Generating Matrix

---

1. Software Environment Dependencies

A) Python: Version 3.8 or above, with the following modules installed: cv2 (version 4.0 or later), matplotlib, seaborn, pandas, tifffile, Cellpose, Pyyaml, scikit-image.

B) STAR: Version 2.6.1d or later.

C) Perl: threads and threads::shared modules.

D) Requires the following R packages: seurat, dplyr, tibble, ggplot2, broom, purrr, cowplot, cluster, ggpubr, plotly, htmlwidgets, kableextra, htmltools, shiny, knitr, rmarkdown, optparse, getopt.

E) Pandoc.

Note: To set up the environment, you could manually install all the tools, or simply follow the guide in BSTMatrix_environment.rst, or use the conda environment file *environment.yaml*. Please see BSTMatrix_Introduction.docx file for details.

2. Input Data

A) Sequencing data: Paired-end sequencing *.fastq* file.

B) Reference genome: Genome sequence *.fasta* file, and the associated *.gtf* file (with exon in the 3rd column) or *.gff* file (with gene and exon in the 3rd column).

C) features.tsv file: generated by using the *.gtf* file with the command below.

`perl ./tools/features_generate.pl -i xxx.gtf -o features.tsv`.

D) STAR genome index files: generated by using the genome sequence file and the *.gtf* file as bellowed.

`STAR --runThreadN 8 --runMode genomeGenerate --genomeDir star/ -genomeFastaFiles genome.fa --sjdbGTFfile gene.gtf`.

E) Fluorescence decoding files and H&E images.

F) Fluorescence image (optional but required for cell segmentation analysis).

Configuration File (config.txt)

```
### Input sequencing data. Supports .gz format.
FQ1     /path/to/read_1.fq.gz
FQ2     /path/to/read_2.fq.gz

### Fluorescence decoding file.
```

```
FLU     /path/to/flu_info.txt           # Decoding file path.

### AllheStat.py setting.
HE      /path/to/HE.tif                 # H&E image path.

# INSIDE  0                   # Whether to recognise the blank region in the image. 0 is no, and 1 is yes.

# GRAY    200              # Gray scale setting for the H&E image analysis. The default is automatic setting.

### Cell segmentation analysis.
## If this analysis is chosen, the fluorescence image path, colour channel, and
path of the high-resolution H&E image should be provided.
CellSplit       True          # Whether to perform cell segmentation analysis. True is yes, otherwise
the analysis is disabled.
Fluorescence    /path/to/fluorescence.tiff              # Fluorescence image path.
fluorescence_channel       0            # Fluorescence image colour channel, default is 0.
# FLGRAY      15                        # Gray scale setting for the fluorescent image analysis.
Default is automatic setting.

# cells_npy    /path/to/cells/npyfile       # Pre-existing cell segmentation result file .npy. If
provided, it will be used for the sell segmentation analysis.

# YAML    /path/to/cell_split/parameter/file      # File in .yaml format with parameters for
cell segmentation analysis. This is optional.

## Reference genome STAR setting
GenomeVer    xxx              # Genome version information. This will be written in the output report.
INDEX   /path/to/STAR/index/dir/          # STAR reference genome index file.
GFF     /path/to/ref/gene/gff3/file       # Reference genome annotation file. File in .gtf format is
also acceptable.


## Reference genome features.tsv file
FEATURE     /path/to/features.tsv

## Output
OUTDIR  /path/to/result/dir/              # Output path.
PREFIX  outfile-prefix                    # Output file prefix.

### Program Parameters
## fastq2BcUmi
BCType          V2                        # Barcode version type (usually V2 version).
BCThreads       8                         # Number of threads.

## Umi2Gene
Sjdboverhang    100                       # Value of -sjdboverhang parameter used during STAR
                                            indexing, default is 100.
STARThreads     8                         # Number of threads used in read alignment in STAR.

## Environment setting. If not provided, the system default path will be used.
Please add a “#” at the beginning of the lines if not set.
```

```
PYTHON   /path/to/python/dir/           # Path to Python.
Rscript    /path/to/Rscript/dir/        # Path to R.

Note: For plant tissue, the fluorescence image is not required for the cell
segmentation analysis. So CellSplit should be set to “True” to enable the analysis,
and the H&E image should be provided. In this case, please comment out the lines of
“Fluorescence” and “fluorescence_channl” by adding a “#” character at the beginning of
the lines.
```

1.4 Running guide

1.4.1 Process steps

The process consists of 8 steps, as outlined below:

Step 1: Run `fastq2BcUmi` to identify barcodes and UMIs in fastq data.
Step 2: Run `LinkBcChip` to decode the chip locations of each barcode.
Step 3: Run `Umi2Gene` to align reads to the reference genome and obtain gene information for each UMI.
Step 4: Run `MatrixMake` to generate the gene expression matrix.
Step 5: Run `AllheStat` to process H&E images.
Step 6: Run `cluster.R` for cluster analysis.
Step 7: Run `CellSplit` to perform cell segmentation analysis.
Step 8: Run `WebReport` to generate a web based report.

1.4.2 Command options

- `-c config.txt` Data configuration file.
- `-s` selection analysis steps to perform: set to 0 to run all eight steps. Or choose specific steps separated by commas.

  Note:

  1. When selecting 0 and performing cell segmentation analysis, the CellSplit parameter in the configuration file should be set to True.
  2. For plant tissue, the fluorescence image is not required for the cell segmentation analysis. So CellSplit should be set to “True” to enable the analysis, and the H&E image should be provided.

1.4.3 Example Command Lines

```
./BSTMatrix -c config.txt -s 0
./BSTMatrix -c config.txt -s 1,2,3,4,5,6,7,8
./BSTMatrix -c config.txt -s 1,3
```

1.5 Description of Result Files

The directory structure and contents of the result files are as follows:

```
outdir/

├── 01.fastq2BcUmi                         Step 1:  Directory for barcode and UMI detection from
                                                     fastq file
│    ├── xxx.bc_dist                               Barcode detection and stats
│    ├── xxx.bc_stat                               Barcode detection and stats
│    ├── xxx.bc_umi_read.tsv                       Barcode type, UMI and read number statistics
│    ├── xxx.bc_umi_read.tsv.id                    Barcode type, and their UMI and read ID
│    ├── xxx.filter                                Reads with fractional barcode
│    ├── xxx.full_stat                             Read and UMI number for barcode types
│    ├── xxx.id_map                                File containing ID mappings
│    ├── xxx.qual.stat                             Read statistics
│    ├── xxx.select_id                             ID of reads with an intact barcode and UMI
│    ├── xxx.stat                                  Barcode detection stats
│    ├── xxx.umi                                   Barcode types and UMIs for each read
│    └── xxx.umi_cor.info                          UMI correction information

├── 02.LinkBcChip                          Step 2:  Directory for barcode location
│    ├── xxx.barcode_pos.tsv                       Barcode location on the chip
│    ├── xxx.barcode.tsv                           Barcode type for the each chip
│    ├── xxx.flu.stat                              Barcode decoding statistics
│    ├── xxx.info                                  Barcode spatial information
│    └── xxx.null                                  Unrecognized chip position information

├── 03.Umi2Gene                            Step 3: Directory for gene expression information
│    ├── xxxAligned.sortedByCoord.out.bam          STAR alignment bam file
│    ├── xxx.cut0.fq                               Read2 sequences used in alignment
│    ├── xxxLog.final.out                          STAR alignment summary
│    ├── xxxLog.out                                STAR log file
│    ├── xxxLog.progress.out                       STAR progress log file
│    ├── xxx.map2gene                              Information of reads mapped to genes
│    ├── xxxSJ.out.tab                             Splice junction info from STAR
│    ├── xxx_STARtmp                               STAR temporary files
│    ├── xxx.stat                                  Preliminary alignment statistics
│    ├── xxx.total.stat                            Alignment summary statistics
│    └── xxx.umi_gene.tsv                          UMIs and genes for each barcode

├── 04.MatrixMake                          Step 4: Directory for expression matrix
│    ├── xxx.matrix.tsv                            Gene expression matrix
│    ├── xxx.matrix.tsv.filt                       Filtered matrix file
│    ├── xxx.select.bc_umi_read.tsv                UMIs and read number for each barcode
│    ├── xxx.select.umi_gene.tsv                   UMI and gene info for each barcode
│    ├── xxx.select.umi_gene.tsv.filter            Filtered barcode and their gene info
│    ├── xxx.sequencing_saturation.stat            Sequencing saturation analysis
│    └── xxx.sequencing_saturation.png             Sequencing saturation graph

├── 05.AllheStat                           Step 5: Directory for tissue expression analysis results
```

```
│   ├── allhe                                Directory for tissue region information
│   │   ├── he_roi_small.png                   PNG image of identified H&E tissue regions
│   │   ├── he_roi.tif                         TIFF image of identified H&E tissue regions
│   │   ├── roi_heAuto.json                    JSON file with tissue region information
│   │   └── stat.txt                           Tissue region statistics
│   ├── all_level_stat.txt                   Statistics for different resolution levels
│   ├── BSTViewer_project                    BSTViewer software input data directory
│   │   ├── cell_split                         Directory for cell segmentation data
│   │   ├── cluster                            Empty directory
│   │   ├── he_roi_small.png                   PNG image of identified H&E tissue regions
│   │   ├── he.tif                             H&E image file
│   │   ├── imgs                               Empty directory
│   │   ├── level_matrix                       Directory for expression matrix at different
                                                 resolution levels
│   │   ├── project_setting.json               BSTViewer project JSON file
│   │   ├── roi_groups                         Directory for tissue and H&E image JSON files
│   │   └── subdata                            Directory for expression matrix at different
                                                 resolution levels in tissue regions
│   ├── heAuto_level_matrix                    Directory for expression matrix at different
                                                 resolution levels in tissue regions
│   │   └── subdata                            Directory for expression matrix at different
                                                 resolution levels in tissue regions
│   ├── level_matrix                         Directory for expression matrix at different
                                               resolution levels for chips
│   │   ├── level_1                            Directory for level 1 expression matrix
│   │   ├── level_13                           Directory for level 13 expression matrix
│   │   ├── level_2                            Directory for level 2 expression matrix
│   │   ├── level_3                            Directory for level 3 expression matrix
│   │   ├── level_4                            Directory for level 4 expression matrix
│   │   ├── level_5                            Directory for level 5 expression matrix
│   │   ├── level_6                            Directory for level 6 expression matrix
│   │   └── level_7                            Directory for level 7 expression matrix
│   ├── stat.txt                             Tissue region analysis statistics
│   └── umi_plot                             Directory for UMI plot results
│       ├── all_umi_count_small.png            PNG image of UMI count in chip regions
│       ├── all_umi_count.tif                  TIFF image of UMI count in chip regions
│       ├── roi_umi_count_small.png            PNG image of UMI count in tissue regions
│       ├── roi_umi_count.tif                  TIFF image of UMI count in tissue regions
│       ├── roi_umi_count_white_small.png      PNG image with white background for UMI
                                                 count in tissue regions
│       └── roi_umi_count_white.tif            TIFF image with white background for UMI
                                                 count in tissue regions
├── 06.Cluster                          Step 6: Directory for clustering analysis results
│   ├── L13                                  Directory for level 13 clustering results
│   │   ├── cluster.csv                        Cluster results
│   │   ├── L13_cluster_files                  Directory for cluster HTML appendix files
│   │   ├── L13_cluster.html                   HTML image of clustering results
│   │   ├── L13_cluster.pdf                    PDF image of clustering results
│   │   ├── L13_cluster.png                    PNG image of clustering results
│   │   ├── L13_umap_clstr.pdf                 Merged PDF image of UMAP results
│   │   ├── L13_umap_clstr.png                 Merged PNG image of UMAP results
│   │   ├── L13_umap_files                     Directory for UMAP HTML appendix files
│   │   ├── L13_umap.html                      HTML image of UMAP results
│   │   ├── L13_umap.pdf                       PDF image of UMAP results
│   │   └── L13_umap.png                          PNG image of UMAP results
│   ├── L3                                      Directory for level 3 clustering
│   │   ├── cluster.csv                           Cluster result file
│   │   ├── L3_cluster_files                      Directory for cluster HTML appendix files
│   │   ├── L3_cluster.html                       HTML image of clustering results
│   │   ├── L3_cluster.pdf                        PDF image of clustering results
│   │   ├── L3_cluster.png                        PNG image of clustering results
│   │   ├── L3_umap_clstr.pdf                     Merged PDF image of UMAP results
│   │   ├── L3_umap_clstr.png                     Merged PNG image of UMAP results
│   │   ├── L3_umap_files                         Directory for UMAP HTML appendix files
│   │   ├── L3_umap.html                          HTML image of UMAP results
│   │   ├── L3_umap.pdf                           PDF image of UMAP results
│   │   └── L3_umap.png                           PNG image of UMAP results
... ...
│   └── L7                                      Directory for level 7 clustering
│       ├── cluster.csv                           Cluster result file
│       ├── L7_cluster_files                      Directory for cluster HTML appendix files
│       ├── L7_cluster.html                       HTML image of clustering results
│       ├── L7_cluster.pdf                        PDF image of clustering results
│       ├── L7_cluster.png                        PNG image of clustering results
│       ├── L7_umap_clstr.pdf                     Merged PDF image of UMAP results
│       ├── L7_umap_clstr.png                     Merged PNG image of UMAP results
│       ├── L7_umap_files                         Directory for UMAP HTML appendix files
│       ├── L7_umap.html                          HTML image of UMAP results
│       ├── L7_umap.pdf                           PDF image of UMAP results
│       └── L7_umap.png                           PNG image of UMAP results

├── 07.CellSplit                         Step 7: Directory for cell segmentation analysis
│   ├── cell_split_result                       Directory for cell segmentation results
│   │   ├── 0_0.npy                               Local cell segmentation results
│   │   ├── 0_0_ori.tif                           Local fluorescent image
│   │   ├── 0_0.tif                               Local fluorescent image after cell recognition
... ...
│   │   ├── 9500_9500.npy                         Local cell segmentation results
│   │   ├── 9500_9500_ori.tif                     Local fluorescent image
│   │   ├── 9500_9500.tif                         Local fluorescent image after cell recognition
│   │   ├── all_barcode_num.txt                   Cell barcode IDs
│   │   ├── all_outline.tif                       Fluorescent image with added cell
                                                    boundaries
│   │   ├── cell_color.tif                        Recognized cell image file
│   │   ├── cellConts.json                        Recognized cell JSON file
│   │   ├── cells.npy                             Recognized cell NPY file
│   │   ├── colors.npy                            Cell and color mapping file
│   │   ├── conts.tif                             Cell segmentation tissue boundary
│   │   ├── fluorescence.tif                      Tissue fluorescent image
│   │   ├── nucleus_color.tif                     Recognized cell nucleus image
│   │   ├── nucleusConts.json                     Recognized cell nucleus JSON
│   │   ├── nucleus.npy                           Recognized cell nucleus NPY
│   │   ├── progress.txt                          Progress percentage file
│   │   └── SegtoBarcode.log                      Log file
│   ├── cluster                                 Directory for clustering results
│   │   ├── cell_cluster_color_img.tif            Cell segmentation clustering image without
                                                     legend in TIFF format
```

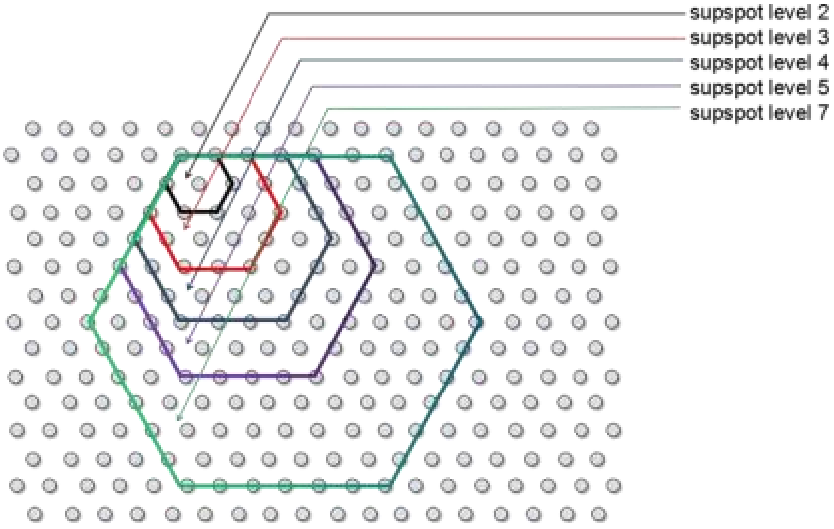
```
│   │   ├── cell_cluster_color_outline_img.tif      Cell segmentation clustering image with
                                                              added cell boundaries in TIFF format
│   │   ├── cell_cluster_with_legend_img.png        Cell clustering image with legend in PNG
                                                         format
│   │   ├── cell_cluster_with_legend_img_small.png  Low-resolution cell clustering image with
                                                         legend in PNG format
│   │   ├── cell_cluster_with_legend_img.tif        Cell clustering TIFF image with legend
│   │   ├── cluster.csv                             Clustering results
│   │   ├── cluster_cells_num.csv                   Cluster cell number count
│   │   ├── clusters_colors.npy                     Cluster category and color mapping results
│   │   ├── colors.npy                              Cell and color mapping results
│   │   ├── legend.tif                              Cluster legend
│   │   ├── marker_gene.csv                         Marker gene information
│   │   ├── object.RDS                              Seurat object from cell segmentation matrix
│   │   ├── UMAP.pdf                                UMAP clustering results in PDF format
│   │   └── UMAP.png                                UMAP clustering results in PNG format
│   ├── images                                    Directory for cell segmentation  related image
│   │   ├── fluorescence_cell_split.png             Fluorescent PNG image cell segmentation
│   │   ├── fluorescence_cell_split_small.png       Low-resolution fluorescent PNG image cell
                                                         segmentation result
│   │   ├── fluorescence_cell_split.tif             Fluorescent TIFF image cell segmentation
                                                         results
│   │   ├── fluorescence.png                        Tissue fluorescent PNG image
│   │   ├── fluorescence_small.png                  Low-resolution tissue fluorescent image in
                                                          PNG format
│   │   ├── fluorescence.tif                        Tissue fluorescent image in TIFF format
│   │   ├── he_cell_split.png                       Tissue H&E staining cell segmentation PNG
                                                           image
│   │   ├── he_cell_split_small.png                 Low-resolution tissue H&E staining cell
                                                           segmentation result in PNG format
│   │   ├── he_cell_split.tif                       Tissue H&E staining cell segmentation result
                                                           in TIFF format
│   │   └── he_hr.tif                               Tissue H&E image in TIFF format
│   └── mtx                                       Directory for cell segmentation matrix results
│       ├── barcodes.tsv.gz                         Cell barcode file
│       ├── cells_center.txt                        Cell center location on chip
│       ├── cells_center.tif                        Cell center image
│       ├── features.tsv.gz                         Cell features file
│       ├── matrix.mtx.gz                           Cell matrix file
│       └── stat.xls                                Cell statistics file
├── 08.WebReport                              Step 8: Directory for web-based report
│   ├── src                                       Directory for web-based report source files
│   ├── xxx.filelist                              List of files used for the web-based report
│   ├── xxx.stat.xls                              Analyses summary
│   ├── xxx.rs_stat.xls                           Analyses summary
│   └── xxx.html                                  Web-based report file
└── xxx                                         Directory for original expression matrix results
    ├── barcode_pos.tsv                           Barcode and corresponding chip position file
    ├── barcode.tsv                               Barcode file
    ├── bc_umi_read.tsv.gz                        UMIs and read counts for each barcode
    ├── features.tsv                              Features file
    ├── matrix.tsv                                Matrix file
    └── umi_gene.tsv.gz                           UMIs and genes corresponding to barcodes
```

Please note that "xxx" represents the specific file or directory names generated during the execution of the pipeline.

2. Multi-level Resolution

After successfully running BSTMatrix, we will receive a spatial expression matrix. By default, the software provides results in eight levels of resolution, namely level 1 to 7 and level 13, in the directory *05.AllheStat*. So how to understand the different levels of resolution?

To better explain this concept, we illustrate the calculation of different levels of resolutions in the below figure. Level 1 represents a single spot level resolution, and level 2 is the combination of the level 1 spot and its surrounding spots. In this case, all the surrounding spots form a perfect hexagon, of which the side length is within the level number of spots. In level 2, the side length of the hexagon is the distance between two spots. Similarly, the side length of the hexagon in level 3 is the distance between three spots. We could calculate resolutions in level 4-13 in the same way. They all use the level 1 spot as the central spot, and the surrounding spots form a perfect hexagon. All the data captured by spots in and on a hexagon are merged for analysis. This merged spot is called “supSpot”. As a result, level 1 presents the highest resolution level, which is within a hexagon with side length of 5μm, while level 13 presents the lowest resolution level, which is in within a hexagon with side length of 100μm.



supSpot and spot

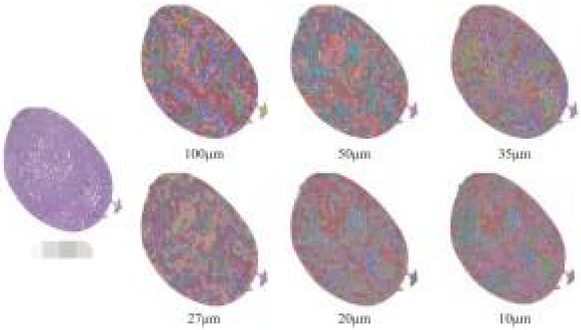
For subsequent analysis in spatial transcriptome data, it is not always the best to use the highest resolution. Instead, choosing an appropriate resolution, which is close to the actual size of sample cells, can yield better understanding in single-cell level transcriptome, which is especially useful in the purpose of restoring the cellular structure in tissue. For assisting in selection of a suitable resolution, we provide several guidelines here:

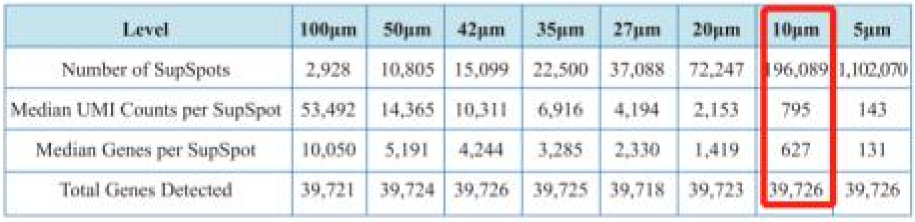
A) Measure and determine the size of the majority of cells in the tissue, and then choose an appropriate resolution which is close to the actual cell size.

B) The resolution level, at which the visualization of clustering results is the most similar to the staining structure of the sections, is considered the optimal resolution.

C) Under the selected resolution, the median number of gene counts for a supSpot exceeds 1000. The acceptable minimum value is 500.

In the example below, Level 2 (10µm resolution) is an appropriate resolution level for subsequent spatial transcriptomic analysis.





Level	100µm	50µm	42µm	35µm	27µm	20µm	10µm	5µm
Number of SupSpots	2,928	10,805	15,099	22,500	37,088	72,247	196,089	1,102,070
Median UMI Counts per SupSpot	53,492	14,365	10,311	6,916	4,194	2,153	795	143
Median Genes per SupSpot	10,050	5,191	4,244	3,285	2,330	1,419	627	131
Total Genes Detected	39,721	39,724	39,726	39,725	39,718	39,723	39,726	39,726

3. Subsequent Analysis

3.1 Seurat Object Creation

The BMKMANU S1000 spatial transcriptomic matrix is different from the 10x Visum spatial matrix, and so we cannot directly use the Seurat spatial plotting functions to further plotting our S1000 matrix. To address this, we could convert our data into the format of 10x spatial matrix and then create a Seurat object. With this converted object, we can perform the analysis using built-in Seurat spatial plotting functions such as SpatialFeaturePlot.

```
Script:
CreateBmkObject.R

Usage:
source("CreateBmkObject.R")
object <- CreateS1000Object(
            matrix_path="BSTViewer_project/subdata/L13_heAuto", # Directory of the
matrix file
     png_path="BSTViewer_project/he_roi_small.png", # Path to the PNG image file
     spot_radius = 1, # Radius of the spots. Can be left unspecified, as it will be
automatically calculated
     min.cells = 5, # Minimum number of cells that a gene is expressed in. Default
 Value is 5, meaning genes expressed in less than five cells will be filtered out.
     min.features = 100 # Minimum number of genes that a cell expresses. Default
value is 100, meaning cells expressing more than 100 genes will be kept.
)
```

Please note that the script "CreateBmkObject.R" is attached as a separate file.

3.2 Automated Annotation using Human/Mouse SingleR

Script: SingleR.r

Parameters:

-i INDIR, --indir=INDIR	Input directory, which is the directory containing the spatial matrix.
-o OUTDIR, --outdir=OUTDIR	Output directory.
--he_png=HE_PNG	Path to the H&E file in PNG format.
--res=RES	Resolution, default is 0.5.
--MinCell=MINCELL	MinCell, default is 5.
--MinFeatures=MINFEATURES	MinFeatures, default is 100.
--point_size=POINT_SIZE	Point size, default is 1.
--ref=REF	Reference dataset for SingleR. Default is 1. 1 for BlueprintEncodeData. 2 for DatabaseImmuneCellExpressionData. 3 for NovershternHematopoieticData. 4 for MonacoImmuneData. 5 for HumanPrimaryCellAtlasData. 6 for ImmGenData. 7 for MouseRNAseqData.
-h, --help	Show the help message and exit.

Output:

```
singleR/
├── cluster.csv                      Clustering results and SingleR annotation
├── singleR_cluster.pdf              SingleR annotation results spatial plot (in PDF format)
├── singleR_cluster.png              SingleR annotation results spatial plot (in PNG format)
├── singleR_cluster_umap.p*          SingleR spatial and UMAP plot (in PDF/PNG format)
├── singleR_UMAP.p*                  SingleR UMAP plot (in PDF/PNG format)
└── UMAP.p*                          Clustering results UMAP plot (in PDF/PNG format)
```

Please note that the script mentioned is attached as a separate file.

3.3 Extraction of Barcode for Adjacent Boundaries of Different Cell Types in BMKMANU S1000 Spatial Transcriptomics

Cell subpopulations within tissues are not independent, and they can engage in extensive ligand-receptor interactions on their contacting boarders. After annotating cell types in spatial transcriptomic data, we could extract barcodes of reads that are collected on the adjacent boundaries of different cell types, to perform the cellular communication and interaction analysis. The following script demonstrates how to extract barcodes of reads on adjacent boundaries of different cell types.

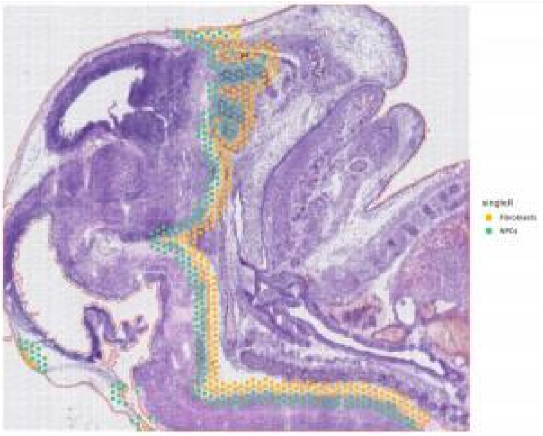
Script: bianjie_script.r

Parameters:

- -i The input directory, e.g., BSTViewer_project/subdata/L13_heAuto.
- -o The output directory.
- --he_png The image path, e.g., BSTViewer_project/he_roi_small.png.
- --point_size The size of plotting points.
- --cluster The cell annotation file with the header “Barcode” and “singleR”, in which the 1st column contains barcode information and the 2nd column contains cell annotation.
- --cell_type1 The first cell type.
- --cell_type2 The second cell type.

Output:

Images of barcodes spatial information for the specified cell adjacent boundaries. An example output image is shown as below.

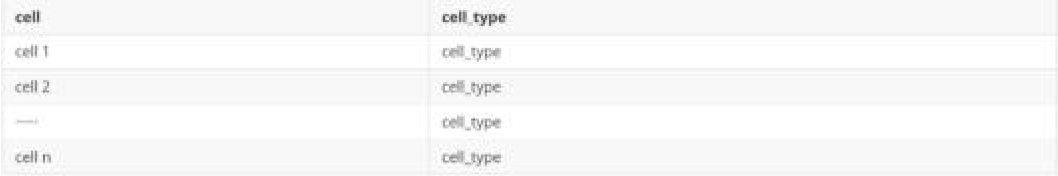


3.4 Cell Communication Analysis

Cell communication analysis is based on the gene expression data and the ligand-receptor database. If the transcriptomic data shows that cell type A and B, as an example, express gene α and β respectively, and if by querying a database α and β are identified as ligand and receptor pairs, we assume that cell A communicates with cell B through the α-β pathway.

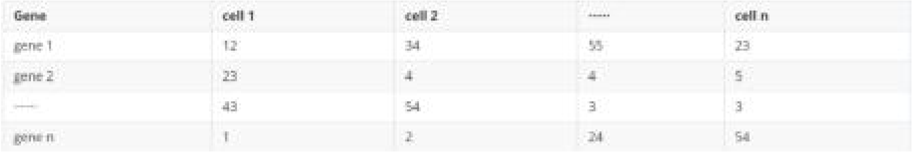
3.4.1 Input Data Format Requirements

A) Cell annotation file (*meta.txt*) with two columns presenting cell IDs and cell types.



cell	cell_type
cell 1	cell_type
cell 2	cell_type
……	cell_type
cell n	cell_type

B) Gene expression matrix (*counts.txt*)



Gene	cell 1	cell 2	……	cell n
gene 1	12	34	55	23
gene 2	23	4	4	5
……	43	54	3	3
gene n	1	2	24	54

3.4.2 Software Installation

Download and Install Software (in the following order)

1. Create a new conda environment:

```
conda create -n cellphonedb2 python=3.7
```

2. Install the required packages:

```
conda install Cython
conda install h5py
conda install scikit-learn
conda install r
conda install rpy2
pip install https://pypi.tuna.tsinghua.edu.cn/simple cellphonedb
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple markupsafe==2.0.1
```

3. Install R packages in R:

```
install.packages('ggplot2')
install.packages('heatmap')
```

Data Conversion and Preprocessing

- Convert data1 to R format:
  - Use the spatial data annotated with RCTD for further analysis.
  - Extract the relevant information and create a meta.txt file:

```
results <- myRCTD@results

results_df <- results$results_df

barcodes = rownames(results_df[results_df$spot_class != "reject" &
puck@nUMI >= 1,])

my_table = puck@coords[barcodes,]

my_table$class = results_df[barcodes,]$first_type

meta = my_table %>% select(class) %>% rownames_to_column(var = 'cell') %>%
dplyr::rename(cell_type = class)

write.table(meta, file = '/output/path/Test/meta.txt', sep = '\t', quote = F,
row.names = T)
```

- Convert data2 to R format:
  - Read the gene expression data in 10x format and create a counts.txt file:

```
expr <- Read10X('/path/BSTViewer_project/subdata/L13_heAuto/',
        cell.column = 1)
object <- CreateSeuratObject(counts = expr, assay = "Spatial") counts
        = object[['Spatial']]@counts %>% as.data.frame() %>%
        rownames_to_column(var = 'Gene')
write.table(counts, file = '/output/path/Test/counts.txt', sep = '\t', quote =
        F, row.names = T)
```

Conversion of Mouse Genes

- CellPhoneDB can only recognize human genes. If you are using mouse genes, you need to convert them to their human homologs.
- Download the human-mouse homologous gene mapping: http://asia.ensembl.org/biomart/martview/b9f8cc0248e4714ba8e0484f0cbe4f02

Reference Steps

Please refer to the following steps for guidance: https://www.jianshu.com/p/922a7f2c21fc

- The downloaded correspondence table should have four columns:
  - Gene_stable_ID | Gene_name | Mouse_gene_stable_ID | Mouse_gene_name

- Replace the correspondence table (human_mouse_gene.txt) in data2 matrix (Shell command):

```
awk -F '\t' 'BEGIN{OFS = '\t'}ARGIND==1{a[$4]=$1}ARGIND==2{if(FNR == 1)
{print}else{$1 = a[$1];print}}' human_mouse_gene.txt counts.txt | sed '/^
/d' | sed '1s/^/Gene\t/' | sed 's/\s/\t/g' > counts_new.txt
```

- List available database versions from the remote repository:

```
cellphonedb database list_remote
```

- View local databases:

```
cellphonedb database list_local
```

- Download the remote database:

```
cellphonedb database download --version <version_spec|latest>
```

- Run CellPhoneDB (Shell command):

```
cellphonedb method statistical_analysis --output-path test_output meta.txt
counts_new.txt
```

- Output files:

  deconvoluted.txt | pvalues.txt | significant_means.txt

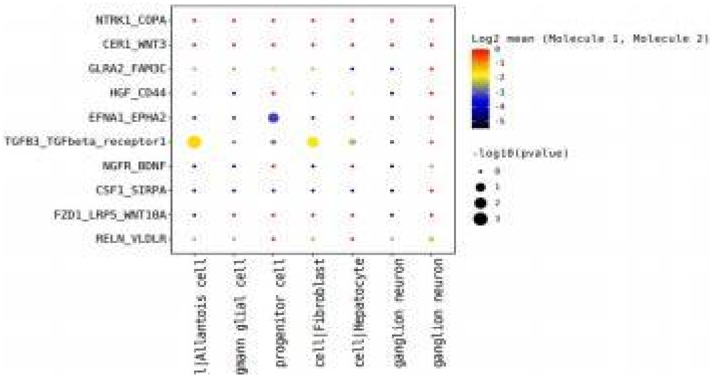
  test.count_network.txt | means.txt | test.interaction_count.txt

- Generate a dot plot (Shell command):

```
cellphonedb plot dot_plot --pvalues-path test_output/pvalues.txt --means-
path test_output/means.txt --output-path test_output --output-name
test.dotplot.pdf
```

- Generate a dot plot for selected cells and their interactions (Shell command):

```
cellphonedb plot dot_plot --pvalues-path test_output/pvalues.txt --means-path
 test_output/means.txt --output-path test_output --output-name
test.dotplot2.pdf --rows test_output/row.txt --columns test_output/col.txt
```



Note: The x-axis represents cell type interactions, the y-axis represents protein interactions. The larger the dots, the smaller the p-values. Different colors present different average expression level.

- Heatmap

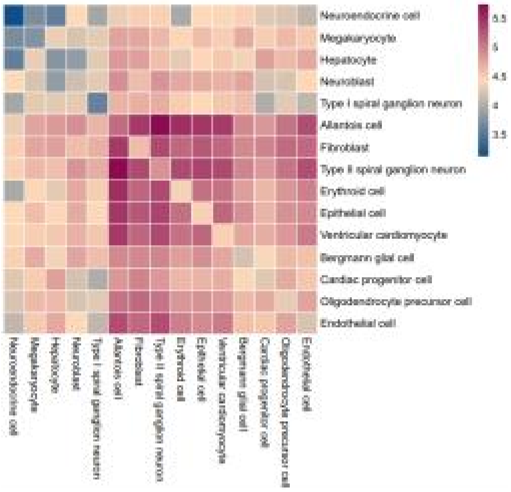
To generate a heatmap using CellPhoneDB, you can use the following command in the shell:

```
cellphonedb plot heatmap_plot --pvalues-path test_output/pvalues.txt --output-path
test_output --pvalue 0.05 --count-name test.heatmap_count.pdf --log-name
test.heatmap_log_count.pdf --count-network-name test.count_network.txt --
interaction-count-name test.interaction_count.txt meta.txt
```

This command will generate the following output files in the specified `test_output` directory:

- `test.heatmap_count.pdf`: Heatmap plot showing the counts of interactions.
- `test.heatmap_log_count.pdf`: Heatmap plot showing the log counts of interactions.
- `test.count_network.txt`: File containing the count network information.
- `test.interaction_count.txt`: File containing the interaction count information.

The heatmap plot visualizes the interactions between different cell types based on the specified p-value threshold (`0.05` in this example). It provides insights into the cell-cell communication patterns.



Note: The x- and y-axes represent cell types, and the color gradient from deep blue to purple-red represents the increasing number of interactions from low to high.

- Cell Type Network Plot ---> R

Here, we will use another cell communication analysis software called CellChat (https://github.com/sqjin/CellChat) for plotting the interaction network.

First, load the required R packages:

```r
library(igraph)
library(tidyr)
library(stats)
library(reshape2)
library(Matrix)
library(grDevices)
library(ggplot2)
```

Next, we will use the plotting function provided by the CellChat to generate the interaction network plot. You can simply copy and paste the following code:

```r
scPalette <- function(n) {
colorSpace <-
c('#E41A1C','#377EB8','#4DAF4A','#984EA3','#F29403','#F781BF','#BC9DCC','#A65628','#54B0E4','#222F75','#1B9E77','#B2DF8A','#E3BE00','#FB9A99','#E7298A','#910241','#00CDD1','#A6CEE3','#CE1261','#5E4FA2','#8CA77B','#00441B','#DEDC00','#B3DE69','#8DD3C7','#999999')
    if (n <= length(colorSpace)) {
colors <- colorSpace[1:n]
    } else {
      colors <- grDevices::colorRampPalette(colorSpace)(n)
    }
    return(colors)
}

netVisual_circle <- function(net, color.use = NULL,title.name = NULL, sources.use = NULL,
targets.use = NULL, id ents.use = NULL, remove.isolate = FALSE, top = 1, weight.scale =
FALSE, vertex.weight = 20, vertex.weight.max = NULL, vertex.size.max = NULL,
vertex.label.cex = 1, vertex.label.color = "black", edge.weight.max = NULL,
edge.width.max = 8, alpha.edge = 0.6, label.edge = FALSE, edge.label.color = 'black',
edge.label.cex = 0.8, edge.curved = 0.2, shape='circle', layout = in_circle(), margin =
0.2, vertex.size = NULL, arrow.width = 1, arrow.size = 0.2){
    if (!is.null(vertex.size)) {
      warning("'vertex.size' is deprecated. Use `vertex.weight`")
    }
    if (is.null(vertex.size.max)) {
      if (length(unique(vertex.weight)) == 1) {
        vertex.size.max <- 5
      } else {
        vertex.size.max <- 15
      }
    }

    options(warn = -1)
    thresh <- stats::quantile(net, probs = 1-top)
    net[net < thresh] <- 0
    if ((!is.null(sources.use)) | (!is.null(targets.use)) | (!is.null(idents.use)) ) {
    if (is.null(rownames(net))) {
      stop("The input weighted matrix should have rownames!")
    }
    cells.level <- rownames(net)
    df.net <- reshape2::melt(net, value.name = "value")
    colnames(df.net)[1:2] <- c("source","target")
    # keep the interactions associated with sources and targets of interest
    if (!is.null(sources.use)){
      if (is.numeric(sources.use)) {
        sources.use <- cells.level[sources.use]
      }
    df.net <- subset(df.net, source %in% sources.use)
    }
    if (!is.null(targets.use)){
      if (is.numeric(targets.use)) {
        targets.use <- cells.level[targets.use]
      }
      df.net <- subset(df.net, target %in% targets.use)
    }
    if (!is.null(idents.use)) {
      if (is.numeric(idents.use)) {
        idents.use <- cells.level[idents.use]
      }
      df.net <- filter(df.net, (source %in% idents.use)  | (target %in% idents.use))
    }
    df.net$source <- factor(df.net$source, levels = cells.level)
    df.net$target <- factor(df.net$target, levels = cells.level)
    df.net$value[is.na(df.net$value)] <- 0
    net <- tapply(df.net[["value"]], list(df.net[["source"]], df.net[["target"]]), sum)
  }
  net[is.na(net)] <- 0


  if (remove.isolate) {
    idx1 <- which(Matrix::rowSums(net) == 0)
    idx2 <- which(Matrix::colSums(net) == 0)
    idx <- intersect(idx1, idx2)
    net <- net[-idx, ]
    net <- net[, -idx]
  }

  g <- graph_from_adjacency_matrix(net, mode = "directed", weighted = T)
  edge.start <- igraph::ends(g, es = igraph::E(g), names = FALSE)
  Coords <- layout_(g,layout)
  if(nrow(coords)!=1){
    coords_scale = scale(coords)
  }else {
    coords_scale <- coords
  }
  if (is.null(color.use)) {
    color.use = scPalette(length(igraph::V(g)))
  }
  if (is.null(vertex.weight.max)) {
    vertex.weight.max <- max(vertex.weight)
  }
  vertex.weight <- vertex.weight/vertex.weight.max*vertex.size.max+5
```

```
loop.angle <- ifelse(coords_scale[igraph::V(g),1] > 0, -atan(coords_scale[igraph::V(g),
2]/coords_scale[igraph::V(g), 1]), pi-atan(coords_scale[igraph::V(g),
2]/coords_scale[igraph::V(g), 1]))

igraph::V(g)$size <- vertex.weight
igraph::V(g)$color <- color.use[igraph::V(g)]
igraph::V(g)$frame.color <- color.use[igraph::V(g)]
igraph::V(g)$label.color <- vertex.label.color
igraph::V(g)$label.cex <- vertex.label.cex

if(label.edge){
  igraph::E(g)$label <- igraph::E(g)$weight
  igraph::E(g)$label <- round(igraph::E(g)$label, digits = 1)
}
if (is.null(edge.weight.max)) {
  edge.weight.max <- max(igraph::E(g)$weight)
}
if (weight.scale == TRUE) {
#E(g)$width<-0.3+edge.width.max/(max(E(g)$weight)-min(E(g)$weight))*(E(g)$weight-
min(E(g)$weight))
  igraph::E(g)$width <- 0.3 + igraph::E(g)$weight/edge.weight.max*edge.width.max
}else {
  igraph::E(g)$width<-0.3 + edge.width.max*igraph::E(g)$weight
}

igraph::E(g)$arrow.width <- arrow.width
igraph::E(g)$arrow.size <- arrow.size
igraph::E(g)$label.color <- edge.label.color
igraph::E(g)$label.cex <- edge.label.cex
igraph::E(g)$color <- grDevices::adjustcolor(igraph::V(g)$color[edge.start[,1]],
alpha.edge)

if(sum(edge.start[,2]==edge.start[,1])!=0){
  igraph::E(g)$loop.angle [which(edge.start[,2]==edge.start[,1])] <- loop.angle
[edge.start[which(edge.start[,2]==edge.star t[,1]),1]]
}
radian.rescale <- function(x, start = 0, direction = 1) {
  c.rotate <- function(x) (x + start) %% (2 * pi) * direction
  c.rotate(scales::rescale(x, c(0, 2 * pi), range(x)))
}

label.locs <- radian.rescale(x=1:length(igraph::V(g)), direction=-1, start=0)
label.dist <- vertex.weight/max(vertex.weight) + 2
plot(g, edge.curved = edge.curved, vertex.shape = shape, layout = coords_scale, margin =
margin, vertex.label.dist = label.dist, vertex.label.degree = label.locs,
vertex.label.family = "Helvetica", edge.label.family = "Helvetica") # "sans"

if (!is.null(title.name)) {
text(0, 1.5, title.name, cex = 1.1)
}
# https://www.andrewheiss.com/blog/2016/12/08/save-base-graphics-as-pseudo-objects-in-r/
# grid.echo()
# gg <-  grid.grab()
gg <- recordPlot()
return(gg)
}
```

Function `scPalette` could help generate a color palette for the network plot.

- Overall Interaction Network Plot

  To create the overall interaction network plot, follow the steps below:

1. Read the analysis data from CellPhoneDB:

```
count_net <- read.delim("/xxx/test_output/test.count_network.txt", check.names =
FALSE)
```

2. Format the data:

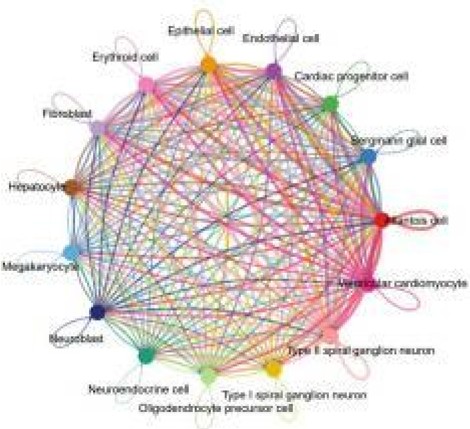
```
count_inter <- count_net
count_inter$count <- count_inter$count / 100
count_inter <- spread(count_inter, TARGET, count)
rownames(count_inter) <- count_inter$SOURCE

count_inter <- count_inter[, -1]
count_inter <- as.matrix(count_inter)
```

3. Plot the network:

```
netVisual_circle(count_inter, weight.scale = T)
```

This will generate the overall interaction network plot as shown below.

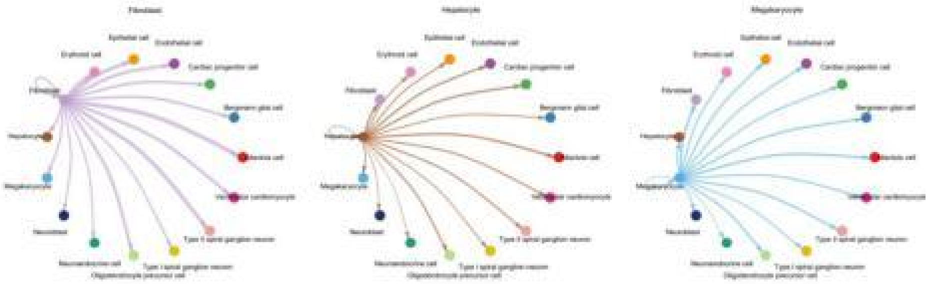


- Cell-Cell Interactions for Each Cell Type

To visualize the interactions of each cell type with other cell types, you can use the following code:

```
par(mfrow = c(1, 3), xpd = TRUE)
for (i in 1:nrow(count_inter)) {
  mat2 <- matrix(0, nrow = nrow(count_inter), ncol = ncol(count_inter), dimnames =
dimnames(count_inter))
  mat2[i, ] <- count_inter[i, ]
  netVisual_circle(mat2,
                   weight.scale = TRUE,
                   edge.weight.max = max(count_inter),
                   title.name = rownames(count_inter)[i],
                   arrow.size = 0.2)
}
```

An example of output of the above code is as shown below.



3.5 Single-Cell Spatial Co-analysis

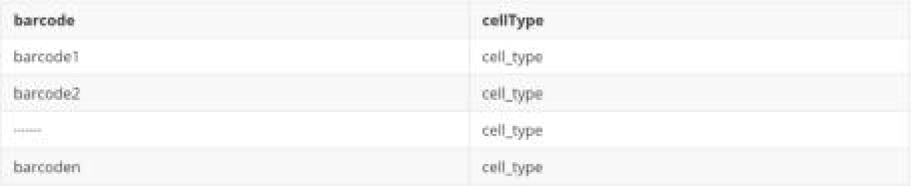
Software: RCTD

RCTD is a software that utilizes supervised learning to define the cell-type specificity of expected cell types in spatial transcriptomics data using annotated scRNA-seq data.

3.5.1 Single-Cell Data Format Requirements

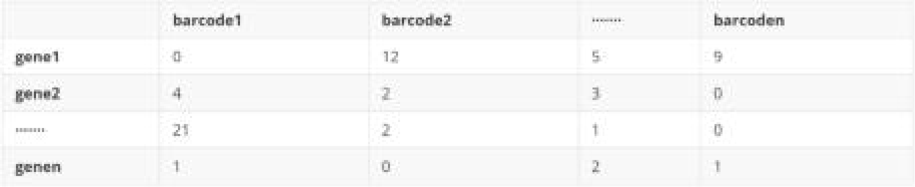
A) Data 1: Cell Annotation Results (cellType)

The cell annotation results should be provided in a format that includes the cell type information (cellType).



barcode	cellType
barcode1	cell_type
barcode2	cell_type
·······	cell_type
barcoden	cell_type

B) Data 2: Gene Expression Matrix (sc_counts)

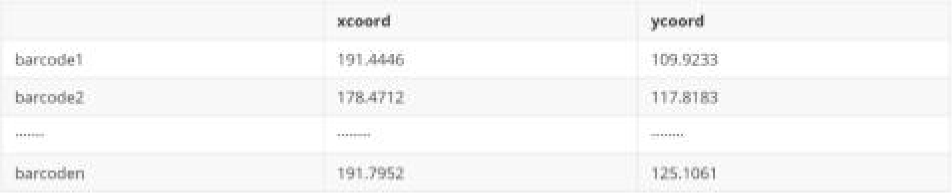


	barcode1	barcode2	·······	barcoden
gene1	0	12	5	9
gene2	4	2	3	0
·······	21	2	1	0
genen	1	0	2	1

3.5.2 Spatial Data Format Requirements

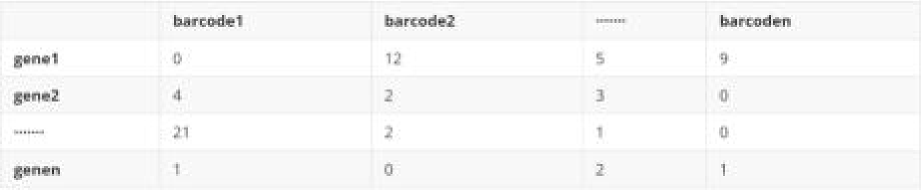
A) Data 1: Spatial Spot Location Information (coords)

The spatial spot location information (coords) should be provided in the specified format.



	xcoord	ycoord
barcode1	191.4446	109.9233
barcode2	178.4712	117.8183
·······	·······	·······
barcoden	191.7952	125.1061

B) Data 2: Gene Expression Matrix (sp_counts)



	barcode1	**barcode2**	·······	**barcoden**
gene1	0	12	5	9
gene2	4	2	3	0
·······	21	2	1	0
genen	1	0	2	1

Installation of spacexr Package

To install the spacexr package, you can use the following code:

```
install.packages("devtools")
devtools::install_github("dmcable/spacexr", build_vignettes = FALSE)
```

Loading R Packages

Before starting the analysis, make sure to load the required R packages:

```
library(Seurat)
library(tidyverse)
library(Matrix)
library(spacexr)
library(ggplot2)
library(ggpubr)
library(gridExtra)
library(reshape2)
library(readr)
library(Seurat)
library(config)
library(ggpubr)
library(gridExtra)
library(reshape2)
library(png)
library(patchwork)
library(SingleR)
library(celldex)
```

Matrix Conversion

To convert a matrix, you can use the following code:

```
Rcpp::sourceCpp(code = '
#include <Rcpp.h>
using namespace Rcpp;
// [[Rcpp::export]]
IntegerMatrix asMatrix(NumericVector rp,
                       NumericVector cp,
                       NumericVector z,
                       int nrows,
                       int ncols){
  int k = z.size();
  IntegerMatrix mat(nrows, ncols);
  for (int i = 0; i < k; i++){
      mat(rp[i],cp[i]) = z[i];
  }
  return mat;
}
')
as_matrix <- function(mat){
  row_pos <- mat@i
  col_pos <- findInterval(seq(mat@x)-1, mat@p[-1])
  tmp <- asMatrix(rp = row_pos, cp = col_pos, z = mat@x,
                       nrows = mat@Dim[1], ncols = mat@Dim[2])
    row.names(tmp) <- mat@Dimnames[[1]]
    colnames(tmp) <- mat@Dimnames[[2]]
    return(tmp)
  }
```

Loading Single-Cell Data

To read the single-cell data, you can use the following code:

```
expr <- Read10X('/xxx/04.QC/filtered_feature_bc_matrix/', cell.column = 1)
sc_data <- CreateSeuratObject(counts = expr, assay = "RNA", min.cells = 5,
min.features = 100)
#counts & nUMI
sc_counts <- as_matrix(sc_data[['RNA']]@counts)
sc_nUMI <- colSums(sc_counts)
```

Please note that you should replace `/xxx/04.QC/filtered_feature_bc_matrix/` with the actual path to your single-cell data.

Annotation

Method 1: SingleR Annotation

```
load('/share/nas1/guochao/database/celldex/MouseRNAseqData.Rdata')
mouseRNA <- ref
sce_for_SingleR <- GetAssayData(sc_data, slot = "data")
pred.mouseRNA <- SingleR(test = sce_for_SingleR, ref = mouseRNA, labels =
mouseRNA$label.fine, assay.type.test = "logcounts", assay.type.ref = "logcounts")
pred.mouseRNA$labels <- as.factor(pred.mouseRNA$labels)
cellType <- data.frame(barcode = sc_data@assays$RNA@counts@Dimnames[2], cell_type
= pred.mouseRNA$labels)
names(cellType) <- c('barcode', 'cell_type')
cell_types <- cellType$cell_type; names(cell_types) <- cellType$barcode # create
cell_types named list
cell_types <- as.factor(cell_types) # convert to factor data type
```

Method 2: Manual Annotation

```
sc_data = NormalizeData(sc_data, normalization.method = 'LogNormalize',
scale.factor = 10000)
sc_data = FindVariableFeatures(sc_data, selection.method = 'vst', nfeatures = 2000)
sc_data <- ScaleData(sc_data)
sc_data <- RunPCA(sc_data, features = VariableFeatures(object = sc_data))
sc_data <- FindNeighbors(sc_data, dims = 1:15)
sc_data <- FindClusters(sc_data, resolution = 0.25)
markers <- FindAllMarkers(sc_data, only.pos = TRUE, min.pct = 0.25,
logfc.threshold = 0.25)
top10 <- markers %>% group_by(cluster) %>% top_n(n = 5, wt = avg_log2FC)
my_df <- sc_data@meta.data %>% as.data.frame() %>% select(seurat_clusters) %>%
rownames_to_column(var = 'barcode') %>% rename(cluster = seurat_clusters)

cellType <- my_df %>% mutate(cell_type = case_when(cluster == '0' ~ 'Allantois
cell', cluster == '1' ~ 'Bergmann glial cell',
       cluster == '2' ~ 'Neuroblast',
       cluster == '3' ~ 'Neuroendocrine cell',
       cluster == '4' ~ 'Fibroblast',
       cluster == '5' ~ 'Type I spiral ganglion neuron',
       cluster == '6' ~ 'Erythroid cell',
       cluster == '7' ~ 'Type II spiral ganglion neuron',
       cluster == '8' ~ 'Epithelial cell',
       cluster == '9' ~ 'Cardiac progenitor cell',
       cluster == '10' ~ 'Oligodendrocyte precursor cell',
       cluster == '11' ~ 'Endothelial cell',
       cluster == '12' ~ 'Megakaryocyte',
       cluster == '13' ~ 'Hepatocyte',
       cluster == '14' ~ 'Ventricular cardiomyocyte'))
cell_types <- cellType$cell_type; names(cell_types) <- cellType$barcode # create
cell_types named list
cell_types <- as.factor(cell_types) # convert to factor data type
```

Building Reference Set

```
reference <- Reference(sc_counts, cell_types, sc_nUMI)
```

Loading Spatial Data

To read the spatial data in, you can use the following code:

```
coords <-
read.table(gzfile('/xxx/05.AllheStat/BSTViewer_project/subdata/L13_heAuto/barcodes
_pos.tsv.gz'), header = F) %>% dplyr::rename(barcodes = V1, xcoord = V2, ycoord =
V3)
rownames(coords) <- coords$barcodes; coords$barcodes <- NULL
```

Please note that you should replace `/xxx/05.AllheStat/BSTViewer_project/subdata/L13_heAuto/barcodes_pos.tsv.gz` with the actual path to your spatial data.

Expression Matrix

```
expr <- Read10X('/xxx/05.AllheStat/BSTViewer_project/subdata/L13_heAuto/',
cell.column = 1)
sp_data <- CreateSeuratObject(counts = expr, assay = "Spatial")
sp_counts <- as_matrix(sp_data[['Spatial']]@counts)
```

nUMI

```
sp_nUMI <- colSums(sp_counts)
```

Building Spatial Experiment Set

```
puck <- SpatialRNA(coords, sp_counts, sp_nUMI)
```

Joint Analysis

```
myRCTD <- create.RCTD(puck, reference, max_cores = 8)
myRCTD <- run.RCTD(myRCTD, doublet_mode = 'doublet')
```

Finding Marker Genes

```
get_marker_data <- function(cell_type_names, cell_type_means, gene_list) {
marker_means = cell_type_means[gene_list,]
marker_norm = marker_means / rowSums(marker_means)
marker_data = as.data.frame(cell_type_names[max.col(marker_means)])
marker_data$max_epr <- apply(cell_type_means[gene_list,], 1, max)
colnames(marker_data) = c("cell_type",'max_epr')
rownames(marker_data) = gene_list
marker_data$log_fc <- 0
epsilon <- 1e-9
for (cell_type in unique(marker_data$cell_type)) {
cur_genes <- gene_list[marker_data$cell_type == cell_type]
other_mean = rowMeans(cell_type_means[cur_genes, cell_type_names != cell_type])
marker_data$log_fc[marker_data$cell_type == cell_type] <- log(epsilon +
cell_type_means[cur_genes, cell_type]) - log(epsilon + other_mean)
}
return(marker_data)
}
cell_type_info_restr = myRCTD@cell_type_info$info
de_genes <- get_de_genes(cell_type_info_restr, puck, fc_thresh = 3, expr_thresh =
.0001, MIN_OBS = 3)
marker_data_de = get_marker_data(cell_type_info_restr[[2]],
cell_type_info_restr[[1]], de_genes)
saveRDS(marker_data_de, '/share/nas1/guochao/Test/221107marker_data_de_standard.RDS')

write.table(marker_data_de, file =
'/share/nas1/guochao/Test/221107marker_data_de_standard.tsv', sep = '\t', quote = F,
row.names = T)
```

Building Plotting Data

```
results <- myRCTD@results
results_df <- results$results_df
barcodes = rownames(results_df[results_df$spot_class != "reject" & puck@nUMI >=
1,])
my_table = puck@coords[barcodes,]
my_table$class = results_df[barcodes,]$first_type
```

Plotting

```
cal_zoom_rate <- function(width, height){
std_width = 1000
std_height = std_width / (46 * 31) * (46 * 36 * sqrt(3) / 2.0)
if (std_width / std_height > width / height){
scale = width / std_width
}
else {
scale = height / std_height
}
return(scale)
}

png <- readPNG('/share/nas1/dengdj/testing/Barcode_YF/20220923-YF-N1295-1-
2/analysis/XSPT-T/05.AllheStat/allhe/he_roi_small.png')
zoom_scale = cal_zoom_rate(dim(png)[2], dim(png)[1])
my_table = my_table %>% mutate(across(c(x, y), ~.x * zoom_scale))
col = c("#F56867", "#FEB915", "#C798EE", "#59BE86", "#7495D3", "#D1D1D1",
"#6D1A9C", "#15821E", "#3A84E6", "#997273", "#787878", "#DB4C6C", "#9E7A7A",
"#554236", "#AF5F3C", "#93796C", "#F9BD3F", "#DAB370")
```

```
p = ggplot(my_table, aes(x = x, y = dim(png)[1] - y)) +
background_image(png) +
geom_point(shape = 16, size = 1.8, aes(color = class)) +
coord_cartesian(xlim = c(0, dim(png)[2]), y = c(0, dim(png)[1]), expand = FALSE) +
scale_color_manual(values = col) +
theme(axis.title.x = element_blank(), axis.text.x = element_blank(), axis.ticks.x =
element_blank(), axis.title.y = element_blank(), axis.text.y = element_blank(),
axis.ticks.y = element_blank()) +
guides(color = guide_legend(override.aes = list(size = 2.5, alphe = 0.1)))

ggsave(p, file = '/share/nas1/guochao/Test/temp/221107_all.png', width = 10, height
= 7, dpi = 300)
ggsave(p, file = '/share/nas1/guochao/Test/temp/221107_all.pdf', width = 10, height
= 7)
```

Please make sure to replace `/xxx/05.AllheStat/BSTViewer_project/subdata/L13_heAuto/` with the actual path to your spatial data. Below is an example output from the code above.