ScreenSeq

Streamlining a pipeline for finding perturbations that reset Mono-Allelic Expression

Alex Gyujin Kim, Gimelbrant Lab
Harvard DBMI Summer Institute

June 11 - August 11, 2018

Background

Random monoallelic expression (MAE) occurs when the expression of one parent's allele is heavily favored over the other by random choice. It had generally been assumed that both maternal and paternal alleles were equally expressed (biallelic expression, BAE) in genes that lied on autosomal chromosomes. X-inactivated genes in females, which is when all genes lying on one X chromosome is silenced epigenetically, were the only genes thought to show any allelic bias in expression. However, it has been shown that a surprising number of autosomal genes display MAE characteristics. Using machine learning techniques to look at epigenetic markers in clonal lymphoblastoid cell lines, 1,264 genes on autosomes were identified to display MAE while 9,428 showed some kind of BAE. In other types of clonal cell lines, 4,227 genes were identified to display MAE and 6,006, BAE.[1]

While the mechanisms behind what affects a cell's independent choice to express the maternal, paternal, or both alleles are not yet fully understood, it has been discovered that MAE genes in clonal cells are remarkably stable throughout development. Clonal cells will exhibit the same allele frequencies as their derivatives. Therefore, reproducible allele specific expressions across clonal generations make it easy to observe the MAE phenomenon *in vitro.* Differential allele expressions is thought to be attributed to DNA sequence polymorphisms, more specifically single nucleotide polymorphisms (SNPs) in regulatory regions of the genome and copy number variations (CNVs), where regions of the genome repeat.[2]

The large number of MAE characterized autosomal genes make this an interesting problem to study. Having a greater understanding of MAE may provide a

---

[1] Savova V, Chun S, Sohail M, et al. Genes with monoallelic expression contribute disproportionately to genetic diversity in humans. *Nature genetics*. 2016;48(3):231-237. doi:10.1038/ng.3493.

[2] Gendrel Anne-Valerie, Marion-Poll Lucile, Katoh Kimiko, Heard Edith. Random monoallelic expression of genes on autosomes: parallels with X-chromosome inactivation. *Seminars in Cell and Developmental Biology* http://dx.doi.org/10.1016/j.semcdb.2016.04.007

mechanism for altering stable allele specific expressions to combat diseases where an allele deficiency may lead to a loss-of-function during development. With the rapid discovery of autosomal disorders that have been associated with detrimental protein levels in cells, the discovery of gene therapies and other treatments to regulate the expression of these autosomal MAE loci may be clinically impactful.

Introduction

My primary focus of research over this summer concerned streamlining a pipeline for finding perturbations that reset monoallelic expression. Part of the effort in trying to understand the mechanism of allelic imbalance in MAE genes involves looking for perturbations that can induce an allelic shift. Because it seems that only a small fraction of MAE genes are significantly affected by the introduction of perturbations, screening techniques that allow biologists to test multiple perturbations at once on several readout genes are valuable in isolating which perturbation induced a significant shift on which readout gene. Examples of perturbations that have been screened in past and ongoing experiments were drugs and RNA interfering shRNA's.

This need for a scalable method to test multiple treatments is satisfied by a technology called ScreenSeq— a systematic way of screening multiple perturbations on a plate of clonal cells.

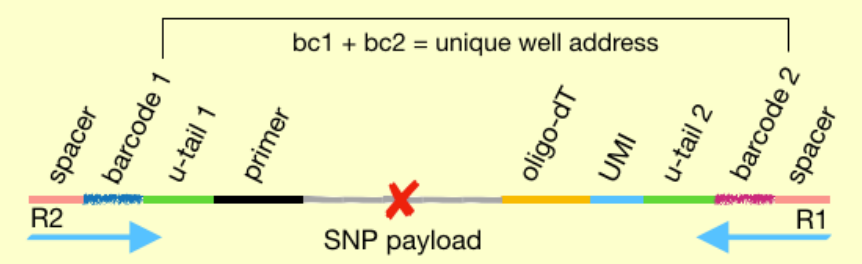


Figure 1. The final product of ScreenSeq. The oligo-dT + UMI + universal PCR tag is added to the right of the SNP payload via reverse transcription to create cDNA from RNA. A primer that binds upstream the SNP binds another PCR u-tail, and a round of PCR uses both u-tails to amplify this SNP region of interest. Barcode identifiers (each 6 bases long) are added via PCR using the u-tails as primers. The combination of the barcodes are unique for each well of the plate. Finally spacers are attached to each end to improve sequencing quality. The downstream paired end that the sequencer reads contains UMI and barcode 2 data. The upstream paired end holds barcode 1 and SNP data. This stepwise construction is accomplished by two rounds of PCR for multiple genes (SNPs) of interest at the same time.

An in-depth knowledge of ScreenSeq is not necessary to understand my contributions to its data pipeline , however a general overview may help contextualize the procedure. A biologist would begin with an 8 by 12 well plate of clonal cells and introduce a different perturbation into each well to induce a change in RNA expression. In each well, the resulting RNA is reverse transcribed into cDNA with UMI tags. The cDNA is then amplified with PCR and multiplex PCR primers for the SNP variants in

MAE genes of interest. Lastly, a barcode plate is added. Two six nucleotide long barcode tags are added to each well and attached via PCR to flank the SNP payload. The combination of the forward and reverse barcode creates a unique 12 base well address that will be sequenced and used to differentiate the data between wells. All in all, the final product in each of these wells will be amplicons of SNP variants, the payload that determines allele variant, that have been surrounded by different elements attached via PCR or Reverse Transcription (Figure 1).

Each plate is then pooled into microPCR tubes to create a RNA sequencing library. Once multiple sequencing libraries are prepared for each plate that was made (i.e., different days after treatment was introduced, different perturbation maps that were used, different barcode maps), these libraries are sent to Illumina to be sequenced and output FASTQ files, R1 and R2 files for each paired end.

Status Quo - The Problem

The overarching objective of data analysis after sequencing is to be able to identify which perturbations had an effect on shifting the allelic imbalances of which gene. Being able to get from pooled plate RNAseq data to SNP counts of each allele variant for each readout gene for each perturbation is a difficult task in data manipulation, as it requires data from multiple sources to be aggregated.

The current pipeline had many disadvantages. A script would align the paired end sequenced reads to a reference genome (mm10) and count the SNP occurrences for each gene for each perturbation. This script took in a YAML configuration file, a text file where parameters could be defined, that required the user to manually copy and paste information regarding which barcode was associated with which perturbation. Unique perturbation identifiers in the format of *{alpha}_{perturbation}_{day}: {fwd_barcode} _{rev_barcode}* had to be either constructed by hand or in Excel using string concatenation formulas. Other equally important parameters to be manually defined were directories where the R1 and R2 FASTQ files were located, output file directories for the SAM alignment files, information on the target genes of interest, where the scripts to be run were located, and other ScreenSeq run specific parameters. Apart from this approach being very human error prone, there were many parameters that were unnecessarily presented to the experimenter creating these files. Figure 2 shows a screenshot of a part of this YAML file. Furthermore, each YAML file contained “pools” of three plates to be aligned and counted together in order to reduce the number of times these YAML files had to be constructed. This meant that every YAML file contained the barcodes and genes data for each well in three 96-well plates.

A minor time consuming step in aligning RNAseq data was also problematic. The Bowtie aligner was used to align to the mouse reference genome, however there was no built in option to include both paired ends if either of them did not align to the reference. In several of the sequencing runs, the paired ends were asymmetric, in that the lengths of R1 and R2 were not the same. R1 was the longer read that contained the

SNP payload. R2 was a shorter read that contained the metadata (barcode + UMI information) for the biological read. Because the R2 could not align to the reference, Bowtie would omit the R2 ends from the alignment files while only keeping the R1 ends. Without this metadata in the corresponding R2 end, it would be impossible to determine which R1 SNP was derived from which well, which is why Dr. Gimelbrant created a small Perl script to manually inject these missing R2 reads into the SAM alignment file that Bowtie outputted. Because some sequenced paired end data were asymmetric while others were symmetric, the experimenter was required at the start of the pipeline to define an asymmetric boolean to tell the pipeline to run the extra script or not.

```
  ## In step2.pl defines % of BAM to subsample. Useful for a speedup when testing
  ## Normally: comment it out!

alignment:
#~ file_names:
  ### the following applies to step 1:::
  fastq_dir: /n/scratch2/ak583/screenseq/FUYANG/fastq/
  R1: R1_test.fastq
  R2: R2_test.fastq
  reference: mm10
  ### the following applies to step 1 + step 2:::
  SAM_name_base: FUYANG_6-5_mm10
  SAM_location: /home/ak583/Gimelbrant_Lab/scratch/screenseq/FUYANG/sam_old

#### Description of targets (step2)

genes:

  NM_010954_Ncam2_MAE:
    SNPFlank_left: CATACAATT
    position: chr16:81596071-81596364
    amplicon_size: 294
  NM_194355_Spire1_MAE:
    SNPFlank_left: ACGGAGTGT
    position: chr18:67490796-67491034
    amplicon_size: 239

samples_barcodes:
  aCtcf_1_D19: AGTCAA_AGTTCC
  bRnf2_1_D19: AGTTCC_ATGTCA
```

Figure 2. Screenshot of a part of the YAML configuration file for the existing pipeline. *genes* represent the readout genes to be targeted by the perturbations. For brevity, only two genes are listed here but a typical YAML file will contain 25 - 30 genes. *samples_barcodes* contains information about the perturbation, replicate, day, and unique well barcode identifiers in one concatenated string. Each row represents a well in a plate.

After the RNAseq data was aligned, step two of this pipeline would loop through the relevant genes in the aligned SAM file. For each perturbation for each gene, it would make SNP counts by assigning a simple A, C, G, T count to each unique perturbation identifier. Lastly, a counts file would be output, this would be pasted into an excel sheet, and an excel formula would calculate the allelic bias (a number from 0 to 1) using these counts. An additional column of cell viability, a qualitative measure of how many cells survived when the perturbation was introduced, would have to be pulled from another Excel worksheet and appended to the counts table. This counts table would then be read in by an R script to produce large master ridge plots for each perturbation.

The disadvantages of the status quo was clear. Copy and pasting from Excel sheets and using Excel formulas is time consuming, error prone, and difficult to document the reconstruction of data. The need to aggregate multiple pieces of

information into one YAML file per pool of plate was a whole day effort. There were extraneous parameters in the YAML file that could contribute to error. The script to inject unpaired end reads into the Bowtie alignment files was a workaround that was time consuming and easily avoidable. The pooling together of plates added yet another level of complexity that required highly structured project directories that were difficult to keep track of. And finally, the master ridge plots (Figure 3), were large and intimidating to analyze. The need for a more streamlined approach was crucial in expediting the screening of new perturbations.

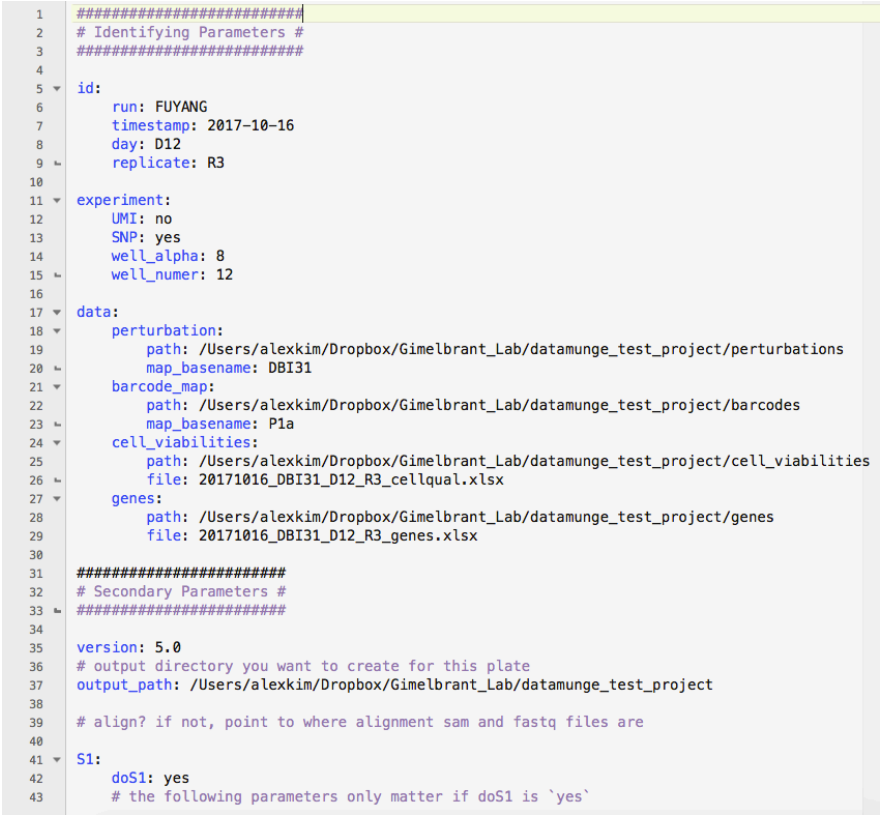
Figure 3. A master ridge plot file for perturbation Atf7ip. One of these master ridge plots is created for each of the 96 perturbations for each plate. Picking out interesting perturbations and readout genes from these visualizations can be tedious and error prone.

Methods

Streamlining the Pipeline

I decided to design the new pipeline so that each plate would be considered separately and not as pools. This decision would anticipate the need to add or remove plates from different project runs, as contaminated plates could easily be discarded or replaced with a new plate prepared on a different day.

The only manual work that an experimenter would need to do is to define the parameters in one YAML file per plate. These YAML files would not contain any parameters that revealed actual biological data (barcode, genes, perturbations); rather, it would abstract these data by requiring pointers to the files that contained this information. This removes the need to munge any data prior to aligning. While in the old pipeline, the unique perturbation identifier and barcode had to be concatenated together with a series of underscores and semicolons (*{alpha}_{perturbation}_{day}: {fwd_barcode}_{rev_barcode}*) for every well, the new pipeline abstracts that whole process. A more in-depth explanation of the pipeline will follow. The new plate YAML file on average comes out to 80 lines, compared to the 300-400 lines of the old YAML file. See Figure 4 in comparison to Figure 2.



```
#########################
# Identifying Parameters #
#########################

id:
    run: FUYANG
    timestamp: 2017-10-16
    day: D12
    replicate: R3

experiment:
    UMI: no
    SNP: yes
    well_alpha: 8
    well_numer: 12

data:
    perturbation:
        path: /Users/alexkim/Dropbox/Gimelbrant_Lab/datamunge_test_project/perturbations
        map_basename: DBI31
    barcode_map:
        path: /Users/alexkim/Dropbox/Gimelbrant_Lab/datamunge_test_project/barcodes
        map_basename: P1a
    cell_viabilities:
        path: /Users/alexkim/Dropbox/Gimelbrant_Lab/datamunge_test_project/cell_viabilities
        file: 20171016_DBI31_D12_R3_cellqual.xlsx
    genes:
        path: /Users/alexkim/Dropbox/Gimelbrant_Lab/datamunge_test_project/genes
        file: 20171016_DBI31_D12_R3_genes.xlsx

#######################
# Secondary Parameters #
#######################

version: 5.0
# output directory you want to create for this plate
output_path: /Users/alexkim/Dropbox/Gimelbrant_Lab/datamunge_test_project

# align? if not, point to where alignment sam and fastq files are

S1:
    doS1: yes
    # the following parameters only matter if doS1 is `yes`
```

Figure 4. Screenshot of a part of the YAML configuration file for the new pipeline. Perturbation, barcode, cell_viability, and gene data are kept discrete. No biological data is revealed. The identifying parameters are used to differentiate plates from each other. The secondary parameters affect several steps of the pipeline.

Once the YAML file has been defined, the user can pass it into the Rscript that begins the pipeline.

The parameters will be parsed, then an empty plate object will be created. The empty plate object is designed to hold information that is specific to each well in a nested list format. Using this data structure gives greater flexibility in getting and setting data about the plate throughout various steps of the pipeline. The empty plate object is

populated step-by-step. First, the barcode file specified in the YAML file is read and added to their corresponding wells in the plate object. Next the perturbations file is read and added. And lastly, the cell viability data. In the end, we are left with a plate object that carries *barcode*, *perturbation*, and *cell viability* data for each well. For this particular plate, we also have information about the *run name*, *timestamp* (date the plate was read), *day* (days after perturbations were introduced), *replicate* (which biological replicate of clonal cells were used), and a *UMI boolean* (whether UMI's were added during ScreenSeq). Having a proper data structure to hold all this data removes the need to manually munge the data together into a concatenated string.

This plate object is automatically written to a configuration YAML file that mimics the file that a user would have had to create in the old pipeline for the aligning and counting pipeline. While this is not the most efficient way to get from one step of the pipeline to the other, it is a quick way to take advantage of an aligning and counting pipeline that has already been established in the lab. Future directions with this would be to rewrite some of these existing scripts to use the data in the plate object without having to write an intermediate YAML. The configuration file that is written is then passed into the existing alignment script, which aligns the FASTQ files to a reference genome.

It is also important to touch on some of the secondary parameters that give the user some flexibility in how the pipeline should run. In addition to the identifying parameters, the secondary parameters contain information about where the project directory should be created, where the sequencer's FASTQ files are, whether or not the FASTQ files should be aligned, and whether or not this plate should be counted again. The last two options were implemented to account for the fact that FASTQ files are not run-specific. Re-aligning FASTQ's that have already been aligned is time consuming. Users may also want to rerun the pipeline on data that has been changed to obtain new counts, which is why the last parameter exists. Both aligning and counting is turned on by default.

In addition, the Bowtie aligner was replaced with STAR aligner, which had a built in option to include both paired ends regardless of whether either end did not align to the reference genome. This change removed the need to define a parameter where the biologist would need to specify whether or not the paired end reads from the sequencer were asymmetric or not. Most importantly, it removed the time consuming step of manually injecting unaligned ends.

Some minor improvements were also made. Excel worksheets can be remarkably cumbersome to work with and read into R. Importing gene names that resemble months and days may even be automatically casted into a Date object when imported into Excel. I circumvented this issue by requiring all data files to be in the format of tab separated value files (TSV), which are both human and computer friendly.

Once the aligning and SNP counting is finished, the new pipeline calculates allelic biases from the counts table that is output from that pipeline. Cell viability data

taken from the filled plate object is added to this table and ultimately a screen summary file is created, which contains the allelic ratios for each readout gene for each perturbation. In other words, this data carries time dependent information about the allelic effect that the perturbation had on a particular gene. This screen summary table can be uploaded into an interactive visualization Shiny app to visualize and analyze the data in slices.

Interactive Visualization Shiny App

The previous pipeline created large master ridge plots for each perturbation (Figure 3) that were difficult to parse and analyze. A more intuitive way of analyzing this information would be in slices, where the biologist would be able to look at only a subset of perturbations and readout genes at a time. At the same time, it would be helpful to produce these comprehensive master ridge plots so that biologists could eyeball interesting signals that stood out. I decided to tackle this visualization problem using R Shiny.

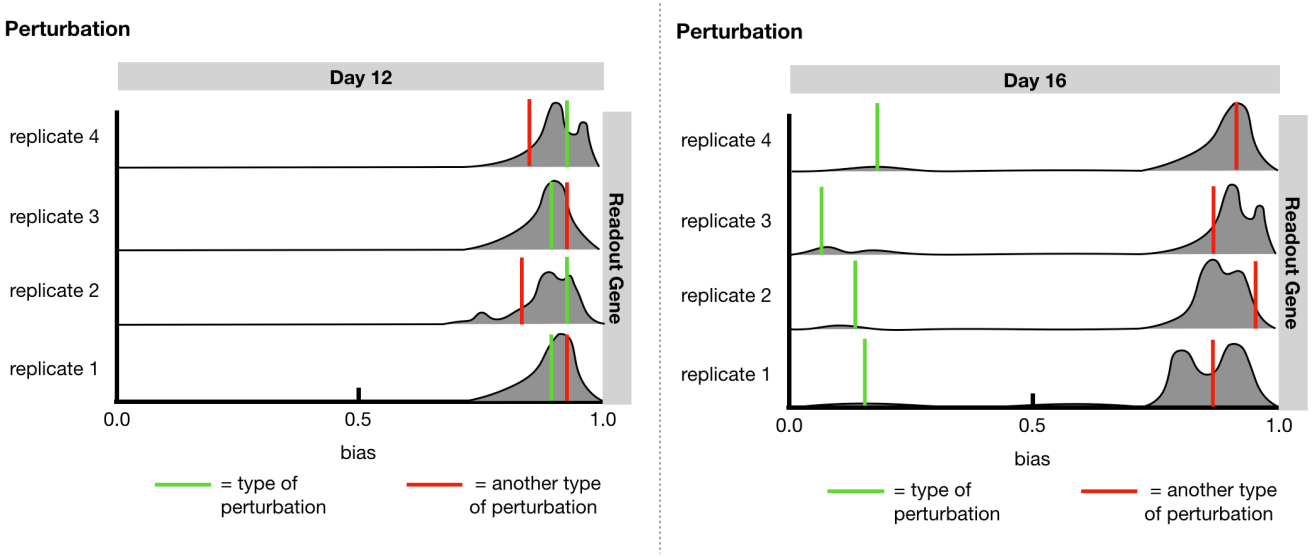


Figure 5. A pictorial representation of ridgeplots that were being produced with the old pipeline. Instead of showing the comprehensive version that shows multiple days and multiple readout genes, this shows the effect of a perturbation on just one readout gene for days 12 and 19 after the perturbation was introduced. In the top left corner is the perturbation that was introduced, the readout gene is on the right, and the day is at the top. The replicates on the y-axis are simply biological replicates, different clonal cell populations taken from the same derivative cell, that represent different trials of the same experiment. The x-axis indicates an allelic bias from 0 to 1. The mountain ridges represent the distribution of all of the cells' allelic biases in the wells of the plate for this particular readout gene. In MAE genes, it is expected to see the ridges shifted significantly towards the 0 or 1. If the mountain is in the middle, the gene would show BAE characteristics. The colored vertical bars are the cells in the well that received a particular perturbation. Analysis of these plots would involve looking at where the vertical bars lie relative to the normal distribution. The different colored bars are different versions of a treatment (different dosages of drugs, different shRNA binding sites of the same target gene).

A pictorial representation of a slice of these master ridge plots may help in understanding the rationale behind the signals that biologists are looking for. Figure 5 is a pictorial representation of a perturbation inducing an allelic shift in the other direction. In Day 12, all wells showed significant bias for one allele over the other. However by Day 19, the cells in wells that had received the green version of this perturbation showed an allelic bias of 0, indicating that the allelic expression of this gene had been reset to favor an allele that had previously been silent. Because allele expressions are very difficult to change as they're stably maintained throughout development, being able to identify which perturbations affect the allelic biases of which genes is an important step towards understanding the mechanisms of MAE.

My application allows users to produce such ridge plots by uploading screen summary files and defining specific parameters that they want to look at. The perturbation, perturbation type, replicate, day, and readout gene can be defined to look at slices of all the data in the screen_summary file that is uploaded. An example of this is in Figure 6. If the user wants to reproduce the master ridge plots to visualize all the readout genes at once, they can simply select all the readout genes.

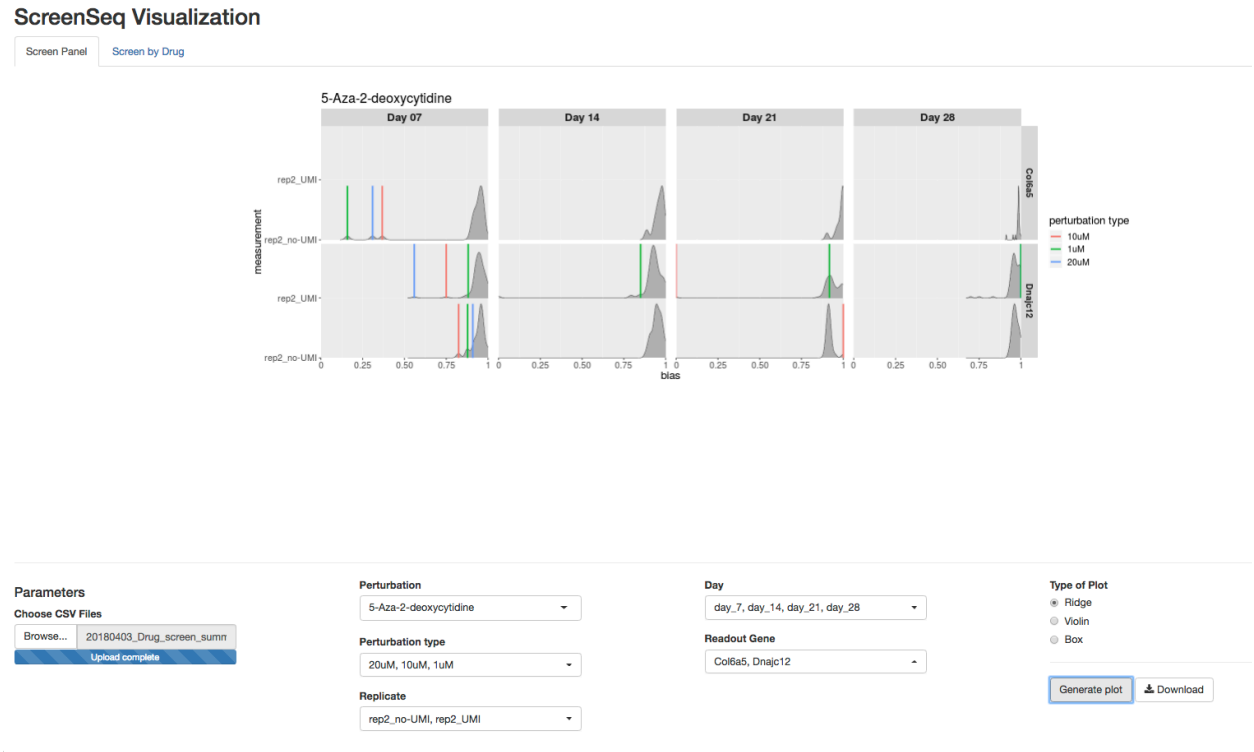


Figure 6. A screenshot of the application that represents the sliced ridge plots for the following parameters: perturbative drug *5-Aza-2-deoxycytidine*, three dosages of the drug, read out genes *Col6a5* and *Dnajc12*, days 7, 14, 21, and 28, and replicate 2 with and without UMI's. Parameters can be defined at the bottom and the plot will be generated above. The plot can also be downloaded as a PNG.

The app also allows for the option to visualize the data as ridge, violin, or box plots (Figure 7).

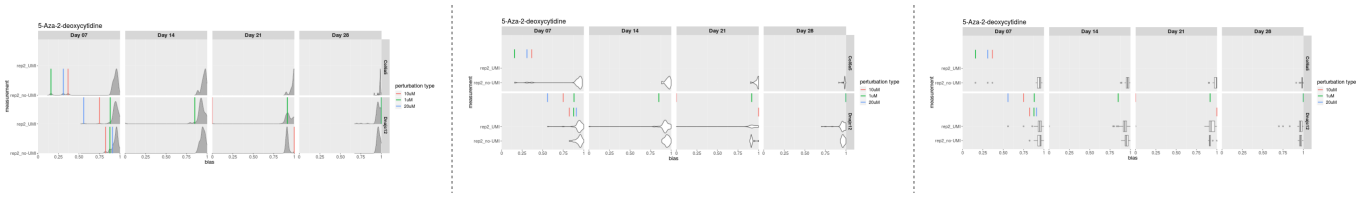


Figure 7. A plot type parameter can be set to visualize the same data in Figure 6 with box plots and violin plots.

Another way to visualize this data is by readout gene rather than by perturbation. It can be especially helpful to see what effect the screen of perturbations had on one particular readout gene. Figure 8 shows how this visualization can be produced by choosing the days and readout genes on the left sidebar.

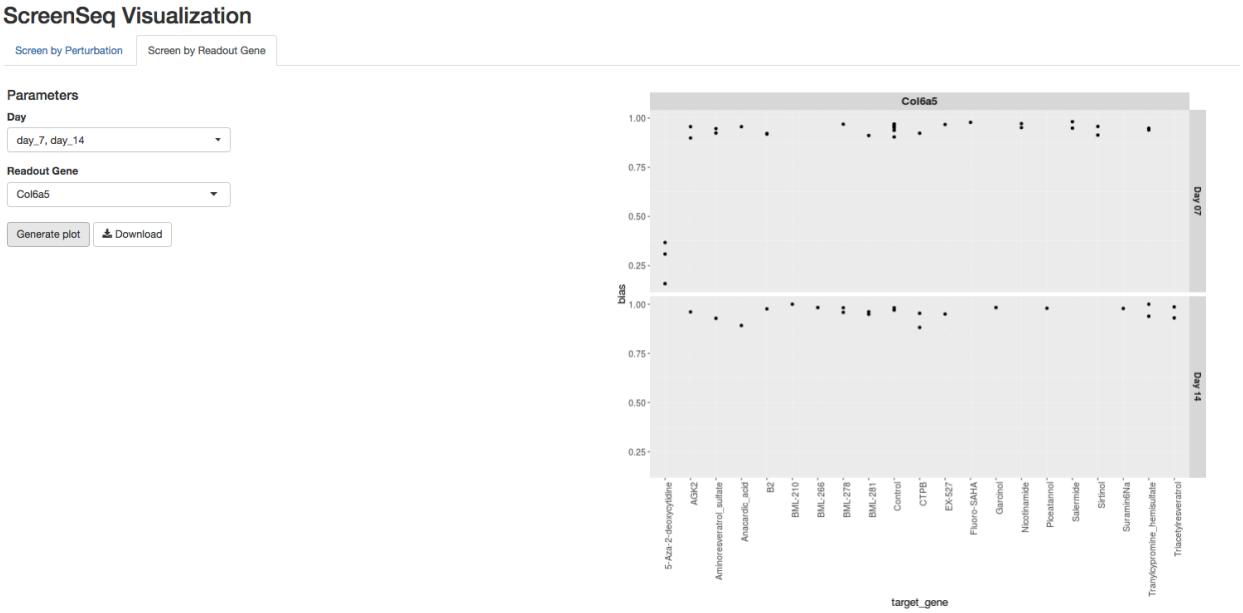


Figure 8. A visualization for one readout gene. This plot displays the allelic effect that a screen of perturbations had on a particular readout gene across days 7 and 14. The drug of interest here would be the first column indicated by *5-Aza-2-deoxycytidine.* By day 7, cells that received the *5-Aza* treatment showed an allelic shift towards 0 away from the norm. By Day 14, the cells died from the perturbation, which is why there are no data points for that day.

This application can be used to parse through a number of different slices of ScreenSeq data in order to gain insight into the experimental results. The next steps for biologists would be to identify which perturbations look promising in their ability to reset allelic expressions, and analyze them further with a single cell capture technique like digital droplet PCR (ddPCR).

Getting Started

1. Go to https://alexkim.shinyapps.io/screenseq-viz/ to access the app.

2. Upload a screen_summary.csv for the plate you want to visualize. Multiple files can be uploaded for visualizing multiple plates.
3. Select relevant parameters and click generate plot. There is an option to download the plot.
4. Move between tabs for the two different visualizations.

Supplementary Apps

I have also developed supplementary apps for biologists to use while preparing barcode plates and reading cell viabilities.

Barcode Helper App

Barcode maps for ScreenSeq are plates that are prepared in a way so that each well of a plate contains a unique combination of two six base sequences. One of these barcodes will be appended upstream of the SNP payload while the other will be appended downstream the SNP payload via multiplex PCR (Figure 1). In order to prepare a standard 8 x 12 ScreenSeq plate, these two barcode tags along with their universal tails must be pipetted into each well of the plate in such a way that no two wells in that plate share a combination of barcodes. This combinatorial puzzle can be abstracted to make plate preparation less tedious and less error prone.

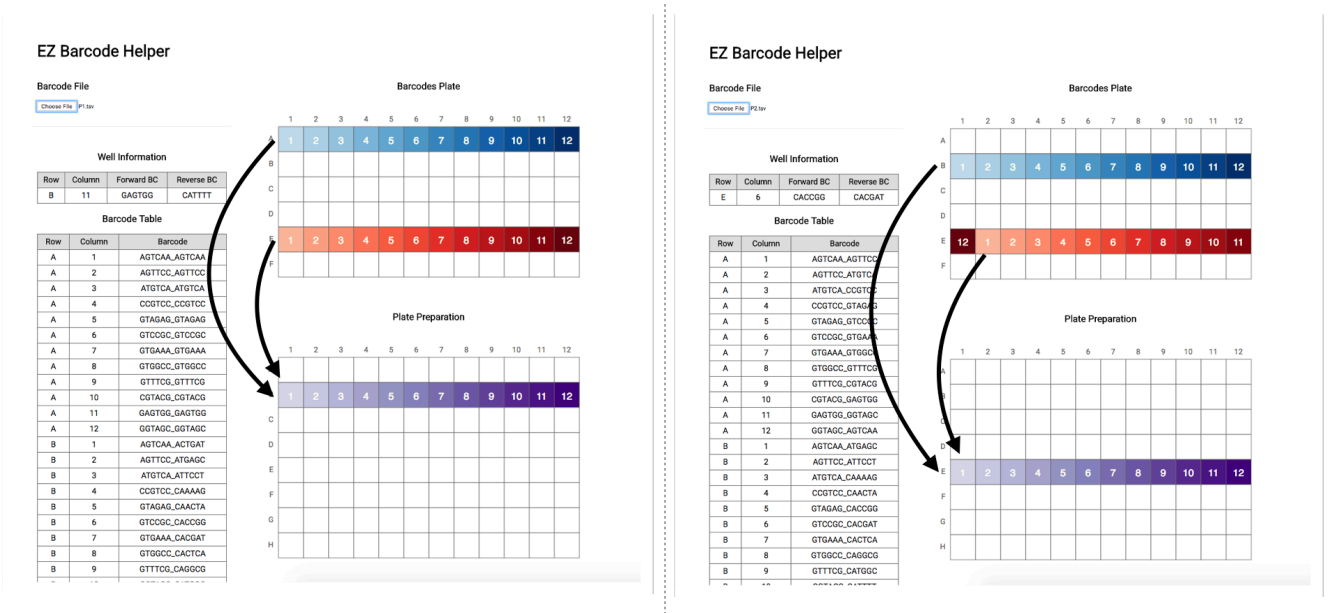


Figure 9a & b. A screenshot of the Barcode Helper App. The arrows indicate which two barcode rows must be combined into which row in the plate to prepare. *Figure 9a.* The picture on the left is a barcode configuration that has been uploaded where we do not see any offset. *Figure 9b.* The picture on the right is a different barcode configuration where the second barcode row must be offset by one to produce a different map of barcodes for a different plate. The colors and numbers are designed to help the user identify this offset.

Before a barcode map is prepared, a standard 6 x 12 well barcode library is made. The first three rows are labeled A, B, and C, and each well in those rows contain unique 6 base DNA sequences attached to universal tail 1. These first three rows carry the forward barcodes that will be placed downstream the SNP. The next three rows are labeled D, E, and F, and these rows carry the same 6 base DNA sequences as the first three rows attached to universal tail 2. These last three rows are the reverse barcodes that will be attached upstream the SNP.

A barcode map can be prepared by combining any two of these library rows. The application, shown in Figure 9, makes this task easier by using color gradients and numbers to make this process intuitive. The plate at the top is the barcode library plate that has already been pre-made, while the plate at the bottom indicates the plate to prepare. When a row from the preparation plate is hovered over, two barcode rows will light up, with numbers and colors indicating which two wells (blue indicates the forward barcodes while red the reverse ones), correspond to which purple well. Well specific information about the forward and reverse barcode sequences will also show up on the left sidebar. Furthermore, there is also a need to produce different barcode maps for different plates, in case multiple plates are to be sequenced at the same time. To accommodate this, a variety of barcode configuration files can be uploaded to visualize different pipetting instructions (Figure 9b).

Getting Started

1. Follow the instructions in the README to get a local server running. (https://github.com/alexkim205/EZ-Barcode-Helper)
2. Upload your barcode configuration file.
3. Hover over the bottom preparation plate to explore interactive features.

Cell Viability Input Shiny App

Cell Viability is a qualitative measure of how many cells survive after the perturbation has been introduced. Levels of viability are manually recorded for each well of a plate using a microscope and Excel, which is a tedious and error prone task in itself. To standardize the input of cell viability data, I developed a Shiny App that would map these levels of viability to a few keystrokes (namely, *1, 2, 3, 4, 5, 6, 7, 8, 9, 0, -*). With one keystroke, a biologist would be able to assign a level of viability to a well. A row in a table will automatically be appended for each well, giving the biologist the advantage of reading multiple plates quickly. The table is directly editable in the case of mistypes and several plate specific parameters can be defined at the top which are concatenated into a filename. The cell viabilities table can then be exported to a TSV file with this filename. See Figure 10.

This application standardizes the collection of cell viability data and puts it into a format that can be easily read by the streamlined pipeline.

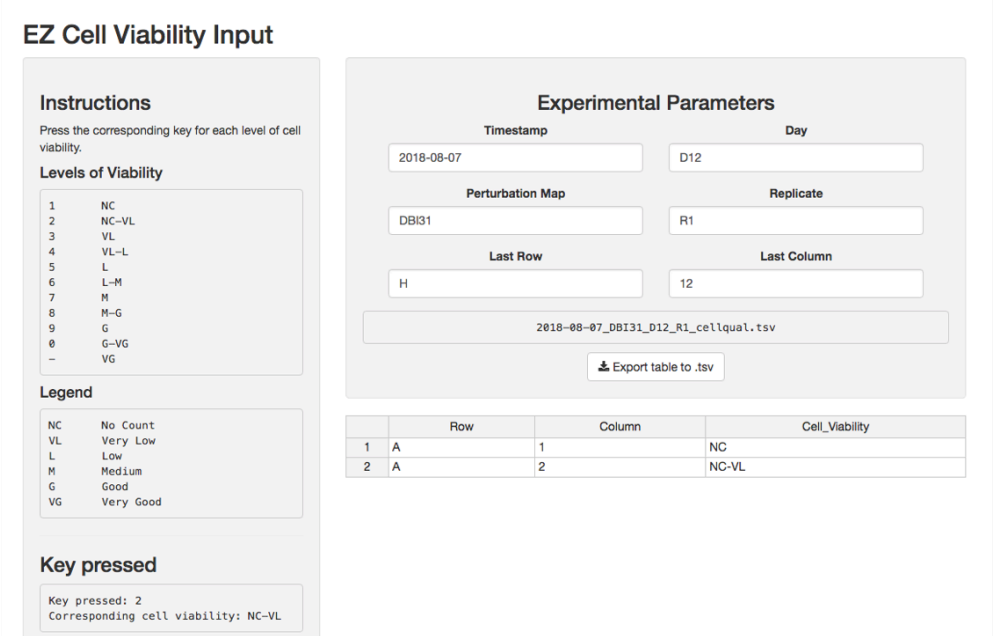


Figure 10. A demonstration of the Cell Viability Shiny App. The left sidebar displays which keystrokes correspond to which levels of viability. These levels range from NC (No Cells/Count) to VG (Very Good). Experimental plate-specific parameters can be set at the top. Every keystroke appends a row corresponding to a well on the plate to the data table below. The data table can be exported as is to a TSV.

Getting Started

1. Go to https://alexkim.shinyapps.io/cell_viability_input/ to access the app.
2. Define parameters to determine file name.
3. Press any key in `123456780-`. Observe the rows that are appended to the table on the right.

Conclusion

ScreenSeq has been a systematic method of screening multiple perturbations across clonal cell lines that the Gimelbrant Lab has used for many years. Saumya Gupta, a postdoc at the lab, has screened multiple drugs and observed their effects on multiple readout MAE genes of interest using this technique. Research on the effect of shRNA perturbations via RNA interference is currently being pushed forward by Clara Pereira, who I worked closely with during the summer to fine tune and optimize several parts of this pipeline. Because of the highly unorganized nature of the data that comes out of ScreenSeq, it is essential that we provide a pipeline that will integrate all the data in an intuitive and clean way. The existing pipeline was insufficiently automated, and the manual data wrangling that compensated for these insufficiencies hurt efficient data exploration.

The supplementary tools and streamlined pipeline should eliminate the time needed to parse through the copious amounts of ScreenSeq data and should also give further insight into identifying interesting signals. Immediate considerations for the future involve coming up with new visualizations of ScreenSeq data that can be implemented into the app for further analysis. Perhaps adding a tab for visualizing ddPCR data for specific perturbations that have been analyzed further will provide additional insight. Implementing this would also mean adding a branch to the pipeline that deals with ddPCR data as well. For the pipeline, the YAML can be abstracted even further with a Shiny App of parameters with dropdown menus and text boxes that will generate a downloadable YAML file. However, there are many more improvements that could be made, most of which will come as I continue to work closely with the biologists of the lab and respond to their needs at the bench.

All of my work has been documented and pushed to the Gimelbrant GitLab as individual projects. Beta versions of the Cell Viability Input and Interactive Visualization Shiny Apps have been deployed on my personal shiny server. Links can be found in the table below.

App	Gitlab	Deployed App
Barcode Helper App	https://gitlab.com/gimelbrant-lab/EZ-Barcode-Helper	Follow instructions in README
Cell Viability Input Shiny App	https://gitlab.com/gimelbrant-lab/Cell-Viability-Input-Shiny	https://alexkim.shinyapps.io/cell_viability_input/
Streamlined ScreenSeq Pipeline	https://gitlab.com/gimelbrant-lab/ScreenSeq-pipeline	Command Line, see README
Ridge Plot Visualization Shiny App	https://gitlab.com/gimelbrant-lab/EZ-Ridgeplot-Analysis-Shiny	https://alexkim.shinyapps.io/screenseq-viz/