Inferring and visualising the hierarchical tree structure of Single-Cell RNA-seq Data data with the *cellTree* package

David duVerle & Koji Tsuda

Graduate School of Frontier Sciences, the University of Tokyo

*Correspondence to dave (at) cb.k.u-tokyo.ac.jp

October 17, 2016

Abstract

Single-cell RNA sequencing, one of the most significant advances in recent genomics [1], is becoming increasingly common, providing unique insights into the exact gene-expression snapshots of cells throughout biological processes such as cell differentiation or tumorigenesis.

A number of methods have been suggested to help organise and visualise the structure of cells measured through single-cell sequencing [2, 3], yet none seem to be able to accurately capture complex differentiation paths over time, or offer a satisfying explanation for the low-dimensional support used to infer the cell distances.

This *R* package implements a new statistical method based on topic modelling techniques, for inferring and visualising the tree structure of single-cell RNA-seq samples and interpreting the sets of genes driving transitions between states.

cellTree version: 1.4.0 [1]

[1] This document used the vignette from *Bioconductor* package *DESeq2* as *knitr* template

Contents

1 Introduction

When considering a number of single-cell expression measurements taken over time (e.g during cell differentiation) or space (e.g. with samples taken across similar tissues), we expect specific pathways, and the genes that compose them, to be more or less active based on the exact state of the cell sampled. This leads us to hypothesise the existence of (possibly overlapping) sets of genes, representing groups of pathways and indirectly characterising specific biological processes under way at sampling time.

In trying to identify the structure connecting these cell measurements, we therefore make the assumption that there exist a latent gene group structure that explains the similarities between cell measurements. Such a (low-dimensional) group structure would additionally provides a support for dimension-reduction of the overall data set that we expect to be both vastly more accurate and more semantically-useful than other statistical procedures such as PCA, ICA or MDS.

We borrowed a method from the field of natural language processing known as Latent Dirichlet Allocation (itself part of the more general field of research of 'topic modelling') to identify this group structure and use it to build a tree structure connecting all cells. In addition to wrapping existing inference methods in a 'bioinformatics-friendly' package, we added a number of functions to take advantage and visualise the fitted model.

Principally, we introduced "backbone trees", a new type of tree structure specifically designed to easily visualise cells along complex differentiation paths, and proposed a heuristic implementation to estimate such a tree from the distance matrix obtained through the fitted model.

Additionally, we implemented a pipeline to run gene set enrichment analysis on the different LDA

"topics", using Gene Ontology terms. Results can be visualised in the form of annotated tables or subgraph of the Gene Ontology DAG.

All tabular results can be exported to LaTeX, for convenient re-use in scientific communication.

2 Installing the cellTree package

cellTree requires the following CRAN-R packages: *topicmodels*, *slam*, *maptpx*, *igraph*, *xtable*, *Rgraphviz* and *gplots*, along with the *Bioconductor* package: *topGO*.

Installing *cellTree* from *Bioconductor* will install all these dependencies:

```
source("http://bioconductor.org/biocLite.R")
biocLite("cellTree")
```

The documentation's examples as well as this vignette's code will further require *Bioconductor* packages: *HSMMSingleCell*, *org.Hs.eg.db* and *biomaRt*:

```
biocLite(c("HSMMSingleCell", "org.Hs.eg.db", "biomaRt"))
```

Then load the package with:

```
library(cellTree)
```

3 Preparing the Gene Expression Data Input

The principal input to *cellTree* is a matrix of gene expression values, with genes as rows and cells as columns. Both gene names (preferably in HGNC format) and cell identifiers should be present as `rownames` and `colnames` for the matrix.

In this vignette, we will be using RNA-seq data for human skeletal muscle myoblasts (HSMM) compiled in the *HSMMSingleCell* package:

```
# load HSMMSingleCell package and load the data set:
library(HSMMSingleCell)
data(HSMM_expr_matrix)

# Total number of genes * cells:
dim(HSMM_expr_matrix)
```

```
## [1] 47192   271
```

Unlike other cell ordering methods, *cellTree*'s functions scale relatively well to very-high-dimensional data, and it is therefore not particularly essential to reduce the set of genes selected. However, the default pipeline will automatically apply a log transformation and remove low-variance values from the

data set. This can be disabled if the data is already treated or if you would prefer to do your own treatment (see the documentation for `compute.lda`).

4 Fitting LDA Model

4.1 Using Latent Dirichlet Allocation for Gene Expression Data

The Latent Dirichlet Allocation (LDA; [4]) model is a Bayesian mixture model initially developed for the analysis of text documents, that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. In natural language processing, given a set of documents and word-occurence counts for each documents, the model assumes that each document is a mixture of topics (with a Dirichlet prior) and each word is the result of one of the document's topic (with a Dirichlet prior on the per-topic word distribution).

For an in-depth explanation of the mathematics behind the general LDA model, we recommend consulting David Blei's original paper [4]. For details on the different inference methods and their implementation, please consult the documentation and vignettes for the *topicmodels* and *maptpx* packages, along with their companion publications [5, 6].

In the context of single-cell gene expression analysis, cells play the role of 'documents' and discretised gene expression levels stand for 'word-occurence counts'. The fitted LDA model for our data is therefore composed of a set of topic distributions for each cell, and per-topic gene distributions. The per-cell topic histograms can then be used as a low-dimension support to compute cell distances and build a structured representation of the cell hierarchy.

4.2 Choosing the Number of Topics

The main parameter to the LDA fitting procedure is the desired number of topics: `k`, (best values for other hyper-parameters are automatically picked by the different fitting methods). As often with such models, a large number of topics (and therefore a more complex statistical model) can lead to overfitting, and it is therefore preferable to use the smallest possible number that provides a good explanation of the data.

Because of the loose significance of the concept of 'topics' in the context of gene expression in a cell, it is difficult to give a reliable estimate of the ideal number, based on biological knowledge alone. A good rule of thumb is that the number of topics should somewhat match the number of major processes (e.g. differentiation steps) undertaken by the cells during the experiment. During our own experiments with a number of single-cell time-series and tissue-based data sets, we found that the optimal number of topics generally stayed between 3 and 7.

The generally-recommended method to select a number of topics is to use cross-validation with different values of `k`, looking at the likelihood for each topic number. However, the computation time for such a method can be prohibitive on large data sets and large range of topic numbers. For convenience, we provide a wrapper to the *maptpx* implementation that uses Matthew Taddy's ingenious method for

model selection through joint MAP estimation [6]: as it fits models for iteratively larger number of topics (using the previous fit's residuals as a basis), this method can exhaustively look at a large range of topic numbers in considerably less time than it takes other methods.

One way to check the sparsity of the model based on biological knowledge, is to examine the gene set enrichment for the different topics (see 6): if two topics share a large amount of identical GO terms, it is quite possible that they are redundant and the model could be made sparser.

4.3 Computing LDA Model Fit

Using the HSMM data set previously loaded, we can use *maptpx* to automatically select the best number of topics and return the fitted model for that number:

```
# Run LDA inference using 'maptpx' method
# finding best number of topics k between 3 and 8:
lda.results = compute.lda(HSMM_expr_matrix, k.topics=3:8, method="maptpx")
```

The argument `k.topics` can only be sent a vector of integers when `method` argument is set to "maptpx" (other methods must be sent a scalar value).

Optionally, we could run the (much slower, though potentially more accurate) collapsed Gibbs sampling method:

```
# Run LDA inference using 'Gibbs' method for k = 6 topics:
lda.results = compute.lda(HSMM_expr_matrix, k.topics=6, method="Gibbs")
```

In order to perform further analysis on the fitted LDA model, it is be preferable for the row names of the input data matrix to contain HGNC-conformant gene names. This can be done by using the *biomaRt* package to convert the original ENSEMBL gene names of the *HSMMSingleCell* package to HGCN (a pre-computed set can also be used: see following paragraph):

```
HSMM_expr_matrix.hgnc = HSMM_expr_matrix

library("biomaRt")
ensembl.ids = sapply(strsplit(rownames(HSMM_expr_matrix), split=".",fixed=TRUE),
                                          "[",
                                          1)
ensembl.mart = useMart(host="www.ensembl.org",
                                            "ENSEMBL_MART_ENSEMBL",
                                            dataset = "hsapiens_gene_ensembl")
gene.map = getBM(attributes = c("ensembl_gene_id", "entrezgene", "hgnc_symbol"),
                                  filters = "ensembl_gene_id",
                                  values = ensembl.ids,
                                  mart = ensembl.mart)
idx = match(ensembl.ids, gene.map$ensembl_gene_id)
hgnc.ids = gene.map$hgnc_symbol[idx]
has.hgnc.ids = !is.na(hgnc.ids)&(hgnc.ids!="")
```

```
rownames(HSMM_expr_matrix.hgnc)[has.hgnc.ids] = hgnc.ids[has.hgnc.ids]

HSMM_lda_model = compute.lda(HSMM_expr_matrix.hgnc, k.topics=6)
```

For convenience, we have packaged a pre-computed LDA model that already includes converted gene names:

```
# Load pre-computed LDA model for skeletal myoblast RNA-Seq data
# from HSMMSingleCell package:
data(HSMM_lda_model)

# Number of topics of fitted model:
print(HSMM_lda_model$K)
```

```
## [1] 5
```

```
# Model uses HGCN gene names:
head(rownames(HSMM_lda_model$theta))
```

```
## [1] "TSPAN6"   "DPM1"     "SCYL3"    "C1orf112" "CFH"      "FUCA2"
```

5 Building a Backbone Tree

Once a model has been fitted to the data using `compute.lda`, it is possible to compute pairwise distances for all cells, based on per-cell topic histograms (we use the Chi-square distance):

```
# Compute pairwise distance between cells
# based on topic distributions in the fitted model:
dists = get.cell.dists(HSMM_lda_model)

print(dists[1:5,1:5])
```

```
##           T0_CT_A01 T0_CT_A03 T0_CT_A05 T0_CT_A06 T0_CT_A07
## T0_CT_A01     0.000     0.578     0.645     0.262     0.598
## T0_CT_A03     0.578     0.000     0.386     0.401     0.462
## T0_CT_A05     0.645     0.386     0.000     0.566     0.441
## T0_CT_A06     0.262     0.401     0.566     0.000     0.539
## T0_CT_A07     0.598     0.462     0.441     0.539     0.000
```

This distance matrix can be used with methods such as `hclust`, to perform hierarchical cluster analysis, or with various tree-building algorithm, to identifying the underlying tree structure of the cells.

In most cases, the cells measured are taken in groups of similar samples (e.g. at specific time-points) that spread along a continuum between the various groups. We expect a small (or at least smaller) variance within groups, and average short distance between samples belonging to neighbouring groups (in time or space). One natural way to visualise such a structure is using a minimum spanning tree (MST).

In order to help properly root the tree, we can provide additional information to the function, in the form of group labels for each cell batch. In this instance, cells were measured at 4 separate time points (0, 24, 48 and 72 hours):

```
# Recover sampling time point for each cell:
library(HSMMSingleCell)
data(HSMM_sample_sheet)
days.factor = HSMM_sample_sheet$Hours
days = as.numeric(levels(days.factor))[days.factor]

# Our grouping annotation (in hours):
print(unique(days))
```

```
## [1]  0 24 48 72
```

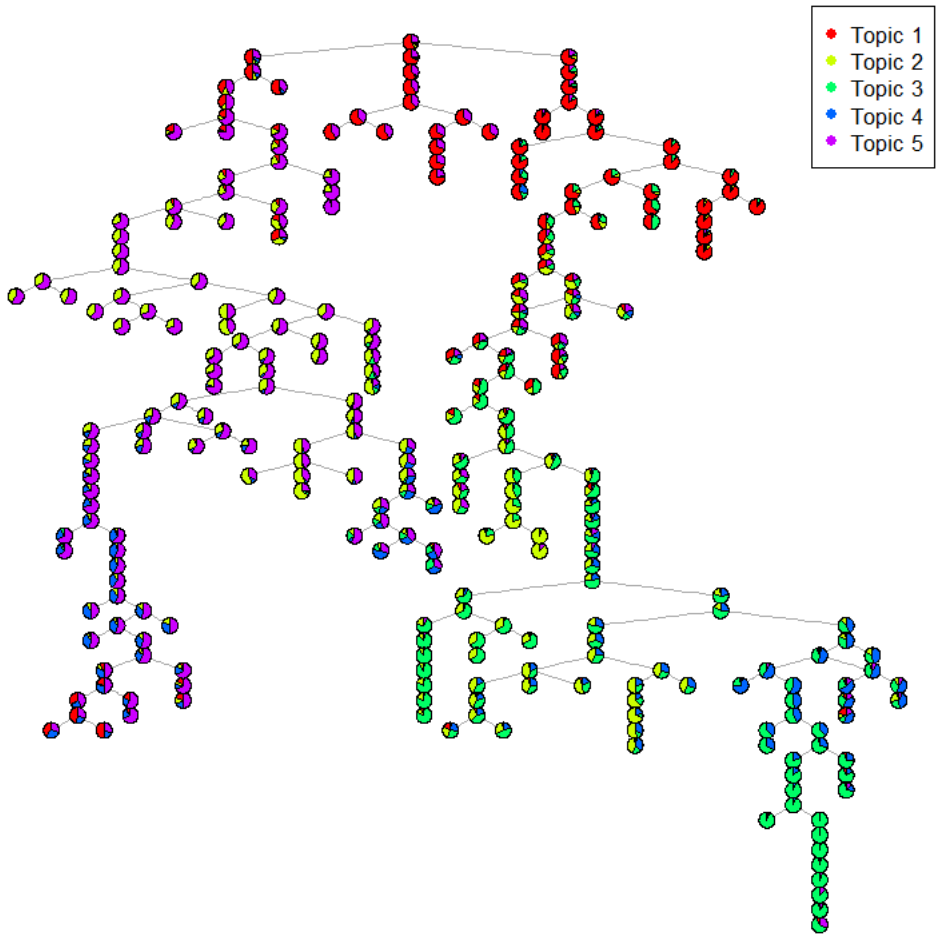
With this time annotation, we can then compute the rooted MST:

```
# compute MST from a fitted LDA model:
mst.tree = compute.backbone.tree(HSMM_lda_model, days, only.mst=TRUE)
```

```
## Using start group: 0 (1)
## Using rooting method: center.start.group
## Using root vertex: 4
## Returning Minimum Spanning Tree
```

```
# plot the tree (showing topic distribution for each cell):
mst.tree.with.layout = ct.plot.topics(mst.tree)
```

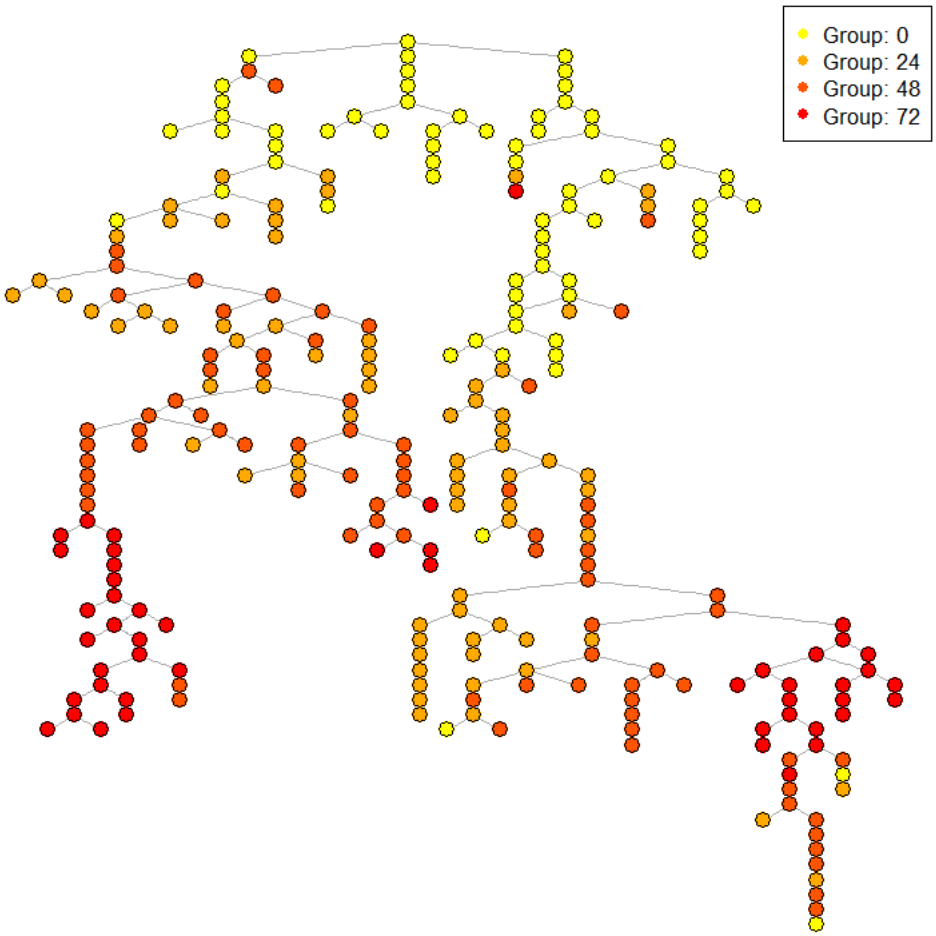
```
## Computing tree layout...
```



To have a better idea of the accuracy of the tree representation, we can plot it with the time group for each cell:

```
# plot the tree (showing time point for each cell):
mst.tree.with.layout = ct.plot.grouping(mst.tree)

## Computing tree layout...
```



As we can see, the inferred tree structure of the cells is somewhat consistent with the time points (i.e. generally follows a chronological order).

However, the MST approach relies to some extent on the assumption that cell distances are uniformly distributed, whereas in fact, we can expect cells inside a same group to have much lower variance than across groups.

The "ideal" structure of a typical cell differentiation experiment would look like a single path from one cell to the next or, in the case of subtype differentiation, a tree with a very small number of branches. Of course, because the samples do in fact represent separate cells, rather than the evolution of a single cell, we must expect small variations around such an idealised continuum. Our suggested approach is to identify cells that are most representative (at the gene expression level) of the biological process continuum, to create a "backbone", with all remaining cells at reasonably small distances from the backbone.

In more formal terms:

Considering a set of vertices V and a distance function over all pairs of vertices: $d : V \times V \to \mathbb{R}^+$, we call *backbone tree* a graph, T with backbone B, such that:

- T is a tree with set of vertices V and edges E.

- B is a tree with set of vertices $V_B \subseteq V$ and edges E_B
  E.
- All vertices in $V \setminus V_B$ are less than distance δ to a vertex in the backbone tree B: $\forall v \in V \setminus V_B, \exists v_B \in V_B$ such that $d(v, v_b) \leq \delta$.
- All 'vertebrae' vertices of T ($v \in V \setminus V_B$) are connected by a single edge to the closest vertex in the backbone tree: $\forall v \in V \setminus V_B, \forall v' \in V : (v, v') \in E \iff v' = argmin_{v' \in V_B} d(v, v')$.

In this instance, we relax the last condition to cover only "most" non-backbone vertices, allowing for a variable proportion of outliers at distance $> \delta$ from any vertices in V_B.

We can then define an optimal backbone tree, T^* to be a backbone tree that minimises the sum of weighted edges in its backbone subtree:

$$T^* = argmin_T \sum_{e \in E_B} d(e) \tag{1}$$

Finding such a tree can be easily shown to be NP-Complete (by reduction to the Vertex Cover problem), but we developed a fast heuristic relying on Minimum Spanning Tree to produce a reasonable approximation. The resulting quasi-optimal backbone tree (simply referred to as 'the' backbone tree hereafter) gives a clear hierarchical representation of the cells relationship: the objective function puts pressure on finding a (small) group of prominent cells (the backbone) that are good representatives of major steps in the cell evolution (in time or space), while remaining cells are similar enough to their closest representative for their difference to be ignored.

Backbone trees provides a very clear visualisation of overall cell differentiation paths (including potential differentiation into sub-types):

```
# compute backbone tree from a fitted LDA model:
b.tree = compute.backbone.tree(HSMM_lda_model, days)
```

```
## Using start group: 0 (1)
## Using rooting method: center.start.group
## Using root vertex: 4
## Adding branch #1:
##  [1]  65  53  45   2  55  47  57  48  44   7  19  25  69  66   9  63  18  62  51
## [20]  56  16  70 136 133 143  89  78 140  94 100 177 194 141 199 201 181 161 204
## [39] 225 236 255 247 246 233 229 259 258 146 235 159 185 191 216 166 149  83 168
## [58] 158   8
## Using branch width: 0.927 (width.scale.factor: 1.2)
## Outliers: 1
## Total number of branches: 1 (forks: 0)
## Backbone fork merge (width: 0.927): 60 ->  60
## Ranking all cells...
```

```
# plot the tree (showing time label for each cell):
b.tree.with.layout = ct.plot.grouping(b.tree)
```

```
## Computing tree layout...
```

In this plot, each backbone cell is represented as a larger disk, with its closest cells around it as smaller disks.

The backbone tree algorithm correctly finds the forked structure we expect in this particular instance, where proliferating cells eventually separate into interstitial mesenchymal and differentiating myoblasts [7]. However, we may expect a longer common trunk at the beginning of the experiment. This can be adjusted by passing a larger `width.scale.factor` argument (default is 1.2):

```
# compute backbone tree from a fitted LDA model:
b.tree = compute.backbone.tree(HSMM_lda_model, days, width.scale.factor=1.5)

## Using start group: 0 (1)
## Using rooting method: center.start.group
## Using root vertex: 4
## Adding branch #1:
##  [1]  65  53  45   2  55  47  57  48  44   7  19  25  69  66   9  63  18  62  51
## [20]  56  16  70 136 133 143  89  78 140  94 100 177 194 141 199 201 181 161 204
## [39] 225 236 255 247 246 233 229 259 258 146 235 159 185 191 216 166 149  83 168
## [58] 158   8
## Using branch width: 1.16 (width.scale.factor: 1.5)
## Outliers: 0
```

```
## Total number of branches: 1 (forks: 0)
## Backbone fork merge (width: 1.16): 60 ->  60
## Ranking all cells...
```

```
# plot the tree (showing time label for each cell):
b.tree.with.layout = ct.plot.grouping(b.tree)
```

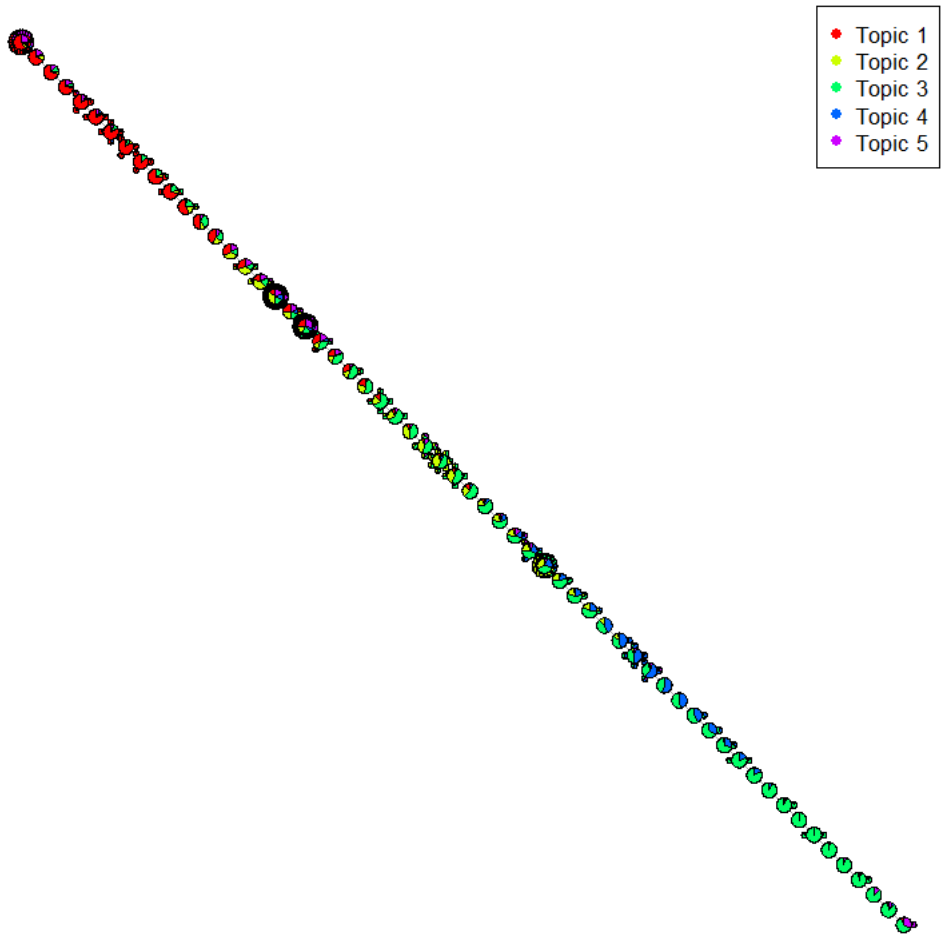
```
## Computing tree layout...
```

The `width.scale.factor` will affect what the backbone tree construction algorithm consider to be "close enough" cells: larger values will lead to less branches and more shared branch segments.

Finally, we can plot the backbone tree with the topic distribution for each cell:

```
# plot the tree (showing topic distribution for each cell):
b.tree.with.layout = ct.plot.topics(b.tree)
```

```
## Computing tree layout...
```



6 Gene Set Enrichment with Gene Ontologies

Because of their Bayesian mixture nature, and despite the slightly misleading name, 'topics' obtained through LDA fitting do not always match clear and coherent groupings (biological or otherwise), depending on sparsity of model and complexity of the input data. In particular, slightly less sparse models (with higher number of topics) can lead to better cell distance computation, but be harder to interpret.

In most cases, however, enrichment analysis of per-topic gene distribution can help characterise a given topic and its role in the cell's process, and even provide potential biological insight, by outlining the general processes most active in specific sections of the cell tree.

Topic analysis is conducted using Gene Ontology (GO) terms [8]. For each topic, *cellTree* orders genes by their per-topic probability and uses a Kolmogorov-Smirnov test to compute a p-value on the matching nodes in the GO graph. Three annotation categories are available: biological processes, cellular components and molecular functions.

To be able to map genes to GO terms, *cellTree* needs the relevant species database, e.g. *org.Hs.eg.db* for *Homo Sapiens* or *org.Mm.eg.db* for *Mus Musculus*:

```
# Load GO mappings for human:
library(org.Hs.eg.db)
```

We can then compute significantly enriched sets for each topic:

```
# Compute GO enrichment sets (using the Cellular Components category)
# for each topic
go.results = compute.go.enrichment(HSMM_lda_model,
                            org.Hs.eg.db, ontology.type="CC",
                            bonferroni.correct=TRUE, p.val.threshold=0.01)
```

```
# Print ranked table of significantly enriched terms for topic 1
# that do not appear in other topics:
go.results$unique[[1]]
```

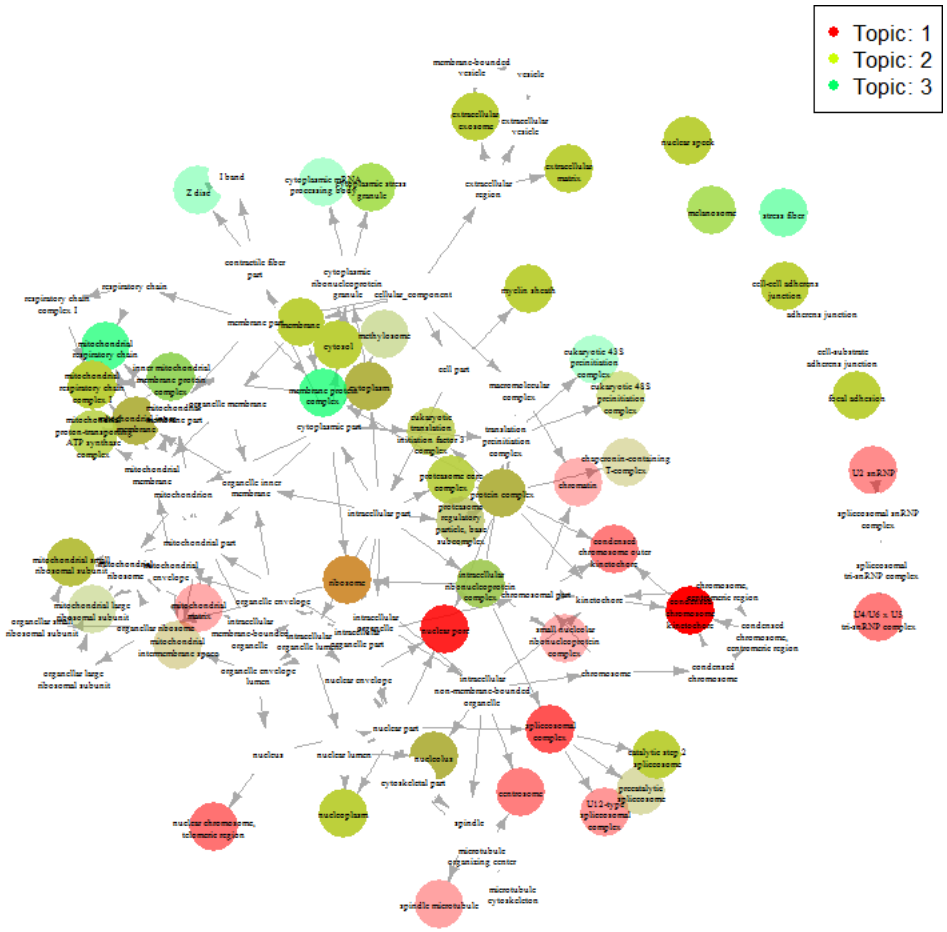
```
##         GO.ID                                     Term Total p-Value
## 17 GO:0000777          condensed chromosome kinetochore    87 2.8e-09
## 23 GO:0005681                     spliceosomal complex   165 3.1e-07
## 26 GO:0000784      nuclear chromosome, telomeric region   102 1.1e-06
## 27 GO:0046540              U4/U6 x U5 tri-snRNP complex    19 1.5e-06
## 28 GO:0000940    condensed chromosome outer kinetochore    12 1.6e-06
## 30 GO:0005813                               centrosome   404 1.7e-06
## 33 GO:0005686                                 U2 snRNP    17 3.0e-06
## 38 GO:0005689             U12-type spliceosomal complex    25 4.6e-06
## 40 GO:0005876                      spindle microtubule    50 7.5e-06
## 42 GO:0005759                     mitochondrial matrix   357 9.7e-06
## 43 GO:0000785                                chromatin   328 1.2e-05
## 44 GO:0005732 small nucleolar ribonucleoprotein complex    19 1.4e-05
```

During enrichment testing, you can have the function plot and output a subgraph of significantly enriched terms for each topic, by using the `dag.file.prefix` argument:

```
# Compute GO enrichment sets (using the Biological Process category)
# for each topic and saves DAG plots to files:
go.results.bp = compute.go.enrichment(HSMM_lda_model,
                                org.Hs.eg.db, ontology.type="BP",
                                bonferroni.correct=TRUE, p.val.threshold=0.01,
                                dag.file.prefix="hsmm_go_")
```

A useful way to visualise GO results is by plotting the subgraph of all enriched terms, coloured according to topic, using function `ct.plot.go.dag`:

```
# plot GO sub-DAG for topics 1 to 3:
go.dag.subtree = ct.plot.go.dag(go.results,
                                up.generations = 2,
                                only.topics=c(1:3))
```



Terms that are enriched for multiple topics are coloured with a mixture of the topics involved (weighted by their significance), making it easy to tell the terms that are exclusive to a small number of topics.

It is also possible to export the entire set of GO enrichment tables to a self-contained LaTeXdocument by using `go.results.to.latex`.

7 Result Summary

In addition to topic and grouping plotting, *cellTree* can output a useful ranked table of all cells in the data set, ordered along the cell tree (non-backbone cells are placed using interpolation between the nearest backbone cells).

```
# Generate table summary of cells, ranked by tree position:
cell.table = cell.ordering.table(b.tree)

# Print first 5 cells:
cell.table[1:5,]

##      branch node.label cell.name    cell.group main.topic topics
```

```
## [1,] "1"    4          "T0_CT_A06"  0          1          Numeric,5
## [2,] "1"    40         "T0_CT_E08"  0          1          Numeric,5
## [3,] "1"    13         "T0_CT_B08"  0          1          Numeric,5
## [4,] "1"    242        "T72_CT_C11" 72         1          Numeric,5
## [5,] "1"    64         "T0_CT_H02"  0          5          Numeric,5
```

There too, an option to create a self-contained LaTeXversion is available:

```
# Generate table summary of cells, ranked by tree position:
cell.table = cell.ordering.table(b.tree,
                                 write.to.tex.file="cell_summary.tex")
```

References

[1] Antoine-Emmanuel Saliba, Alexander J Westermann, Stanislaw A Gorski, and Jörg Vogel. Single-cell rna-seq: advances and future challenges. *Nucleic acids research*, page gku555, 2014.

[2] Cole Trapnell, Davide Cacchiarelli, Jonna Grimsby, Prapti Pokharel, Shuqiang Li, Michael Morse, Niall J Lennon, Kenneth J Livak, Tarjei S Mikkelsen, and John L Rinn. The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nature biotechnology*, 32(4):381–386, 2014.

[3] Miguel Juliá, Amalio Telenti, and Antonio Rausell. Sincell: an r/bioconductor package for statistical assessment of cell-state hierarchies from single-cell rna-seq. *Bioinformatics*, page btv368, 2015.

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[5] Kurt Hornik and Bettina Grün. topicmodels: An r package for fitting topic models. *Journal of Statistical Software*, 40(13):1–30, 2011.

[6] Matthew A Taddy. On estimation and selection for topic models. *arXiv preprint arXiv:1109.4518*, 2011.

[7] Cole Trapnell, Davide Cacchiarelli, Jonna Grimsby, Prapti Pokharel, Shuqiang Li, Michael Morse, Niall J Lennon, Kenneth J Livak, Tarjei S Mikkelsen, and John L Rinn. Pseudo-temporal ordering of individual cells reveals dynamics and regulators of cell fate decisions. *Nature biotechnology*, 32(4):381, 2014.

[8] Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T Eppig, et al. Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29, 2000.

8 Session Info

```
sessionInfo()
```

```
## R version 3.3.1 (2016-06-21)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows Server 2012 R2 x64 (build 9600)
##
## locale:
## [1] LC_COLLATE=C
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] org.Hs.eg.db_3.4.0     HSMMSingleCell_0.107.2 cellTree_1.4.0
##  [4] topGO_2.26.0           SparseM_1.72           GO.db_3.4.0
##  [7] AnnotationDbi_1.36.0   IRanges_2.8.0          S4Vectors_0.12.0
## [10] Biobase_2.34.0         graph_1.52.0           BiocGenerics_0.20.0
## [13] knitr_1.14
##
## loaded via a namespace (and not attached):
##  [1] igraph_1.0.1       magrittr_1.5       xtable_1.8-2       lattice_0.20-34
##  [5] stringr_1.1.0      highr_0.6          caTools_1.17.1     tools_3.3.1
##  [9] grid_3.3.1         maptpx_1.9-2       KernSmooth_2.23-15 DBI_0.5-1
## [13] gtools_3.5.0       matrixStats_0.51.0 formatR_1.4        bitops_1.0-6
## [17] slam_0.1-38        evaluate_0.10      RSQLite_1.0.0      gdata_2.17.0
## [21] stringi_1.1.2      gplots_3.0.1       BiocStyle_2.2.0
```