Reference manual:

RaceID2 (Rare Cell Type Identification)
and
StemID (Stem Cell Identification)

Developed by Dominic Grün and Alexander van Oudenaaarden

Contents

1 RaceID2

1.1 Prerequisites

This manual describes in detail how to run the R-code for the RaceID2 (**Ra**re **Ce**ll Type **ID**entification) algorithm, an advanced version of RaceID[1], including k-medoids clustering of single cell expression data, identification of outlier cells, and inference of the final clusters that represent distinct cell types or cell states.

The algorithm is implemented as an `S4` class object, named `SCseq`. The R script `RaceID2_StemID_class.R` contains the specifications of the class object and all methods. The additional R script `RaceID2_StemID_sample.R` lists sample commands to run a complete analysis on example data. These data are provided as text file `transcript_counts_intestine.xls` and should be stored in the working directory together with the script `RaceID2_StemID_class.R`. After opening an R workspace in the same directory the commands in `RaceID2_StemID_sample.R` can be executed.

The algorithm requires a number of R packages that have to be installed:

```
>install.packages(c("tsne", "pheatmap", "MASS", "cluster", "mclust",
"flexmix", "lattice", "fpc",  "RColorBrewer", "permute", "amap", "locfit",
"vegan"))
```

Prior to running the code these packages have to be loaded into the workspace:

```
> require(tsne)
> require(pheatmap)
> require(MASS)
> require(cluster)
> require(mclust)
> require(flexmix)
> require(lattice)
> require(fpc)
> require(amap)
> require(RColorBrewer)
> require(locfit)
> require(vegan)
```

The `S4` class object is specified in the R script `RaceID2_StemID_class.R`, which has to be loaded into the workspace. If the file is deposited in the working directory, this is done by the command:

```
> source("RaceID2_StemID_class.R")
```

Otherwise, the full path has to be indicated.

Please note the final paragraph “Remarks on running RaceID2” containing important advice on how to choose parameters for running RaceID2.

1.2 Input data

Input data have to be provided as a `data.frame` object with expression values, i. e. raw read or transcript counts, with genes as rows and cells as columns. Row names should correspond to gene identifiers and column names to identifiers of single cell samples. It is recommended to remove any spike-in RNAs from the expression table, if clustering due to variability of non-endogenous transcripts should be avoided.

The provided example data can be loaded using the following commands:

```
> x <-read.csv("transcript_counts_intestine.xls", sep="\t", header=TRUE)
```

The first column contains unique gene ids and has to be assigned as row names:

```
> rownames(x) <- x$GENEID
```

The gene ids of the non-endogenous spike-in RNAs starts with “ERCC” and these transcripts have to be removed:

```
> prdata <- x[grep("ERCC",rownames(x),invert=TRUE),-1]
```

Now, the input data have the correct format:

```
> head(prdata[,1:4])
                         I_1      I_2      I_3      I_4
0610005C13Rik__chr7  2.007853 1.001958 0.000000 5.049473
0610007N19Rik__chr15 0.000000 0.000000 0.000000 0.000000
0610007P14Rik__chr12 1.001958 0.000000 1.001958 3.017717
0610008F07Rik__chr2  0.000000 0.000000 0.000000 1.001958
0610009B14Rik__chr12 0.000000 0.000000 0.000000 0.000000
0610009B22Rik__chr11 1.001958 0.000000 1.001958 0.000000
```

The primary input data object is denoted as `prdata` in the following.

Given a data.frame `prdata` of transcript counts, an `SCseq` object can be created:

```
> sc <- SCseq(prdata)
```

The transcript count data.frame initializes the slots `sc@expdata`, `sc@ndata`, and `sc@fdata`. The contents of a slot, for example `expdata`, can be written out by the command:

```
> sc@expdata
```

1.3 Preprocessing of input data

As a first step the input data are subject to filtering by applying the method `filterdata`.

```
> sc <- filterdata(sc, mintotal=3000, minexpr=5, minnumber=1, maxexpr=500,
downsample=TRUE, dsn=1, rseed=17000)
```

The function call displays the default values of all arguments.
After discarding cells with less than `mintotal` transcripts, the expression data are normalized. Normalization is performed by dividing transcript counts in each cell by the total number of transcripts in this cell followed by multiplication with the median of the total number of transcripts across cells (median normalization). If `downsample` is set to `TRUE`, then transcript counts are downsampled to `mintotal` transcripts per cell, instead of the normalization. Downsampled versions of the transcript count data are averaged across `dsn` samples. If `dsn` equals one, sampling noise should be comparable across cells, while for high numbers of `dsn` the data will become similar to the median normalization. To ensure reproducibility of downsampling the random number generator is initialized with a fixed seed `rseed` that can be changed. By default `downsample` is set to `TRUE.`
After normalization or downsampling a pseudocount of `0.1` is added to the expression data, and the output overwrites the slot `sc@ndata`.
Finally, genes that are not expressed at `minexpr` transcripts in at least `minnumber` cells are discarded.
If expression data are derived based on unique molecular identifiers[2–4] (UMIs) it is recommended to filter out genes with saturated UMI counts, since differences in saturation between cells will affect the transcriptome correlation and potentially introduce spurious clusters. This is done by discarding genes with at least `maxexpr` transcripts in at least a single cell after normalization or downsampling. To disable this filtering step, `maxexpr` should be set to `Inf`. By default `maxexpr` is set to `Inf`. After gene filtering, the transcript count data overwrite the slot `sc@fdata`.
All filtering parameters used are written to the slot `sc@filterpar`.

1.4 k-medoids clustering

The preprocessed data are clustered by k-medoids clustering.

```
> sc <- clustexp(sc, clustnr=30, bootnr=50, metric="pearson", do.gap=FALSE,
sat=TRUE, SE.method="Tibs2001SEmax", SE.factor=.25, B.gap=50, cln=0,
rseed=17000, FUNcluster="kmedoids")
```

The function call displays the default values of all arguments. These arguments define how the clustering is performed when calling the method:

`metric:` the input data `sc@fdata` are transformed to a distance object. Distances can be computed based on different metrics. Possible values are `"pearson"`, `"spearman"`, `"kendall"`, `"euclidean"`, `"maximum"`, `"manhattan"`, `"canberra"`, `"binary"` or `"minkowski"`. Default is `"pearson"`. In case of the correlation based methods, the distance is computed as 1 – correlation. The distance object is written to the slot `sc@distances`. K-medoids clustering is performed on this distance object.

`cln:` the number of clusters for k-medoids clustering. Default is `0`. In this case, the cluster number is determined based on the gap statistic[5] and `do.gap` has to be `TRUE`.

`do.gap:` if `do.gap` equals `TRUE` the number of clusters is determined based on the gap statistic[5], which is defined as the difference between within-cluster dispersion of the original data and a background distribution modeled by a uniform distribution within the limits of the original data. Default is `FALSE`.

`sat:` if `sat` equals `TRUE` the number of clusters is determined based on finding the saturation point of the within-cluster dispersion as a function of the cluster number. If `do.gap` equals `TRUE` the number derived from gap statistic[5] is overwritten. Default is `TRUE`.

`clustnr:` maximum number of clusters for the computation of the gap statistic or derivation of the cluster number by the saturation criterion. Default is `30`. If more major cell types are expected a higher number should be chosen.

`B.gap:` number of bootstrap runs for the calculation of the gap statistic. Default is `50`.

`SE.method:` the clustering routine calls a modified version of the `maxSE` function from the `cluster` package to determine the first local maximum of the gap statistic. By default, we use the method `"Tibs2001SEmax"` for calling the first local maximum (see specification of `maxSE`). This method requires that the maximum exceeds the values of its neighbors by a fraction of their standard deviation. This fraction is defined by the parameter `SE.factor`. All methods defined for the original `maxSE` function can also be used.

`SE.factor:` fraction of the standard deviation by which the local maximum is required to differ from the neighboring points it is compared to. Default is `0.25`.

`FUNcluster:` the clustering method applied. One of the following methods can be selected: `kmedoids`, `kmeans`, `hclust`. RaceID2 is designed for k-medoids clustering and therefore it is recommended to use only the `kmedoids` method. Default is `kmedoids`.

Given the desired number of clusters, k-medoids clustering is performed using the `clusterboot` function from the `fpc` package. The `clusterboot` function performs bootstrapping of k-medoids clustering and quantifies the robustness of all clusters by Jaccard's similarity.

`bootnr:` number of booststrapping runs for `clusterboot`. Default is `50`.

`rseed:` to obtain exactly reproducible results between different runs the random number generator for the `clusterboot` is seeded manually. Default is `17000`.

All filter parameters used are written to the slot `sc@clusterpar`. Results of the `clustexp` method are written to the slot `sc@cluster,` which contains a list of three objects:

`gap`: a `clusGap` object with gap statistic (`NULL` if `do.gap` is `FALSE`).

`jaccard`: Jaccard's similarity for each cluster computed by bootstrapping (see below).

`kpart`: clustering partition computed by k-medoids clustering.

If `clustexp` was executed with `sat=TRUE`, it is recommended to examine the saturation behavior. The within-cluster dispersion as a function of the cluster number can be plotted with the function `plotsaturation`.

```
> plotsaturation(sc,disp=TRUE)
```

The cluster number determined based on the saturation criterion is circled in blue. The cluster number is inferred based on the saturation of the average within-cluster dispersion. In this approach the number of clusters is the minimal number k_i such that the change of the within-cluster dispersion upon further increase of the cluster number $k_{i+1} = k_i + 1$ is equal, within the estimated error interval, to the average change upon further increase of the cluster number quantified by a linear regression across $k_{i+2}, \ldots, k_{\max}$. In other words, the cluster number is determined such that adding more clusters only leads to a linear decrease of the within cluster-dispersion. The change of the within-cluster dispersion can be plotted with the function `plotsaturation`.

```
> plotsaturation(sc,disp=TRUE)
```

The change of the within-cluster dispersion and the within-cluster dispersion itself as a function of the cluster number are shown in Figure 1.

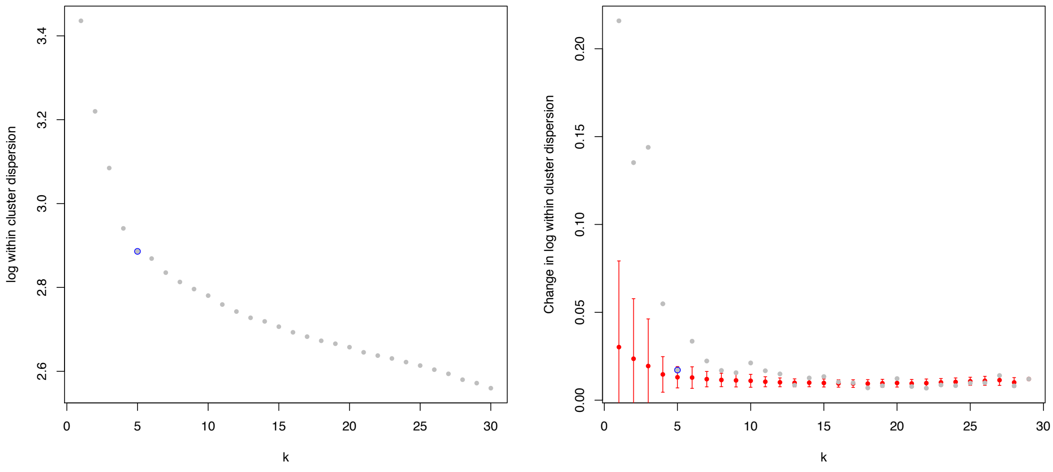

Figure 1. Within-cluster dispersion (left) and change of the within-cluster dispersion (right) as a function of the number of cluster. The average change at cluster number *k* upon further increase is shown in red.

If `clustexp` was executed with `do.gap=TRUE`, it is recommended to examine the gap statistic:

```
> plotgap(sc)
```

In Figure 2 an example is shown. Here, the algorithm determines a cluster number of five. The saturation based approach is recommended since it is more robust and faster to compute.

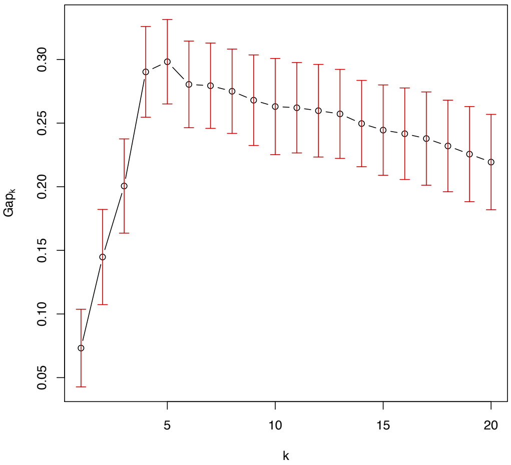

Figure 2. Gap statistic.

If the saturation of the within-cluster distance exhibits further gains at larger cluster numbers (grey dots far above the red error bars in Figure 1) or if the first local maximum of the gap statistic is not clearly identifiable the cluster number can be changed manually by setting the argument `cln` to the desired number. A good choice will be the minimal cluster number where a clear saturation is apparent. We note that the outlier calling procedure can introduce new clusters and therefore corrects for an initial underestimation. However, if the initial number of clusters is too low, the clusters will be very heterogeneous and the outlier calling procedure potentially does not perform well.

Robustness of the clusters can be assessed in various ways. The reproducibility of individual clusters across bootstrapping runs is reflected by Jaccard's similarity (intersect of two clusters divided by the union). A barchart plot of Jaccard's similarities can be generated (see Figure 3):

```
> plotjaccard(sc)
```

Stable clusters should have a Jaccard's similarity greater than 0.6. Clusters with lower values can be resolved by the outlier identification procedure. If too many clusters (more than two) have low values, a smaller number of clusters should be considered by setting the argument `cln` to the desired number.

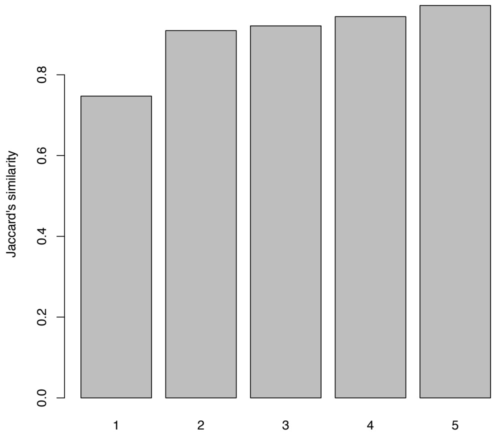

Figure 3. Jaccard's similarities of k-medoids clusters.

An alternative method that reflects the resolution of the clustering is a silhouette plot. The silhouette provides a representation of how well each point is represented by its cluster in comparison to the closest neighboring cluster. It computes for each point the difference between the average similarity to all points in the same cluster and to all points in the closest neighboring cluster. This difference is normalize such that it can take values between -1 and 1 with higher values reflecting better representation of a point by its cluster.

A silhouette plot for the k-medoids clusters can be generated (see Figure 4):

```
> plotsilhouette(sc)
```

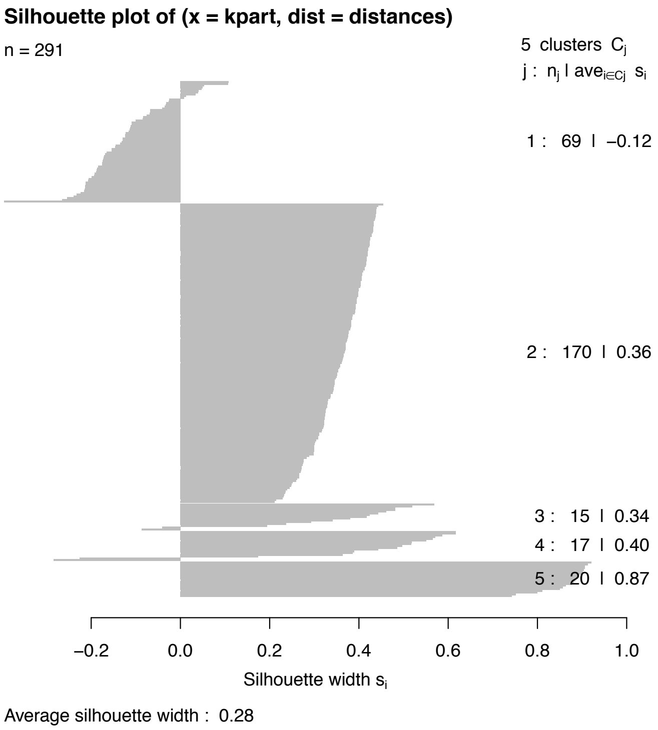

Figure 4. Silhouette plot.

The clusters can also be plotted in a heatmap representation of the cell-to-cell distances (Figure 5):

```
> x <- clustheatmap(sc,final=FALSE,hmethod="single")
```

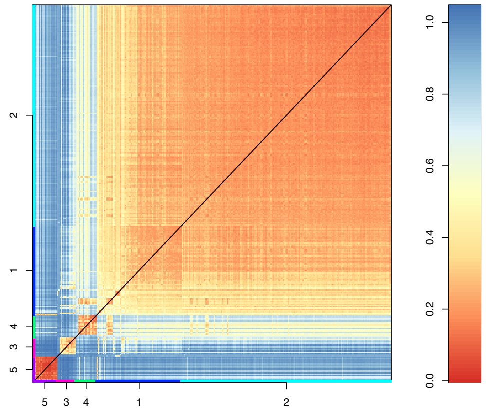

Figure 5. Heatmap of the k-medoids clusters. Individual clusters are highlighted with rainbow colors along the x- and y-axes.

The logical argument `final` controls if the k-medoids clusters or the final clusters including the outliers (see below) are plotted. The hierarchical clustering method used for ordering cluster centers can be selected by setting `hmethod` to one of the methods used by the `hclust` function. Default is `"single"`. The output argument is a vector containing the clustering partition in the same order as indicated along the axes of the heatmap. The colours used for highlighting the clusters in this plot and each of the following plots where clusters are highlighted are internally generated in the `clustexp` and the `findoutliers` method, respectively, and stored in the slot `sc@fcol`. If another color scheme is desired a vector with the corresponding color names has to be assigned to the slot. For example, in case of five clusters:

```
> sc@fcol <- rainbow(5)
```

1.5 Outlier identification

After the `clustexp` method has been executed, the outlier identification and the redefinition of the final clusters can be performed by calling the method `findoutliers`:

```
> sc <- findoutliers(sc, outminc=5,outlg=2,probthr=1e-3,thr=2**-(1:40),
outdistquant=.95)
```

The function call displays the default values of all arguments.

`outminc :` expression cutoff for the identification of outlier genes is defined. Default is `5`.

`probthr:` defines the probability threshold for outlier calling. If the probability of observing a given expression level for a gene in a cell is lower than this cutoff (based on the negative binomial distribution for the calibrated noise model), the cell is considered an outlier for this gene. Default is 10^{-3}.

`outlg:` minimal number of outlier genes required to identify a cell as an outlier. Default is `2`.

`outdistquant:` outlier cells are merged to outlier clusters if their similarity exceeds the `outdistquant`-quantile of the similarity distribution for all pairs of cells that are together in one of the original clusters. Default is `0.95`.

`thr`: probability values for which the number of outliers is computed in order to plot the dependence of the number of outliers on the probability threshold (see Figure 6).

The method consists of three steps. The first step is the calibration of background model. The outlier identification is based on the distribution of transcript counts within a cluster. For each gene, the transcript count distribution is assumed to be a negative binomial, governed by two parameters: the mean and the dispersion parameter. While the mean is determined directly by averaging expression of a gene across all cells in a cluster, the dispersion parameter is derived from the ensemble of all cells based on the average variance-mean dependence. This dependence is modeled by a second order polynomial in logarithmic space. The approach relies on the idea that the majority of genes are expressed at similar levels across different clusters and biologically meaningful expression differences of a gene between cells in the same cluster should strongly exceed this expected variability.
The regression of the variance-mean dependence and the derivation of the dispersion parameter as a function of the mean is performed in the noise model calibration step of the algorithm. The results are written to the slot `sc@background`, which is a list of three objects:

`vfit`: object of the class `lm` containing the regression of the variance on the mean.

`lvar:` function that describes the dependence of the variance on the mean expression.

`lsize:` function that describes the dependence of the dispersion parameter on the mean expression.

The quality of the model fit can be inspected by plotting `lvar` as a function of the mean (Figure 6):

```
> plotbackground(sc)
```

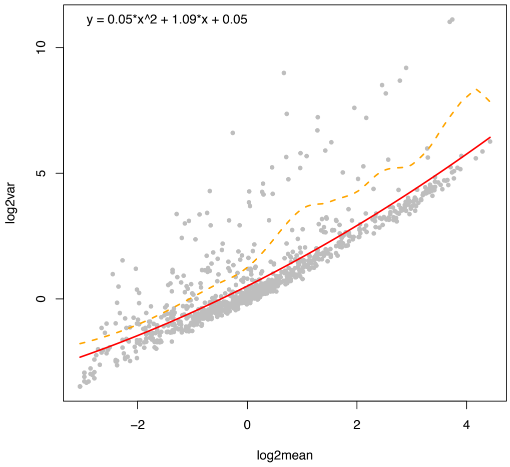

Figure 6. Regression of the variance on the mean by a second order polynomial in logarithmic space.

The plot also contains a local regression for comparison (orange broken line). The polynomial model fit is indicated in the upper left corner.

The second step is the identification of outlier cells. In this step for each gene a probability is computed for the observed transcript counts in every cell of a cluster, inferring the background distribution based on the mean expression in this cluster. If a minimum number of `outlg` genes in a single cell have a transcript count with a probability less than `probthr` after multiple testing correction by the Benjamini-Hochberg method (across cells), this cell is flagged as an outlier. Only genes with more than `outminc` transcripts in at least a single cell within the cluster are included in this test. The results of the outlier detection are written to the slot `sc@out`, which is a list of four elements:

`out:` a vector with the names of all outlier cells.

`stest:` a vector with the number of outliers as a function of decreasing values for `probthr` stored in `thr`.

`thr:` a vector of probability thresholds used to calculate the number of outliers stored in `stest`.

`cprobs:` a vector with the probability of the outlier gene with the largest probability, i. e. from a list of genes ranked by outlier probability in increasing order the probability of the gene at rank `outlg` is given.

The number of outliers as a function of the probability threshold can be plotted (Figure 7).

```
> plotsensitivity(sc)
```

The probability threshold should be chosen such that the tail of the distribution is captured as outliers to ensure maximum sensitivity of the method.

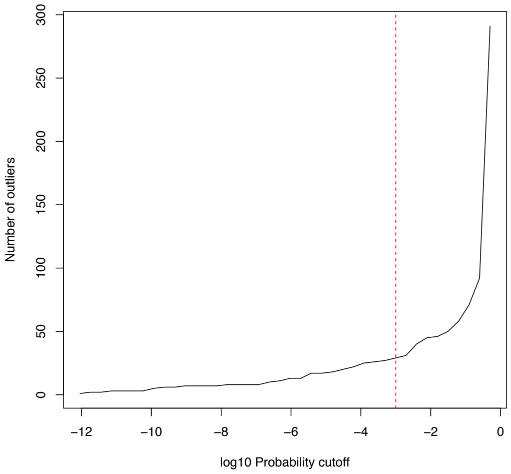

Figure 7. Number of outliers as a function of the probability threshold.

Depending on the expectation for a given dataset the parameter `outlg` can be varied. For instance, if the sensitivity of the sequencing experiment was low and only a few highly expressed genes were reliably quantified it could be necessary to reduce `outlg` to 1, i. e., to require only a single outlier gene to identify a cell as an outlier.

The method `plotoutlierprobs` can be executed to produce a barplot of the outlier probabilities of all cells across all clusters (Figure 8):

```
> plotoutlierprobs(sc)
```

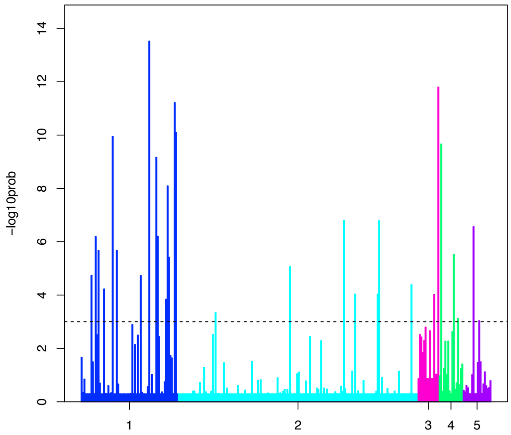

Figure 8. Outlier probabilities of all cells across clusters.

In the final step of the algorithm outlier cells are merged into clusters based on their similarity: outlier cells are merged to outlier clusters if their similarity exceeds the `outdistquant` quantile of the similarity distribution for all pairs of cells within one of the original clusters. After the outlier cells are merged, new cluster centers are defined for the original clusters after removing the outliers and for the additional outlier clusters. Subsequently, each cell is assigned to the nearest cluster center. The final clusters are denoted by increasing number, where the number of the remaining original clusters are the same as in the original partition `sc@cluster$kpart`. The final partitioning of the cells into clusters is stored in the slot `sc@cpart` and can be written to a text file:

```
> x <- data.frame( CELLID=names(sc@cpart), cluster=sc@cpart )
> write.table(x[order(x$cluster,decreasing=FALSE),], "cell_clust.xls",
row.names=FALSE, col.names=TRUE, sep="\t", quote=FALSE)
```

All parameters used by the method `findoutliers` are stored in the slot `sc@outlierpar`.

1.6 Analysis of final clusters

The function `clustdiffgenes` identifies differentially regulated genes for each cluster in comparison to the ensemble of all cells:

```
> cdiff <- clustdiffgenes(sc,pvalue=.01)
```

It returns a list with a `data.frame` element for each cluster that contains the mean expression across all cells not in the cluster (`mean.ncl`) and in the cluster (`mean.cl`), the fold-change in the cluster versus all remaining cells (`fc`), and the p-value for observing the difference in mean expression between the cluster and all other cells based on binomial counting statistics (`pv`). To compute this probability the median total transcript count in each cell not in the cluster was rescaled to match the original median transcript count of all cells in the cluster. This step ensures that the p-value is not artificially inflated, e. g., due to an over-representation of cells with low transcript counts in a given cluster. Only genes with `pv` lower than `pvalue` are shown. The table is ordered by increasing `pv`, i. e. the most significantly up- or down-regulated genes will appear at the top of the list.

```
> head(cdiff$cl.3,10)
                 mean.ncl   mean.cl          fc           pv
Apoa1__chr9     2.1121661 47.486690 22.48246000 0.000000e+00
Apoa4__chr9     1.5820320 33.207170 20.99020184 0.000000e+00
Fabp1__chr6     2.2516751 52.149391 23.16026435 0.000000e+00
Fabp2__chr3     3.9629854 66.574620 16.79910819 0.000000e+00
Plac8__chr5     4.4884693 31.312948  6.97630888 0.000000e+00
```

```
Reg1__chr6      0.6287205 23.736060 37.75295858 0.000000e+00
Fth1__chr19    11.6266793 47.778109  4.10935125 1.110223e-15
Rbp2__chr9      0.5775672 12.953565 22.42780462 7.394085e-14
Defa24__chr8   31.9903416  3.919583  0.12252394 7.241276e-11
AY761184__chr8 27.7818204  2.462489  0.08863669 3.407436e-10
```

The tables with differentially expressed genes can be written to tab-separated text files:

```
> for ( n in names(cdiff) ) write.table(
data.frame(GENEID=rownames(cdiff[[n]]), cdiff[[n]]), paste(
paste("cell_clust_diff_genes",sub("\\.","\\_",n), sep="_"), ".xls",
sep=""), row.names=FALSE, col.names=TRUE, sep="\t", quote=FALSE)
```

To specifically examine expression differences between two clusters or two sets of clusters, the function `diffgenes` can be used:

```
> d <- diffgenes(sc,cl1=1,cl2=c(2,3),mincount=5)
```

Input arguments are two vectors with sets cluster numbers, `cl1` and `cl2`, that should be compared, for instance, cluster 1 with the union of clusters 2 and 3. Only genes with more than `mincount` transcript in at least a single cell of `cl1` or `cl2` are evaluated.
The function computes the z-score for expression differences between the two groups using the average of the standard deviation in `cl1` and `cl2`. If `cl1` or `cl2` contains only a single cluster, the standard deviation is estimated by the squareroot of the mean. The function returns a list of five elements:

`z:` a vector of z-scores in decreasing order with genes up-regulated in `cl1` appearing at the top of the list.

`cl1:` a `data.frame` with expression values for cells in `cl1`.

`cl1n:` a vector of cluster numbers in `cl1`.

`cl2:` a `data.frame` with expression values for cells in `cl2`.

`cl2n:` a vector of cluster numbers in `cl2`.

The function `plotdiffgenes` can be used to plot expression in the two groups for a given gene (Figure 9).

```
> plotdiffgenes(d ,gene="Dmbt1__chr7")
```

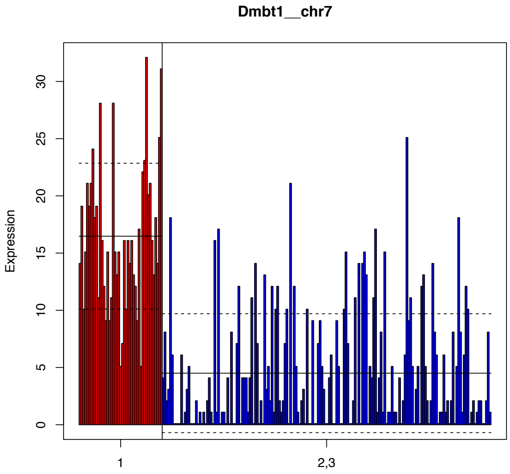

Figure 9. Expression profile of the gene *Dmbt1* in two sets of clusters (`cl1` and `cl2`) as computed by diffgenes. Cells in `cl1` and `cl2` are highlighted in red and blue, respectively. Mean expression and standard deviation are indicated by the black solid and broken lines, respectively.

The final clusters can also be plotted in a heatmap (Figure 9):

```
> x <- clustheatmap(sc,final=TRUE,hmethod="single")
```

Input arguments are explained in paragraph II. To plot the final clusters instead of the k-medoids clustering, the logical argument `final` has to be set to `TRUE`. The return vector x contains all cluster numbers in the same order as shown in the heatmap.

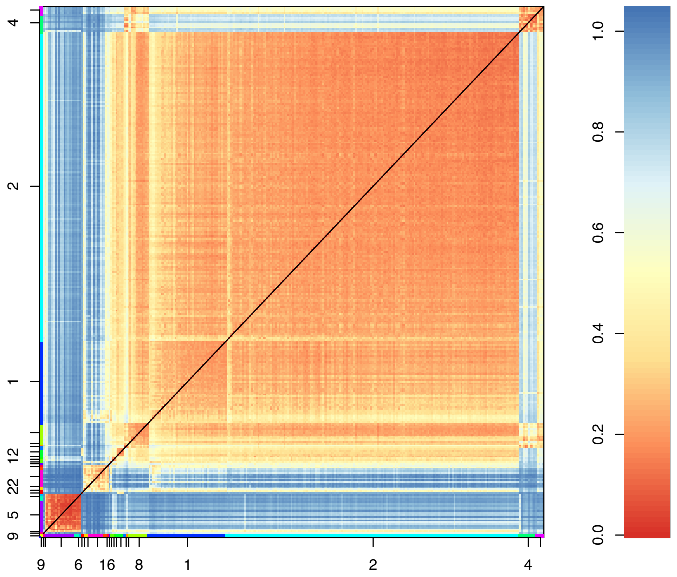

Figure 9. Heatmap of final clusters. Individual clusters are highlighted with rainbow colors along the x- and y-axes. Since not all cluster labels can be shown in the plot, the function `clustheatmap` returns a vector with all clusters in the same order as depicted in the heatmap.

1.7 Using t-SNE maps for cluster examination

In order to visualize the partition of the cell population into clusters, the t-Distributed Stochastic Neighbor Embedding (t-SNE) algorithm[6] can be used, implemented in the R package `tsne`. A t-SNE map reduces the dimension of the input data to a low dimensional space, in our case to a two-dimensional space, and preserves all pairwise distances between the data-points as good as possible.

A t-SNE map is computed by the following command:

```
> sc <- comptsne(sc,rseed=15555,sammonmap=FALSE,initial_cmd=TRUE)
```

The method is executed with a fixed seed for the random number generator to yield exactly reproducible maps across different runs. The seed can be varied by changing the numeric argument `rseed` (default is `15555`). If `sammonmap` is set to `TRUE` dimensional reduction is computed with the Sammon's map algorithm instead of t-SNE. If `initial_cmd` is set to `TRUE` the t-SNE map is initialized with point-to-point distances computed by classical multidimensional scaling instead of a random seed. A data.frame with the coordinates in the two-dimensional embedded space is stored in the slot `sc@tsne`.

A t-SNE map can now be drawn for the original cluster and for the final clusters

Original clusters:

```
> plottsne(sc,final=FALSE)
```

Final clusters (Figure 11):

```
> plottsne(sc,final=TRUE)
```

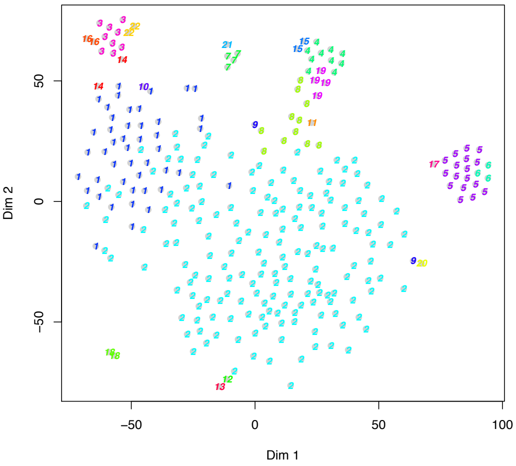

Figure 11. t-SNE map representation of all cells. All final clusters are highlighted in different colors and the cluster number is written on top of each cell.

The t-SNE map representation can also be used to analyze expression of a gene or a group of genes, to investigate cluster specific gene expression patterns (Figure 12):

```
> g <- c("Apoa1__chr9", "Apoa1bp__chr3", "Apoa2__chr1", "Apoa4__chr9", 
"Apoa5__chr9")
> plotexptsne(sc,g,n="Apoa genes",logsc=TRUE)
```

The function requires a vector `g` of gene identifiers (or a single gene identifier) and highlights the aggregated expression level of these genes by superimposing a color scale on the t-SNE map. With the argument `n` a title of the plot can be specified, which by default corresponds to the first element of `g`. If the logical argument `logsc` is set to `TRUE` the log-transformed transcript counts are highlighted.

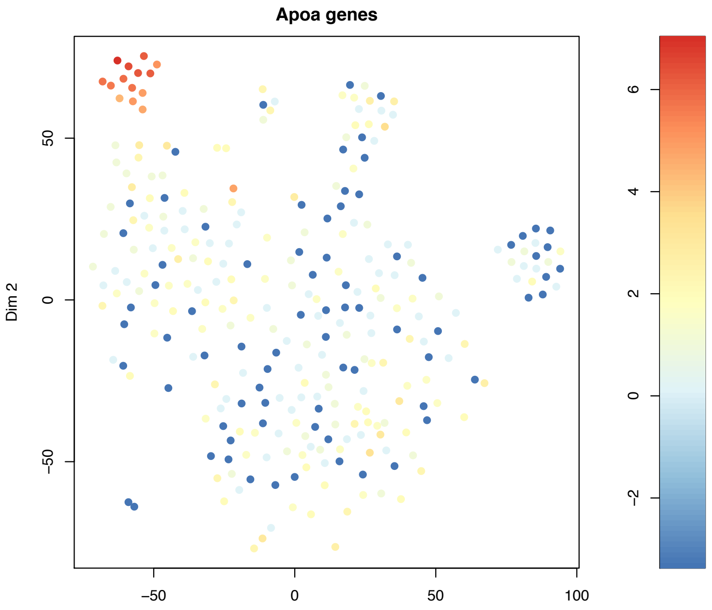

Figure 12. t-SNE map with color code representation of log-transformed gene expression aggregated across all Apoa genes.

The method `plotlabelstsne` allows plotting labels on top of each cell in the t-SNE map. By default, the column names of `sc@ndata`, i. e. the names of all single cells are plotted (Figure 13):

```
> plotlabelstsne(sc, labels=names(sc@ndata))
```

The argument labels can be any vector of labels with the same length as the column number of `sc@ndata`, `ncol(sc@ndata)`.

The method `plotsymbolstsne` can be used to label groups of cells in the t-SNE map by different symbols and colors (Figure 14).

```
> plotsymbolstsne(sc,types=sub("\\_\\d+$","", names(sc@ndata)))
```

For instance, cells from different replicates can be labeled if analyzed together. For this purpose a vector with common identifiers for all cells of a group has to be supplied as the `types` argument. For examples, if column names denote the experiment and a numeric identifier for the cell, a `types` vector can be created by a pattern substitution:

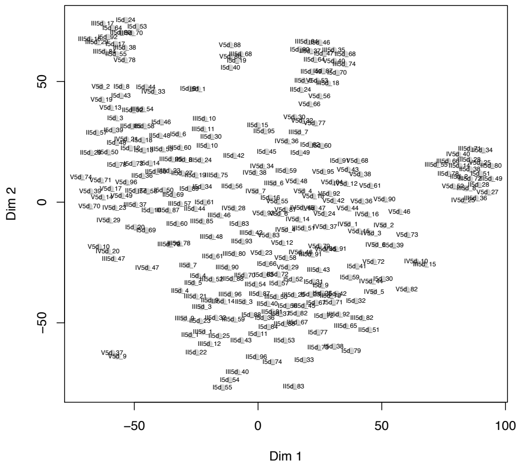

Figure 13. t-SNE map with labels corresponding to the column names of the transcript count data.

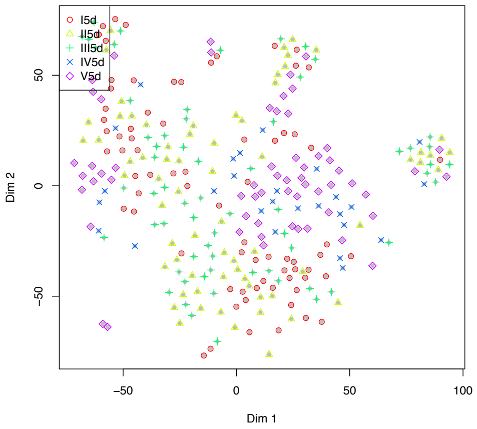

Figure 14. t-SNE map with symbols corresponding to different replicates analyzed together.

1.8 Remarks on running RaceID2

The RaceID2 algorithm can be run on single cell sequencing data produced by different techniques on material from arbitrary sources. Since the algorithm relies on statistics valid for absolute transcript counts, the sequencing method should incorporate unique molecular identifiers. The performance for read based expression quantification has not been tested.

In order to assess the performance of the method, the diagnostic plots have to be inspected carefully and it could be necessary to adjust parameter values.

It is crucial that the background model provides a reasonable fit to the data (Figure 6). If the variance of transcript counts is not a convex function of the mean (in logarithmic space) the background model will not yield a good fit and the algorithm cannot perform well. Moreover, it is assumed that a negative binomial provides a reasonable fit to the transcript count data for a given cell type and that expression of the majority of genes will not (strongly) depend on the cell type. All these assumption will most likely be valid for most systems and single cell sequencing techniques.

However, problems could arise if libraries with different complexities are combined and jointly analyzed by RaceID2. In this case, it is strongly advised to eliminate cell-to-cell differences in technical noise by using downsampling instead of normalization in the `filterdata` method by setting the argument `downsample` to `TRUE`. This could also be necessary if the analyzed sample comprises cells with highly variable total transcript count (several orders of magnitude). The minimum total transcript count `mintotal` as well as the minimum expression `minexpr` for a given number (`minnumber`) of cells should be chosen according to the sensitivity of the dataset. To obtain robust results `mintotal` should be at least of the order of ~1000. In general, only genes should be included, which are robustly expressed (>4 transcripts) in at least a single cell. More lowly expressed genes tend to introduce more noise and do not add statistical power. The parameter `maxepr` can be used to discard highly expressed genes that saturate the unique molecular identifiers (UMI). For these genes UMIs cannot be used to reliably infer transcript counts. The choice depends on the length of the UMI. For UMIs of length 4 genes with >500 transcripts per cell cannot be reliably quantified anymore[2].

The number of clusters in the initial k-medoids clustering step is determined based on the saturation of the within-cluster distance of points (Figure 1) or by the gap statistic (Figure 2). However, both statistics can be noisy and a clear saturation point or a pronounced first local maximum of the gap statistic, respectively, might not exist. In this case, one should choose a number of clusters by setting `cln` to a number above which no more jumps in the saturation behavior occur or where the gap statistic starts saturating, respectively. If the within-cluster dispersion does not saturate at the maximal number of clusters given by `be` , a larger value of this argument needs to selected to run the saturation statistics and the clustering using the `clustexp` function. The precise number will not be crucial, since the outlier identification step can introduce additional clusters. However, if the Jaccard's similarities (Figure 3) or the silhouette (Figure 4) of several clusters are low (Jaccard's similarity < 0.6 or silhouette < 0.1) the number of clusters should be lowered.

In the outlier identification step of the `findoutliers` method a crucial parameter is the probability threshold `probthr`, by default set to 10^{-3}. Visual inspection of the sensitivity plot (Figure 7) allows determining a reasonable value for this parameter. It should separate the broad tail from the initial steep decay of the distribution.

Additionally, the minimum number of outlier genes `outlg` can be varied. This parameter will also control the resolution. If only major cell types should be resolved that differ by a larger number of genes, `outlg` should be increased to higher numbers.

Another important parameter is the minimum transcript count `outminc` of a gene that has to be observed in at least a single cell in a cluster to include this gene in the statistical test. It will influence the size of the dataset (number of genes) and the multiple testing correction of the p-value. If a large number of outliers are detected, the stringency of the multiple testing correction can be increased by lowering this number.

For bug reports and any questions related to RaceID2 please email directly to gruen@ie-freiburg.mpg.de.

2 StemID

StemID is an algorithm based on RaceID2 for the inference of differentiation trajectories and the prediction of the stem cell identity. As an initial step, the algorithm embeds the space of transcript counts for each gene, in which every cell can be represented, into a lower dimensional space in order to maintain only the number of dimensions necessary to represent all point-to-point distances. For the Euclidean metric, only n-1 dimensions are necessary to embed n data points from a high dimensional space (>n dimensions) with exactly conserved distances. For a correlation-based metric as used by RaceID2 this is not true. Here, we embed into $k<n-1$ dimensions, with k being the number of positive eigenvalues of the squared double-centered distance matrix. The distance $d_{i,j}$ between cells i and j is defined as $d_{i,j} = 1 - \rho_{i,j}$, where $\rho_{i,j}$ equals Pearson's correlation coefficient of the transcriptome of these cells. The embedding is computed in R using the function cmdscale.

For the derivation of differentiation trajectories the medoid m_i of cluster i is connected to the medoids m_j of all other clusters j (j = 1, …, i–1, i+1, …, N) in the embedded space. Subsequently, for each cell k in cluster i the vector $z_{i,k} = y_{i,k} - m_i$ connecting its position $y_{i,k}$ to m_i is projected onto each link $l_{i,j} = m_j - m_i$ between cluster i and j (j = 1, …, i–1, i+1, …, N, i. e. the component of this vector (anti-)parallel to each connection is calculated. Projections $p_{k,i,j}$ correspond to the cosine of the angle $\alpha_{k,i,j}$ between $z_{i,k}$ and $l_{i,j}$ times the length of $l_{i,j}$ and are computed based on the dot product of the two vectors:

$$p_{k,i,j} = \left|l_{i,j}\right| \cdot \cos\alpha_{k,i,j} = \frac{z_{i,k} \cdot l_{i,j}}{\left|z_{i,k}\right|}$$

If the vector component is anti-parallel to a projection it will be negative. The respective cell is then assigned to the connection with the longest projection using the coordinate computed from the projection. This procedure is repeated for every cell in each cluster. To determine connections with significantly more assigned cells than expected by chance, the computation is repeated after randomizing the cell positions in the embedded space. Randomization is performed by sampling new cell positions from a uniform interval with boundaries given by the real data for each embedded dimension. Cluster centers are kept constant to maintain the topology of the configuration.

Outgoing and incoming links are distinguished for the p-value calculation, i. e. for each cluster it is computed how many of its cells are assigned to each link to another cluster. The distribution of expected cells on each outgoing link is sampled by repeating the randomization procedure 2,000 times. A p-value for each link is derived as the quantile of this distribution corresponding to the actual number of cells on the link. In general, a cluster can have an enriched outgoing link, which is at the same time a depleted incoming link. We consider a link significantly enriched if this is true for either the outgoing or the incoming link.

To compute a p-value, the sampling is repeated sufficiently often. For instance, if a p-value threshold of $P<0.01$ is chosen to assign significance to a link, the randomization is repeated 2,000 times to calculate the 1%-quantile with sufficient confidence. For lower p-values the number of randomizations needs to be increased.

The ensemble of significant connections can be interpreted as a predicted lineage tree comprising all commonly used differentiation trajectories of a system.

To predict the stem cell identity the algorithm also takes into account the transcriptome entropy of each cell. The entropy E_j of cell j is computed as

$$E_j = \sum_{i=1}^{N} p_{i,j} \log_N p_{i,j} ,$$

where $p_{i,j} = n_{i,j}/N$ and $n_{i,j}$ equals the number of transcripts of gene i in cell j. N equals the total number of transcripts in each cell, which is the same for all cells due to the downsampling (or median-normalization) performed by RaceID2. Next, the median delta-entropy ΔE_k is computed for each cluster k, defines as

$$\Delta E_k \equiv \text{median}_{j\in k}\left(E_j\right) - \min_l\left(\text{median}_{j\in l}\left(E_j\right)\right).$$

To predict the stem cell identity, StemID computes a score for each cluster k given by

$$s_k = l_k \cdot \Delta E_k ,$$

where l_i denotes the number of significant links of cluster j.

2.1 Initializing a lineage tree object

To run StemID it is required to first run RaceID2 on the dataset of interest. The algorithm is implemented as an `S4` class object, named `Ltree`. This object needs to be initialized

```
> ltr <- Ltree(sc)
```

2.2 Entropy

The entropy of each cell type, which is required to compute the StemID score, can be computed using the function `compentropy`.

```
> ltr <- compentropy(ltr)
```

The entropy of each cell is written to a vector stored in slot `ltr@entropy`.

2.3 Cell projections

The dimensionality reduction and the calculation of the projections of each cell onto all inter-cluster links are performed by the function `projcells`.

```
> ltr <- projcells(ltr,cthr=2,nmode=FALSE)
```

This function has two additional arguments:

`cthr:` Threshold for the minimum number of cells required to include a cluster into the lineage analysis. Only cluster with more than `cthr` cells are included. Default is `0`.

`nmode:` Boolean argument. If `nmode` is set to `TRUE` the assignment to inter-cluster links for each cell is not done based on the longest projection, but based on identifying the cluster (other than the cluster the cell belongs to) that contains the nearest neighbor of the cell, i. e. the cell with the most similar transcriptome. The coordinate on the assigned link is still derived based on the projection. Default is `FALSE`.

The output is written to two slots of the `ltr` object. The slot `ltr@ldata` contains a list of several elements:

`lp:` vector with the filtered partition into clusters after discarding clusters with `cthr` cells or less.

`pdi:` matrix with the coordinates of all cells in the embedded space. Clusters with `cthr` transcripts or less were discarded. Rows are medoids and columns are coordinates.

`cn:` data.frame with the coordinates of the cluster medoids in the embedded space. Clusters with `cthr` transcripts or less were discarded. Rows are medoids and columns are coordinates.

`m:` vector with the numbers of the clusters which survived the filtering.

`pdil:` data.frame with coordinates of cells in the two-dimensional t-SNE representation computed by RaceID2. Clusters with `cthr` transcripts or less were discarded. Rows are cells and columns are coordinates.

`cnl:` data.frame with the coordinates of the cluster medoids in the two-dimensional t-SNE representation computed by RaceID2. Clusters with `cthr` transcripts or less were discarded. Rows are medoids and columns are coordinates.

The other slot `ltr@trproj` contains a list of two data.frames:

`res:` data.frame with three columns for each cell. The first column `o` shows the cluster of a cell, the second `l` shows the cluster number for the link the cell is assigned to, and the third column `h` shows the projection as a fraction of the length of the inter-cluster link. Parallel projections are positive, while anti-parallel projections are negative.

`rma:` data.frame with all projection coordinates for each cell. Rows are cells and columns are clusters. Projections are given as a fraction of the length of the inter-cluster link. Parallel projections are positive, while anti-parallel projections are negative. The column corresponding to the originating cluster of a cell shows `NA`.

The randomization can subsequently be computed by applying the function `projback`.

```
> ltr <- projback(ltr,pdishuf=2000,nmode=FALSE,rseed=17000)
```

This function has three additional arguments:

`pdishuf:` positive integer. This is the number of randomizations to be performed. As a rule of thumb this number should be at least one order of magnitude larger than the desired p-value on the significance of the number of cells on a connection. Default is `2000`.

`nmode:` See `projcells`.

`rseed:` positive integer. This is the seed to initialize random number generation for the randomization of cell positions. Default is `17000`.

The results of the randomization are written to the slot `ltr@prback`. The slot contains a data.frame of the same structure as the data.frame `ltr@trproj$res`, which contains the maximal projections of the actual data. The projections of all `pdishuf` randomizations are appended to this data.frame and therefore the number of rows corresponds to the number of cells multiplied by `pdishuf`.
The slot `ltr@prbacka` is a data.frame reporting the aggregated results of the randomization. It has four columns. Column `n` denotes the number of the randomization sample, column `o` and `l` contain the numbers of the originating and the terminal cluster, respectively, for each inter-cluster link and column `count` shows the number of cells assigned to this link in randomization sample `n`. The discrete distribution for the computation of the link p-value is given by the data contained in `ltr@prbacka`.

2.4 Computing the lineage tree

Next, the function `lineagetree` has to be executed in order to assemble the lineage tree.

```
> ltr <- lineagetree(ltr,pthr=0.01,nmode=FALSE)
```

This function has two additional arguments:

`pthr:` positive number. This number corresponds to the p-value threshold, which is used to determine, whether the magnitude of an observed trajectory is significantly larger than observed for the randomized background distribution. This criterion is not used to infer significance of a link, but shown in a graphical representation of the tree (see below).

`nmode:` See `projcells`.

The output of this function is written to several slots. The slot `ltr@ltcoord` stores the projection coordinates of all cells in the two-dimensional t-SNE space and is used for graphical visualization. The slot `ltr@prtree` contains a list with two elements. The first element `l` stores a list with the projection coordinates for each link. The name of each element identifies the link and is composed of two cluster numbers separated by a dot. The second element `n` is a list of the same structure and contains the cell names corresponding to the projection coordinates stored in `l`. The slot `object@cdata` is a list with several elements. At this point only the first element `counts` is initialized, which contains a data.frame showing the number of cells on the links between each pair of clusters. The slot `ltr@sigcell` is a logical vector indicating for each cell, if the projection magnitude is significantly larger than for the randomized background distribution.

Finally, the function comppvalue has to be executed in order to identify connections at a given level of significance.

```
ltr <- comppvalue(ltr,pethr=0.01,nmode=FALSE)
```

The function has two additional arguments.

`pethr:` positive number. This number corresponds to the p-value threshold, which is used to determine for each link if it is populated by a number of cells significantly larger than expected for the randomized background distribution.

This p-value threshold determines, which connections are considered valid differentiation trajectories in the derived lineage tree.

`nmode:` See `projcells`.

The results of this function are written to the slot `ltr@cdata`, which is a list of several elements. The element `counts` has already been initialized by the function `lineagetree`. All other elements of this list have the same structure like `counts`. The element `counts.br` contains the cell counts on cluster connections averaged across the randomized background samples. Values of the element `pv.e` correspond to an enrichment p-value, and equal 0 if the observed number of cells on the respective link exceeds the (1 – `pethr`)-quantile of the randomized background distribution and 0.5 otherwise. Values of the element `pv.d` correspond to a depletion p-value, and equal 0 if the observed number of cells on the respective link is lower than the `pethr`-quantile of the randomized background distribution and 0.5 otherwise. The elements `pvn.e` and `pvn.d` are estimates of the 1-quantile or quantile, respectively, corresponding to the number of cells on a link as derived from the randomized background distribution. If `nmode` is set to `TRUE` all p-values are computed based on a binomial test.

2.5 Visualization of the lineage tree

The function `plotdistanceratio` can be used to plot a histogram of the ratio between the cell-to-cell distances in the embedded space and the input space.

```
> plotdistanceratio(ltr)
```

An example is shown in Figure 15.

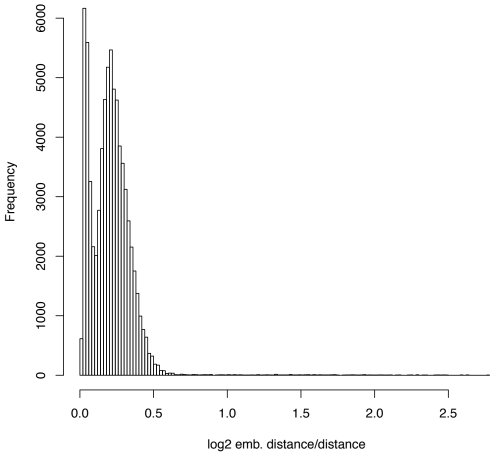

Figure 15. Histogram of the $\log_2$ratio between the cell-to-cell distances in the embedded space and the input space. Values close to zero indicate perfect conservation of the distances in the embedded space.

The function `plotmap` shows a t-SNE map of all cells similar to the one computed by RaceID2, but after discarding all clusters filtered out during the lineage tree inference, i. e. clusters with `cthr` cells or less.

```
> plotmap(ltr)
```

For comparison, the plot also depicts a minimum spanning tree connecting the cluster medoids computed based on the correlation-based distances between the cluster medoids. An example is shown in Figure 16.

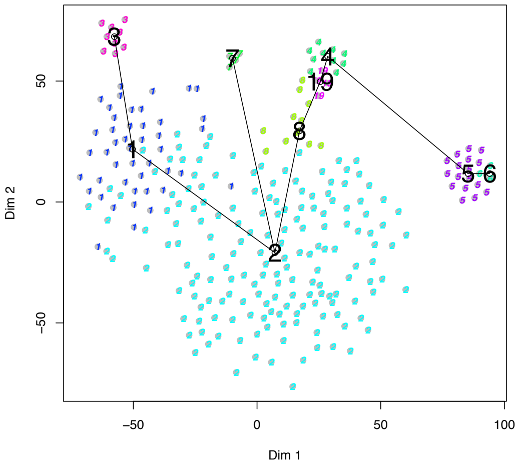

Figure 16. t-SNE map of all RaceID2 clusters with more than `cthr` cells. A minimum spanning tree connecting the cluster-medoids is shown in black.

The function `plotmapprojections` shows the projections of cells onto the inter-cluster links in t-SNE space. Projections are computed in the high-dimensional embedded space, but each cell is depicted in the t-SNE space using the relative coordinate on the inter-cluster link inferred from the projection.

```
> plotmapprojections(ltr)
```

For comparison, the plot also depicts a minimum spanning tree connecting the cluster medoids computed based on the correlation-based distances between the cluster medoids. An example is shown in Figure 17.

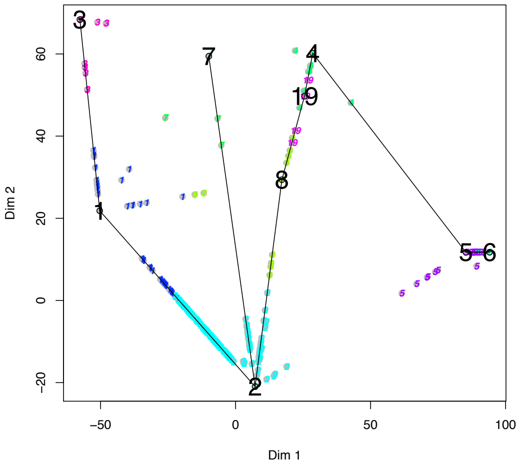

Figure 17. Projections of cells onto inter-cluster links visualized in t-SNE space. Cluster-medoids keep the positions computed by the t-SNE map RaceID2. Cluster with `cthr` cells or less are discarded. A minimum spanning tree connecting the cluster-medoids is shown in black.

The inferred lineage tree can be plotted with the function `plottree`.

```
> plottree(ltr,showCells=TRUE,nmode=FALSE)
```

It has two additional arguments:

`showCells:` logical. If `showCells` is set to `TRUE` single cells are shown in the plotted lineage tree. Otherwise, only the significant branches of the tree are shown. Default is `TRUE`.

`nmode:` See `projcells`.

`scthr:` positive number between 0 and 1. Only links are drawn with link score greater than `scthr`. The link score corresponds to 1 – fraction of link not covered by a cell. Link score close to zero correspond to a situation where most cells on a link reside close to the connected cluster center, while a score close to one arises in a situation where the link is covered uniformly with cells. At higher values of `scthr` only links are shown that represent higher confidence predictions for actual differentiation trajectories. Default is `0`.

Examples are shown in Figure 18.

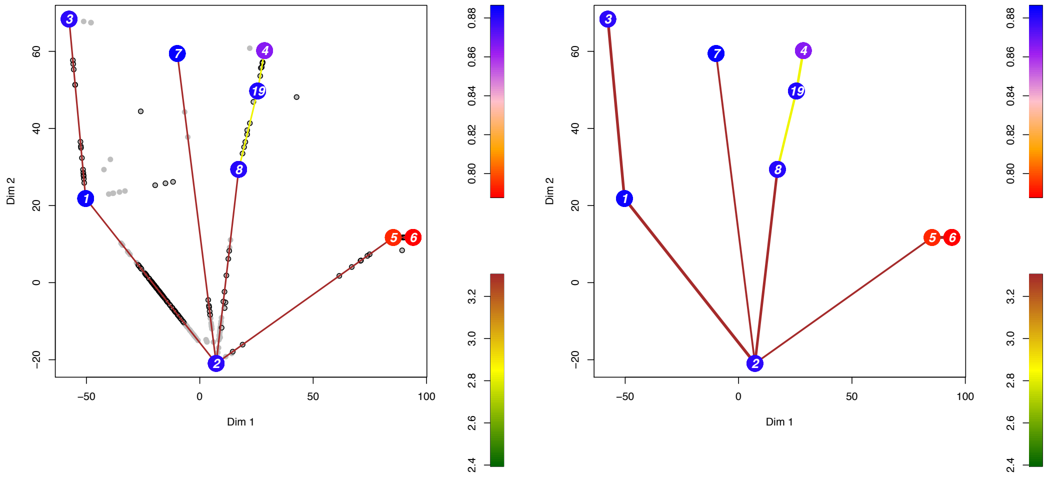

Figure 18. Examples of the lineage tree plotted with plottree for for `showCells=TRUE` (left) and `showCells=FALSE` (right). If `showCells=TRUE`, cells are shown as grey dots. If the projection magnitude is significantly larger than expected based on the randomized background distribution, cells are circled in black. In both plots, the color of the links indicates the –$\log_{10}$p-value of the link and the color of the vertices indicates the delta-entropy. The width of the connections in the right panel indicates the link score (line width corresponds to link score multiplied by 5, see Figure 21 and explanation below).

The function `plotlinkpv` can be used to plot a heatmap of enrichment p-values along clustering dendograms.

```
> plotlinkpv(ltr)
```

The plot shows the enrichment $\log_2$p-value computed as one minus the probability for the observed number of cells on a link based on the randomized background model. Rows correspond to outgoing links, i. e. enrichment of cells from the cluster corresponding to a row on the links to all other clusters represented by the columns. An example is shown in Figure 19.

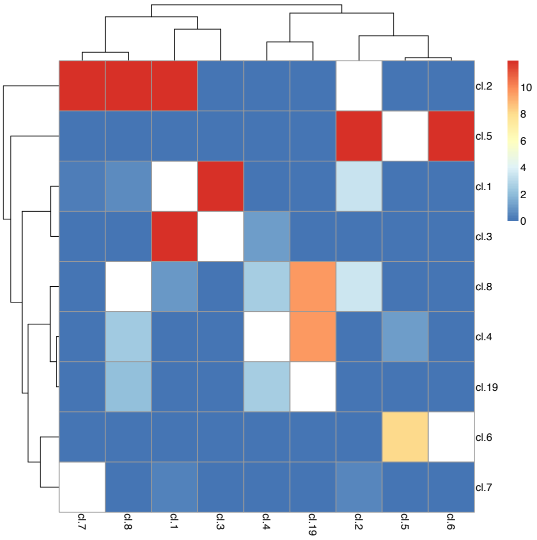

Figure 19. Two-dimensional clustering heatmap of enrichment log$_2$p-values.

The function `projenrichment` can be used to plot a heatmap of the fold change in cell number compared to the randomized background distribution.

```
> projenrichment(ltr)
```

The plot shows the log$_2$ratio between the number of cells on each link and the average number computed from the randomized background distribution. Only values for significantly enriched or depleted links are color-coded. Non-significant values are set to zero. Rows correspond to outgoing links, i. e. ratios of cells from the cluster corresponding to a row on the links to all other clusters represented by the columns. An example is shown in Figure 20.

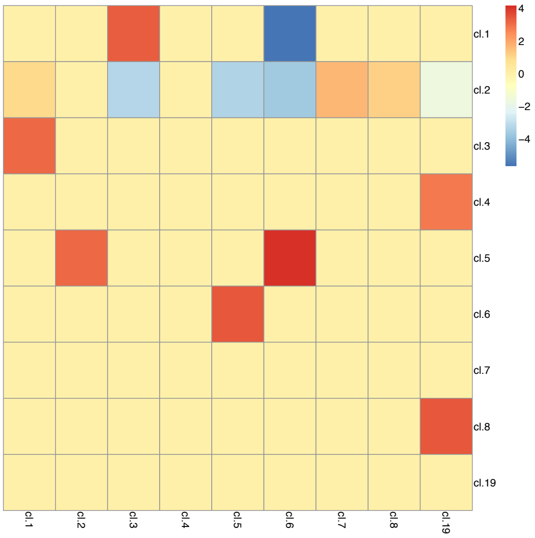

Figure 20. Heatmap showing the log$_2$ratio of the number of cells on the links from a cluster (row) to all other clusters (columns) and the corresponding number derived from the randomized background distribution.

The function `plotlinkscore` can be used to plot a heatmap reflecting the population of each significant link

```
> plotlinkscore(ltr)
```

The plot shows the link score. This number is calculated by taking the difference between the positions of each pair of neighboring cells on a link after rescaling the link length to one. The maximum difference on a link is then subtracted from one and this number is defined as the link score reflecting the coverage of a link. A high coverage indicates a higher likelihood that the link is an actual differentiation trajectory. An example is shown in Figure 21.

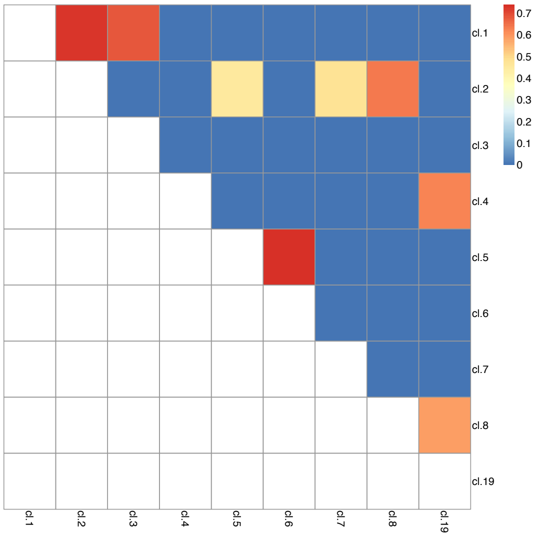

Figure 21. Heatmap showing the link score for each link between cell clusters.

2.6 Analysis of the lineage tree

The function `getproj` permits visual inspection of the differentiation trajectories. To extract the projections onto all connections for all cells of a given cluster i, the cluster number i has to be given as a second argument to `getproj`. For instance, the projection coordinates for all cells in cluster 1 can be extracted by the following command.

```
> x <- getproj(ltr,i=1)
```

The function returns a list of two elements. The element `pr` is a data.frame with the projections, where each row corresponds to a cell and columns correspond to the connected cluster. The element `prz` has the same structure, but shows projections after a z-score transformation in each row.

The projections can be visualized as a heatmap, for example, using the function `pheatmap`.

```
> pheatmap(x$prz)

> pheatmap(x$pr)
```

Examples are shown in Figure 22.

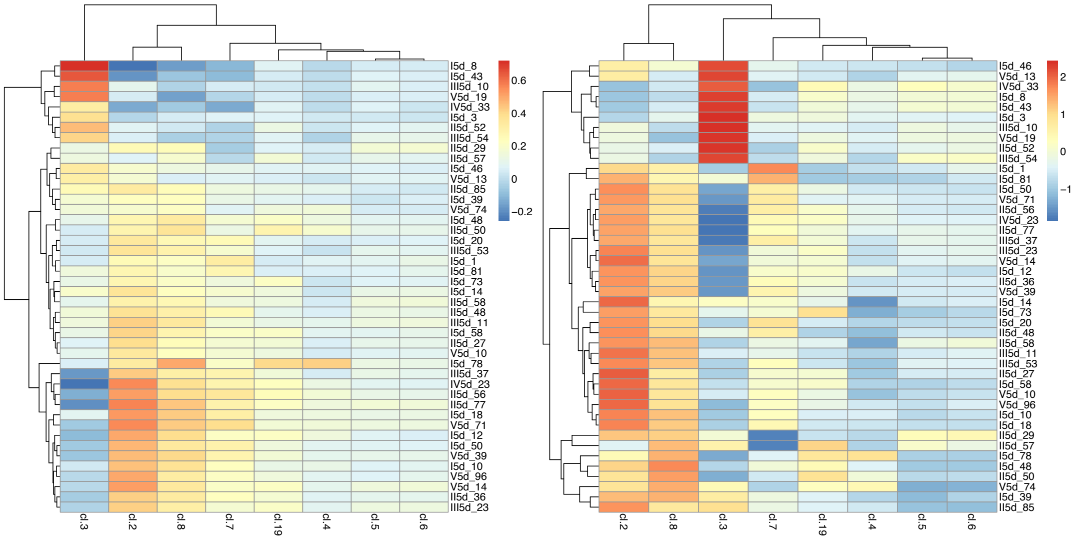

Figure 22. Two-dimensional clustering heatmaps of the projections (left) and the z-score of the projections (right).

The function `branchcells` can be used to find differences between cells of the same cluster residing on different links.

```
> x <- branchcells(ltr,br=list("1.3","1.2"))
```

The second argument `br` is a list of two inter-cluster links with one shared cluster (in the example cluster 1). A link is denoted by two cluster numbers in increasing order, separated by a period. The function returns a list of several elements. The element `n` is a list of two vectors containing the cells on the two branches belonging to the shared cluster. The second argument `scl` is an `SCseq` class element corresponding to the RaceID2 element `ltr@sc` with the only difference that additional cluster numbers are assigned to the cells on the two links in `br` that belong to the shared cluster. These cluster numbers were assigned counting onward from the largest RaceID2 cluster number and are stored in the return element `k`. The last element `diffgenes` stores the output of the RaceID2 `diffgenes` function performed on the two groups of cells from the shared cluster that were assigned to the two links. For example, `x$diffgenes$z` shows an ordered list of z-scores indicating the degree of up-regulation on link `"1.3"` compared to link `"1.2"` for cluster 1 cells.

```
> head(x$diffgenes$z)
```

As described in 1.6 the expression of these genes across the two groups can be plotted by the `plotdiffgenes` function. For example, the top up-regulated gene on the first branch `"1.3"` can be plotted by the command (Figure 22):

```
> plotdiffgenes(x$diffgenes,names(x$diffgenes$z)[1])
```

The cells from the two branches can be highlighted in a t-SNE map (see Figure 23).

```
> plottsne(x$scl)
```

They can be recognized by the additional cluster numbers stored in `x$k`.

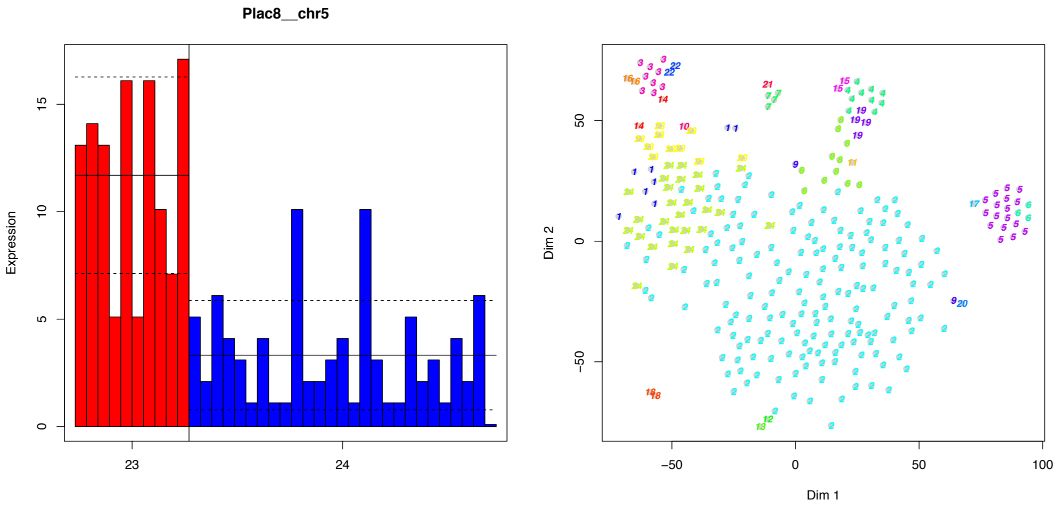

Figure 23. Differential gene expression between two branches. The novel cluster numbers in this example are 23 and 24 for cells on the branch `"1.3"` and `"1.2"`, respectively.

The StemID score is computed by executing the function `compscore`.

```
> x <- compscore(ltr,nn=1)
```

This function multiplies for each cluster the number of significant links to neighbouring clusters by the delta-entropy. The additional argument `nn` denotes the degree of neighborhood included for calculating the connectivity, i. e. up to the `nn`-th nearest neighbors are included. By default, (`nn`=1) only significant links to next nearest neighbors are counted. Increasing this argument allows analyzing the situation where a stem cell gives rise to different branches not directly but via a multipotent progenitor population. It is always recommended to inspect the behavior of the StemID score with increasing values of `nn`.

The StemID score can be plotted by the function `plotscore`, which has the same additional argument `nn` as `compscore` and calls this function.

```
> x <- plotscore(ltr,nn=1)
```

The function plots three histograms: the number of links, the delta-entropy and the StemID score. An example is shown in Figure 24.

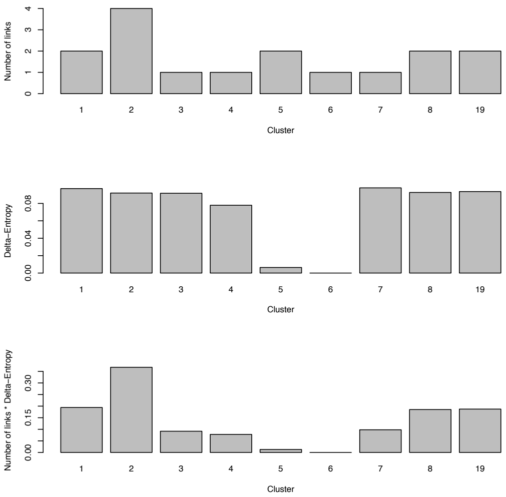

Figure 24. Histograms for the number of links, the delta-entropy, and the StemID score.

For bug reports and any questions related to StemID please email directly to gruen@ie-freiburg.mpg.de.

References

1. Grün, D. *et al.* Single-cell messenger RNA sequencing reveals rare intestinal cell types. *Nature* **525,** 251–5 (2015).

2. Grün, D., Kester, L. & van Oudenaarden, A. Validation of noise models for single-cell transcriptomics. *Nat. Methods* **11,** 637–40 (2014).

3. Jaitin, D. A. *et al.* Massively Parallel Single-Cell RNA-Seq for Marker-Free Decomposition of Tissues into Cell Types. *Science (80-.).* **343,** 776–779 (2014).

4. Islam, S. *et al.* Quantitative single-cell RNA-seq with unique molecular identifiers. *Nat. Methods* **11,** 163–6 (2014).

5. Tibshirani, R., Walther, G. & Hastie, T. Estimating the number of clusters in a data set via the gap statistic. *J. R. Stat. Soc. Ser. B (Statistical Methodol.* **63,** 411–423 (2001).

6. Van der Maaten, L. & Hinton, G. Visualizing Data using t-SNE. *J. Mach. Learn. Res.* **9,** 2570–2605 (2008).