SCclust T10 Tutorial

Alex Krasnitz, Jude Kendall, Junyan Song, Lubomir Chorbadjiev

2018-09-12

Contents

1 Introduction

The *SCclust* package performs phylogenetic analysis for sets of integer-valued genomic DNA copy-number profiles, each representing a single nucleus. For this purpose,*SCclust* determines a joint set of change points present in the profiles and uses these as features. An incidence table is then set up, with profiles as columns and features as rows and the binary values indicating, for each profile and feature, the presence of the feature in the profile. Dissimilarities among the profiles are computed as Fisher test p-values for pairs of columns. For these, false discovery rates (FDR) are computed using permutations within the table. A hierarchical tree is derived from the dissimilarity matrix, and its branches are identified as clones or sub-clones depending on the maximal dissimilarity and on the number of features shared within the branch.

In this tutorial we show how to use *SCclust* package using data, prepared by *sgains* pipeline as described in Example usage of sGAINS pipeline. *SCclust* package is called as the last step in processing data from *sgains* pipeline. In this tutoral we show how *SCclust* package could be used independently from *sgains* pipeline.

We assume that you have an R environment and have installed *SCclust* package as described in the `README.md`.

2 Data

2.1 Data for the T10 case

This tutorial is based on data published in: Navin N, Kendall J, Troge J, et al. Tumor Evolution Inferred by Single Cell Sequencing. Nature. 2011;472(7341):90-94. doi:10.1038/nature09807. In particular we will use single-cell data for polygenomic breast tumor T10 case available from SRA. Description of samples for T10 could be found in Supplementary Table 1 | Summary of 100 Single Cells in the Polygenomic Tumor T10

Here we will execute *SCclust* package on input prepared by *sgains* pipeline `varbin` step. Alternatively, the reader can go through the data pre-processing steps following *sgains* T10 tutorial. For the purposes of this tutorial we recommend that you download already prepared `varbin` data from example data. For convenience, we briefly describe the `varbin` data format later in this tutorial. Apart from `varbin` T10 data you will need the binning scheme used in the analysis, which could be found here. Finally, we will need `cytoBand.txt`, the cytoband annotation file for the HG19 version of the human genome, which can be downloaded from the UCSC Genome Browser.

2.2 Collect the Neccessary Data

We first create a directory where the data used in this tutorial will be stored:

```
mkdir T10data
cd T10data
```

Next, we download and extract T10 `varbin` data:

```
wget -c \
  https://github.com/KrasnitzLab/SCclust/releases/download/v1.0.0RC3/\
  navin_t10_varbin_data.tar.gz
tar zxvf navin_t10_varbin_data.tar.gz
rm navin_t10_varbin_data.tar.gz
```

We also download and extract the binning scheme used in preparation of `varbin` data:

```
wget -c \
  https://github.com/KrasnitzLab/SCclust/releases/download/v1.0.0RC3/\
  hg19_R50_B20k_bins_boundaries.txt.gz
gunzip hg19_R50_B20k_bins_boundaries.txt.gz
```

And finally we download the `cytoBand.txt`, a cytoband annotation table for Human reference genome *hg19*:

```
wget -c \
  http://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/cytoBand.txt.gz
gunzip cytoBand.txt.gz
```

The data directory should have following structure:

```
.
|-- T10data
    |-- cytoBand.txt
    |-- hg19_R50_B20k_bins_boundaries.txt
    |-- varbin
        |-- SRR052047.varbin.20k.txt
        |-- SRR052148.varbin.20k.txt
```

```
        |-- SRR053437.varbin.20k.txt
        ...
```

2.3 Explore the Downloaded Data

Next, we attach *SCclust* to the R session:

```
library("SCclust")
```

```
library(futile.logger)
flog.threshold(ERROR)
#> NULL
```

2.3.1 Binning scheme

The binning scheme is a table describing a partition of the genome into bins to sort the sequence reads into. The bin boundaries are defined such that each bin contains an approximately equal number of uniquely mapping reads. Also tabulated is the GC content of each bin.

```
gc_df <- read.csv("T10data/hg19_R50_B20k_bins_boundaries.txt",
                  header = T, sep='\t')
gc_df$chrom.numeric <- chrom_numeric(gc_df$bin.chrom)
pander::pandoc.table(head(gc_df))
```

Table 1: Table continues below

bin.chrom	bin.start	bin.start.abspos	bin.end	bin.length
chr1	0	0	859077	859077
chr1	859077	859077	999002	139925
chr1	999002	999002	1141973	142971
chr1	1141973	1141973	1280121	138148
chr1	1280121	1280121	1435418	155297
chr1	1435418	1435418	1603686	168268

mappable.positions	gc.content	chrom.numeric
131390	0.4358	1
131390	0.6281	1
131391	0.6027	1
131390	0.6284	1
131390	0.5758	1
131391	0.5691	1

We are using a binning scheme with 20000 bins:

```
dim(gc_df)
```

[1] 20000 8

2.3.2 Cytobands and Centromeres for HG19

The `cytoBand.txt` file contains a table of cytoband boundary coordinates as per HG19.

```
cytobands <- read.csv("T10data/cytoBand.txt", header = F, sep='\t')
knitr::kable(head(cytobands))
```

V1	V2	V3	V4	V5
chr1	0	2300000	p36.33	gneg
chr1	2300000	5400000	p36.32	gpos25
chr1	5400000	7200000	p36.31	gneg
chr1	7200000	9200000	p36.23	gpos25
chr1	9200000	12700000	p36.22	gneg
chr1	12700000	16200000	p36.21	gpos50

The main reason we need `cytoBand.txt` is to get the location of centromeres. Centromeric regions of the genome often contain repetitive sequence, leading to mapping artefacts and distorted read counts. For this reason, we opt to mask them prior to copy-number profile segmentation. In the present version we mask cytobands p11 through q11.

To identify centromeric regions to be masked we use `calc_centroareas` function passing the list of bands to be masked:

```
centroareas <- calc_centroareas(cytobands, centromere=c("p11", "q11"))
knitr::kable(head(centroareas))
```

	chrom	from	to
33	1	120600000	128900000
393	2	83300000	102700000
508	3	87200000	98300000
556	4	48200000	52700000
604	5	46100000	58900000
655	6	57000000	63400000

The locations of centromeric regions are now tabulated in `centroareas`.

2.3.3 Varbin Samples Data

The `SGAINS` pipeline produces separate `varbin` file for each sample. Each 'varbin' file is a comma-separated table with each row representing a bin and with columns for chromosome, position of the bin in the chromosome, absolute position of the bin, how many reads are mapped in this bin and the ratio. Here the absolute position means the position relative to the start of chromosome 1, assuming that sequences of chromosomes 1,2,...,Y are concatenated.

```
sample_df <- read.csv("T10data/varbin/SRR052047.varbin.20k.txt",
                      header=T, sep='\t')
knitr::kable(head(sample_df))
```

chrom	chrompos	abspos	bincount	ratio
chr1	0	0	51	0.3327000
chr1	859077	859077	57	0.3718412

chrom	chrompos	abspos	bincount	ratio
chr1	999002	999002	89	0.5805941
chr1	1141973	1141973	53	0.3457471
chr1	1280121	1280121	99	0.6458294
chr1	1435418	1435418	63	0.4109824

```
sample_df <- read.csv("T10data/varbin/SRR052148.varbin.20k.txt",
                      header=T, sep='\t')
knitr::kable(head(sample_df))
```

chrom	chrompos	abspos	bincount	ratio
chr1	0	0	125	0.5530061
chr1	859077	859077	69	0.3052594
chr1	999002	999002	90	0.3981644
chr1	1141973	1141973	57	0.2521708
chr1	1280121	1280121	98	0.4335568
chr1	1435418	1435418	84	0.3716201

3 Segmentation of Varbin Data

3.1 Prepare list of bins that are inside or intersect with centromeric regions

Next, we use the `calc_regions2bins` function to determine which bins overlap with regions tabulated in `centroareas`:

```
centrobins <- calc_regions2bins(gc_df, centroareas)
length(centrobins)
```

[1] 1513

3.2 Exclude centromeric bins from the binning scheme

After excluding centromeric bins from the binning scheme, the new binning scheme has fewer bins:

```
gc_df <- gc_df[-centrobins, ]
dim(gc_df)
```

[1] 18487 8

3.3 Collect Varbin files for all samples

With the help of `varbin_input_files` we create a vector of character strings, each representing a path to a `varbin` data file in `T10data/varbin` directory:

```
varbin_files <- varbin_input_files("T10data/varbin", "*.varbin.20k.txt")
knitr::kable(head(varbin_files))
```

names	cells	paths
SRR052047.varbin.20k.txt	SRR052047	T10data/varbin/SRR052047.varbin.20k.txt
SRR052148.varbin.20k.txt	SRR052148	T10data/varbin/SRR052148.varbin.20k.txt
SRR053437.varbin.20k.txt	SRR053437	T10data/varbin/SRR053437.varbin.20k.txt
SRR053600.varbin.20k.txt	SRR053600	T10data/varbin/SRR053600.varbin.20k.txt
SRR053602.varbin.20k.txt	SRR053602	T10data/varbin/SRR053602.varbin.20k.txt
SRR053604.varbin.20k.txt	SRR053604	T10data/varbin/SRR053604.varbin.20k.txt

For the purpose of this tutorial we will use a subset of the first 10 samples out of a 100 total found by `varbin_input_files` function:

```
varbin_files <- varbin_files[seq(10),]
dim(varbin_files)
#> [1] 10  3
```

3.4 Segment varbin files

Next, we approximate the copy number ratio as a function of the bin number by a piecewise-constant function. This process is called segmentation. First we normalize bincount based on the GC content using LOWESS smoothing and after that we use Circular Binary Segmentation (CBS) algorithm, as implemented in the R package `DNAcopy`, but modified for a more consistent elimination of very short segments.

mention the quantal procedure

```
res <- segment_varbin_files(varbin_files, gc_df, centrobins)
#> Analyzing: Sample.1
#> Analyzing: Sample.1
#> Analyzing: Sample.1
#> Analyzing: Sample.1
#> Analyzing: Sample.1
#> Analyzing: Sample.1
#> Analyzing: Sample.1
#> Analyzing: Sample.1
#> Analyzing: Sample.1
#> Analyzing: Sample.1
```

The `segment_varbin_files` function returns the segmented copy number ratio along with the input ratios for all the samples

```
knitr::kable(head(res$seg[,c(1:8)]))
```

chrom	chrompos	abspos	SRR052047	SRR052148	SRR053437	SRR053600	SRR053602
1	0	0	1.914278	1.961422	1.981773	1.959143	1.96083
1	859077	859077	1.914278	1.961422	1.981773	1.959143	1.96083
1	999002	999002	1.914278	1.961422	1.981773	1.959143	1.96083
1	1141973	1141973	1.914278	1.961422	1.981773	1.959143	1.96083
1	1280121	1280121	1.914278	1.961422	1.981773	1.959143	1.96083
1	1435418	1435418	1.914278	1.961422	1.981773	1.959143	1.96083

```
knitr::kable(head(res$ratio[,c(1:8)]))
```

chrom	chrompos	abspos	SRR052047	SRR052148	SRR053437	SRR053600	SRR053602
1	0	0	0.6593663	1.079580	1.222141	1.108080	1.079869
1	859077	859077	1.8631489	1.781506	2.062367	1.765271	1.896897
1	999002	999002	2.4473278	1.908000	2.396548	2.878972	2.471616
1	1141973	1141973	1.7385504	1.479967	1.654073	1.247160	2.736819
1	1280121	1280121	2.2813321	1.692508	1.706780	1.688706	2.188923
1	1435418	1435418	1.3992734	1.382124	1.448576	1.665827	1.627200

3.5 Construct file names for the segmented output and store segmentation and ratio results.

The file names as constructed will be required for visualization.

```
filenames <- case_filenames("T10data/results", "NavinT10")
```

```
save_table(filenames$seg, res$seg)
save_table(filenames$ratio, res$ratio)
```

```
cells <- uber_cells(res$seg)$cells
save_table(filenames$cells, data.frame(cell=cells))
```

```
dir("T10data/results")
#> [1] "NavinT10.cells.txt"       "NavinT10.clone.txt"
#> [3] "NavinT10.featuremat.txt" "NavinT10.features.txt"
#> [5] "NavinT10.ratio.txt"       "NavinT10.seg.txt"
#> [7] "NavinT10.sim_pv.txt"      "NavinT10.tree.txt"
#> [9] "NavinT10.true_pv.txt"
```

4 Construct features and feature matrix

At this point, we convert the set of segmented copy number profiles into a feature matrix. This process includes a number of steps. First, since single-cell copy number must be integer, we round the segmented values to the nearest integer. Some of the neighboring segments may merge as a result. As part of this procedure we calculate the ploidy of each cell. Ploidy here means the predominant integer value of copy number in the cell genome. We compute this value for three alternative definitions of ploidy: the genome-wide median, the genome-wide mode and the mode of all chromosome-wide modes. Also, at this point we compute the percentage of the genome with copy number 0, i.e., lost homozygously. A high percentage of homozygous loss is likely a sign of DNA degradation ex-vivo. Such cells are removed from further processing (this is controlled using `homoloss` parameter of `calc_pinmat` fuction).

Display the ploidies table here?

Next, we replace each profile by a set of change points, i.e., discontinuities in the integer segmented value. Each change point has two values associated with it: position (bin number) and the sign of change going in the direction of increasing bin number. The positive and negative change points are handled separately. Further, we turn change points into short intervals centered at the change point positions, with the interval length is an integer number of bin sdetermined by the `smear` parameter. For each sign of the change points, the features are a minimal set of points such that at least one of them is contained in each of the short intervals centered at the change points. The feature matrix is then a binary matrix with cells as columns and features as rows, indicating, for each cell and feature, the presence of a short change-point centered interval containing the feature. The results are stored in a list called `pins`, with two items: `pins` is a tabulation of the feature locations and signs and `pinmat` is the feature matrix.

```
pins <- calc_pinmat(gc_df, res$seg, dropareas=centroareas)
pinmat_df <- pins$pinmat
pins_df <- pins$pins
save_table(filenames$featuremat, pinmat_df)
save_table(filenames$features, pins_df)
```

```
dir("T10data/results")
#> [1] "NavinT10.cells.txt"       "NavinT10.clone.txt"
#> [3] "NavinT10.featuremat.txt" "NavinT10.features.txt"
#> [5] "NavinT10.ratio.txt"       "NavinT10.seg.txt"
#> [7] "NavinT10.sim_pv.txt"      "NavinT10.tree.txt"
#> [9] "NavinT10.true_pv.txt"
```

5 Dissimilarity matrix and significance of dissimilarities

In the next step, we use the feature matrix to compute dissimilarities among cells. These are defined, for each pair of cells, as (one-sided) Fisher test p-values for the corresponding two columns of the feature matrix. To analyze the significance of dissimilarities, we will need to compare their observed distribution to a sampling from a null model. For the latter, we perform `nsim` permutations of the feature set, preserving the column and, approximately, the row sums and recomputing the dissimilarity matrix each time. Each permutation consists of `nsweep` sweeps. A sweep consists of a Metroplolis-Hastings step for every row and a swapping move for pairs of columns, such that every column is involved. Positive and negative feature submatrices are permuted separately. The entire procedure is implemented as `sim_fisher_wrapper` function, which returns a list, with the vectors of the observed dissimilarities and those sampled from the null distribution as items.

```
fisher <- sim_fisher_wrapper(
      pinmat_df, pins_df, njobs=30, nsim=150, nsweep=10)
true_pv <- fisher$true
sim_pv <- fisher$sim
```

These distributuions can be saved if so desired.

```
save_mat(filenames$true_pv, true_pv)
save_mat(filenames$sim_pv, sim_pv)
```

Next, we use function `fisher_fdr` to determine which of the observed dissimilarities are outliers on the left in the null distribution, i.e., much smaller than expected. The value is a matrix of false discovery rates (FDR) for every pair of cells. We also arrange the observed dissimilarities in a distance matrix.

```
# devtools::load_all()
# flog.threshold(DEBUG)
mfdr <- fisher_fdr(true_pv, sim_pv, cells)
mfdr[1:5,1:5]
#>           SRR052047 SRR052148 SRR053437 SRR053600 SRR053602
#> SRR052047         0         0         0         0         0
#> SRR052148         0         0         0         0         0
#> SRR053437         0         0         0         0         0
#> SRR053600         0         0         0         0         0
#> SRR053602         0         0         0         0         0
mdist <- fisher_dist(true_pv, cells)
mdist[1:5,1:5]
#>               SRR052047  SRR052148  SRR053437     SRR053600    SRR053602
```

```
#> SRR052047  0.000000e+00 -0.1048271  0.0000000 -9.469255e-05 -0.094775405
#> SRR052148 -1.048271e-01  0.0000000 -0.2742585  0.000000e+00 -0.181475429
#> SRR053437  0.000000e+00 -0.2742585  0.0000000  0.000000e+00 -0.126980095
#> SRR053600 -9.469255e-05  0.0000000  0.0000000  0.000000e+00 -0.004500073
#> SRR053602 -9.477540e-02 -0.1814754 -0.1269801 -4.500073e-03  0.000000000
```

6 Hierarchical clustering

At this point, we use the distance matrix derived as above to compute hierarchical clustering of cells. The function `hclust_tree` invokes the R core function `hclust` with the linkage choice as indicated (here `'average'`, the default value) and with `mdist` as the dissimilarity matrix. The value is an extension of the `R hclust` object, to include additional items, as required for the clone identification in the following. The more important ones are `nodesize` (the number of leaves for each node), `mergefdr` (maximal pairwise FDR anywhere in the node) and `sharing`, a matrix specifying for each node and each feature the fraction of leaves in the node with the feature. As an interface with Python software, the function `tree_py` is provided, which outputs the same hierarchical tree in the format for Python hierarchical clustering suite.

```
hc <- hclust_tree(pinmat_df, mfdr, mdist,hcmethod='average')
names(hc)
#>  [1] "merge"       "height"      "order"       "labels"      "method"
#>  [6] "call"        "dist.method" "mergefdr"    "meanfdr"     "nodesize"
#> [11] "leaflist"    "labellist"   "sharing"     "complexity"
tree_df <- tree_py(mdist, method='average')
head(tree_df)
#>      index1 index2      height clustersize
#> [1,]      1      6 -1.97022074           2
#> [2,]      2      9 -0.63406817           2
#> [3,]      8     10 -0.62533815           3
#> [4,]      4     12 -0.36620370           4
#> [5,]     11     13 -0.14854512           6
#> [6,]      5     14 -0.09879872           7
```

```
save_table(filenames$tree, tree_df)
```

7 Finding clones and subclones

Finally, we determine which nodes of the tree represent clones. Two types of clones are considered: hard and soft. A hard clone is defined as a node with at least `minsize` leaves for which `mergefdr` is below `fdrthresh` (the FDR condition), the number of features shared by at least `sharemin` leaves is at least `nshare` (the sharing condition), and such that the parent node does not have at least one of these two properties. Once the hard clones are identified, they can be expanded as follows: ask whether the parent of the hard clone meets the sharing condition as above; if not, the hard clone is also the soft clone; if so, expand the clone to the parent and iterate.

```
hc <- find_clones(hc)
```

The suction `find_clones` supports multiple parameters that control its behavior such as `fdrthresh` that defines maximal allowed value for log10(FDR) for any pair of leaves in a clone node, `sharemin` that defines when a feature is considered 'widely shared' if present in sharemin fraction of leaves in a

node, `nshare` that defines minimal number of 'widely shared' features in a hard clone. For the full list of parameters please consult the manual of SCclust package.

As a result of this computation, an item called `softclones` is added to `hc`. It is a matrix with two rows named `hard` and `soft` and as many columns as there are hard clones. In the `hard` row the node numbers for all hard clones are listed, whereas the `soft` row contains the node numbers for the soft clones obtained by expanding each of the hard clones. An expansion from distinct hard clones may result in the same soft clone.

```
hc$softclones
#>
#> hard
#> soft
```

To complete the analysis, we compute subclones. In essence, this computation is a loop over clones. If the number of cells in the clone exceeds a minimal value (e.g., 6), we extract a submatrix of the feature matrix, restricted to the cells in the clone. The clonal structure analysis is then performed for this sub-matrix as described above, starting with the computation of dissimilarities.

```
subclones <- find_subclones(hc, pinmat_df, pins_df, nsim=nsim)
```

For the full list of parameters that control the behavior of `find_sublones` please consult the manual pages of SCclust package.

```
save_table(filenames$clone, subclones)
```