Chapter 13

Lineage Inference and Stem Cell Identity Prediction Using Single-Cell RNA-Sequencing Data

Sagar and Dominic Grün

Abstract

With the advent of several single-cell RNA-sequencing (scRNA-seq) techniques, it has become possible to gain novel insights into the fundamental long-standing questions in biology with an unprecedented resolution. Among the various applications of scRNA-seq, (1) discovery of novel rare cell types, (2) characterization of heterogeneity among the seemingly homogenous population of cells described by cell surface markers, (3) stem cell identification, and (4) construction of lineage trees recapitulating the process of differentiation remain at the forefront. However, given the inherent complexity of these data arising from the technical challenges involved in such assays, development of robust statistical and computational methodologies is of major interest. Therefore, here we present an in-house state-of-the-art scRNA-seq data analyses workflow for de novo lineage tree inference and stem cell identity prediction applicable to many biological processes under current investigation.

Key words Single-cell RNA sequencing, Single-cell data analysis, Stem cell identification, Lineage inference, Pseudo-temporal ordering, Fate bias, Multipotent, RaceID3, StemID2, FateID

1 Introduction

During embryonic development as well as adult tissue homeostasis and regeneration, cells move from an immature naïve state to a mature functional state through a series of complex transcriptional changes governed by various cell-intrinsic and cell-extrinsic factors. Quantification of genome-wide transcription of individual cells using scRNA-seq provides a unique opportunity to capture and characterize distinct cell states occurring during such biological processes. Moreover, since the process of differentiation is not completely synchronous in complex multicellular organisms, single-cell expression data often comprises of not only immature and mature cell types but also transient cell types with unique transcriptional signature representing the intermediate states of cellular differentiation. Therefore, statistical methods and computational tools de novo or in combination with prior biological

Patrick Cahan (ed.), *Computational Stem Cell Biology: Methods and Protocols*, Methods in Molecular Biology, vol. 1975, https://doi.org/10.1007/978-1-4939-9224-9_13,

knowledge can be used to predict the identity of stem cells and infer the lineage tree by aligning the intermediate cell states along a trajectory, thereby tracking the global gene expression changes occurring during the process of differentiation.

During the last few years, several methods have been developed for lineage inference and stem cell identification from scRNA-seq data [1–7]. Here we describe our in-house step-by-step workflow of relevant aspects of previously published RaceID3, StemID2, and FateID algorithms including the sample R-codes for de novo lineage tree construction and stem cell identity prediction, thereby providing important tools to facilitate scRNA-seq-driven discoveries.

2 Materials

The analysis workflow described in this protocol is implemented in R. Analyses can be performed either on a private workstation or compute servers [8].

1. R (https://www.r-project.org/) and RStudio (optional, https://www.rstudio.com/).
2. RaceID3 and StemID2 algorithms (https://github.com/dgrun/RaceID3_StemID2). The link contains two R files: RaceID3_StemID2_class.R and RaceID3_StemID2_sample.R as well as an example data in .xls format.
3. FateID algorithm available as an R package through CRAN package repository.

3 Methods

This section provides a step-by-step workflow using a previously published dataset where 5-day-old progeny of *Lgr5*$^{+}$ mouse intestinal cells was sequenced [9]. In order to identify stem cells in the scRNA-seq data, first robust identification of all cell types in the dataset is necessary. This is done by applying the RaceID3 algorithm, which performs k-medoids clustering and a subsequent outlier identification step to recover all cell types, including the rare ones. Afterward, the StemID2 algorithm is used to construct a lineage tree. The StemID2 algorithm is also used to identify the putative stem/progenitor cell cluster(s) in the dataset. Additionally, based on the StemID2-predicted differentiation trajectory, cells in the dataset can be arranged from the progenitor state to the differentiated state, thereby constructing a pseudo-temporal profile of cells recapitulating the process of differentiation [8]. This approach provides an important informative tool to visualize the dynamic

transcriptional changes of known and unknown genes occurring during differentiation. Furthermore, we also present another independent approach based on our recently developed FateID algorithm to quantify the level of multipotency based on the entropy level of fate bias. Note that we discuss only the most relevant aspects of our analysis pipeline in this protocol due to space constraints and users are advised to go through the manuals available online to explore the full functionality of our computational tools.

3.1 Package Installation and Data Preprocessing

1. After installing R, install various R packages required for the proper functioning of the RaceID3 and StemID2 algorithms using the following R code:

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

2. Additionally, install the following bioconductor packages, scran, DESeq2, and biomaRt, using the following command:

```
source("https://bioconductor.org/biocLite.R")
biocLite("scran")
biocLite("DESeq2")
biocLite("biomaRt")
```

3. Create a directory of your desired name, and copy the following downloaded files using the link https://github.com/dgrun/RaceID3_StemID2 into it:

```
RaceID3_StemID2_class.R,
RaceID3_StemID2_sample.R and
transcript_counts_intestine_5days_YFP.xls
```

4. Load the Rscript `RaceID3_StemID2_class.R` containing all functions required to run the algorithm implemented as an S4 class object `SCseq` using the following command:

```
source("RaceID3_StemID2_class.R")
```

5. Load the example data `transcript_counts_intestine_5days_YFP.xls` as a data frame object in the working environment using the following commands (*see* **Note 1**):

```
expr.data <-
read.csv("transcript_counts_intestine_5days_YFP.xls",
sep="\t", header=TRUE)
rownames(expr.data) <- expr.data$GENEID
```

```
expr.data <-
expr.data[grep("ERCC",rownames(expr.data),invert=TRUE),-
1]
```

6. Create an `SCseq` object called `sc` using the `expr.data` data frame using the following command: `sc <- SCseq(prdata)`.

3.2 Data Normalization and Removal of Unwanted Variability

After having the data in the correct format and an `SCseq` object being created, the next steps are performed by calling the function `filterdata`. It has several important arguments for data normalization and removal of unwanted variability. The steps below provide a brief explanation of the most important arguments.

1. `mintotal` defines the minimum total number of transcripts per cell. Only cells with a total number of transcripts greater than `mintotal` are included in the analysis. It is quite likely to have a few cells displaying very low number of total transcripts due to various technical reasons. Such cells should be discarded from the further downstream analysis.
2. Data normalization can be achieved using two different methods – downsampling and rescaling. If `downsample` is set to `TRUE`, the number of transcripts in each cell will be downsampled to a value of `mintotal`. Rescaling is done by dividing the number of transcript for each gene per cell by the total number of transcripts detected in this cell followed by multiplication with the minimum of the total number of transcripts across all cells present in the analysis after removing cells having less than `mintotal` number of transcripts (*see* **Note 2**).
3. In order to avoid handling zeros present in the expression matrix for further calculations, e.g., log transformation of the expression matrix, a pseudocount of 0.1 is added to the expression data. The data after normalization and pseudocount addition is saved in the `sc@ndata` slot.
4. `minexpr` and `minnumber` arguments define the minimum number of transcript of a particular gene in a cell and minimum number of cells, respectively. Only if a gene is expressed at a value greater than `minexpr` in at least `minnumber` of cells, it is included in the clustering analysis. The expression data containing the remaining genes with their corresponding transcript counts in all the cells included in the analysis is saved in the `sc@fdata` slot.
5. In order to remove further genes, which may artificially influence the clustering (such as analyzing cells from a transgenic mouse, which overexpress a particular gene), the argument `FGenes` can be initialized with gene names whose exclusion from clustering is desired (*see* **Note 3**).

6. Furthermore, instead of excluding a single gene or set of genes using `FGenes`, if removal of all the genes correlating with certain gene(s) is necessary, then argument `CGenes` can be initialized. `CGenes` removes all the genes positively or negatively correlated with gene(s) provided in `CGenes` argument with a threshold defined by the argument `ccor` (*see* **Note 4**).
7. The following R-code can used to initialize all the parameters for data normalization and filtering steps:

```
sc <- filterdata(sc, mintotal=3000, minexpr=5,
minnumber=1, maxexpr=Inf, downsample=FALSE, sfn=FALSE,
hkn=FALSE, dsn=1, rseed=17000, CGenes=NULL,
FGenes=NULL, ccor=.4)
```

8. RaceID3 provides two optional functions to remove unwanted variability: `CCcorrect` and `varRegression`. To use `CCcorrect`, any GO annotation, for example, "cell cycle" and "cell proliferation," can be retrieved using the bioconductor package `biomaRt`. The associated variability to the gene sets is removed using a principal component analysis (PCA) or an independent component analysis (ICA) using argument `mode`. The following commands can be executed to remove the variability associated with "cell cycle" and "cell proliferation" using `biomaRt`:

```
require(biomaRt)
g <- sub("__chr.+","",rownames(sc@fdata))
k <- getBM(attributes = c("external_gene_name",
"go_id", "name_1006"),
filters = "external_gene_name", values = g, mart = mart)
gCC <- name2id( k$external_gene_name[k$name_1006 == "cell
cycle"],rownames(sc@fdata))
gCP <- name2id( k$external_gene_name[k$name_1006 == "cell
proliferation"],rownames(sc@fdata))
vset <- list(gCC,gCP)
x <- CCcorrect(sc@fdata, vset=vset, CGenes=NULL, ccor=.4,
nComp=NULL, pvalue=.05, quant=.01, mode="pca")
```

The function returns a list consisting of three objects. `x$n` returns the number of components that have been removed, `x$pca` contains all the information associated with the PCA analysis, and `x$cor` contains the corrected expression data which has to be used to overwrite the `sc@fdata` slot using the following command:

```
sc@fdata <- x$xcor
```

9. RaceID3 also offers an optional function `varRegression` to remove batch-associated variability through linear regression. The following command can be used to remove batch-associated variability in the given dataset:

```
sc@fdata <- varRegression(sc@fdata,vars)
```

The argument `vars` defines a data frame having batch information as columns and cell names/identification codes as rows. For example, for the given mouse intestinal dataset, the following command can be used to extract the batch information:

```
vars <- data.frame(row.names=names(sc@fdata),
batch=sub("(_|)\crd.+","",names(sc@fdata)))
```

3.3 k-Medoids Clustering, Outlier Identification, and Data Visualization

Once the expression data is normalized and filtered as well as any unwanted variability, if required, is removed, the next step involves a robust clustering of the expression data to identify different cell types. This is achieved using k-medoids clustering. However, standard clustering approaches such as k-means clustering and k-medoids clustering require a previous knowledge of the number of clusters that can be inferred from the dataset, thereby preventing them to identify rare cell types. To circumvent this problem, RaceID3 algorithm includes an extra outlier identification step to identify rare cell types after the initial k-medoids clustering is performed. The outlier identification step is based on the transcript count distribution for each gene in a given cluster. Such distribution is assumed to be negative binomial governed by two parameters: the mean and the dispersion. While the mean is directly calculated as the average expression of the given gene across all the cells in a given cluster, the dispersion parameter is derived by fitting a second-order polynomial regression on the variance-mean relationship for each gene in the whole dataset. Based on this fit, for each gene a probability value is calculated for the observed transcript counts in each cell of a given cluster. If the transcript count of `x` number of genes in a cell have a probability value less than `p`, which is multiple testing corrected by the Benjamini-Hochberg method (across cells), this cell is flagged as an outlier. `x` and `p` can be user-defined keeping certain guidelines in consideration described below. Finally, the outlier cells are merged based on their similarity to each other. After the outlier cells are merged, new cluster centers are defined for the original clusters after removing the outliers and for the newly identified outlier clusters. Subsequently, each cell is assigned to the nearest cluster center. The final clusters are given numerical values in an increasing order. Thereafter, the visualization of cell clusters is required. This goal is achieved

via reducing the dimensions of the expression data into two-dimensional space using the t-Distributed Stochastic Neighbor Embedding (tSNE) algorithm, which preserves the local structure of the higher-dimensional data into the two-/three-dimensional space [10]. This section describes the step-by-step workflow of clustering, outlier identification, and data visualization using tSNE in detail.

1. k-medoids clustering is performed using `clustexp` function using the following R-code:

```
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",
FSelect=TRUE)
```

   The next steps describe the important arguments of `clustexp` function in detail.

2. To perform k-medoids clustering, the expression data saved in `sc@fdata` object is transformed into a distance matrix. The distance matrix is calculated as `1-correlation matrix`, with a cell-to-cell correlation matrix. The method to calculate correlation can be changed using the `metric` function. The default method is `pearson` but can be changed to `spearman`, `logpearson`, `euclidean`, `kendall`, `maximum`, `manhattan`, `canberra`, `binary`, or `minkowski`. The resulting distance matrix is saved in `sc@distances` slot.

3. As mentioned previously, since k-medoids clustering requires the number of clusters to be defined beforehand, the next important step is to determine the number of clusters defining the dataset, which in theory should reflect the number of abundant cell types captured during the scRNA-seq experiment. The algorithm is designed to determine the cluster number using two different approaches: within-cluster dispersion and gap statistic (*see* **Note 5**). By default, the cluster number is determined by the saturation of the average within-cluster dispersion. The cluster number is determined such that adding more clusters only leads to a linear decrease of the within-cluster dispersion. The logarithmic within-cluster dispersion and the change in logarithmic within-cluster dispersion can be visualized as a function of the cluster number using the following two commands, respectively (Fig. 1a, b):

```
plotsaturation(sc,disp=TRUE) and
plotsaturation(sc,disp=FALSE)
```

4. If the saturation of the within-cluster dispersion exhibits further gains at larger cluster numbers (gray dots far above the red

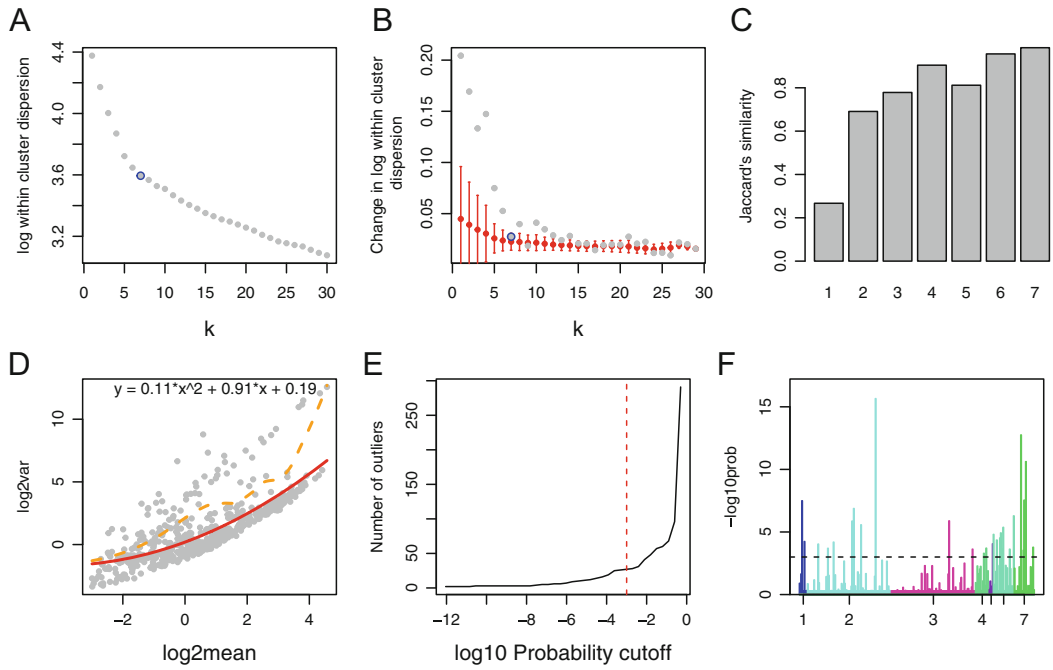

Fig. 1 k-Medoids clustering and outlier identification. (**a**) Log-transformed within-cluster dispersion and (**b**) change in it as a function of the number of clusters. (**c**) Jaccard's similarity bar plot of identified clusters. (**d**) Second-order polynomial regression of the variance on the mean of transcript expression across all cells in logarithmic space. (**e**) Number of outliers as a function of the probability threshold. (**f**) Outlier probabilities of all cells across clusters

error bars in Fig. 1b), 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.

5. Further, to assess the robustness of the inferred clusters, there are two methods implemented in the algorithm: Jaccard's similarity and silhouette plot. In this protocol chapter, only Jaccard's similarity score is explained. To plot a bar chart depicting Jaccard's similarity score for each cluster, use the following command (Fig. 1c): `plotjaccard(sc)`.
6. Jaccard's similarity score is proportional to the stability of the clusters. Stable clusters should have a score 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 manually setting the argument `cln` to the desired number.
7. After k-medoids clustering is performed, the unique feature of the RaceID3 algorithm enables the identification of outliers in the given scRNA-seq dataset. As mentioned before, the outlier identification step consists of three steps—calibration of a

background model, calculation of probability value for each gene in every cell of a given cluster based on which a cell is identified as an outlier, and the final clustering of the outlier cells based on their similarity. These steps are executed using the following R-code:

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

`outminc` defines the minimum transcript count for the identification of outlier genes, `outlg` is the minimum number of genes required to classify a cell as an outlier, `probthr` is outlier probability threshold only below which a cell is considered an outlier for a given gene, and `outdistquant` defines the quantile of the distance distribution for all pairs of cells within the original clusters after outlier removal. Outlier cells having distances lower than the `outdistquant` value are grouped together in the outlier clusters.

8. After executing `findouliers` with desired parameters, the quality of the background model fit can be visualized by plotting the second-order polynomial regression of the variance on the mean in logarithmic space using the following command (Fig. 1d): `plotbackground(sc)`. Furthermore, the number of outliers as a function of probability values can be plotted using (Fig. 1e) `plotsensitivity(sc)`. To ensure maximum sensitivity of the method, it is recommended to choose a `probthr` value, which covers the tail of this distribution as outliers. To visualize the outlier probabilities of all cells in the dataset as a bar plot, the following command can be used (Fig. 1f): `plotoutlierprobs(sc)`.
9. After the optimization of clustering including the identification of rare cells/clusters using RaceID3, the high-dimensional expression data need to be reduced to two dimensions for cluster visualization. To achieve this, RaceID3 implements tSNE algorithm using the following command:

```
sc <- comptsne(sc,rseed=15555,sammonmap=FALSE,initial
_cmd=TRUE)
```

10. After the tSNE map is computed, it can be visualized by the following two commands: `plottsne(sc,final=FALSE)` and `plottsne(sc,final=TRUE)`. If `final` is set to `FALSE`, the original k-medoids clusters are shown, whereas if it is set to `TRUE`, the outlier clusters/cells are also depicted in the tSNE map representation (Fig. 2a, b).

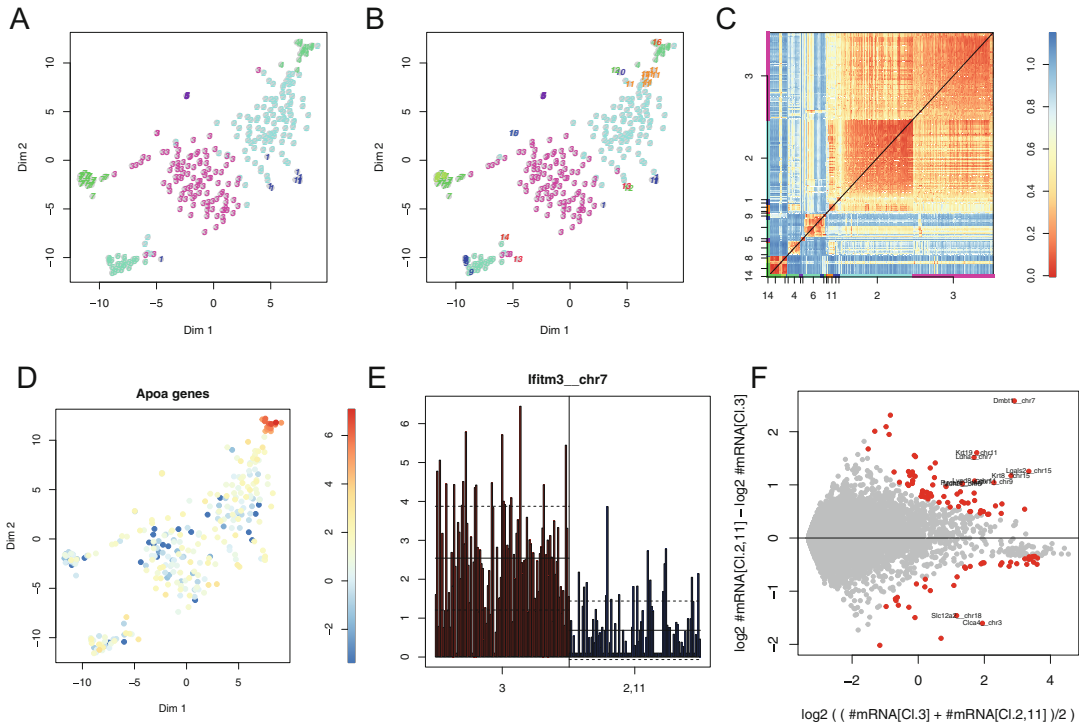

Fig. 2 Visualization and analysis of final clusters. tSNE map representation (**a**) before and (**b**) after outlier identification. (**c**) Heatmap representation of final clusters based on the cell-to-cell distance matrix. (**d**) tSNE map showing the log-transformed expression of *Apoa* genes in the dataset. (**e**) Expression of *Ifitm3* gene in cluster 3 versus clusters 2 and 11 calculated by `diffgenes` function. (**f**) MA plot showing differentially expressed genes between cluster 3 and 2 and 11 calculated by the function `diffexpnb`

11. The clusters can also be plotted in a heatmap representation of the cell-to-cell distances using the following command (Fig. 2c):

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

If `final` is set to `FALSE`, only k-medoids clusters prior to outlier identification are plotted (not shown).

12. In order to visualize the number of transcripts of a gene or set of genes on the tSNE map representation, the following command can be used:

```
plotexptsne(sc,g, logsc=FALSE)
```

`g` represents a character vector containing gene name similar to the row names of the expression matrix. If `logsc` argument is set to `TRUE`, the number of transcripts is

log-transformed. The expression pattern of, e.g., all *Apoa* genes can be plotted using the following command (Fig. 2d):

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

3.4 Gene Expression Analysis of the Clusters

In order to identify genes characterizing a cluster, there are three functions provided in the RaceID3 algorithm. They are named `clustdiffgenes`, `diffgenes`, and `diffexpnb`. Below we provide an explanation of each one of them and the corresponding arguments they require for their proper execution.

1. The function `clustdiffgenes` compares each cluster with all the other clusters present in the dataset to find the differentially expressed genes and can be executed using the following code: `cdiff <- clustdiffgenes(sc,pvalue=.01)`. It returns a list containing a data frame for each cluster, consisting of the following five columns: `mean.ncl` (the mean expression of all cells not in the cluster), `mean.cl` (the mean expression of all cells in the cluster), `fc` (fold-change: ratio of `mean.cl` and `mean.ncl`), `pv` (*p*-value for differential expression between all cells in a cluster and all remaining cells), and `padj` (the Benjamini-Hochberg corrected false discovery rate). Only genes with `pv` lower than the argument `pvalue` are included in the output.
2. The tables containing information about the differentially expressed genes can be exported as `.xls` files using the following command:

```
for ( n in names(cdiff) ) write.table( data.frame(GENEI-
D=rownames(cdiff[[n]]), cdiff[[n]]), paste( paste("cell_-
clust_diff_genes",sub("\cr.","\cr_",n), sep="_"), ".xls",
sep=""), row.names=FALSE, col.names=TRUE, sep="\t", quo-
te=FALSE)
```

3. In order to specifically examine expression differences between two clusters or two sets of clusters, the function `diffgenes` can be used using the following command:

```
d <- diffgenes(sc,cl1=3,cl2=c(2,11),mincount=1)
```

The arguments `cl1` and `cl2` are numerical vectors defining the cluster numbers to be compared. For instance, in the above code, we compare cluster 3 against clusters 2 and 11 in the example dataset. 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 cell, the standard deviation is estimated by the square root of the mean. The function returns a list of five elements: (1) z, a vector of z-scores in decreasing order with genes upregulated in cl1 appearing at the top of the list; (2) cl1, a data frame with expression values for cells in cl1; (3) cl1n, a vector of cluster numbers in cl1; (4) cl2, a data frame with expression values for cells in cl2; and (5) 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 (here first gene with the highest z-score, Fig. 2e) using the following command:

```
plotdiffgenes(d,gene=names(d$z)[1])
```

The last function `diffexpnb` provides a tool to perform differential expression analysis between two groups. This function implements an approach similar to DESeq [11] by which negative binomial distributions are fitted to transcript counts in two populations, representing, for example, two different cell types. These two groups can also be two different clusters identified by RaceID3. The code to execute the function is as follows:

```
x <- diffexpnb(sc@ndata, A=A, B=B, method="pooled"
,norm=FALSE, DESeq=FALSE, vfit=sc@background$vfit,
locreg=FALSE)
```

In the code above, we utilize the background model generated in the RaceID3 run which is saved in `sc@background $vfit` slot. As input, the normalized expression data frame `sc@ndata` is used. The function has the following other input arguments: (1) A, vector of cell IDs corresponding column names of x, and (2) B, vector of cell IDs corresponding column names of x. For example, to find the differentially expressed genes in cluster 3 versus clusters 2 and 11, A and B argument of the `diffexpnb` function can be initialized with the following command:

```
A <- names(sc@cpart)[sc@cpart == 3]
B <- names(sc@cpart)[sc@cpart %in% c(2,11)]
```

The results can be visualized as an MA plot using the following function (Fig. 2f): `plotdiffgenesnb(x, pthr=.05, lthr=1, mthr=1, show_names=TRUE, padj=TRUE)`.

3.5 Lineage Inference and Stem Cell Identity Prediction

After applying RaceID3 to identify different cell types and characterizing the gene expression patterns, StemID2 can be applied for the inference of differentiation trajectories and the prediction of the stem cell identity. StemID2 starts with embedding the higher-dimensional 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 reliably represent all point-to-point distances. For example, only $n - 1$ dimensions are necessary to embed n data points from a high-dimensional space ($>n$ dimensions) with exactly conserved distances for a Euclidean metric. Since RaceID3 uses a correlation-based metric, this task is achieved by embedding this correlation matrix 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 $\rho_{i,j}$ is the Pearson's correlation coefficient of these cells calculated based on the transcriptome. The major improvement of StemID2 over the first version is a fast mode (*see* **Note 6**), which allows running the entire algorithm on large datasets in substantially reduced computational time at a cost of a minor loss in sensitivity.

To derive the differentiation trajectories, the medoid m_i of cluster i is connected to the medoids m_j of all other clusters j ($j = 1, \ldots, i-1, i+1, \ldots, N$) 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 ($j = 1, \ldots, i-1, i+1, \ldots, N$), 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:

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

If the vector component is antiparallel 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 many times (*see* below). 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 2000 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 from the lineage tree, StemID2 assumes stem cells to be multipotent and tries to identify the cell population with highest degree of multipotency [12]. For this purpose, the algorithm computes a score called StemID2 score, s_k, for each cluster using the following formula:

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

where l_k defines the number of significant links of cluster k and ΔE_k is the median delta-entropy. The first parameter, l_k, takes into account the number of links each cluster has to the other clusters. The higher the number of links, the more likely it is that such a cell type can fluctuate toward a diversity of cell fate biases, whereas a cell type with fewer links is much more canalized. The second parameter, ΔE_k, deals with the uniformity of transcriptome as calculated by Shannon's entropy [13]. Here we hypothesize that stem cells possess higher uniformity in their transcriptome, whereas mature cells express relatively smaller number of genes albeit at a higher level which are important for their cell type-specific functions. For example, pancreatic beta cells express high levels of insulin.

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. Next, the median delta-entropy ΔE_k is computed for each cluster k, using the following formula:

$$\Delta E_k = \text{median}_{j \in k}(E_j) - \min_l(\text{median}_{j \epsilon l}(E_j))$$

Below we provide a step-by-step protocol with the R-code to construct a lineage tree and predict the cluster with higher multipotency, thereby reflecting its stemness.

1. In order to use StemID2, it is required to run RaceID3 before. Like RaceID3, StemID2 is also implemented as an S4 class object named `Ltree` and requires the `SCseq` object, `sc`, to be created previously (*see* Subheading 3.1). The following command can be used for initialization: `ltr <- Ltree(sc)`.
2. The entropy of each cell is calculated using the function `compentropy`, and the following command can be executed for this calculation: `ltr <- compentropy(ltr)`.
3. The embedding of the correlation matrix into a lower-dimensional space and the calculation of the projections of each cell onto all inter-cluster links are performed by the function `projcells` using the following command:

```
ltr <- projcells(ltr,cthr=5,nmode=FALSE)
```

   `cthr` defines the minimum number of cells that a cluster should comprise of in order to be included in the lineage tree inference. If the argument `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`.
4. In order to analyze the ratio between the cell-to-cell distances in the embedded space and the input space, a histogram can be plotted using the following command (Fig. 3a):

```
plotdistanceratio(ltr)
```

5. The randomization is performed using the function `projback`. The following command can be executed for this purpose:

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

   The argument `pdishuf` defines the number of randomizations to be performed. Importantly, if `fast` is set to `TRUE`, StemID2 operates in the fast mode (*see* **Note 6**). The fast mode is recommended for large datasets (>3000 cells).
6. To compute the lineage tree, the function `lineagetree` can be executed as follows:

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

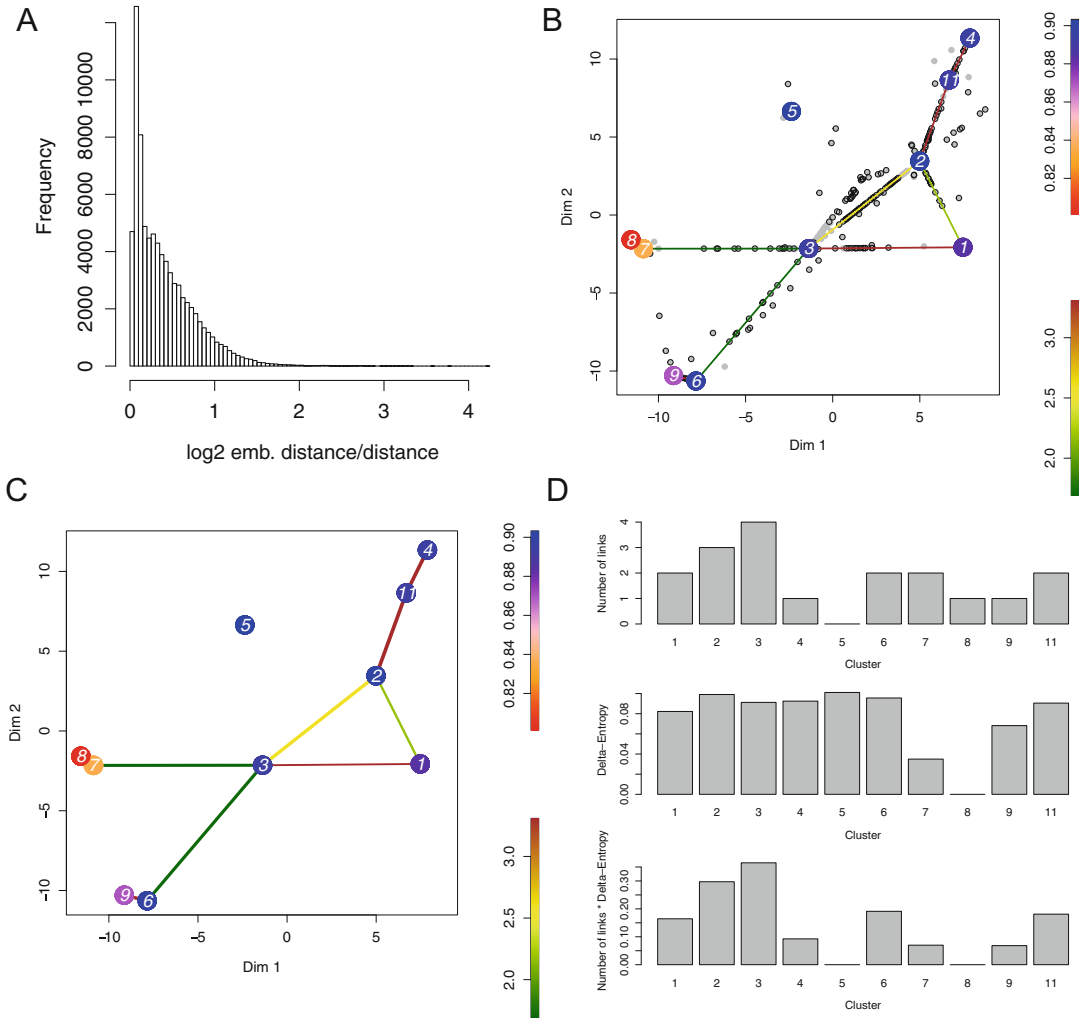

Fig. 3 Lineage tree inference and stem cell identity prediction. (**a**) Histogram depicting the $\log_2$ratio of the cell-to-cell distances in the lower-dimensional embedded space and the higher-dimensional input space. Values close to zero indicate a perfect conversion of the distances in the embedded space. Inferred lineage tree (**b**) with cells and (**c**) without cells. Cells are circled in back to indicate a statistically significant projection coordinate compared to the randomized background model. 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 links indicates the link score. (**d**) Bar plots showing the number of links, the delta-entropy, and the StemID2 score for all the clusters. Cluster 3 shows the highest StemID2 score

where pthr corresponds to the *p*-value threshold, which is used to determine whether the magnitude of an observed cell coordinate on a link 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.

7. Afterward, the function `comppvalue` has to be executed in order to identify connections at a given level of significance. The following R-code can be executed for this purpose:

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

`pethr` corresponds to a *p*-value threshold to determine if a given link is populated by number of cells significantly larger than expected for the randomized background distribution.

8. Finally the inferred lineage tree can be visualized with the function `plottree` using the following command (Fig. 3b, c):

```
plottree(ltr,showCells=TRUE,nmode=FALSE,scthr=.3)
```

`showCells` is a logical argument; if set to `TRUE`, single cells will also be shown in the plotted lineage tree. Otherwise, only the significant branches of the tree will be shown, and `scthr` can be defined as a link score, which can be given any value between 0 and 1. The link score corresponds to `1 -- fraction of link not covered by a cell`. Link score close to zero corresponds 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.

9. Finally, the StemID2 score is computed by executing the function `compscore` using the following R-code: `x <- compscore (ltr,nn=1,scthr=.3).`

10. The StemID2 score can be plotted by the function `plotscore` using the following command: `x <- plotscore(ltr,nn=1, scthr=.3)`. The function plots three histograms: the number of links, the delta-entropy, and the StemID2 score (Fig. 3d).

3.6 Pseudo-Temporal Ordering

StemID2 can also be used for the visualization and analysis of pseudo-temporal gene expression changes. Below we explain the steps necessary to create such pseudo-temporal profiles using example dataset provided with the StemID2 algorithm.

1. In the first step, the StemID2 object `ltr` and the information about the cells along a StemID2-predicted differentiation trajectory are provided using the following R-code:

```
n <- cellsfromtree(ltr,c(3,2,11,4))
```

Here, as an example, the differentiation trajectory predicted by StemID2 on the example data is used. Cluster

3, which has the highest StemID2 score, is used as a starting point, while cluster 4 is considered an endpoint.

2. Afterward, normalized expression data stored in the slot `sc@ndata` is given as an input to the function `filterset` in order to remove lowly expressed genes. The function `filterset` can be executed using the following command:

```
fs <- filterset(ltr@sc@ndata,n=n$f,minexpr=2,minnumber=1)
```

The function removes those genes, which are expressed at a value less than `minexpr` in at least `minnumber` of cells.

3. Now a self-organizing map (SOM) for grouping similar expression profiles into modules is created using the function `getsom`. The function requires the filtered expression data generated in the previous step and few other arguments such as number of nodes of the SOM (`nb`), window-size (`k`) for calculating the running mean of the expression values in the ordered cells, and also an option to generate such profiles using local regression (`locreg`). The following command is used to create a SOM: `s1d <- getsom(fs,nb=1000,k=5, locreg=TRUE,alpha=.5)`.

4. In order to group the nodes of the SOM into larger modules and to produce additional z-score transformed and binarized pseudo-temporal expression profiles, function `procsom` is executed using the following code:

```
ps <- procsom(s1d,corthr=.85,minsom=3)
```

`corthr` defines the minimum correlation threshold required for merging the neighboring two nodes, and `minsom` defines the minimum number of transcripts that a node should have to be included in the map.

5. Now the plots can be visualized using the `plotheatmap` function. First a color scheme corresponding to the particular cluster number is defined using the following two commands: `fcol <- ltr@sc@fcol` and `y <- ltr@sc@cpart[n$f]`. The `plotheatmap` function provides visualization of following four different types of pseudo-temporal plots ordered by SOM—(1) average z-score of all modules, (2) z-score profile of each gene, (3) normalized expression profile of each gene, and (4) binarized expression profile of each gene. The following commands can be executed to visualize the ordered plots (Fig. 4a–d):

```
plotheatmap(ps$nodes.z, xpart=y, xcol=fcol, ypart=unique
(ps$nodes), xgrid=FALSE, ygrid=TRUE, xlab=FALSE)
```

```
plotheatmap(ps$all.z, xpart=y, xcol=fcol, ypart=ps$nodes,
xgrid=FALSE, ygrid=TRUE, xlab=FALSE)
plotheatmap(ps$all.e, xpart=y, xcol=fcol, ypart=ps$nodes,
xgrid=FALSE, ygrid=TRUE, xlab=FALSE)
plotheatmap(ps$all.b, xpart=y, xcol=fcol, ypart=ps$nodes,
xgrid=FALSE,ygrid=TRUE, xlab=FALSE)
```

6. In order to visualize the dynamics of gene expression changes of a particular node or an individual gene, the following command can be used:

```
g <- names(ps$nodes)[ps$nodes == 1]
```

where g extracts the genes present in node 1. Now it can be plotted using the following command (Fig. 4e):

```
plotexpression(fs, y, g, n$f, k=25, col=fcol, name="Node
1", cluster=FALSE, locreg=TRUE, alpha=.5, types=NULL)
```

To visualize the expression profile changes of a particular gene, e.g., Clac4 here, the following command can be used (Fig. 4f):

```
plotexpression(fs, y, "Clca4__chr3", n$f, k=25, col=fcol,
cluster=FALSE, locreg=TRUE, alpha=.5, types=NULL)
```

3.7 Prediction of Stem Cell Identity Using Fate Bias Probabilities

As clear from the above section, StemID2 implementation relies on the clustering analysis performed by RaceID3. Although RaceID3 has an outlier identification step to refine the clustering, it is plausible that clustering analysis may not reveal the transcriptional differences, which are subtle and gradual in nature. For example, homogenous progenitor populations can be transcriptionally primed toward different lineages, but these differences could be so subtle that clustering analysis is not sensitive enough to detect such early priming and fate biases in the multipotent cells. As an attempt to overcome this challenge, we have recently developed an algorithm called FateID to quantify fate bias manifested in terms of subtle lineage-specific transcriptional characteristics within a multipotent progenitor population [8].

FateID utilizes prior knowledge to identify committed stages of all lineages and tracks differentiation trajectories backward in time, in order to predict the likelihood of multipotent progenitors to give rise to each lineage. FateID utilizes random forests, a supervised learning method known for its robustness to overfitting, to learn the identity of cells in a test set given the training data. FateID starts with target clusters of committed cell populations, identified by clustering or marker gene analysis. Cells in the

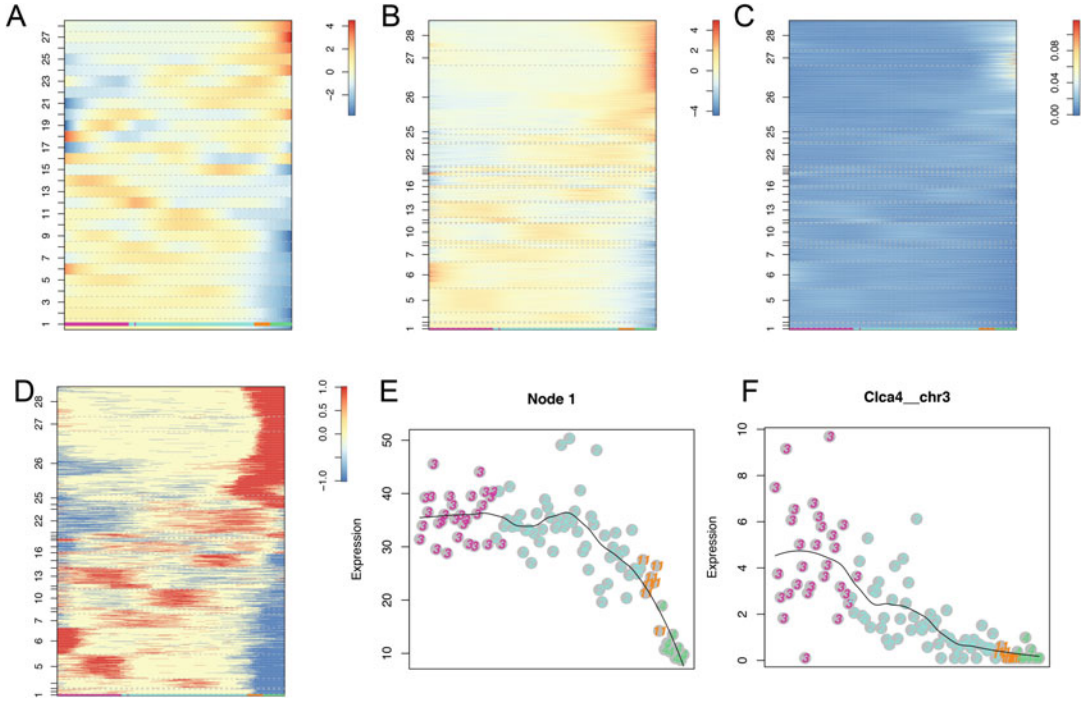

Fig. 4 Pseudo-temporal ordering. Heatmaps showing (**a**) average z-score of all modules, (**b**) z-scores of all genes, (**c**) expression profile of each gene, and (**d**) binarized expression profile of each gene ordered by SOM. (**e**) Pseudo-temporal expression profiles of all the genes in module 1 and (**f**) *Clac4*, a stem cell marker gene

local neighborhood of each of these target clusters are selected to build a test set. Next, a defined number of cells from the target clusters, most similar to the test set, are used as training set to derive the probability of each cell in the test set to belong to any of the target clusters based on random forests votes. Cells with a significant bias toward a particular lineage become part of the respective target cluster and are utilized for classification in the next iteration. FateID proceeds until the fate biases of all cells have been inferred.

In principle, the fate bias probabilities calculated for each cell using FateID can be used to predict the level of multipotency in a given population of cells. For example, if a cell has equal probabilities of differentiation toward all target clusters, it is more likely to be more multipotent compared to a cell which has a high fate bias toward one particular lineage. Furthermore, based on fate biases for the different target clusters, an entropy value for each cell is calculated. An entropy value of the fate biases will indicate the level of multipotency with larger fate bias entropies corresponding to more multipotent cell states.

Below we describe a step-by-step approach to calculate the fate bias and entropy based on these values to predict the level of multipotency of the cells present in the scRNA-seq dataset.

1. The FateID algorithm demands three important inputs in order to calculate fate bias, (1) the expression dataset `x`, (2) a vector `y` containing the information about the partitioning of cells into various clusters, and (3) information about the target clusters `tar`, which represent the mature cell types in the dataset (*see* **Note 7**). The following commands can be executed to initialize these three objects:

```
y <- sc@cpart[sc@cpart %in% c(4,11,2,3,7,8,6,9)]
x <- sc@fdata[,colnames(sc@fdata) %in% names(y)]
tar <- c(8,9,4)
```

   Please note that we have used the RaceID3 cluster partitioning as an input to the FateID algorithm. Moreover, one can also choose the final number of clusters to be included in the analysis. Here we have included clusters 4, 11, 2, 3, 7, 8, 6, and 9 in the further analysis and set clusters 8, 9, and 4 as target clusters representing the most mature cell types.

2. The next step of the algorithm utilizes `fateBias` function to calculate the fate biases. The following command can be executed to achieve this goal:

```
fb <- fateBias(x, y, tar, z=NULL, minnr=5, minnrh=10,
nbfactor=5, use.dist=FALSE, seed=12345, nbtree=NULL)
```

3. In order to visualize the high-dimensional data into two- and three-dimensional space (`k=2` for 2D and `k=3` for 3D), the algorithm utilizes various dimensional reduction algorithms (argument `m`) such as tSNE (`tsne`), classical multidimensional scaling (`cmd`), locally linear embedding (`lle`), and diffusion maps (`dm`). Dimensional reduction can be achieved using the following code:

```
dr <- compdr(x, z=NULL, m=c("tsne","cmd","dm","lle"), k=2,
lle.n=30, dm.sigma="local", dm.distance="euclidean", tsne.
perplexity=30, seed=12345)
```

4. The low-dimensional data can be now visualized, for example, using the t-SNE coordinates through the following command (Fig. 5a): `plotFateMap(y,dr,k=2,m="tsne").`

5. The following command can be used to visualize the fate bias in the desired dimensional reduction method map for target cluster 9 (Fig. 5b): `plotFateMap(y,dr,k=2,m="tsne", fb=fb,g="t9").`

   The additional argument `g` provides the name of the target cluster concatenated with the letter `t`. To plot the fate bias for

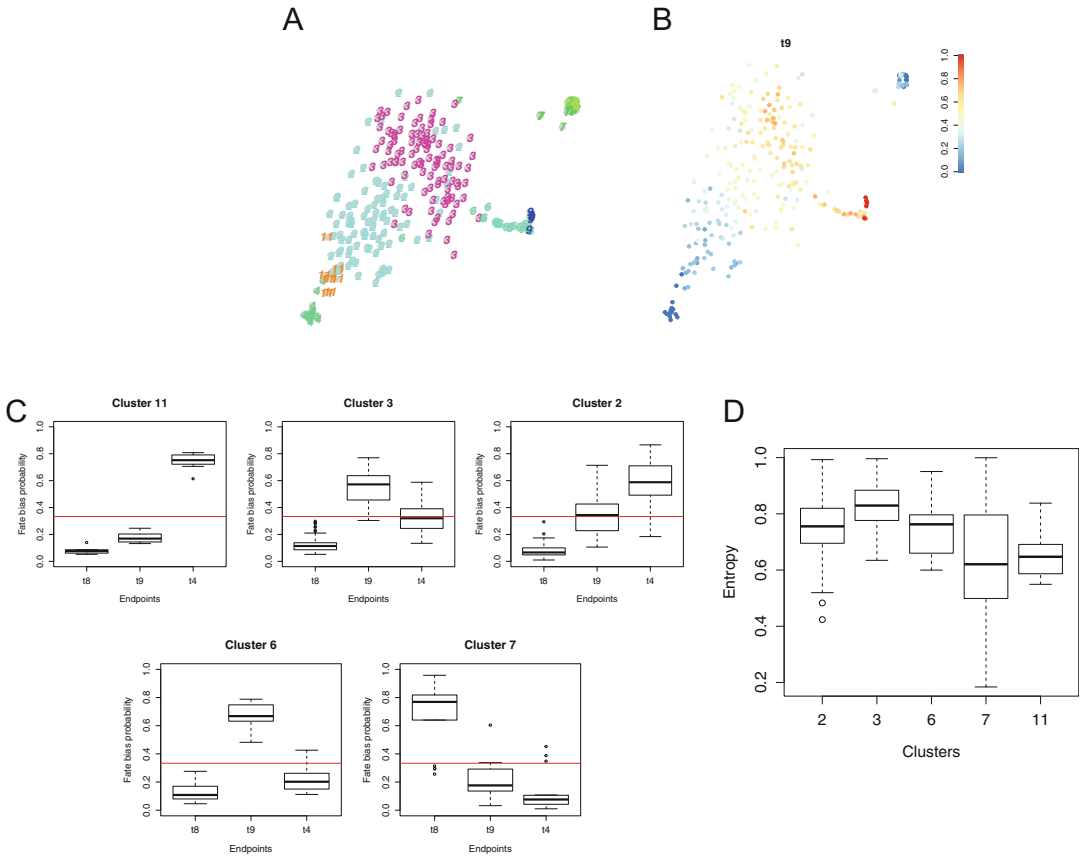

Fig. 5 Identifying multipotency levels of clusters using FateID. (**a**) tSNE representation of the clusters included in the FateID analysis. (**b**) tSNE representation of fate bias of cells toward cluster 9. Color-coding shows the fate probabilities on a scale of 0–1. (**c**) Boxplots depicting the fate probabilities of the progenitor clusters toward different mature states. The horizontal red line depicts an equal fate bias toward the three mature states (1/3 = 0.33). (**d**) Boxplot showing the entropy values calculated based on the fate bias. Cluster 3 shows the highest entropy level

clusters 8 and 4, `g` is initialized as `"t8"` and `"t4,"` respectively (not shown).

6. Next, the fate bias boxplots of each cluster included in the analysis toward the target clusters can be visualized using the following command (Fig. 5c):

```
data.fate.bais <- as.data.frame(cbind(y,fb$probs))
rc <- ceiling(sqrt(length(unique(data.fate.bais$y))))
par(mfrow =c(rc,rc))
for(i in c(1:length(unique(data.fate.bais$y))))
 {
boxplot(subset(data.fate.bais, data.fate.bais$y %in%
unique(data.fate.bais$y)[i])[,-1], main =
```

```
paste("Cluster",unique(data.fate.bais$y)[i], sep = " " ),
xlab="Endpoints", ylab="Fate bias probability",
ylim=c(0,1))
abline(h = 1/length(tar), col="red")
}
```

7. In order to calculate the entropy based on fate bias probabilities, the following command can be executed: `E <- plotFateMap(y,dr,k=2,m="tsne",g="E",fb=fb)`. Finally, to visualize the entropy boxplots for each cluster included in the analysis, use the following command (Fig. 5d):

```
data.fate.bais.ent <- as.data.frame(cbind(y,E))
boxplot(data.fate.bais.ent$E~data.fate.bais.ent$y,
xlab="Clusters", ylab="Entropy")
```

4 Notes

1. The first column of the `.xls` example file contains unique gene ids and has to be assigned as row names of the R data frame object. Also, the gene ids of the non-endogenous spike-in RNAs starts with "ERCC," and these transcripts have to be removed.
2. We recommend using downsampling only if batch effects due to differences in complexity between distinct libraries in the dataset are observed. Normalization by rescaling is in general more sensitive.
3. We routinely initialize `FGenes` with *Hspa1a*, *Hsp90aa1*, *Jun*, *Fos*, *Zfp36*, *Junb*, *Fosb*, *Socs3*, *Atf3*, *Jund*, *Hspe1*, *Hspa1b*, *Hsp90ab1*, *Hspb1*, *Egr1*, *Hspa8*, *Cebpd*, and *Cebpb*, since these genes were recently shown to be dissociation-induced [14].
4. Sometimes user may want to remove cell cycle-associated variability, and in this case, `CGenes` can be initialized with genes such as *Pcna* and *Mki67*.
5. If the argument `do.gap` is set to `TRUE`, the algorithm determines the number of clusters as the first local maximum of gap-statistic. The graph showing the value of gap-statistic as a function of the number of clusters can be plotted using the following command: `plotgap(sc)`. Since gap-statistic calculation requires longer computational time, we recommend users to opt within-cluster dispersion approach to choose a desired cluster number.

6. In this mode, the *p*-value is not computed by performing a large number of randomizations. Here, an expression table is first generated, which contains for each cluster a large number of cells (corresponding to the size of the entire dataset). Subsequently, the positions of these cells are randomized within the boundaries dictated by the actual data and projected on the links between the cluster of origin an all other clusters and assigned to the link with the longest projection. Next, a probability to fall on each link is derived from the occupancy of this link, and the number of cells on that link in the actual data is evaluated using a binomial test with this probability parameter as input. This way, a *p*-value for all links is derived in a much more efficient way than the multiple shuffling-based approach. However, and in contrast to the multiple shuffling-based approaches, the underlying model does not imply the correlation structure between the occupancy of links and is thus less precise than the original approach.
7. If a partitioning into cell types and states from a prior clustering analysis is not available, FateID can derive a partitioning based on marker gene information which defines the most mature cell types using the function `FMarker`.

References

1. Bendall SC, Davis KL, Amir el AD, Tadmor MD, Simonds EF, Chen TJ, Shenfeld DK, Nolan GP, Pe'er D (2014) Single-cell trajectory detection uncovers progression and regulatory coordination in human B cell development. Cell 157(3):714–725
2. Trapnell C, Cacchiarelli D, Grimsby J, Pokharel P, Li S, Morse M, Lennon NJ, Livak KJ, Mikkelsen TS, Rinn JL (2014) The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. Nat Biotechnol 32(4):381–386
3. Marco E, Karp RL, Guo G, Robson P, Hart AH, Trippa L, Yuan GC (2014) Bifurcation analysis of single-cell gene expression data reveals epigenetic landscape. Proc Natl Acad Sci U S A 111(52):E5643–E5650
4. Shin J, Berg DA, Zhu Y, Shin JY, Song J, Bonaguidi MA, Enikolopov G, Nauen DW, Christian KM, Ming GL, Song H (2015) Single-cell RNA-Seq with waterfall reveals molecular cascades underlying adult neurogenesis. Cell Stem Cell 17(3):360–372
5. Qiu X, Mao Q, Tang Y, Wang L, Chawla R, Pliner HA, Trapnell C (2017) Reversed graph embedding resolves complex single-cell trajectories. Nat Methods 14(10):979–982
6. Guo M, Bao EL, Wagner M, Whitsett JA, Xu Y (2017) SLICE: determining cell differentiation and lineage based on single cell entropy. Nucleic Acids Res 45(7):e54
7. Grun D, Muraro MJ, Boisset JC, Wiebrands K, Lyubimova A, Dharmadhikari G, van den Born M, van Es J, Jansen E, Clevers H, de Koning EJP, van Oudenaarden A (2016) De novo prediction of stem cell identity using single-cell transcriptome data. Cell Stem Cell 19(2):266–277
8. Herman JS, Sagar, Grun D (2018) FateID infers cell fate bias in multipotent progenitors from single-cell RNA-seq data. Nat Methods 5:379–386. https://doi.org/10.1101/218115
9. Grun D, Lyubimova A, Kester L, Wiebrands K, Basak O, Sasaki N, Clevers H, van Oudenaarden A (2015) Single-cell messenger RNA sequencing reveals rare intestinal cell types. Nature 525(7568):251–255

10. van der Maaten L, Hinton G (2008) Visualizing data using t-SNE. J Mach Learn Res 9:2570–2605
11. Anders S, Huber W (2010) Differential expression analysis for sequence count data. Genome Biol 11:R106
12. Ridden SJ, Chang HH, Zygalakis KC, MacArthur BD (2015) Entropy, ergodicity, and stem cell multipotency. Phys Rev Lett 115:208103
13. Shannon CE (1948) A mathematical theory of communication. Bell Syst Tech J 27:379–423, 623–656.
14. van den Brink SC, Sage F, Vértesy A, Spanjaard B, Peterson-Maduro J, Baron CS, Robin C, van Oudenaarden A (2017) Single-cell sequencing reveals dissociation-induced gene expression in tissue subpopulations. Nat Methods 14:935–936