Supplementary Manual 1
CellTagR Tutorial

R Package - CellTagR

This is a wrapped R package of the workflow (https://github.com/morris-lab/CellTagWorkflow) with additional assessment of the complexity of the Celltag Library sequences.

For details regarding development and usage of CellTag, please refer to the following paper - *Biddy et. al. Nature, 2018*, https://www.nature.com/articles/s41586-018-0744-4

Install devtools

```
install.packages("devtools")
```

Install the package from GitHub.

```
library("devtools")
devtools::install_github("morris-lab/CellTagR")
```

Load the package

```
library("CellTagR")
```

Assessment of CellTag Library Complexity via Sequencing

In the first section, we would like to evaluate the CellTag library complexity using sequencing. Following is an example using the sequencing data we generated in lab for pooled CellTag library V2.

1. Read in the fastq sequencing data and extract the CellTags

The extracted CellTag will be stored as attribute (fastq.full.celltag & fastq.only.celltag) in the result object.

```
# Read in data file that come with the package
fpath <- system.file("extdata", "V2-1_R1.zip", package = "CellTagR")
extract.dir <- "."
# Extract the dataset
unzip(fpath, overwrite = FALSE, exdir = ".")
full.fpath <- paste0(extract.dir, "/", "V2-1_S2_L001_R1_001.fastq")
# Set up the CellTag Object
test.obj <- CellTagObject(object.name = "v2.whitelist.test", fastq.bam.directory = full.fpath)
# Extract the CellTags
test.obj <- CellTagExtraction(celltag.obj = test.obj, celltag.version = "v2")
```

2. Count the CellTags and sort based on occurrence of each CellTag

```
# Count and Sort the CellTags in descending order of occurrence
test.obj <- AddCellTagFreqSort(test.obj)
# Check the stats
test.obj@celltag.freq.stats
```

3. Generation of whitelist for this CellTag library

Here are are generating the whitelist for this CellTag library - CellTag V2. This will remove the CellTags with an occurrence number below the threshold. The threshold (using 90th percentile as an example) is determined: floor[(90th quantile)/10]. The percentile can be changed while calling the function. A plot of CellTag reads will be plotted and it can be used to further choose the percentile. If the output directory is offered, whitelist files will be stored in the provided directory. Otherwise, whitelist files will be saved under the same directory as the fastq files with name as _whitelist.csv (Example: v2_whitelist.csv).

```
# Generate the whitelist
test.obj <- CellTagWhitelistFiltering(celltag.obj = test.obj, percentile = 0.9, output.dir = NULL)
```

The generated whitelist for each library can be used to filter and clean the single-cell CellTag UMI matrices.

Single-Cell CellTag Extraction and Quantification

In this section, we are presenting an alternative approach that utilizes this package to carry out CellTag extraction, quantification, and generation of UMI count matrices. This can be also accomplished via the workflow supplied - https://github.com/morris-lab/CellTagWorkflow.

Note: Using the package could be slow for the extraction part. For reference, it took approximately an hour to extract from a 40Gb BAM file using a maximum of 8Gb of memory.

1. Download the BAM file

Here we would follow the same step as in https://github.com/morris-lab/CellTagWorkflow to download the a BAM file from the Sequence Read Archive (SRA) server. Again, this file is quite large. Hence, it might take a while to download. The file can be downloaded using wget in terminal as well as in R.

```
# bash
wget https://sra-download.ncbi.nlm.nih.gov/traces/sra65/SRZ/007347/SRR7347033/hf1.d15.possorted_genome_bam.bam
```

OR

```
download.file("https://sra-download.ncbi.nlm.nih.gov/traces/sra65/SRZ/007347/SRR7347033/hf1.d15.possorted_genome_bam.b
```

2. Create a CellTag Object

In this step, we will initialize a CellTag object with a object name and the path to where the bam file is stored.

```
# Set up the CellTag Object
bam.test.obj <- CellTagObject(object.name = "bam.cell.tag.obj", fastq.bam.directory = "./hf1.d15.bam")
```

Note: The following tutorials are only intended for ONE CellTag version. To obtain information for all three version of CellTags, it is required to run the following pipeline for each CellTag version independently, i.e. finishing process for V1 and then repeat the procedure for V2 and so on. By the end of running through the pipeline for various CellTag versions, the clonal information of each will be stored in the same object, which can be used to carry out network construction and visualization.

3. Extract the CellTags from BAM file

In this step, we will extract the CellTag information from the BAM file, which contains information including cell barcodes, CellTag and Unique Molecular Identifiers (UMI). The result generated from this extraction will be a data table containing the following information. The result will then be saved into the slot "bam.parse.rslt" in the object in the following format.

Cell Barcode	Unique Molecular Identifier	CellTag Motif
Cell.BC	UMI	Cell.Tag

```
# Extract the CellTag information
bam.test.obj <- CellTagExtraction(bam.test.obj, celltag.version = "v1")
# Check the bam file result
head(bam.test.obj@bam.parse.rslt[["v1"]])
```

4. Quantify the CellTag UMI Counts and Generate UMI Count Matrices

In this step, we will quantify the CellTag UMI counts and generate the UMI count matrices. This function will take in two inputs, including the barcode tsv file generated by 10X and celltag object processed from Step 2. The barcode tsv file can be either filtered or raw. **However, note that using the raw barcodes file could result in a requirement of large memory for using this function**. If filtered barcodes files are used, **only cell barcodes that appear in the filtered barcode file** will be preserved. The result will also be saved as a *dgCMatrix* in a slot - "raw.count" - under the object. At the same time, a initial celltag statistics will be saved as another slot under the object. The matrix will be in the format as following.

	CellTag Motif 1	CellTag Motif 2	<all tags detected>	CellTag Motif N
Cell.BC	Motif 1	Motif 2	<all tags detected>	Motif N

```
# Generate the sparse count matrix
bam.test.obj <- CellTagMatrixCount(celltag.obj = bam.test.obj, barcodes.file = "./barcodes.tsv")
# Check the dimension of the raw count matrix
dim(bam.test.obj@raw.count)
```

The generated CellTag UMI count matrices can then be used in the following steps for clone identification.

Single-cell CellTag UMI Count Matrix Processing

In this section, we are presenting an alternative approach that utilizes this package that we established to carry out clone calling with single-cell CellTag UMI count matrices. In this pipeline below, we are using a subset dataset generated from the full data (Full data can be found here: https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE99915). Briefly, in our lab, we reprogram mouse embryonic fibroblasts (MEFs) to induced endoderm progenitors (iEPs). This dataset is a single-cell dataset that contains cells collected from different time points during the process. This subset is a part of the first replicate of the data. It contains cells collected at Day 15 with three different CellTag libraries - V1, V2 & V3.

1. Read in the single-cell CellTag UMI count matrix

This object is what we have generated from the above steps using bam files. As mentioned before, bam files take a long time to process. Hence, in this repository, we include a sample object saved as .Rds file from the previous steps, in which raw count matrix is included in the slot - "raw.count"

```
# Read the RDS file and get the object
dt.mtx.path <- system.file("extdata", "Demo_V1.Rds", package = "CellTagR")
bam.test.obj <- readRDS(dt.mtx.path)
```

(RECOMMENDED) Optional Step: CellTag Error Correction

NOTE: If CellTag error correction was **NOT** planned to be performed, skip this step and move to the *Step 2 - binarization*, in which the raw matrix will be used. Otherwise, before binarization and additional filtering, we will carry out the following error correction step via Starcode, from which collapsed matrix will be used further for binarization.

In this step, we will identify CellTags with similar sequences and collapse similar CellTags to the centroid CellTag. For more information and installation, please refer to starcode software - https://github.com/gui11aume/starcode. Briefly, starcode clusters DNA sequences based on the Levenshtein distances between each pair of sequences, from which we collapse similar CellTag sequences to correct for potential errors occurred during single-cell RNA-sequencing process. Default maximum distance from starcode was used to cluster the CellTags.

I. Prepare for the data to be collapsed

First, we will prepare the data to the format that could be accepted by starcode. This function accepts two inputs including the CellTag object with raw count matrix generated and a path to where to save the output text file. The output will be a text file with each line containing one sequence to collapse with others. In this function, we concatenate the CellTag with cell barcode and use the combined sequences as input to execute Starcode. The file to be used for Starcode will be stored under the provided directory.

```
# Generating the collapsing file
bam.test.obj <- CellTagDataForCollapsing(celltag.obj = bam.test.obj, output.file = "~/Desktop/collapsing.txt")
```

II. Run Starcode to cluster the CellTag

Following the instruction for Starcode, we will run the following command to generate the result from starcode.

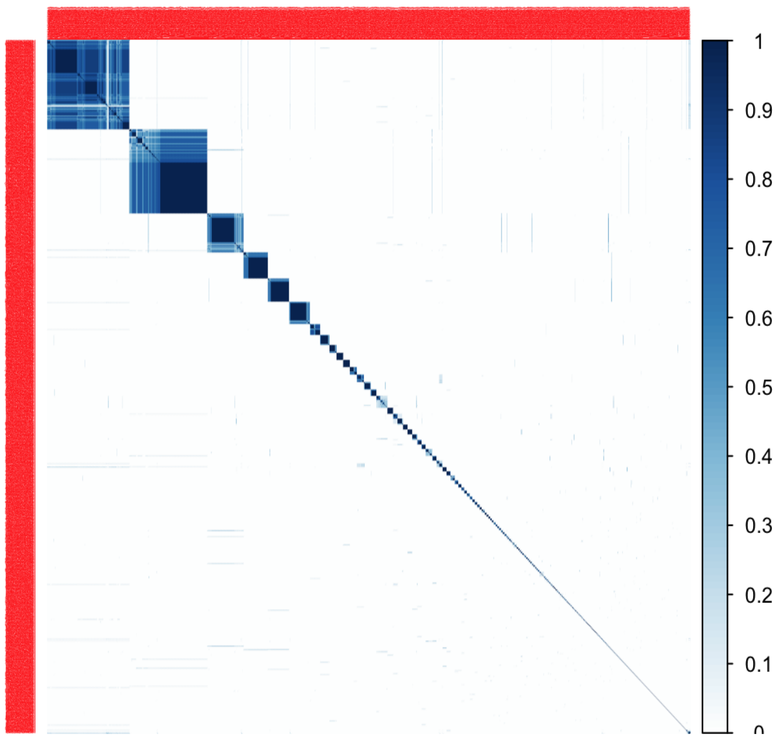
```
./starcode -s --print-clusters ~/Desktop/collapsing.txt > ~/Desktop/collapsing_result.txt
```

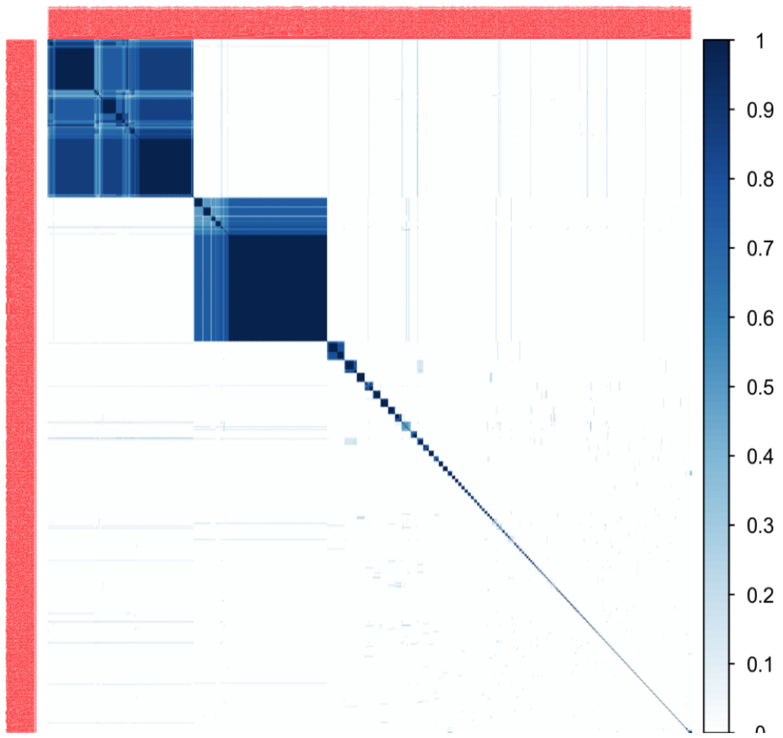
III. Extract information from Starcode result and collapse similar CellTags

With the collapsed results, we will regenerate the CellTag x Cell Barcode matrix. The collpased matrix will be stored in a slot - "collapsed.count" - in the CellTag object. This function takes two inputs including the CellTag Object to modify and the path to th result file from collapsing.

```
# Recount and generate collapsed matrix
bam.test.obj <- CellTagDataPostCollapsing(celltag.obj = bam.test.obj, collapsed.rslt.file = "~/Desktop/collapsing_rslt
# Check the dimension of this collapsed count.
head(bam.test.obj@collapsed.count)
```

Below is an example Jaccard Analysis result with Error Correction using Starcode collapsing (top - without collapsing, bottom - with collapsing):





2. Binarize the single-cell CellTag UMI count matrix

Here we would like to binarize the count matrix to contain 0 or 1, where 0 indicates no such CellTag found in a single cell and 1 suggests the existence of such CellTag. The suggested cutoff that marks existence or absence is at least 2 counts per CellTag per Cell. For details regarding to the cutoff choice, please refer to the paper - https://www.nature.com/articles/s41586-018-0744-4. The binary matrix will be stored in a slot - 'binary.mtx' - as a *dgCMatrix*. **Note: If collapsing was performed, binarization will based on the collapsed count matrix. Otherwise, it will be based on the raw count matrix**

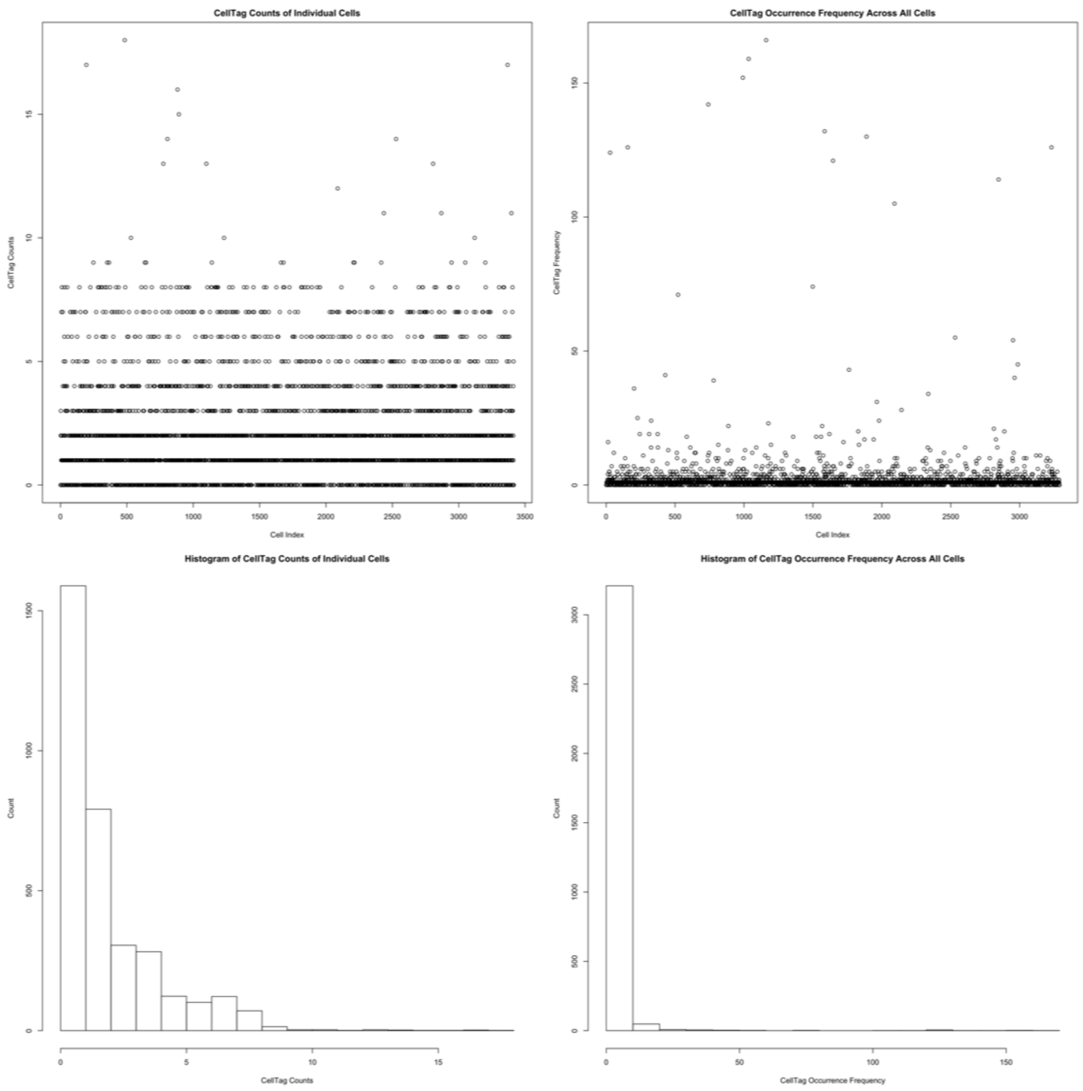
```
# Calling binarization
bam.test.obj <- SingleCellDataBinatization(bam.test.obj, 2)
```

3. Metric plots to facilitate for additional filtering

We then generate scatter plots for the number of total celltag counts in each cell and the number each tag across all cells. These plots could help us further in filtering and cleaning the data.

```
MetricPlots(bam.test.obj)
```

Below is an example plot that you could obtain from this object



4. Apply the whitelisted CellTags generated from assessment

Based on the whitelist generated earlier, we filter the UMI count matrix to contain only whitelisted CelTags for the current version under processing. The function takes in two inputs including the CellTag object with binarization performed and the path to the whitelist csv file. The whitelist result will be saved in a slot - "whitelisted.count".

```
# Read the RDS file and get the object
dt.mtx.whitelist.path <- system.file("extdata", "v1_whitelist.csv", package = "CellTagR")
bam.test.obj <- SingleCellDataWhitelist(bam.test.obj, dt.mtx.whitelist.path)
```

5. Check metric plots after whitelist filtering

Recheck the metric similar to Step 3

```
MetricPlots(bam.test.obj)
```

6. Additional filtering

Filter out cells with more than 20 CellTags

```
bam.test.obj <- MetricBasedFiltering(bam.test.obj, 20, comparison = "less")
```

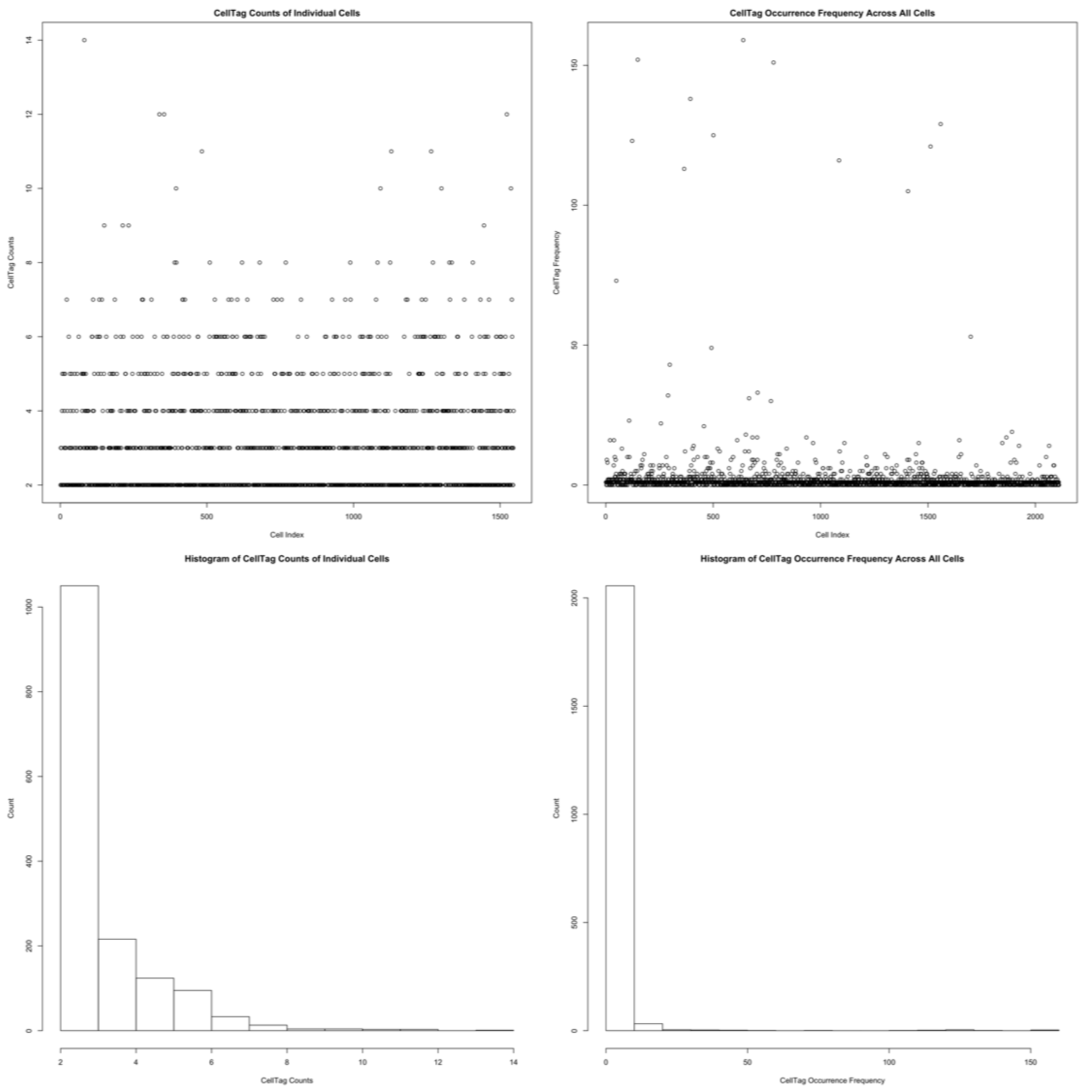
Filter out cells with less than 2 CellTags

```
bam.test.obj <- MetricBasedFiltering(bam.test.obj, 2, comparison = "greater")
```

7. Last check of metric plots

```
MetricPlots(bam.test.obj)
```

Example plot of last check!



If it looks good, proceed to the following steps to call the clones.

8. Clone Calling

I. Jaccard Analysis

This calculates pairwise Jaccard similarities among cells using the filtered CellTag UMI count matrix. This function takes the CellTag object with metric filtering carried out. This will generate a Jaccard similarity matrix, which is saved as a part of the object in a slot - "jaccard.mtx". It also plots a correlation heatmap with cells ordered by hierarchical clustering.

```
bam.test.obj <- JaccardAnalysis(bam.test.obj)
```

II. Clone Calling

Based on the Jaccard similarity matrix, we can call clones of cells. A clone will be selected if the correlations inside of the clones passes the cutoff given (here, 0.7 is used. It can be changed based on the heatmap/correlation matrix generated above). Using this part, a list containing the clonal identities of all cells and the count information for each clone will be stored in the object in slots - "clone.composition" and "clone.size.info".

Clonal Identity Table `clone.composition`

clone.id	cell.barcode
Clonal ID	Cell BC

Count Table `clone.size.info`

Clone.ID	Frequency
Clonal ID	The cell number in the clone

```
# Call clones
bam.test.obj <- CloneCalling(celltag.obj = bam.test.obj, correlation.cutoff=0.7)
# Check them out!!
bam.test.obj@clone.composition[["v1"]]
bam.test.obj@clone.size.info[["v1"]]
```

Network Construction And Visualization

Having all three CellTag version analyzed and stored in one CellTag object, we will construct network of each individual clone connecting to its descendents. As well as connections between clones, cells in each clone will be visualized on the network as leaf nodes. In the network, each center node denotes a clone. Connections between those nodes suggest a "parent-child" relationship between the clones. Each leaf node denotes a cell. Connections between leaf nodes and center nodes suggest a "belonging" relationship. Additionally, we allow users to further construct a stacked bar chart to facilitate further analysis of the dynamics of different timepoints.

Note: As mentioned previously, the pipeline above could be time consuming majorly due to the bam file reading step. And it is required to execute through for three times to obtain all versions (V1, V2, V3) of CellTag information. Here, we provide a demo object in .Rds format that is generated with all three versions processed. The R notebook used to process all three versions are included in the Examples folder.

1. Read in the object

```
# Read the RDS file and get the object
dt.mtx.path <- system.file("extdata", "bam_v123_obj.Rds", package = "CellTagR")
bam.test.obj <- readRDS(dt.mtx.path)
```

2. Calculate the link list

Here, we are converting the CellTag Matrix into a form of link list, which will be further used to construct the linkages in the network

```
bam.test.obj <- convertCellTagMatrix2LinkList(bam.test.obj)
```

3. Get nodes from the link list

```
bam.test.obj <- getNodesfromLinkList(bam.test.obj)
```

4. Add additional information

For each leaf node (each cell), other information, such as cluster/cell types, can be available via other analysis. In this step, we will add these information into each node such that these information can be visualized on the network as well. In this scenario, for demo purposes, we used a simulation data frame to serve as a mock cluster information for each node.

```
# Simulate some additional data
additional_data <- data.frame(sample(1:10, size = length(rownames(bam.test.obj@celltag.aggr.final)), replace = TRUE),
colnames(additional_data) <- "Cluster"
# Add the data to the object
bam.test.obj <- addData2Nodes(bam.test.obj, additional_data)
```

5. Network visualization and plot

Here, we will visualize the network!

```
# Network Visualization
bam.test.obj <- drawSubnet(tag = "CellTagV1_2", overlay = "Cluster", celltag.obj = bam.test.obj)
bam.test.obj@network
```

Additionally, the network can be saved to a html file, allowing better visualization and overview.

```
saveNetwork(bam.test.obj@network, "~/Desktop/presentation/Demo/hf1.d15.network.construction.html")
```

6. Stack bar chart generation

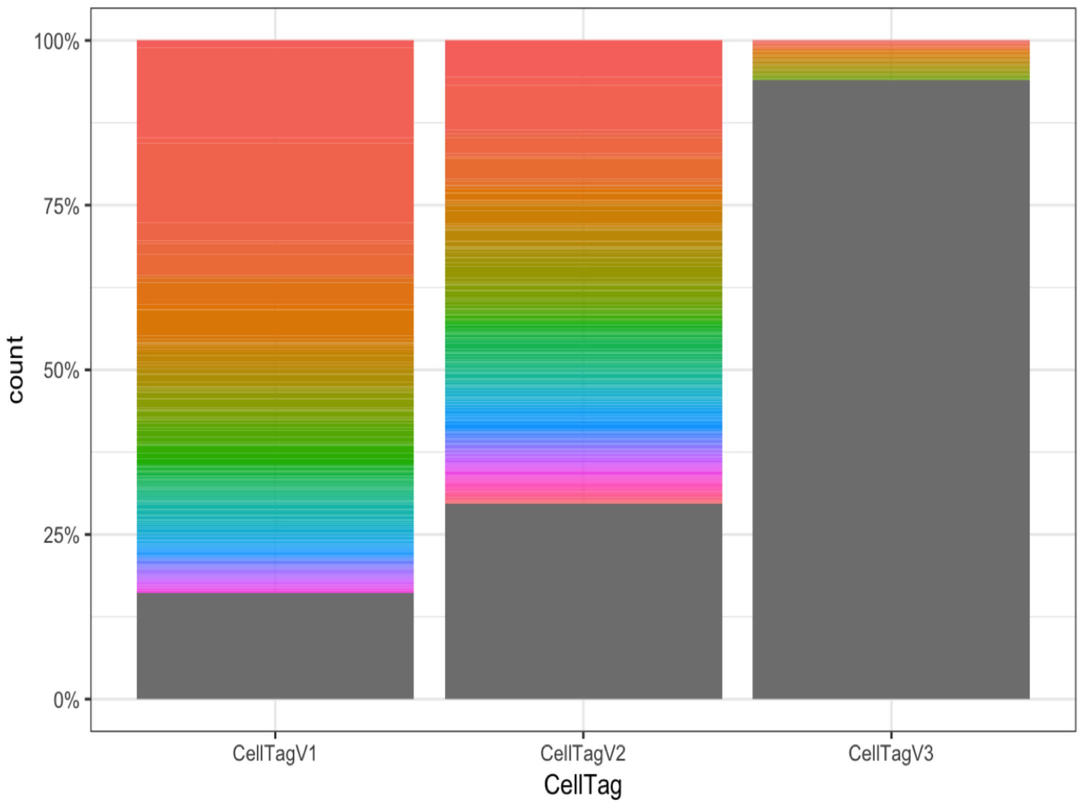
An important aspect of using CellTag is to analyze the clonal dynamics of a population of cells. Here, we provide a stack bar chart option to look at and potentially provide some insights.

```
# Get the data for ploting
bar.data <- bam.test.obj@celltag.aggr.final
bar.data$Cell.BC <- rownames(bar.data)

bar.data <- gather(bar.data, key = "CellTag", value = "Clone", 1:3, na.rm = FALSE)

# Using ggplot to plot
ggplot(data = bar.data) +
  geom_bar(mapping = aes(x = CellTag, fill = factor(Clone)), position = "fill", show.legend = FALSE) +
  scale_y_continuous(labels = scales::percent_format()) +
  theme_bw()
```

Below is a sample bar chart!


100%
75%
50%
25%
0%
count
CellTagV1
CellTagV2
CellTagV3
CellTag