James Nemesh
Drop-seq informatics cookbook, v1.0
McCarroll Lab, May 2015

Introduction

Drop-seq sequencing libraries produce paired-end reads: read 1 contains a cell barcode and a molecular barcode (also known as a UMI); read 2 is aligned to the reference genome. This document provides step-by-step instructions for using the software we have developed to convert these sequencing reads into a digital expression matrix, containing integer counts of the number of transcripts for each gene, in each cell.

Overview of Alignment

Next generation sequencers emit reads in many formats that can be converted into fastq format. Since there are many sequencers and pipelines available to do this, we leave this step to the user. For example, we use either Picard's **IlluminaBasecallsToFastq** or Illumina's **bcl2fastq**. Once you have a pair of fastq files, you can follow this set of steps to align your raw reads and have a BAM file that is suitable to produce digital gene expression (DGE) results.

1. Fastq -> aligned and tagged BAM
    a. Fastq -> BAM
    b. Tag molecular barcodes
    c. Tag cell barcodes
    d. Trim 5' primer sequence
    e. Trim 3' polyA sequence
    f. SAM -> Fastq
    g. STAR alignment
    h. Sort STAR alignment in queryname order
    i. Merge STAR alignment tagged SAM to recover cell/molecular barcodes
    j. Add gene/exon and other annotation tags

The next sections will explain the meta data needed to follow this workflow, as well as explain each of the programs that has been developed to run these steps. Some of these programs are developed by the Drop-seq team, and others take advantage of existing Picard Tools or aligners like STAR.

Meta Data

To follow this set of processes from raw unaligned reads to an aligned BAM, it's necessary to have a number of different metadata files. These provide information about the sequence of the organism(s) you're running your experiment on, as well as genomic features like genes,transcripts, and exons that help extract DGE data from the reads.

We organize our meta data using a set of conventions we suggest you follow, as it makes it easier to keep track of what files are used for particular processes. In the software section, we'll refer to these files using these conventions.

The first convention is that we establish a root name for all of our files that encodes information about the organism and the genome build used to derive that meta data. For example, mm10 is the Dec. 2011 *Mus musculus* assembly. All files for mouse use this as the root name, followed by a ".", then the type of file.

Meta Data file types:

- *fasta*: The reference sequence of the organism. Needed for most aligners.
- *dict*: A dictionary file as generated by Picard's CreateSequenceDictionary. Needed for Picard Tools.
- gtf: The principle file to determine the location of genomic features like genes, transcripts, and exons. Many other meta data files we use derive from this original file. We download our GTF files from ensembl, which has a handy description of the file format here. Ensembl has a huge number of prepared GTF files for a variety of organisms here.
- refFlat: This file contains a subset of the the same information in the GTF file in a different format. Picard tools like the refFlat format, so we require this as well. To make life easy, we provide a program ConvertToRefFlat that can convert files from GTF format to refFlat for you.
- genes.intervals: The genes from the GTF file in interval list format. This file is optional, and useful if you want to go back to your BAM later to see what gene(s) a read aligns to.
- exons.intervals: The exons from the GTF file in interval list format. This file is optional, and useful if you want to go back to your BAM and view what exon(s) a read aligns to.
- rRNA.intervals: The locations of ribosomal RNA in interval list format. This file is optional, but we find it useful to later assess how much of a dropseq library aligns to rRNA.
- reduced.gtf: This file contains a subset of the information in the GTF file, but in a far more human readable format. This file is optional, but can be generated easily by the supplied ReduceGTF program that will take a GTF file as input.

<note to software devs: it would be nice to have a program written in java that extracts the rRNA intervals (or even genes/exons as well) from GTF/RefFlat annotation files.>

Alignment Pipeline Programs

FastqToSAM

This picard program takes your paired FastQ files and converts them into an unaligned BAM. We do this so we can tag the *genome read* with a a number of BAM tags, such as the molecular and cel barcode. The unaligned BAM is created in queryname order. See the linked Picard documentation for further information.

Example

java -Xmx4g -jar /path/to/picard/picard.jar FastqToSam
FASTQ=my_fastq_1.gz
FASTQ2=my_fastq_2.gz
QUALITY_FORMAT=Standard
OUTPUT=my_unaligned_data.bam
SAMPLE_NAME=my_experiment
SORT_ORDER=queryname

TagBamWithReadSequenceExtended

This Drop-seq program extracts bases from the cell/molecular barcode encoding read (BARCODED_READ), and creates a new BAM tag with those bases on the *genome read*. By default, we use the BAM tag XM for molecular barcodes, and XC for cell barcodes, using the TAG_NAME parameter.

This program is run once per barcode extraction to add a tag. On the first iteration, the molecular barcode is extracted from bases 13-20 of the barcode read. On the second iteration, the cell barcode is extracted from bases 1-12. This is controlled by the BASE_RANGE option. This program has an option to drop a read (DISCARD_READ), which we use after both barcodes have been extracted, which makes the output BAM have unpaired reads with additional tags.

Additionally, this program has a BASE_QUALITY option, which is the minimum base quality of all bases of the barcode being extracted. If more than NUM_BASES_BELOW_QUALITY bases falls below this quality, the read pair is discarded.

Example Molecular Barcode:

java -Xmx4g -jar /path/to/dropseq/TagBamWithReadSequenceExtended.jar

INPUT=my_unaligned_data.bam
OUTPUT=my_unaligned_data_tagged_Cell.bam
SUMMARY=unaligned_taggedMolecular.bam_summary.txt
BASE_RANGE=13-20
BASE_QUALITY=10
BARCODED_READ=1
DISCARD_READ=False
TAG_NAME=XM
NUM_BASES_BELOW_QUALITY=1

Example Cell Barcode:
java -Xmx4g -jar /path/to/dropseq/TagBamWithReadSequenceExtended.jar
INPUT=my_unaligned_data_taggedCell.bam
OUTPUT=my_unaligned_data_tagged_CellMolecular.bam
SUMMARY=unaligned_taggedCell.bam_summary.txt
BASE_RANGE=1-12
BASE_QUALITY=10
BARCODED_READ=1
DISCARD_READ=True
TAG_NAME=XC
NUM_BASES_BELOW_QUALITY=1

TrimStartingSequence

This Drop-seq program is one of two sequence cleanup programs designed to trim away any extra sequence that might have snuck it's way into the reads. In this case, we trim the SMART Adapter that can occur 5' of the read. In our standard run, we look for at least 5 contiguous bases (NUM_BASES) of the SMART adapter (SEQUENCE) at the 5' end of the read with no errors (MISMATCHES) , and hard clip those bases off the read.

Example:
java -Xmx4g -jar /path/to/dropseq/TrimStartingSequence.jar
INPUT=my_unaligned_data_tagged_CellMolecular.bam
OUTPUT=my_unaligned_data_tagged_CellMolecular_trimmedSmart.bam
OUTPUT_SUMMARY=adapter_trimming_report.txt
SEQUENCE=AAGCAGTGGTATCAACGCAGAGTGAATGGG
MISMATCHES=0
NUM_BASES=5

PolyATrimmer

This Drop-seq program is the second sequence cleanup program designed to trim away trailing polyA tails from reads. It searches for at least 6 (NUM_BASES) contiguous A's in the read with 0 mismatches (MISMATCHES), and hard clips the read to remove these bases and all bases 3' of the polyA run.

Example:

```
java -Xmx4g -jar /path/to/dropseq/PolyATrimmer.jar
INPUT=my_unaligned_data_tagged_CellMolecular_trimmedSmart.bam
OUTPUT=my_unaligned_data_tagged_trimmed_filtered.bam
OUTPUT_SUMMARY=polyA_trimming_report.txt
MISMATCHES=0
NUM_BASES=6
```

SamToFastq

Now that your data has had the cell and molecular barcodes extracted, the reads have been cleaned of SMARTSeq primer and polyA tails, and the data is now unpaired reads, it's time to align. To do this, we extract the FASTQ files using Picard's SamToFastq program.

Example:

```
java -Xmx4g -jar /path/to/picard/picard.jar SamToFastq
INPUT=my_unaligned_data_tagged_CellMolecular_trimmedSmart_polyAFiltered.bam
FASTQ=my_prepared_reference.fastq
```

Alignment - STAR

We use STAR as our RNA aligner. The manual for STAR can be found here. There are many potential aligners one could use at this stage, and it's possible to substitute in your lab's favorite. We haven't tested other aligners in methodical detail, but all should produce valid BAM files that can be plugged into the rest of the process detailed here.

If you're unsure how to create an indexed reference for STAR, please read the STAR manual. Below is a minimal invocation of STAR. Since STAR contains a huge number of options to tailor alignment to a library and trade off sensitivity vs specificity, you can alter the default settings of the algorithm to your liking, but we find the defaults work reasonably well for Drop-seq.

Example:

```
/path/to/STAR/STAR
--genomeDir /path/to/STAR_REFERENCE
--readFilesIn my_prepared_reference.fastq
--outFileNamePrefix star
```

SortSam

This picard program is invoked after alignment, to guarantee that the output from alignment is sorted in queryname order. As a side bonus, the output file is a BAM (compressed) instead of SAM (uncompressed.)

Example:
java -Xmx4g -jar /path/to/picard/picard.jar SortSam
I=starAligned.out.sam
O=starAligned.out.bam
SO=queryname

MergeBamAlignment

This Picard program merges the sorted alignment output from STAR (ALIGNED_BAM) with the unaligned BAM that had been previously tagged with molecular/cell barcodes (UNMAPPED_BAM). This recovers the BAM tags that were "lost" during alignment. The REFERENCE_SEQUENCE argument refers to the fasta meta data file.

We ignore secondary alignments, as we want only the best alignment from STAR (or another aligner), instead of assigning a single sequencing read to multiple locations on the genome.

Example:
java -Xmx4g -jar /path/to/picard/picard.jar MergeBamAlignment
REFERENCE_SEQUENCE=my_fasta.fasta
UNMAPPED_BAM=my_unaligned_data_tagged_trimmed_filtered.bam
ALIGNED_BAM=starAligned.out.bam
OUTPUT=out.bam
INCLUDE_SECONDARY_ALIGNMENTS=false
PAIRED_RUN=false VALIDATION_STRINGENCY=SILENT

TagReadWithGeneExon

This is a Drop-seq program that adds a BAM tag "GE" onto reads when the read overlaps the exon of a gene. This tag is the name of the gene, as reported in the annotations file. You can use either a GTF or a RefFlat annotation file with this program, depending on what annotation data source you find most useful. This is used later when we extract digital gene expression (DGE) from the BAM.

Example:
java -Xmx4g -jar /path/to/dropseq/TagReadWithGeneExon.jar

```
I=/dev/stdin
O=out_gene_exon_tagged.bam
ANNOTATIONS_FILE=${refFlat}
TAG=GE
```

End of Alignment

At this point, the alignment is completed, and your raw fastq reads have been changed from paired reads to single end reads with the cell and molecular barcodes extracted, cleaned up, aligned, and prepared for DGE extraction.

Going with the flow - using Unix pipes to simplify alignment

If you're on a Unix or OS X operating system, you may be familiar with pipes. Drop-seq programs extend the Picard API, and so like Picard are able to use pipes to redirect output from one program to the next. Why is this useful? It's a little bit faster, but more importantly it saves an incredible amount of space by not generating a large number of temporary files, as the examples above have. It also simplifies writing pipelines, as there are fewer named files - intermediate data flows through the pipeline without being saved.

A simple example of using pipes would be to extract both cell and molecular barcodes within a single process. We start with the SAM file produced by FastqToSam, and apply both tags to the BAM. The output of the first call is directed to /dev/stout, and picked up by the second program, which reads /dev/stdin. The summary files are still directed to actual files on the disk.

```
java -Xmx4g -jar /path/to/dropseq/TagBamWithReadSequenceExtended.jar
INPUT=/my_unaligned_data.bam
OUTPUT=/dev/stdout
COMPRESSION_LEVEL=0
SUMMARY=unaligned_taggedMolecular.bam_summary.txt
BASE_RANGE=13-20 BASE_QUALITY=10 BARCODED_READ=1 DISCARD_READ=False
TAG_NAME=XM NUM_BASES_BELOW_QUALITY=1 |
java -Xmx4g -jar /path/to/dropseq/TagBamWithReadSequenceExtended.jar
INPUT=/dev/stdin
OUTPUT=/dev/stdout
COMPRESSION_LEVEL=0
SUMMARY=unaligned_taggedCell.bam_summary.txt
BASE_RANGE=1-12 BASE_QUALITY=10 BARCODED_READ=1 DISCARD_READ=True
TAG_NAME=XC NUM_BASES_BELOW_QUALITY=1
```

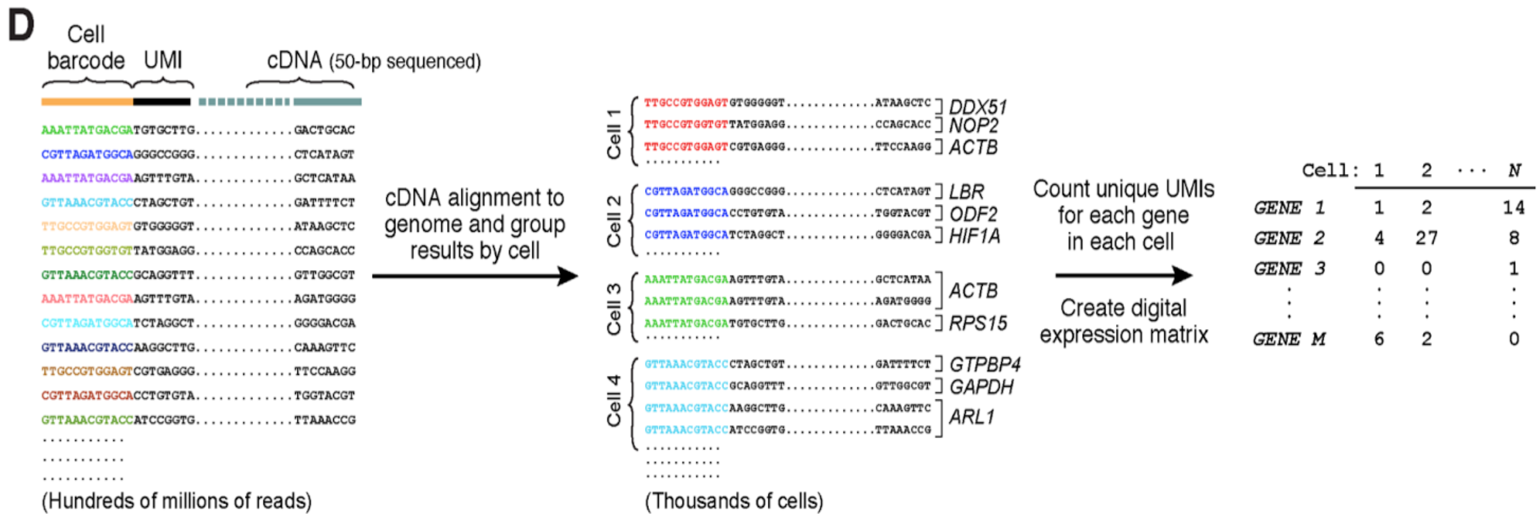
Most Picard and Drop-seq programs can be piped the same way. However, it's important to note that some outputs need to be saved. In our example, those would be:

1) my_unaligned_data_tagged_trimmed_filtered.bam
2) my_prepared_reference.fastq
3) starAligned.out.sam
4) starAligned.out.bam

#1 is saved because it needs to be merged with the aligned BAM file. 2,3 are saved because we've had difficulty using Unix pipes with STAR.

<Question for Steve/Evan: Do we want to include a whole bash script pipeline? What about Alec's nice python based pipeline?>

Overview of DGE extraction



To digitally count gene transcripts, a list of UMIs in each gene, within each cell, is assembled, and UMIs within ED = 1 are merged together. The total number of unique UMI sequences is counted, and this number is reported as the number of transcripts of that gene for a given cell.

Digital Gene Expression

Extracting Digital Gene Expression (DGE) data from an aligned library is done using the Drop-seq program DigitalExpression. The input to this program is the aligned BAM from the alignment workflow. There are two outputs available: the primary is the DGE matrix, with each a row for each gene, and a column for each cell. The secondary analysis is a summary of the DGE matrix on a per-cell level, indicating the number of genes and transcripts observed.

Primary Output Example:

GENE	ATCAGGGACAGA	AGGGAAAATTGA	TTGCCTTACGCG	TGGCGAAGAGAT	TACAATTAAGGC
LOXL4	0	0	0	0	0
PYROXD2	1	0	1	1	0
HPS1	23	12	9	8	3
CNNM1	0	2	1	0	0
GOT1	22	6	7	9	3

Summary Output Example:

CELL_BARCODE	NUM_GENES	NUM_TRANSCRIPTS
ATCAGGGACAGA	12128	232831
AGGGAAAATTGA	12161	185418
TTGCCTTACGCG	10761	173547
TGGCGAAGAGAT	10036	108545
TACAATTAAGGC	9889	99771
CTAAGTAGCTTT	9244	91563

DGE Extraction Options:

There are a large number of options in the DGE program, as we've performed large amounts of experimentation with the outputs to this program. Most of these parameters have default settings, and are the correct setting for a standard Drop-seq experiment. Outlined below are some of the parameters that you might change.

READ_MQ The minimum map quality of a read to be used in the DGE calculation. For aligners like STAR, the default (10) is higher than what's needed to eliminate all multi-mapping reads. If you use a different aligner, you might want to set a different threshold.

EDIT_DISTANCE. By default we collapse UMI barcodes with a hamming distance of 1.

RARE_UMI_FILTER_THRESHOLD This is an implementation of the rare UMI filter implemented by Islam, et al. We leave this off by default, and use edit distance collapse instead. If desired, one can set EDIT_DISTANCE=0 and enable this filter instead at some threshold, like 0.01.

Options for selecting sets of cells

When running DGE, we don't select every cell barcode observed. This is because the aligned BAM can contain hundreds of thousands of cell barcodes; most reads will be on either

STAMPs (beads exposed to a cell in droplets) or “empties” (beads that were exposed only to ambient RNA in droplets). There will also be a lot of cell barcodes with just a handful of reads. Because a huge matrix might be difficult to work with, these options limit the number of cell barcodes that are emitted by DGE extraction. *You must use one of these options.*

MIN_NUM_GENES_PER_CELL. DigitalExpression runs a single iteration across all data, and selects cells that have at least this many genes.

MIN_NUM_TRANSCRIPTS_PER_CELL. DigitalExpression runs a single iteration across all data, and selects cells that have at least this many transcripts.

NUM_CORE_BARCODES. DigitalExpression counts the number of reads per cell barcode (thresholded by READ_MQ), and only includes cells that have at least this number of reads.

CELL_BC_FILE. Instead of iterating over the BAM and discovering what cell barcodes should be used, override this with a specific subset of cell barcodes in a text file. This file has no header and a single column, containing one cell barcode per line. Since this option doesn't have to iterate through the BAM to select barcodes, DGE extraction is significantly faster when using this option.

Example:
In this example, we extract the DGE for the top 100 most commonly occurring cell barcodes in the aligned BAM.

```
java -Xmx4g -jar /path/to/dropseq/DigitalExpression.jar
I=out_gene_exon_tagged.bam
O=out_gene_exon_tagged.dge.txt.gz
SUMMARY=out_gene_exon_tagged.dge.summary.txt
NUM_CORE_BARCODES=100
```

Cell Selection

A key question to answer for your data set is how many cells you want to extract from your BAM. One way to estimate this is to extract the number of reads per cell, then plot the cumulative distribution of reads and select the “knee” of the distribution.

We provide a tool to extract the reads per cell barcode in the Drop-seq software called BAMTagHistogram. This extracts the number of reads for any BAM tag in a BAM file, and is a general purpose tool you can use for a number of purposes. For this purpose, we extract the cell tag “XC”:

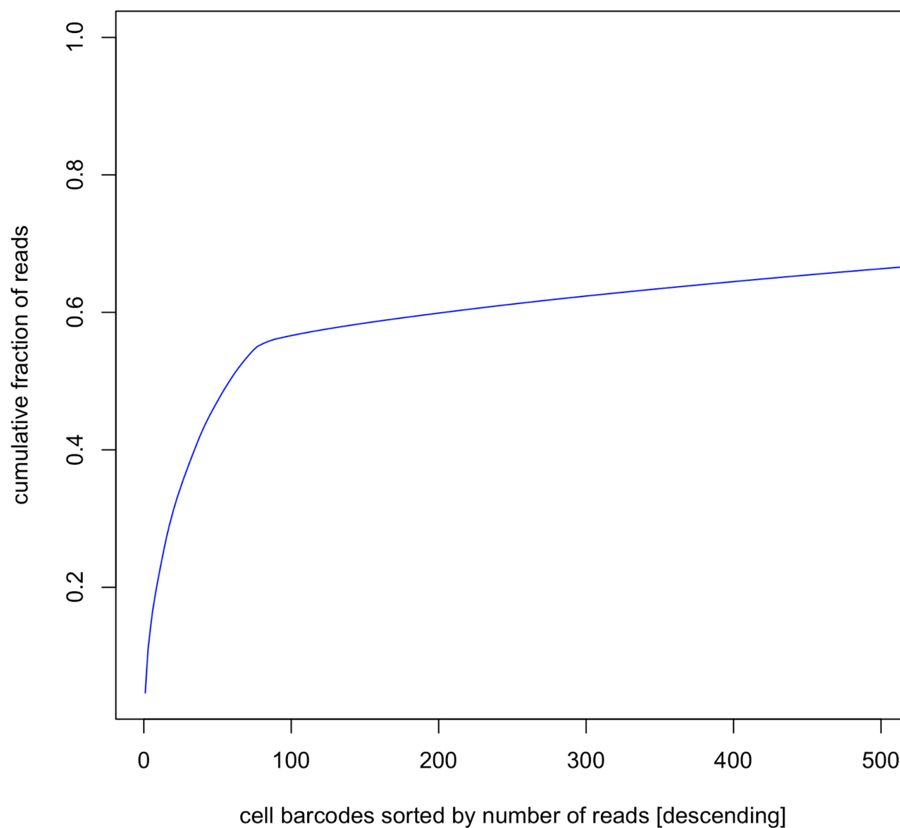
Example:

```
java -Xmx2g -jar /path/to/dropseq/BAMTagHistogram.jar
```

```
I=out_gene_exon_tagged.bam
O=out_cell_readcounts.txt.gz
TAG=XC
```

Once we run this program, a little bit of R code can create a cumulative distribution plot. Here's an example using the 100 cells data from the Drop-seq initial publication (Figures 3C and 3D):

```
a=read.table("100cells_numReads_perCell_XC_mq_10.txt.gz", header=F, stringsAsFactors=F)
x=cumsum(a$V1)
x=x/max(x)
plot(1:length(x), x, type='l', col="blue", xlab="cell barcodes sorted by number of reads [descending]", ylab="cumulative fraction of reads", xlim=c(1,500))
```



In this example, the number of STAMPs are the number of cell barcodes to the left of the inflection point; to the right of the inflection point are the empty beads that have only been exposed to ambient RNA. Figure S3A of Macosko et al., 2015 provides additional justification and explanation for how we identify the number of cells sequenced.