dropEst Documentation

Release 0.8.5

Viktor Petukhov, Peter Kharchenko

Jun 07, 2019

Table of Content:

Pipeline for estimating molecular count matrices for droplet-based single-cell RNA-seq measurements.

CHAPTER 1

News

1.1 [0.8.5] - 2018-11-14

- Fixed several bugs
- Support for SPLiT-seq
- dropTag now able to trim and filter gene reads based on quality
- Pipeline can be installed with `make install`

See Changelog for the full list.

CHAPTER 2

General processing steps

1. **dropTag**: extraction of cell barcodes and UMIs from the library. Result: demultiplexed .fastq.gz files, which should be aligned to the reference.
2. **Alignment** of the demultiplexed files to reference genome. Result: .bam files with the alignment.
3. **dropEst**: building count matrix and estimation of some statistics, necessary for quality control. Result: .rds file with the count matrix and statistics. *Optionally: count matrix in MatrixMarket format.*
4. **dropReport** - Generating report on library quality.
5. **dropEstR** - UMI count corrections, cell quality classification

CHAPTER 3

Examples

Complete examples of the pipeline can be found at EXAMPLES.md.

Here are results of processing of neurons_900 10x dataset.

CHAPTER 4

Supported protocols

- 10x
- inDrop
- iCLIP
- SPLiT-seq
- Seq-Well
- Drop-seq

CHAPTER 5

Citation

If you find this pipeline useful for your research, please consider citing the paper:

Petukhov, V., Guo, J., Baryawno, N., Severe, N., Scadden, D. T., Samsonova, M. G., & Kharchenko, P. V. (2018). dropEst: pipeline for accurate estimation of molecular counts in droplet-based single-cell RNA-seq experiments. Genome biology, 19(1), 78. doi:10.1186/s13059-018-1449-6

5.1 Setup

5.1.1 System requirements

- Boost >= 1.54
- BamTools library >= 2.5.0
  - Note that some linux distributions have separate packages for the library and the executable, i.e. on Ubuntu you need `libbamtools-dev`, but not `bamtools`.
  - or you can build it locally and then specify the location of the build when running cmake (e.g. `cmake -D BAMTOOLS_ROOT=/home/username/bamtools .`)
- Zlib *(was tested on 1.2.11 version)*
- Bzip2 *(was tested on 1.0.5 version)*
- R >= 3.2.2 with packages:
- Rcpp
- RcppEigen
- RInside
- Matrix
- Compiler with c++11 support *(was tested with gcc >= 4.8.5 and CLang 3.9.1)*
- CMake >= 3.0

5.1.2 Installation

Install R packages:

```
install.packages(c("Rcpp","RcppEigen", "RInside", "Matrix"))
```

Clone this repository:

```
git clone https://github.com/hms-dbmi/dropEst.git
```

Build (**replace ‘‘installation/path‘‘ with path to the folder, where you want to install the pipeline**):

```
mkdir dropEst/build
cd dropEst/build
cmake .. && make
```

Install:

```
make install
```

After this, `droptag`, `dropest` and `dropReport.Rsc` binaries must be available at `installation/path/bin/` folder.

5.1.3 Manual installation of dependencies

Here is the instruction on how to install specific versions of libraries to the local folder (i.e. `~/local/`). Let's store this directory in `LOCAL_LIBS` variable:

```
export LOCAL_LIBS=$HOME/local/
```

To create this directory use:

```
mkdir $LOCAL_LIBS
```

To add installed libraries to `PATH` use:

```
export PATH=$LOCAL_LIBS/bin:$LOCAL_LIBS/usr/local/bin/:$PATH
```

CMake

Download version 3.12:

```
wget https://cmake.org/files/v3.12/cmake-3.12.0-rc1.tar.gz
tar xvf cmake-3.12.0-rc1.tar.gz
cd cmake-3.12.0-rc1
```

Build and install:

```
./bootstrap --prefix=$LOCAL_LIBS
make
make install
```

For the detailed instruction see instruction page.

Zlib

Download version 1.2.11:

```
wget https://zlib.net/zlib-1.2.11.tar.gz
tar xvf zlib-1.2.11.tar.gz
cd zlib-1.2.11
```

Build and install:

```
./configure --prefix=$LOCAL_LIBS
make
make install
```

BamTools

Clone repository, version 2.5.0:

```
git clone https://github.com/pezmaster31/bamtools.git
cd bamtools
git reset --hard 94f072
```

Build and install:

```
mkdir build && cd build
cmake ../
make
make install DESTDIR=$LOCAL_LIBS
```

For the detailed instruction see instruction page.

Bzip2

Download version 1.0.6:

```
wget http://www.bzip.org/1.0.6/bzip2-1.0.6.tar.gz
tar xvf bzip2-1.0.6.tar.gz
cd bzip2-1.0.6
```

Build and install:

```
make -f Makefile-libbz2_so
make install PREFIX=$LOCAL_LIBS
cp -a libbz2.so* $LOCAL_LIBS/lib/
ln -s $LOCAL_LIBS/lib/libbz2.so.1.0 $LOCAL_LIBS/lib/libbz2.so
```

For the detailed instruction see this page.

Boost

Download version 1.60:

```
wget http://sourceforge.net/projects/boost/files/boost/1.60.0/boost_1_60_0.tar.gz
tar xzf boost_1_60_0.tar.gz
cd boost_1_60_0
```

Build and install:

```
./bootstrap.sh --with-libraries=filesystem,iostreams,log,system,thread,test
./b2 cxxflags="-std=c++11" include="$LOCAL_LIBS/include/" search="$LOCAL_LIBS/lib/"␣
↪link=shared threading=multi install --prefix=$LOCAL_LIBS
```

For the detailed instruction see tutorial page.

5.1.4 Dockers

Alternatively, you can use dropEst through Docker. Dockerfiles for the most popular linux distributions are provided (see `dropEst/dockers/`). You can either build and run these dockers or just read dockerfiles for the further instructions on dropEst installation for specific distribution.

To install docker on your system see installation instruction.

To pull pre-built CentOS-based docker from DockerHub use:

```
docker pull vpetukhov/dropest:latest
```

Dockers for older dropEst versions are available on DockerHub.

Or, you can build docker by hands, using the following commands:

```
cd dropEst/dockers/centos7
docker build -t dropest .
```

Or, for CentOS 7:

```
cd dropEst/dockers/centos7
docker build -t dropest .
```

To run the docker, use

```
docker run --name dropest -it dropest
```

You can find more info about dockers at Docker Cheat Sheet

Updating dropEst inside docker

Please note, that docker container isn't wired to a specific dropEst version, it just builds the latest commit from the master branch of the git repo. To update the code inside a compiled container, you need to log into it, pull the latest version and rebuild the code:

```
docker exec -it dropest /bin/bash

cd /home/user/dropEst/build
rm -rf ./*
git pull origin master
cmake .. && make
```

5.1.5 Troubleshooting

CMake can't find installed libraries

If `cmake` can't find one of the libraries, or you want to use some specific versions, which are currently not in the default path, use corresponding cmake variables: * Boost: BOOST_ROOT. * BamTools: BAMTOOLS_ROOT. * R: R_ROOT. Can be found by running the `cat(R.home())` in R.

These variables should be set to the path to the installed library. It can be done either by using command line options: `cmake -D R_ROOT="path_to_r"` or by adding the variable declaration to the beginning of CMakeLists.txt: `set(R_ROOT path_to_r)`.

In case you have some issues with the linker for specific library, please build this library manually with the version of compiler, which you're going to use for dropEst build.

Problems with std::__cxx11::string

If you have messages like "*(path to some library)*: undefined reference to *(some name)* for std::__cxx11::basic_ostringstream, std::allocator >", it means that you're trying to link a library, built with gcc < 5.0, while dropEst is built with gcc >= 5.0. It's a compiler issue, and you have to guarantee consistency of compiler versions by rebuilding either the library or dropEst. For more details see question on stackoverflow.

If you have several compilers in your system, please use cmake flags `-DCMAKE_CXX_COMPILER=(c++ compiler)` and `-DCMAKE_C_COMPILER=(c compiler)` to choose a compiler. Here, `(c++ compiler)` and `(c compiler)` denotes path to the prefered compiler version.

Boost 1.65

CMake < 3.10 has known issues with boost 1.65. If you have such combination, please try either to upgrade cmake or to downgrade boost.

5.2 dropTag

```
droptag -- generate tagged fastq files for alignment
```

Example command:

```
./droptag -c config.xml [-S] [-s] reads1.fastq reads2.fastq [...]
```

Positional arguments of the dropTag phase contain paths to the read files, obtained with a sequencer. These files can be either in *.fastq* of *.fastq.gz* format. The file order depends on the type of used protocol. Below is the instruction on the usage for currently supported protocols.

Note. Algorithm of UMI correction requires information about base-call quality of UMIs. To save this information, use `-s` option. In this case, in addition to fastq files with reads, the pipeline saves separate gzipped file with read parameters. These files must be passed to dropEst with `-r` option.

5.2.1 Protocols

inDrop v1 & v2

- File 1: barcode reads. Structure:
- Cell barcode, part 1

- Spacer
- Cell barcode, part 2
- UMI
- File 2: gene reads

Example config file is located at "*dropEst/configs/indrop_v1_2.xml*".
Example command:

```
./droptag -c [-S] [-s] ~/dropEst/configs/indrop_v1_2.xml barcode_reads.fastq gene_
↪reads.fastq
```

inDrop v3

- File 1: cell barcode, part 1 *(default length: 8bp)*
- File 2: cell barcode + UMI, part 1 *(default length: >= 14bp)*
- File 3: gene read
- *File 4 (optional): library tag*

If a file with library tags provided, option "-t" is required. ***This option wasn't tested properly, so it's better to avoid using it.***

Example config file is located at "*dropEst/configs/indrop_v3.xml*".
Example command:

```
./droptag -c ~/dropEst/configs/indrop_v3.xml [-S] [-s] [-t library_tag] barcode1_
↪reads.fastq barcode2_reads.fastq gene_reads.fastq [library_tags.fastq]
```

10x

- File 1: library tag *(default length: 8bp)*
- File 2: cell barcode + UMI, part 1 *(default length: 16+10=26bp)*
- File 3: gene read

Example config file is located at "*dropEst/configs/10x.xml*".
Example command:

```
./droptag -c ~/dropEst/configs/10x.xml [-S] [-s] lib_tag_reads.fastq barcode_reads.
↪fastq gene_reads.fastq
```

While dropTag provides way to demultiplex 10x data, Cell Ranger is still recommended tool for this. *dropEst* phase can be ran on the Cell Ranger demultiplexed .bam file to obtain data in the format, optimized for the subsequent analysis.

Note. Sometimes 10x CellRanger isn't able to determine gene, from which a read originated. In this cases it fills gene info with a list of possible genes, separated by semicolon (e.g. 'ENSG00000255508;ENSG00000254772'). These genes **must be filtered out** prior to further analysis:

```
holder <- readRDS('./cell.counts.rds')
cm <- holder$cm[grep("^[^;]+$", rownames(holder$cm)),]
```

iCLIP

- File 1: Gene reads with barcodes at the beginning of the sequence

Example config file is located at "*dropEst/configs/iclip.xml*".
Example command:

```
./droptag -c ~/dropEst/configs/iclip.xml [-S] [-s] data.fastq
```

Note. Implementation of iCLIP wasn't tested properly. Please, be careful using it. Anyone who used it is very welcome to comment it either in Issues or by e-mail.

SPLiT-seq

- File 1: Gene reads
- File 2: UMI + cell barcode (3 parts)

Example config file is located at "*dropEst/configs/split_seq.xml*". Example command:

```
./droptag -c ~/dropEst/configs/split_seq.xml [-S] [-s] gene_reads.fastq barcode_reads.
↪fastq
```

Seq-Well

- File 1: Gene reads
- File 2: Cell barcode + UMI

Example config file is located at "*dropEst/configs/seq_well.xml*". Example command:

```
./droptag -c ~/dropEst/configs/seq_well.xml [-S] [-s] gene_reads.fastq barcode_reads.
↪fastq
```

Drop-seq

Currently, processing of Drop-seq is supported only from bam files, obtained with Drop-seq tools. To get more information see Usage of tagged bam files.

5.2.2 Command line arguments for dropTag

- -c, –config filename: xml file with droptag parameters
- -l, –log-prefix prefix: logs prefix
- -n, –name name: alternative output base name
- -p, –parallel number: number of threads (usage of more than 6 threads should lead to significant speed up)
- -r, –reads-per-out-file : maximum number of reads per output file; (0 - unlimited). Overrides corresponding xml parameter.
- -S, –save-stats : save stats to rds file. This data is used on the dropReport phase.
- -t, –lib-tag library tag : (for IndropV3 with library tag only)
- -q, –quiet : disable logs

Please, use `./droptag -h` for additional help.

5.3 Alignment

dropTag writes the tagged reads into multiple files. All these files must be aligned to reference, and all bam files with the alignments must be provided as input for the dropEst stage. In the paper we used TopHat2 aligner, however any RNA-seq aligners (i.e. Kallisto or STAR) can be used.

5.3.1 Alignment with TopHat

1. Install bowtie index for the sequenced organism.
2. Download corresponding gene annotation in the gtf format.
3. Run TopHat2 for each file:

```
tophat2 -p number_of_threads --no-coverage-search -g 1 -G genes.gtf -o output_dir␣
↪Bowtie2Index/genome reads.fastq.gz
```

4. The result needed for the count estimation is *./output_dir/accepted_hits.bam*.

5.3.2 Alignment with Kallisto

Prior to v0.42.4, Kallisto didn't use BAM flags to mark primary/secondary alignments. Only versions :math:'geq' 0.42.4 are supported.

1. Download transcript sequences in .fasta format (i.e. Ensembl genes).
2. Build Kallisto index: `kallisto index -i genes.fa.gz`.
3. Run `kallisto quant --pseudobam --single -i genes.index -o out -l mean_length -s std_length reads.fastq.gz`. Here, *mean_length* is mean length of RNA fragment (not read length) and *std_length* is standard deviataion of RNA fragment length. You should specify values, according to the experiment design.

5.4 dropEst

```
dropest: estimate molecular counts per cell
```

This phase requires aligned .bam files as input and uses them to estimate count matrix. These files must contain information about cell barcode and UMI for each read (reads, which don't contain such information are ignored). Three possible ways to encode this information are acceptable: 1. Encode it in read names (as a result of dropTag phase). 2. Use .bam tags (i.e. output of 10x Cell Ranger). Tag names can be configured in **config.xml* file*. 3. Save this information in a separate file using `-s` option on dropTag phase and pass it to dropEst with `-r` option.

Note. To run Bayesian algorithm of UMI error correction with dropestr, you need to pass information about UMI base call quality to dropEst. You can do it either using .bam tags or `-s` dropTag option (i.e. option 1 isn't available in this case). You still can run simpler algorithms (i.e. *cluster* or *directional*, see paper) without this information.

Count matrix estimation also requires information about the source gene for the reads. It can be provided in three ways: 1. Use gene annotation in either *.bad* or *.gtf* format. To provide such file, "*-g*" option should be used. 2. Use .bam tags. Tag name can be configured in **config.xml* file*. 3. Pseudoaligners encode gene names as chromosome names

Another crucial moment in estimation of count matrix is correction of cell barcode errors. Most protocols provide the list of real barcodes, which simplifies the task of correction. If such file is available, path to the file **should be specified in the *config.xml* file** (*Estimation/Merge/barcodes_file*). This can dramatically increase quality of the result. Lists for inDrop protocols can be found at *dropEst/data/barcodes/*. Two algorithms of barcode correction are available: 1. Simple, "*-m*" option. This algorithm is recommended in the case, where barcodes list is supplied. 2. Precise, "*-M*" option. Doesn't requires list of real barcodes to obtain high-quality results, however has lower performance for large datasets.

Example command:

```
dropest [options] [-m] [-r pipeline_res.params.gz] -g ./hg38/genes.gtf -c ./config.
↪xml ./alignment.*/accepted_hits.bam
```

5.4.1 Usage of tagged bam files (e.g. 10x, Drop-seq) as input

Some protocols provide pipelines, which create .bam files with information about CB, UMI and gene. To use these files as input, specify "*-f*" option. Example:

```
dropest [options] -f [-g ./genes.gtf] -c ./config.xml ./pipeline_res_*.bam
```

If "*-g*" option is provided, genes are parsed from the gtf, and information about genes from the bam file is ignored.

To specify corresponding .bam tag names, use "*Estimation/BamTags*" section in the config (see *configs/config_desc.xml*).

5.4.2 Usage of pseudoaligners

Pseudoaligners, such as Kallisto store gene / transcript names in the field of chromosome name. To parse such files, use "*-P*" option. Example: `bash dropest [options] -P [-r pipeline_res.params.gz] -c ./config.xml ./kallisto_res_*.bam`

5.4.3 Count intronic / exonic reads only

One feature of the pipeline is the ability to count only UMIs, which reads touch only specific parts of a genome. Option *"-L"* allows to specify all acceptable types of regions: * e: UMIs with exonic reads only * i: UMIs with

intronic reads only * E: UMIs, which have both exonic and not annotated reads * I: UMIs, which have both intronic and not annotated reads * B: UMIs, which have both exonic and intronic reads * A: UMIs, which have exonic, intronic and not annotated reads

Thus, to count all UMIs with exonic **or** not annotated reads, use *"-L eE"*. Default value: *"-L eEBA"*, i.e. to count all UMIs, which have at least one reads, touching exon.

Example commands: * Intronic reads only: `bash dropest [-f] [-g ./genes.gtf] [-r pipeline_res.params.gz] -L i -c ./config.xml ./alignment_*.bam` * Exonic reads only: `bash dropest [-f] [-g ./genes.gtf] [-r pipeline_res.params.gz] -L e -c ./config.xml ./alignment_*.bam` * Exon/intron spanning reads: `bash dropest [-f] [-g ./genes.gtf] [-r pipeline_res.params.gz] -L BA -c ./config.xml ./alignment_*.bam`

The pipeline can determine genome regions either using .gtf annotation file or using .bam tags, i.e. for CellRanger output (see *Estimation/BamTags/Type* in *configs/config_desc.xml*). If .gtf file isn't provided and .bam file doesn't containt annotation tags, all reads with not empty gene tag are considered as exonic.

Note. There is no way to extract information about intronic reads from pseudoalignment, so you can't use pseudoaligners at this stage.

Velocyto integration

For some purposes (i.e. velocyto) it can be useful to look separately at the fraction of intronic and exonic UMIs. Option *"-V"* allows to output three separate count matrices, each of which contains only UMIs of a specific type: intronic, exonic or exon/intron spanning. These matrices are stored in the separate file *"cell.counts.matrices.rds"*.

Note. Please **ensure that you provided gtf file with genes** with `-g` option.

5.4.4 Command line arguments for dropEst

- -b, –bam-output: print tagged bam files
- -c, –config filename: xml file with estimation parameters
- -C, –cells num: maximal number of output cells
- -f, –filled-bam: bam file already contains genes/barcodes tags
- -F, –filtered-bam: print tagged bam file after the merge and filtration
- -g, –genes filename: file with genes annotations (.bed or .gtf)
- -G, –genes-min num: minimal number of genes in output cells
- -l, –log-prefix : logs prefix
- -m, –merge-barcodes : merge linked cell tags
- -M, –merge-barcodes-precise : use precise merge strategy (can be slow), recommended to use when the list of real barcodes is not available
- -o, –output-file filename : output file name
- -P, –pseudoaligner: use chromosome name as a source of gene id
- -q, –quiet : disable logs
- -r, –read-params filenames: file or files with serialized params from tags search step. If there are several files, they should be provided in quotes, separated by space: "file1.params.gz file2.params.gz file3.params.gz"
- -R, –reads-output: print count matrix for reads and don't use UMI statistics

- -u, –merge-umi: apply 'directional' correction of UMI errors. This option prevents output of `reads_per_umi_per_cell`. If you want to apply more advanced UMI correction, don't use '-u', but use follow up R analysis.
- -V, –velocyto : save separate count matrices for exons, introns and exon/intron spanning reads
- -w, –write-mtx : write out matrix in MatrixMarket format

5.4.5 Output

Result of this phase is cell.counts.rds file with the next fields: * **cm** (sparse matrix): count matrix in sparse format * **reads_per_chr_per_cell** (list of data.frame): number of reads per cell (row) for each chromosome (column): * **Exon** (data.frame): exonic reads * **Intron** (data.frame): intronic reads * **Intergenic** (data.frame): intergenic reads * **mean_reads_per_umi** (vector): mean number of reads per UMI for each cell * some additional info

To additionaly print the file with count matrix in MatrixMarket format use "*-w*" option.

5.5 dropReport

To run the report you have to install: * *dropestr* R package with all dependencies (`dependencies = T`). * pandoc * R packages: `R install.packages(c("optparse","rmarkdown"))`

To run the report use:

```
./dropReport.Rsc cell.counts.rds
```

To see full list of options use:

```
./dropReport.Rsc -h
```

5.5.1 Troubleshooting

If you get the error *"pandoc version 1.12.3 or higher is required and was not found"*, try to set path in the corresponding environment variable: `export RSTUDIO_PANDOC=/path/to/pandoc`.

5.6 dropEstR package

This package implements UMI errors corrections and low-quality cells filtration, which are not performed during dropEst phase.

To install the package, use:

```
devtools::install_github('hms-dbmi/dropEst/dropestr', dependencies = T)
```

Package content: * filtration of low-quality cells (see vignette "*low-quality-cells*") * correction of UMI errors (see vignette "*umi-correction*") * quality control (see *dropReport.Rsc*)

5.7 Configuration

Description of the fields for the config file is provided in *dropEst/configs/config_desc.xml*.

5.8 For developers

5.8.1 Adding new protocols to dropTag

For each new protocol you need to write new c++ class, which inherits TagsFinderBase (*"TagsSearch/TagsFinderBase.cpp"*). There are several examples for existed protocols (see all classes with suffix "TagsFinder"). All you really need there is to define function `parse_fastq_records`. After that you need to add new protocol type to function `get_tags_finder` in *"droptag.cpp"*. It takes only several lines, e.g.:

```
if (protocol_type == "ddSEQ")
    return ...
```

Having this your are able to set "TagsSearch/protocol" to "ddSEQ" in the config.xml and work with ddSEQ as with any other supported protocol in very efficient and parallel manner.

5.8.2 General workflow of dropTag under the hood

Entry point is *"droptag.cpp"* file, `main()` function. Most of it is just parsing CLI parameters and logging. So, there are only two important places: (1) `get_tags_finder`: factory-like function, which creates TagsFinder for a specific protocol based on configs, and (2) `finder->run`: TagsFinder's method, which actually does the whole work.

Workflow of `finder->run` is the following:

1. Read fastq records from the files, which contain gene and barcode reads. Reading is performed synchronously over all files, as records in different files correspond to each other.
2. This set of fastq records is parsed to a single gene record with its parameters (i.e. cell barcode, UMI and its quality). There are two options on how to store the parameters: it can either be done in the read name of the gene record or in a separate data structure. In the later case, this information is stored as a gzipped tsv file (see `-s` option).
3. Parsed record is converted to a string and gzipped.
4. Gzipped info is written to the output file.

Though code is more complicated then that, because it's implemented in a multi-threaded way, and there it can't be done by simple "parallel map" style. So, parallelism style looks more like MPI (though of course it's implemented in C++11 threads), and works as the follows (see `TagsFinderBase::run_thread` function):

- All data is stored in concurrent queues, which have limited maximal size.
- Each workers independently iterates over all 4 tasks and check if it can do some work on it. If task is single-threaded and another worker is already doing it, or if corresponding queue is already full, the worker goes to the next task.

Such scheme allows to achieve ~10 times higher performance.