Drop-seq Core Computational Protocol

version 1.0.1 (6/11/15)

James Nemesh
Steve McCarroll's lab, Harvard Medical School

Introduction

The following is a manual for using the software we have written for processing Drop-seq sequence data into a "digital expression matrix" that will contain integer counts of the number of transcripts for each gene, in each cell. This software pipeline performs many analyses including massive de-multiplexing of the data, alignment of reads to a reference genome, and processing of cellular and molecular barcodes.

Drop-seq sequencing libraries produce paired-end reads: read 1 contains both 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 that contains integer counts of the number of transcripts for each gene, in each cell.

We may release updates to this manual as we learn from users' experiences. If a revision simply contains additional hints or advice or detail, then we will update the date on the protocol but not the version number. Whenever we implement a substantive change to the software or protocol, we will increment the version number.

We hope this is helpful and that you are soon generating exciting data with Drop-seq.

Drop-seq Software and Hardware Requirements

The Drop-seq software provided is implemented entirely in Java. This means it will run on a huge number of devices that are capable of running Java, from large servers to laptops. We require 4 gigabytes of memory for each program to run, which is also sufficient for Picard programs we use as part of alignment and analysis. Disk space will be determined by your data size plus the meta-data and aligner index. 50 gigabytes of disk space will be sufficient to store our meta data plus a STAR index.

Overview of Alignment

The raw reads from the sequencer must be converted into a Picard-queryname-sorted BAM file for each library in the sequencer run. Since there are many sequencers and pipelines available to do this, we leave this step to the user. For example, we use either Picard IlluminaBasecallsToSam

(preceded by Picard ExtractIlluminaBarcodes for a library with sample barcodes); or Illumina's bcl2fastq followed by Picard FastqToSam**.** Once you have an unmapped, queryname-sorted BAM, you can follow this set of steps to align your raw reads and create a BAM file that is suitable to produce digital gene expression (DGE) results.

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

The next sections will explain the metadata needed to follow this workflow, as well as explain each of the programs that have been developed to run these steps. Some of these programs are developed by us, and others take advantage of existing Picard Tools or aligners like STAR.

Metadata

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 metadata 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 metadata. 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.

metadata 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 metadata 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.

On the Drop-Seq website you will find a set of pre-made meta data for human, mouse and human/mouse experiments. In a later release of this software we'll provide you with the tools to generate metadata for the organism(s) of your choice.

Premade Meta Data links @GEO.
MIXED MOUSE HUMAN

Alignment Pipeline Programs

On the Drop-seq website you will find a zipfile containing the programs described below. The zipfile also contains a script Drop-seq_alignment.sh that executes the process described below. Because of differences in computing environments, this script is not guaranteed to work for all users. However, we hope it will serve as an example of how the various programs should be invoked.

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 cell barcode is extracted from bases 1-12. This is controlled by the BASE_RANGE option. On the second iteration, the molecular barcode is extracted from bases 13-20 of the barcode read. 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 Cell Barcode:
TagBamWithReadSequenceExtended
INPUT=my_unaligned_data.bam
OUTPUT=unaligned_tagged_Cell.bam
SUMMARY=unaligned_tagged_Cellular.bam_summary.txt
BASE_RANGE=1-12
BASE_QUALITY=10
BARCODED_READ=1
DISCARD_READ=False
TAG_NAME=XC
NUM_BASES_BELOW_QUALITY=1

Example Molecular Barcode:
TagBamWithReadSequenceExtended
INPUT=unaligned_tagged_Cell.bam
OUTPUT=unaligned_tagged_CellMolecular.bam
SUMMARY=unaligned_tagged_Molecular.bam_summary.txt
BASE_RANGE=13-20
BASE_QUALITY=10
BARCODED_READ=1
DISCARD_READ=True
TAG_NAME=XM
NUM_BASES_BELOW_QUALITY=1

FilterBAM:
This Drop-seq program is used to remove reads where the cell or molecular barcode has low quality bases. During the run of TagBamWithReadSequenceExtended, an XQ tag is added to each read to represent the number of bases that have quality scores below the BASE_QUALITY threshold. These reads are then removed from the BAM.

Example:
FilterBAM
TAG_REJECT=XQ
INPUT=unaligned_tagged_CellMolecular.bam
OUTPUT=unaligned_tagged_filtered.bam

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:
TrimStartingSequence
INPUT=unaligned_tagged_filtered.bam
OUTPUT=unaligned_tagged_trimmed_smart.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:
PolyATrimmer
INPUT=unaligned_tagged_trimmed_smart.bam
OUTPUT=unaligned_mc_tagged_polyA_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=unaligned_mc_tagged_polyA_filtered.bam
FASTQ=unaligned_mc_tagged_polyA_filtered.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. Be aware that STAR requires roughly 30 gigabytes of memory to align a single human sized genome, and 60 gigabytes
for our human/mouse reference.

Example:

```
/path/to/STAR/STAR
--genomeDir /path/to/STAR_REFERENCE
--readFilesIn unaligned_mc_tagged_polyA_filtered.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=aligned.sorted.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 metadata 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=unaligned_mc_tagged_polyA_filtered.bam
ALIGNED_BAM=aligned.sorted.bam
```

OUTPUT=merged.bam
INCLUDE_SECONDARY_ALIGNMENTS=false
PAIRED_RUN=false

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 contains 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:
TagReadWithGeneExon
I=merged.bam
O=star_gene_exon_tagged.bam
ANNOTATIONS_FILE=${refFlat}
TAG=GE

DetectBeadSynthesisErrors - Detecting and repairing barcode synthesis errors

In June 2015, we noticed that a recently purchased batch of ChemGenes beads generated a population of cell barcodes (about 10-20%) with sequences that shared the first 11 bases, but differed at the last base. These same cell barcodes also had a very high percentage of the base "T" at the last position of the UMI. Based on these observations, we concluded that a percentage of beads in the lot had not undergone all twelve split-and-pool bases (perhaps they had stuck to some piece of equipment or container, and the been re-introduced after the missing synthesis cycle). Thus, the 20-bp Read 1 contained a mixed base at base 12 (in actuality, the first base of the UMI) and a fixed T-base at base 20 (in actuality, the first base of the polyT segment).

To correct for this, we generated DetectBeadSynthesisErrors, which identifies cell barcodes with aberrant "fixed" UMI bases. If only the last UMI base is fixed as a T, the cell barcode is corrected (the last base is trimmed off) and all cell barcodes with identical sequence at the first 11 bases are merged together. If any other UMI base is fixed, the reads with that cell barcode are discarded.

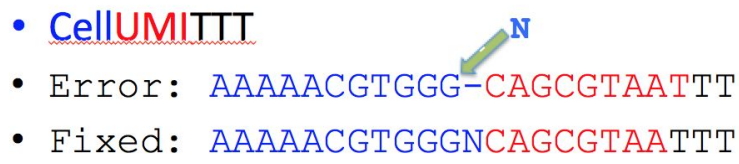
The program asks the user to select a number of barcodes on which to perform the correction. We use roughly 2 times the anticipated number cells, as we empirically found that this allows us to recover nearly every defective cell barcode that corresponds to a STAMP (rather than an empty bead cell barcode).

Example:
DetectBeadSynthesisErrors
I=my.bam
O=my_clean.bam

OUTPUT_STATS=my.synthesis_stats.txt
SUMMARY=my.synthesis_stats.summary.txt
NUM_BARCODES= <roughly 2x the number of cells>
PRIMER_SEQUENCE=AAGCAGTGGTATCAACGCAGAGTAC

This program reads in the BAM file, and looks at the distribution of bases at each position of all UMIs for a cell barcode. It detects unusual distributions of base frequency, where a base with >=80% frequency at any position is detected as an error. Barcodes with less than 25 total UMIs are ignored. There are a number of different errors that are categorized:

1. SYNTHESIS_MISSING_BASE - 1 or more bases missing from cell barcode, resulting in fixed T's at the end of UMIs. This counts the maximum number of fixed sequential T's in the UMIs at the end. This error type is cleaned up by the software for situations where there is a single base missing, and is by far the most common error. The fix involves inserting an "N" base before the last cell barcode base, effectively shifting the reading frame back to where it should be. This will both collapse these beads back together in further analysis, as well as repair the UMIs for these bead barcodes.



2. SINGLE_UMI_ERROR - At each position of the UMIs, the base distribution is highly skewed, i.e. at each position, a single base appears in >= 80% of the UMIs for that cell. There's no fix for this currently. Cell barcodes with this property are dropped. These cells have the interesting property that the number of genes and transcripts are at a close to 1:1 ratio, as there's generally only 1 UMI for every gene.

3. PRIMER_MATCH - Same as SINGLE_UMI_ERROR, but in addition the UMI perfectly matches one of the PCR primers. These cell barcodes are dropped. These errors are only detected if a PRIMER_SEQUENCE argument is supplied.

4. 4) OTHER - UMIs are extremely skewed towards at least one base (and not T at the last base), but not at all 8 positions. These cell barcodes are dropped.

The file my.synthesis_stats.txt contains a bunch of useful information:

1. CELL_BARCODE - the 12 base cell barcode
2. NUM_UMI - the number of total umis observed
3. FIRST_BIASED_BASE - the first base position where any bias is observed. -1 for no detected bias
4. SYNTH_MISSING_BASE - as #3 but specific to runs of T's at the end of the UMI
5. ERROR_TYPE - see error type definitions above

6. For bases 1-8 of the UMI, the observed base counts across all UMIs. This is a "|" delimited field, with counts of the A,C,G,T,N bases.

The file my.synthesis_stats.summary.txt contains a histogram of the SYNTHESIS_MISSING_BASE errors, as well as the counts of all other errors, the number of total barcodes evaluated, and the number of barcodes ignored.

End of Alignment

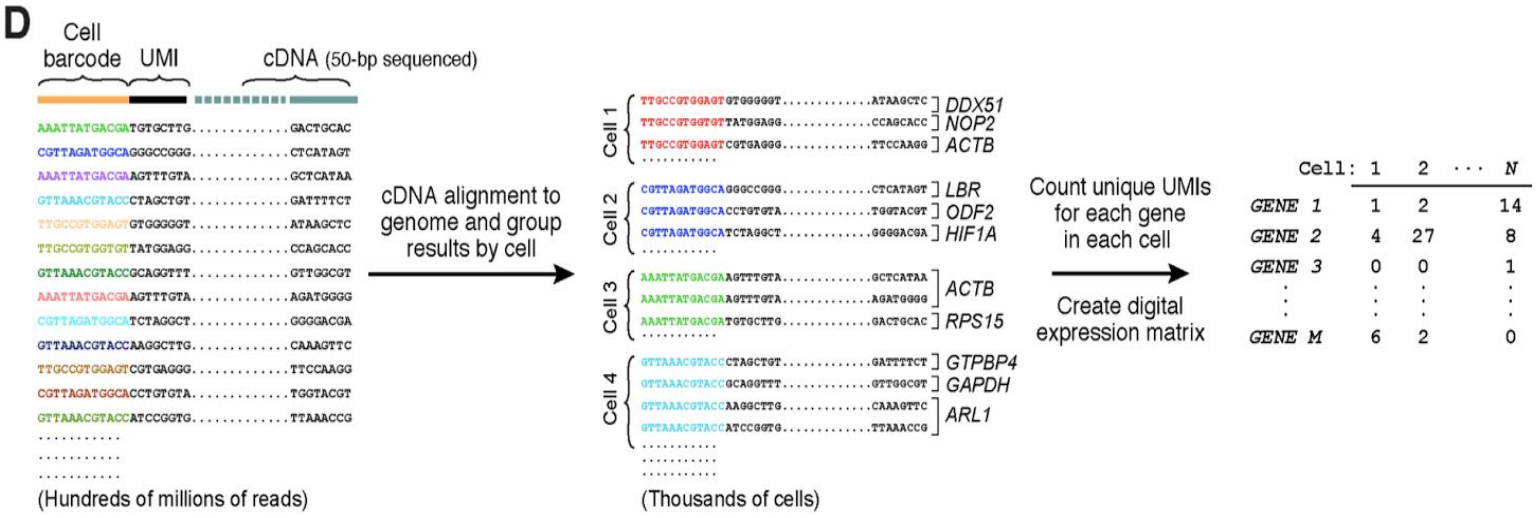
At this point, the alignment is completed, and your raw reads have been changed from paired reads to single end reads with the cell and molecular barcodes extracted, cleaned up, aligned, and prepared for DGE extraction.

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 a significant amount of disk 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. The tradeoff is that executing several programs in a pipeline requires more RAM and more processing power, so if your computer does not have a lot of RAM and lots of processors, this might not be useful.

There are some limitations to the amount of pipelining that can be done, because some files must be read more than once, and because STAR does not have the ability to write to standard output. The script Drop-seq_alignment.sh has an option (-p) that runs the programs in pipelines to the degree that is possible. If you are interested in using pipes, you can try this option, or examine the script to see what steps can be connected via pipes.

Overview of DGE extraction



To digitally count gene transcripts, a list of UMIs in each gene, within each cell, is assembled, and UMIs within edit distance = 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.

```
DigitalExpression
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

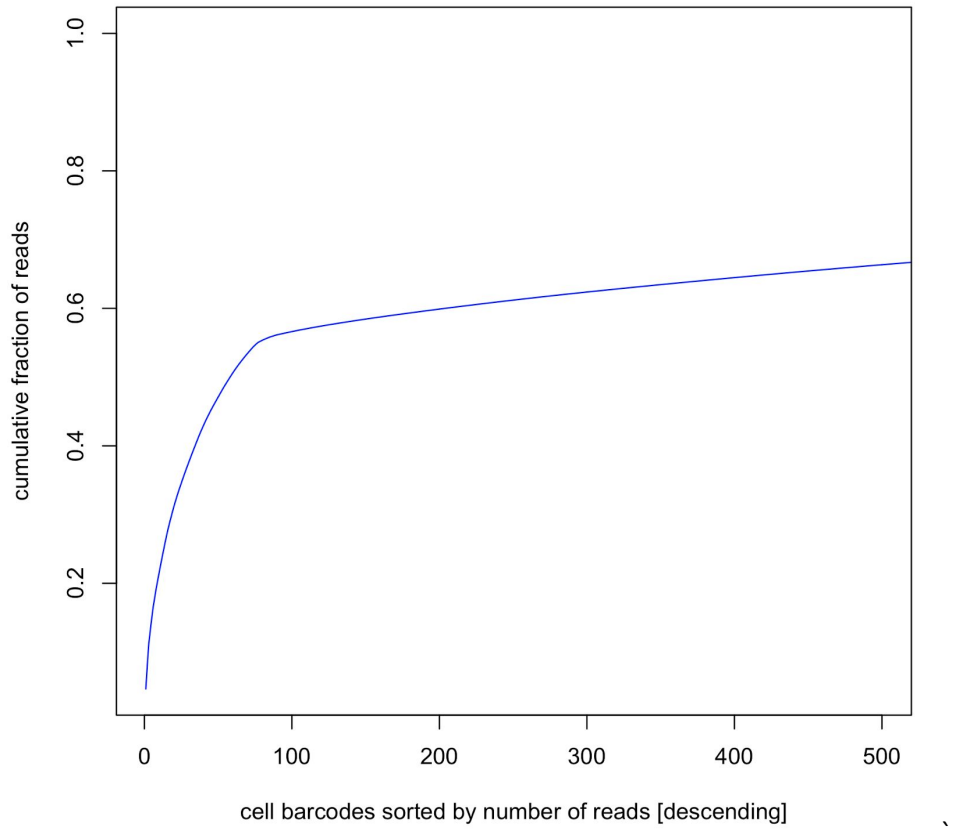
A key question to answer for your data set is how many cells you want to extract from your BAM. One way to estimate this is to extract the number of reads per cell, then plot the cumulative distribution of reads and select the "knee" of the distribution.

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:
BAMTagHistogram
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.

Mixed-species plots

To create the mixed species plots used in the paper, we suggest the following steps:

1. Align your data to a mixed species reference. There is meta data at the bottom of one of the pages of our GEO submission
2. Determine how many cells are in your BAM. See **Cell Selection** in this document and the BAMTagHistogram program. Put that list of cell barcodes in a file that has a single column of cell barcodes, 1 per line.
3. At this point, if you are using our human/mouse metadata, your BAM has chromosomes that are prepended with HUMAN or MOUSE, i.e.: HUMAN_11. Filter your BAM into 2 organism specific BAMs using FilterBAM with the argument REF_SOFT_MATCHED_RETAINED=HUMAN or REF_SOFT_MATCHED_RETAINED=MOUSE (this is about as fancy as running grep on your BAM.)
4. Run DigitalExpression on each organism specific BAM with the CELL_BC_FILE argument, using the file generated in step #2.
5. You now have two summary files that have the number of genes/transcripts contained by each cell, in an organism specific manner. Merge them into one file and plot.

Conclusion

With successful execution of our software you have hopefully transformed a pile of hundreds of millions of sequence reads into a digital expression matrix that has genome-wide expression measurements (digital counts) for each gene in each individual cell.

What to do next? We expect analysis of massive single-cell expression data to become a lively field. We think very strongly of the Seurat package developed by our colleague Rahul Satija. We used Seurat to perform all of the downstream analyses (cell clustering, etc) in the Cell paper. Seurat is available on Rahul's web site (http://www.satijalab.org/seurat.html), where Rahul will also have protocols for the specific analyses in the paper

But what if everything doesn't go perfectly?

One of the big challenges with releasing a new software toolkit to the world is that people will always do things you didn't anticipate, with data sets you never imagined. While we feel the Drop-seq software produces the computationally correct (at least to our intentions) answers, it's possible that you will discover a bug, or documentation of a particular software parameter will be unclear.

If you find part of this document unclear, let us know and we'll do our best to update it and add clarity. If parameters of our software have unclear documentation, let us know which ones are unclear, and we'll do our best to buff up those descriptions.

If you run into software behavior you think is a bug, then you can help to be part of the solution. To do this, you'll need to give us the following information

- The program you were running, and the exact command line arguments you supplied to that program
- A small test data set that can replicate the problem you observed
- The behavior that you think was faulty, and if possible what you expected to see. This can be very useful when a computation produces an answer that doesn't make sense.