changed somewhat. Importantly, the definition of certain statistics has changed slightly, so quality metrics cannot be directly compared between the the new and old versions. In addition, we no longer provide normalized match counts.

Command-line interface changes

The PoolQ command-line interface has been reworked significantly in PoolQ 3.0. There is no current migration guide between the two interfaces; however virtually every possible configuration of PoolQ 2.0 is also possible with PoolQ 3.0. To update your command-line arguments, consult the commands specified in the test data `Makefile` for each scenario. Furthermore, there are extensive explanations of the various parameters in subsequent sections of this document.

Counts file changes

One of the key internal changes to PoolQ 3.0 was to avoid using the terms "construct barcode" and "sample" barcode prescriptively. PoolQ 3.0 favors the terms *row barcode* and *column barcode*, since the actual concepts these barcodes represent should be of limited concern to PoolQ. As such, the default counts file format changes its terminology slightly. Where the PoolQ 2.X counts file header read:

```
ConstructBarcode        Construct IDs
```

the PoolQ 3.0 counts file header reads:

```
Row Barcode             Row Barcode IDs
```

Note that the structure of the files have not changed, so any parser which ignores these columns in the header need not make any changes. Parsers that use these columns should be able to adapt trivially to the new terminology. Moreover, PoolQ 3.0 supports a command line option `--compat` which specifies that the counts file contain the former terminology.

Quality file changes

- Changed the "Total Reads" metric to report the total number of reads in the file i.e., do not discard reads with Ns in the sample (conditions) barcode region. These values will be higher in PoolQ 3 than they would have been previously for the same input data.
- Changed the "Matching Reads" metric to include only reads that matched both a known row and column barcode; formerly this number included all reads that matched a row barcode even if they did not match a column barcode. These values will be lower in PoolQ 3 than they would have been previously for the same input data.
- Changed the "1-base mismatch reads" metric to report the number of one-base mismatches used to match a row barcode only if the column barcode also matched; as above, this formerly included reads that matched a row barcode even if the column barcode did not match. These values will be lower in PoolQ 3 than they would have been previously for the same input data.
- Changed the "Overall % Match: (Construct+Sample Barcode)/total reads" heading to simply "Overall % match", and report the % as the number

of reads matching both a row and column barcode divided by the total number of reads in the file (i.e., do not exclude reads with Ns in the sample barcode region from the denominator). These values will be lower in PoolQ 3 than they would have been previously for the same input data.
- Removed the metric "Avg. counts for unknown sample barcodes"
- Removed the metric "Normalized avg. counts for unknown sample barcodes"
- Changed the "Read counts for sample barcodes without associated conditions" metric to be specific to cases where a row barcode matched but a column barcode did not; report only the top 100 unexpected sample barcodes.
- Removed the "Construct barcodes matching multiple construct IDs" section

PoolQ Use Cases

The most common usages of PoolQ falls into one of six basic scenarios, defined by the number of sequencing files it needs to read and the way it will locate barcodes within those files. PoolQ can process sequencing results with reads wholly contained within one sequencing file, or with each read partially represented within two sequencing files. It can locate barcodes that occur at a constant offset within the reads or that occur at a variable offset within the reads. For barcodes whose position varies within the reads, PoolQ has two approaches for identifying the barcode region: first, by locating a fixed DNA sequence known to immediately precede the construct barcode; and second, by tiling a template sequence along the read until a sequence of DNA matches the template.

We describe the combination of these two dimensions in the following way:

- Scenario 1: Fixed offset, one sequencing file
- Scenario 2: Variable offset, one sequencing file
- Scenario 3: Fixed offset, two sequencing files
- Scenario 4: Variable offset, two sequencing files
- Scenario 5: Template match, one sequencing file
- Scenario 6: Template matching, two sequencing files

These scenarios require different command-line options. The PoolQ distribution contains sample data representing each scenario and presents example command lines for processing each case.

PoolQ Inputs

PoolQ requires 3 input files to run: the (FASTQ or SAM/BAM) file containing the reads (the reads file); a file mapping barcodes to conditions (the conditions file); and a file mapping construct barcode sequences to construct IDs (the reference file). PoolQ can also take an optional input file (the platform reference file) describing known barcodes that are not expected to be present in the sequencing reads. There are also a variety of optional settings that specify how

PoolQ processes the reads.

Reads File(s)

The reads file is a either a standard FASTQ file or a SAM/BAM file, containing the reads. It may or may not be compressed with gzip. If the file is a FASTQ file, then the file extension should be `.fastq` or `.txt` (case-insensitive). If it is a BAM file, then the file extension should be `.bam`, and for a SAM file, the extension should be `.sam` (case-insensitive). Files that do not follow these naming conventions can still be used, but you must specify the file type explicitly.

If the file is compressed with gzip, then the file must have an extra `.gz` extension after the `.fastq`, `.txt`, or `.bam` extension.

A barcode representing a sample and a construct are parsed out of each read. The length of the sample barcode is inferred from the contents of the conditions file. The length of the construct barcode is inferred from the contents of the reference file. By default, the sample barcode starts at the first base of the read, and the construct barcode starts at the 17th base of the read, but both of these defaults can be overridden with an optional flag.

In some cases, reads are divided into two separate FASTQ files (BAM/SAM is not supported in this case). One file contains the sample barcodes, while the other contains the construct barcodes. In this case, the file with shorter reads is assumed to contain the sample barcodes, while the file with longer reads is assumed to contain the construct barcodes. This mode of operation requires that FASTQ record IDs match between the two input files.

Reference Files

Reference files map DNA barcodes to their associated identifiers. PoolQ uses two reference files to define the rows and columns found in the counts file.

Column Reference File   The column reference file (formerly known as the "conditions" file) maps DNA barcodes to the IDs that will be presented as the columns in the counts files. Generally the columns correspond to sample barcodes, which identify unique gDNA samples representing different experimental conditions.

- It is a file with at least two columns
- The first column contains the sample barcodes sequence and the second column contains the condition descriptions
- All other columns are ignored
- Every row must have the same number of columns
- The columns can be separated by either commas or tabs
- The file may not include any column headers or any extra columns
- Barcodes must contain only A, T, C or G
- A barcode cannot occur more than once in the file

4

- Every barcode in the conditions file must have the same length
- You can have multiple barcodes mapping to the same condition, but be aware that if you do so, the reads for those barcodes will be counted together, resulting in a single column in the counts file for that condition

Row Reference File The row reference file (formerly simple the "reference file") contains the DNA barcodes that will make up the rows in the counts file. Generally, the rows correspond to barcodes representing shRNAs, sgRNAs, ORFs, or other constructs utilized in a pooled experiment.

- It is a file with at least two columns
- The first column contains the construct barcode sequences, and the second column contains the construct IDs
- All other columns are ignored
- Every row must have the same number of columns
- The columns can be separated by either commas or tabs
- The file may not include any column headers or any extra columns
- Barcode sequences must contain only A, T, C or G
- A construct ID cannot occur more than once in the file
- Every construct barcode in the reference file must have the same length
- You can have multiple construct IDs mapping to the same barcode sequence; the counts file will report the counts for the construct barcode alongside a comma-separated list of associated construct IDs.

The Platform Reference File The platform reference file is an optional input file whose format is identical to that of the reference file. It consists of a master list of known construct sequences and their construct IDs. This file is used to provide construct IDs for barcodes encountered during the PoolQ run that were not expected to occur. A construct is expected to occur only if it is present in the reference file.

Row Matcher

This argument determines how PoolQ matches DNA barcodes to those in its row reference database. There are currently two supported values, "exact" and "mismatch". Using "exact" requires every base to match perfectly; "mismatch" allows up to one base of mismatch between them (no gaps or deletions allowed).

Column Matcher

This argument determines how PoolQ matches DNA barcodes to those in its column reference database. There are currently two supported values, "exact" and "mismatch". Using "exact" requires every base to match perfectly; "mismatch" allows up to one base of mismatch between them (no gaps or deletions allowed).

Barcode policy

Barcode policies indicate how PoolQ should locate various types of barcodes in the reads. PoolQ supports 3 basic policies: fixed-location, search prefix, template.

Fixed Location The fixed-location policy indicates that a barcode exists at a single, known position within the read. For a 0-based index `i`, you can specify a fixed strategy by passing `FIXED@i`. So a barcode located at the start of the read is specified as `FIXED@0`, while a barcode located at the twelfth base into the read is specified `FIXED@11`. Using the fixed location policy, you can optionally specify how many bases to read as the barcode (sometimes reads are too short to contain a full barcode). The length `n` is specified by adding `:n` to the end of the policy; so the policy for finding a 6-base barcode twelve bases into the read is specified as `FIXED@11:6`. If you do not specify a length, PoolQ will choose the length based on the length of barcodes found in the corresponding reference file.

Search Prefix The search prefix policy is useful when barcodes may occur at different positions within each read, assuming the barcode region is always immediately preceded by some known DNA sequence. When specifying a prefix policy, you must provide the prefix that PoolQ should look for. Optionally, you can provide additional parameters specifying a range of bases where the search prefix could occur within the read; as with the fixed location policy, you may also give a barcode length.

To use the prefix policy for a DNA prefix `s`, specify `PREFIX:s`. For example, a common DNA prefix is `CACCG`, which is specified `PREFIX:CACCG`. To indicate that the prefix should only be located beginning at the 12th base of the read, specify `PREFIX:CACCG@11`. To indicate that the prefix must occur before the 20th base of the read, specify `PREFIX:CACCG@-19`. These two parameters may be specified in conjunction: `PREFIX:CACCG@11-19`. You can optionally specify how many bases to read as the barcode (sometimes reads are too short to contain a full barcode). The length `n` is specified by adding `:n` to the end of the policy; so the policy for finding a 6-base barcode twelve bases into the read is specified as `PREFIX@11-19:6`. If you do not specify a length, PoolQ will choose the length based on the length of barcodes found in the corresponding reference file.

Template The template matching policy is useful when barcodes may occur at different positions from read to read, but the barcode can be identified by examining the surrounding context. Template matching also allows for the possibility that barcodes are not made up of contiguous regions. Templates are specified using IUPAC codes in upper- and lowercase. Lowercase bases are used in matching, but are not extracted as part of the barcode. Uppercase bases are used in matching *and* are extracted as part of the barcode. For example the template: `caccgNNNNctcnnnNNNNa` indicates that we are looking for a 20-base region that: * begins with `CACCG` * followed by the first four

bases of the barcode * then the sequence `CTC`, * then three more arbitrary bases * followed by the next four bases of the barcode * then a single `A` Given the read sequence `TTGCACCGTTGTCTCATGACCTATGTG`, notice that the template `caccgNNNNctcnnnNNNNa` matches only at the 4th base (indicated `3` on the ruler)

```
0         1         2
01234567890123456789012345  6
TTGCACCGTTGTCTCATGACCTATGTG
    caccgNNNNctcnnnNNNNa
```

The barcode that will be extracted is `TTGTACCT`.

To indicate that PoolQ should search for barcodes using a template `t`, specify `TEMPLATE:t`. For example, to use the template `caccgNNNNctcnnnNNNNa`, give `TEMPLATE:caccgNNNNctcnnnNNNNa`. As with the search prefix policy, it is possible to limit where PoolQ searches within each read for template matches. To indicate that the template should only be located beginning at the 12th base of the read, specify `TEMPLATE:caccgNNNNctcnnnNNNNa@11`. To indicate that the prefix must occur before the 20th base of the read, specify `TEMPLATE:caccgNNNNctcnnnNNNNa@-19`. These two parameters may also be specified in conjunction: `TEMPLATE:caccgNNNNctcnnnNNNNa@11-19`. Unlike with the fixed-location and prefix search policies, you cannot limit the length of the matched barcode.

Row Barcode Policy   The row barcode policy specifies how row barcode are located. For details on barcode policies, see the previous section.

Column Barcode Policy   The column barcode policy specifies how column barcodes are located. For details on barcode policies, see the previous section.

Count Ambiguous

This is an optional input flag that, when present, controls the handling of barcodes encountered in reads that fuzzy-match to more than one construct in the reference file. The default behavior is to discard ambiguously matching reads, so they will not be included in the counts file. If this behavior is not desired, specify this flag and all ambiguous reads will be counted for every possible matching construct barcode. As a consequence of this behavior, the sum of the counts for a given column may add up to more than the number of reads for a particular condition.

Unexpected Sequence Threshold

This is an optional argument that, when present, specifies the number of unexpected barcodes to include in the unexpected sequence report. PoolQ will report the most frequently occurring barcodes. The default value is 100.

Reads File Type

This is an optional input flag that, when present, specifies how PoolQ should treat the reads file type. By default, PoolQ will attempt to guess whether the reads file is a FASTQ, BAM, or text file based on the file name. If the filename is misleading, you can specify the file type explicitly using this flag. Valid values include BAM, FASTQ, and RAW (for plain text).

Skip Short Reads

This is an optional input flag that, when present, specifies that PoolQ should simply ignore reads that are too short to contain both barcode sequences. By default, PoolQ considers files containing short reads to be badly formed and exits. By specifying this flag, you indicate that PoolQ should simply skip these short reads; a count of the number of skipped short reads will be available in the quality file. Currently, this flag is only supported if you have selected the fixed barcode policy.

Compatibility mode

This is an optional input flag that, when present, specifies that PoolQ should write counts files that are identical to those emitted by PoolQ 2.0. As described above, the difference is in the header text. This flag is provided to make comparing counts between PoolQ 2.0 and PoolQ 3.0 easier, and to support users who parse counts files as part of their processing pipelines. This option is mutually exclusive of the GCT mode.

GCT mode

This is an optional flag that directs PoolQ to emit counts in the http://software.broadinstitute.org/cancer/software/genepattern/file-formats-guide#GCT file format, for use with tools such as GenePattern or RIGER. It is mutually exclusive of the PoolQ 2.0 compatibility mode flag described above.

PoolQ Outputs

PoolQ generates output files representing the matrix of read counts (or counts) for expected sequences, a report of read counts for unexpected sequences, and a report containing simple metrics used to help assess the overall quality of the sequencing data. There are two optional output files that contain alternative representations of the counts matrix. One contains the counts in log normalized form and the other contains read counts by barcode rather than by condition.

Counts File

The counts file is a text file that contains a simple matrix of the read counts. The columns of the matrix represent the experimental conditions, and the rows

of the matrix correspond the construct barcode sequences. The individual values in each row are separated by tabs.

If you plan on loading the counts 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 checkbox for "Tab" is selected.

Quality Report

The quality report is a simple text file containing some extra information gathered during the PoolQ run. The information reported here is intended to help you assess the quality of your data, and spot problems such as an unacceptably high frequency of uncounted reads, or mistakes in barcode tracking. We currently report:

- The total number of reads
- The total number of reads that were successfully counted
- Out of the counted reads, the total number of single base mismatches to the construct barcode
- Out of the counted reads, the percent that matched to both a known sample barcode and a known construct barcode
- The average frequency of unknown sample barcode sequences
- For each sample barcode mapped to a condition, we report:
    - the barcode
    - the condition
    - the total number of reads matching the sample barcode plus an expected construct barcode
    - the total number of reads matching the sample barcode
    - the percent of the reads for the sample barcode that matched an expected construct barcode
    - the log normalized number of matches
- For each sample barcode not mapped to a condition, we report the barcode and the total number of reads

Barcode Counts File

The barcode counts file has a similar format to the counts file, except that the columns in the matrix represent the read counts for individual DNA barcodes rather than for experimental conditions. If, based on the quality file, a particular PCR appear to have been of low quality, it is possible to reaggregate counts by condition by excluding the counts from the barcode corresponding to the failed PCR. The barcode counts file is an optional output intended to provide support for loading PoolQ data into the RNAi Informatics database. However, it is available for any consumer of PoolQ data.

Log Normalized Counts File

The log normalized counts file has the same format as the counts file, but every score is normalized according to the following procedure:

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

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. It is an optional output. Tracking unexpected sequences currently results in a substantial performance penalty. We recommend that you only generate an unexpected sequence for forensic or investigative purposes.

The unexpected sequence report contains two sections. The first section consists of a table whose rows correspond to unexpected construct barcode sequences and whose columns indicate the number of times each sequence was found for each barcode. An additional column lists the construct IDs for these sequences, if the IDs are known. These construct IDs can be provided to PoolQ via the platform reference file, described above.

The second section describes unexpected sample barcodes and the number of times an unexpected sequence appeared with each unexpected barcode. The unexpected sample barcodes are listed in descending order of the number of occurrences.

Correlation File

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

Running PoolQ

There are two different ways you can run PoolQ: using the Graphical User Interface (GUI), or using the Command Line Interface (CLI). But before you can run it, you need to download the zip file and unzip it.

Prerequisites

PoolQ is built for Java 8. To run PoolQ, you will need a JRE for version 8 or later. To compile PoolQ you will need a Java 8 JDK. You can download an appropriate JRE or JDK from Oracle at:

http://www.oracle.com/technetwork/java/javase/downloads/index
.html

Downloading and Unzipping PoolQ

You can download PoolQ from an as yet undetermined location. The file you download is a ZIP file that you will need to unzip. In most cases, this is as simple as right-clicking on the zip file, and selecting something like "extract contents" from the popup menu. This will create a new folder on your computer named `poolq-3.2.0`, with the following contents:

- `poolq3.jar`
- `poolq3.bat`
- `poolq3.sh`
- `test-data/`

Feel free to rename the folder, and to move it to wherever you want. Be aware, however, that the `.sh` and `.bat` files will only function properly if they can find the `poolq.jar` file in the same folder.

The test-data folder contains four sets of sample input data and a Makefile that will run PoolQ on each set, printing the full command line in the process. If you have GNU Make installed, you can test all four sample datasets by running:

```
% cd test-data
% make
```

You can also test them individually by running:

```
% (cd test-data; make test-scenario1)
% (cd test-data; make test-scenario2)
% (cd test-data; make test-scenario3)
% (cd test-data; make test-scenario4)
% (cd test-data; make test-scenario5)
% (cd test-data; make test-scenario6)
% (cd test-data; make test-long-template)
```

Recommended JVM Settings

We recommend the following JVM settings be provided when running PoolQ:

- `-Xmx4G`
- `-XX:+UseG1GC`

This document contains a number of example command-lines for running PoolQ; however, we only list the full JVM options once, since typing the full command becomes unwieldy and the JVM options distract somewhat from the command-line arguments that are passed to PoolQ itself. You can copy and paste the full Java command from here:

```
java -Xmx4G -XX:+UseG1GC -jar poolq3.jar
```

Running PoolQ

You can run PoolQ from any Windows, Mac, or Linux machine, but it requires
some understanding about how to launch programs from the command line on
your given operating system.

1. Open a terminal window for your operating system
2. Change directories to the `poolq-3.2.0` directory

- On Windows, run:

  ```
  poolq3.bat
  ```

- Or, on a UNIX-based machine, run:

  ```
  ./poolq3.sh
  ```

- Or, on any machine, run:

  ```
  java -jar poolq3.jar
  ```

If you successfully launched PoolQ, you should see a usage message explaining
all of the command-line options:

```
poolq 3.2.0
Usage: poolq [options]

  --row-reference <row-reference-file>
                           reference file for row barcodes (i.e., constructs)
  --col-reference <col-reference-file>
                           reference file for column barcodes (i.e., conditions)
  --global-reference <global-reference-file>

  --row-reads <row-reads-file>
                           required if reads are split between two files
  --col-reads <col-reads-file>
                           required if reads are split between two files
  --reads <reads-file>     required if reads are contained in a single file
  --row-matcher <matcher>  function used to match row barcodes against the row reference dat
  --col-matcher <matcher>  function used to match column barcodes against the column referen
  --count-ambiguous <value>
                           when true, counts ambiguous fuzzy matches for all potential row b
  --row-barcode-policy <barcode-policy>

  --col-barcode-policy <barcode-policy>

  --quality <file>
  --counts <file>
```

```
--normalized-counts <file>

--barcode-counts <file>
--correlation <file>
--unexpected-sequence-threshold <number>

--unepected-sequences <file>

--unexpected-sequence-cache <cache-dir>

--skip-unexpected-sequence-report <value>

--skip-short-reads

--compat                    Enable PoolQ 2.X compatibility mode
--gct                       Output counts in GCT format
```

At this point, you are ready to run PoolQ, supplying file names and locations for the 3 file inputs and 2 or 3 file outputs. For example:

- On Windows, run:

  ```
  poolq3.bat --reads reads.fastq --col-reference conditions.txt
  --row-reference reference.txt --row-barcode-policy
  PREFIX:CACCG@12 --col-barcode-policy FIXED:0
  ```

- Or, on a UNIX-based machine, run:

  ```
  ./poolq3.sh --reads reads.fastq --col-reference conditions.txt
  --row-reference reference.txt --row-barcode-policy
  PREFIX:CACCG@12 --col-barcode-policy FIXED:0
  ```

- Or, on any machine, run:

  ```
  java -jar poolq3.jar --reads reads.fastq --col-reference
  conditions.txt --row-reference reference.txt --row-barcode-policy
  PREFIX:CACCG@12 --col-barcode-policy FIXED:0
  ```

The Scoring Algorithm

The reads file contains the sequencing reads. PoolQ supports any of the following formats:

- FASTQ (including Solexa/Illumina variants)
- SAM
- BAM

PoolQ currently ignores any read sequence content besides the sample barcode and the construct barcode. In the future, we may check the remaining sequence to help confirm the quality of the read.

Most often, the entire construct barcode is included in the read. However, for files with very short read lengths the construct barcode sequence may be truncated. In the case of truncated construct barcode sequences, PoolQ will attempt to match based on the available construct barcode sequence prefix.

If the read does not have a sample barcode that is an exact match to a barcode found in the conditions file, then the line is not counted, except in the section of the quality report devoted to counting reads for barcodes not found in the conditions file.

Counting Reads that Match a Barcode

The PoolQ scoring algorithm always attempts to match construct barcodes exactly to to one of the sequences provided in the reference file first. If an exact match is found, then only the exact match is counted.

If an exact match is not found, PoolQ will attempt to match to a known construct barcode sequence allowing a single nucleotide mismatch. An N in the construct barcode sequence is considered a single nucleotide mismatch. The exact match setting allows you to override the single nucleotide mismatch behavior and score only exact matches.

If a construct barcode sequence is a single nucleotide mismatch to two or more different barcodes, PoolQ will discard the read by default. It is possible to override this behavior as well with the include ambiguous setting, in which case PoolQ counts the read for every construct barcode sequence that is a single nucleotide mismatch.

If PoolQ matches a read to a sample barcode that is mapped to a condition, and a construct barcode that is mapped to one or more construct IDs, then the counts are incremented for all of the matching condition/construct ID pairs.

Construct barcodes are counted as unexpected sequences if they are not successfully matched by the above procedure.

Contact Us

Your feedback of any kind is much appreciated. Please email us at rnaiinformatics@broadinstitute.org.