SCTDA Tutorial

Pablo G. Cámara

July 16, 2015

SCTDA is an object oriented python library for topological data analysis of high-throughput single-cell RNA-seq data. It includes tools for the preprocessing, analysis and exploration of single-cell RNA-seq data, based on condensed topological representations produced by software such as Mapper or Ayasdi. This tutorial illustrates the basic SCTDA workflow using the motor neuron dataset of [1] as case study. It is based on an IPython notebook.

In [1]:
```
import SCTDA
```

1 Preliminary steps

1.1 Pre-processing

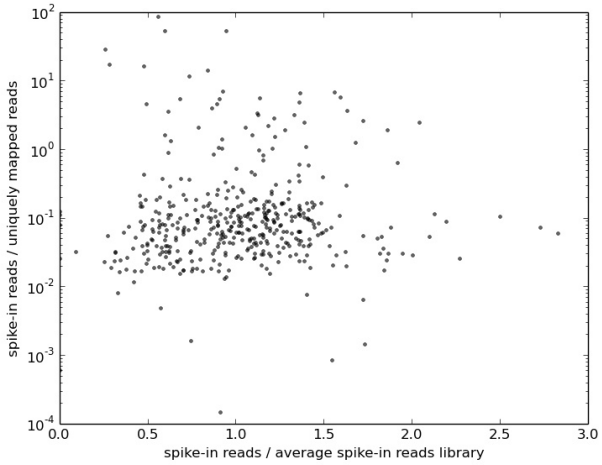
For convenience, SCTDA includes a class for preparing and filtering single-cell expression data, based on RNA spike-in read counts. It takes as input one or more files with read counts in the format of HTSeq-count. Files can be all from a same time point or from multiple time points of a continuous biological process. In this tutorial we consider the case of in vitro motor neuron differentiation from mouse embryonic stem cells. The dataset consists of 11 files corresponding to day 2 (2 files), 3 (2 files), 4 (2 files), 5 (2 files) and 6 (3 files). Each file contains whole-transcriptome read counts of 40 individual cells. ERCC spike in's from Life Technlogies were used in all samples.

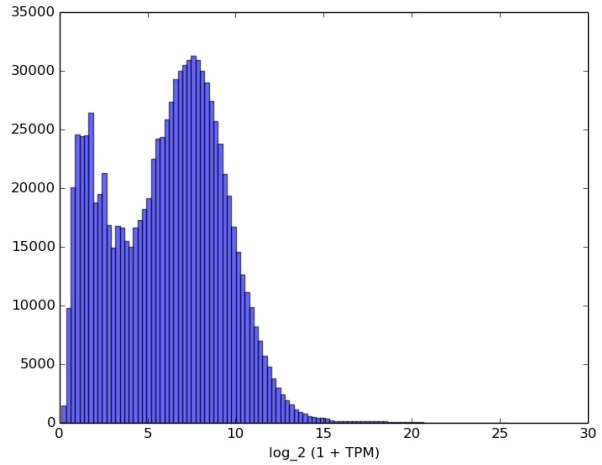
The class `SCTDA.Preprocess` allows to read, filter and organize the data, putting it in the appropriate form for SCTDA. Instances are initialized by providing a list of files with the corresponding timepoints and library id's, as well as the number of cells per file and the common identifier for RNA spike-in read counts:

In [2]:
```
files = ['D2_TM009.txt', 'D2_TM010.txt', 'D3_TM011.txt', 'D3_TM012.txt',
         'D4_TM013.txt', 'D4_TM015.txt', 'D5_TM016.txt', 'D5_TM019.txt',
         'D6_TM017.txt', 'D6_TM018.txt', 'D6pos_TM014.txt']

days = [2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 6]
libs = ['TM009', 'TM010', 'TM011', 'TM012', 'TM013', 'TM015',
         'TM016', 'TM019', 'TM017', 'TM018', 'TM014']

p = SCTDA.Preprocess(files, days, libs, 40, spike='ERCC')
```

Some simple data statistics can be shown using method `SCTDA.Preprocess.show_statistics()`:

In [3]:
```
p.show_statistics();
```





From the first plot we observe that some cells have very low ratio between spike-in reads and average spike-in reads in the library, presumably due to low sequencing depth. Some other cells have large ratio of spike-in reads and uniquely mapped reads, presumably due to large amounts of degraded RNA. Finally, some cells have very low ratio of spike-in reads and uniquely mapped reads, being large outliers. From the second plot we observe a large amount of transcripts near the detection limit, presumably due to noise. When normalizing read counts to transcripts per million (TPM), SCTDA assumes that read counts have been obtained by CEL-seq or any other single-cell RNA-seq method independent of gene lengths. We can filter out all those cells using the method `SCTDA.Preprocess.save()`:

In [4]:
```
p.save('table_tpm.tsv', filterXlow=0.1, filterYlow=0.01, filterYhigh=0.8,
       filterZlow=4.0)
```

Out [4]:
```
(373,
 {'D2_TM009.txt': 2,
  'D2_TM010.txt': 5,
  'D3_TM011.txt': 6,
  'D3_TM012.txt': 4,
  'D4_TM013.txt': 15,
  'D4_TM015.txt': 6,
```

```
'D5_TM016.txt': 4,
'D5_TM019.txt': 6,
'D6_TM017.txt': 7,
'D6_TM018.txt': 11,
'D6pos_TM014.txt': 1})
```

Parameters `filterXlow` and `filterXhigh` set respectively lower and upper bounds in the ratio between spike-in reads and the average number of spike-in reads in the library. Parameters `filterYlow` and `filterYhigh` set respectively lower and upper bounds in the ratio between spike-in reads and uniquely mapped reads. Parameters `filterZlow` and `filterZhigh` set respectively lower and upper bounds on the number of read counts for each gene and sample. Out of the 440 cells, 373 pass all the above filters. The number of cells that have been filtered out in each file is returned as a dictionary.

The above command also produces a tab separated file called `table_tpm.tsv`, where each row correspond to a cell passing all filters. The first column contains a unique identifier of the cell, the second column contains the sampling day, the third column contains the library id and the remaining columns contain $\log_2(1 + \text{TPM})$ expression values for each gene, after filtering out reads according to `filterZlow` and `filterZhigh` parameters. This table can be used by dedicated software, such as Mapper or Ayasdi, to build a topological representation, that can be then analysed using SCTDA.

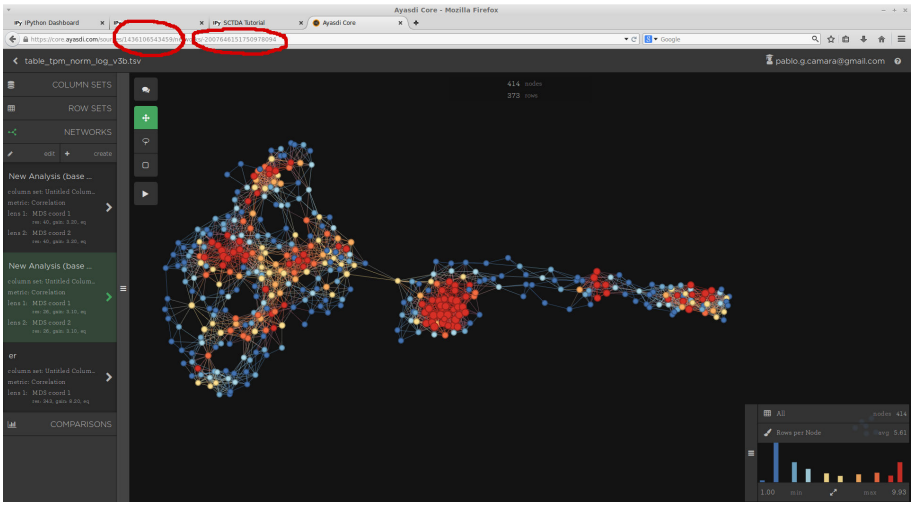
1.2 Parsing the topological graph

Apart from the above tab separated table, SCTDA makes use of a topological condensed representation of the table. This can be produced using software such as Mapper or Ayasdi. The graph should be in GEXF format, and should be accompanied by two JSON files with the same name as the graph and extension `.json` and `.groups.json`, specifying respectively the rows of the table that are associated to each node of the graph and, optionally, groups of nodes to be considered for subsequent analysis.

SCTDA provides a method to generate these files from an existing Ayasdi Core session:

```
In [5]: SCTDA.ParseAyasdiGraph("analysis1", "1436106543459",
                               "-2007646151750978094", "username", "password");
```

This will produce the files `analysis1.gexf`, `analysis1.json` and `analysis1.groups.json`. "`1436106543459`" and "`-2007646151750978094`" are the identifiers of the Ayasdi Core session that we are importing into SCTDA. They can be obtained from the Ayasdi Core session as indicated in the following screenshot:



2 Analysis

SCTDA provides two major classes for the analysis of single cell RNA-seq expression data. These are `SCTDA.UnrootedGraph` and `SCTDA.RootedGraph`, respectively for non-longitudinal and longitudinal single cell RNA-seq data. `SCTDA.RootedGraph` is an inherited class from `SCTDA.UnrootedGraph`, so all methods of `SCTDA.UnrootedGraph` are also included in `SCTDA.RootedGraph`, in addition to some methods that are specific of `SCTDA.RootedGraph`. In this tutorial we will make use of `SCTDA.RootedGraph`, as we are dealing with longitudinal data. The class is initialized with the data produced in the preliminary steps described above:

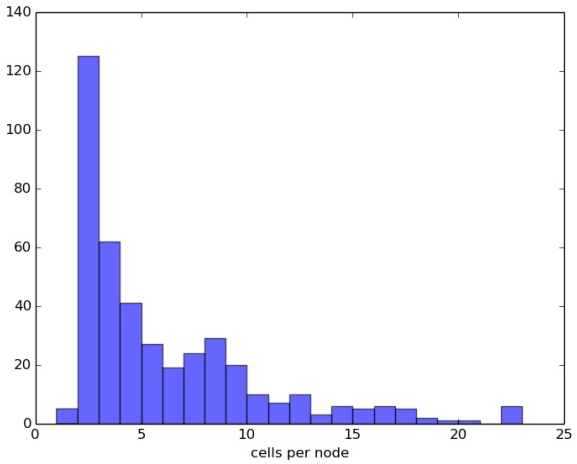
```
In [6]: c = SCTDA.RootedGraph("analysis1", "table_tpm.tsv");
```

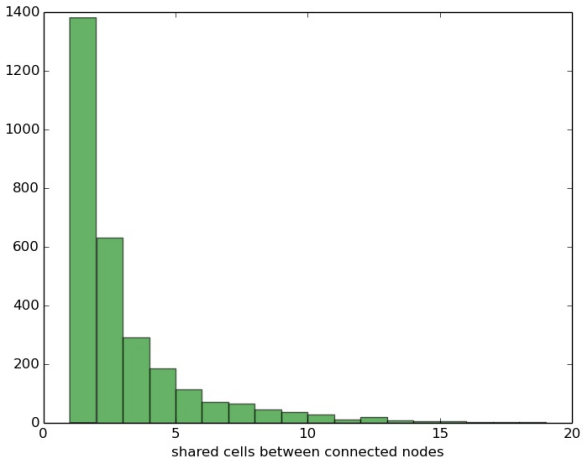
The class includes several methods for the analysis and visualisation of data. These are described in what follows.

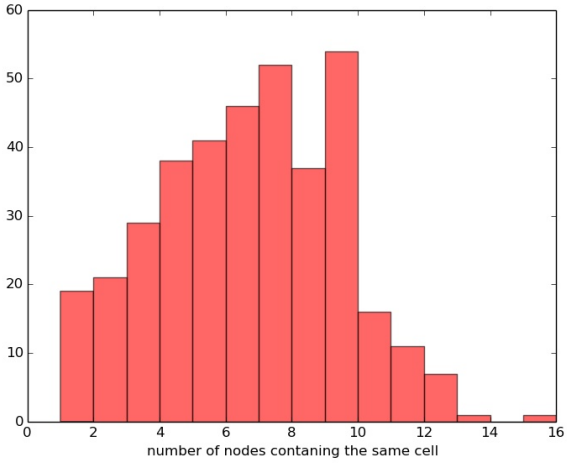
2.1 Exploring and visualizing the data

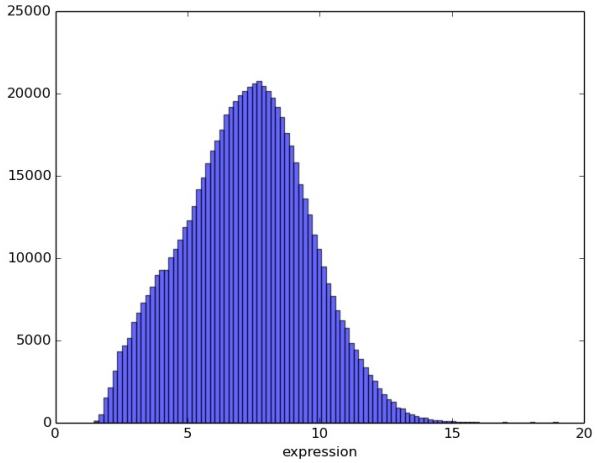
We can display some general statistics using the method `SCTDA.RootedGraph.show_statistics()`:

```
In [7]: c.show_statistics();
```





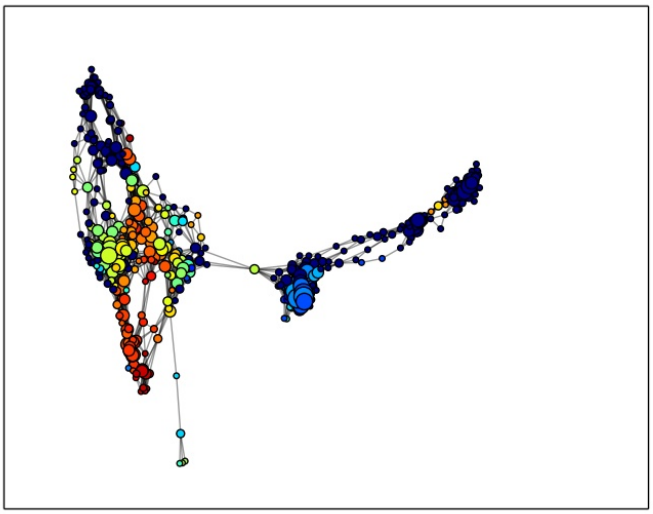




The first plot displays the distribution of the number of cells per node in the topological representation. The second plot presents the distribution of the number of common cells between nodes that share an edge in the topological representation. The third plot contains the distribution of the number of nodes that contain the same cell. Last plot shows the distribution of transcripts in $\log_2(1 + \text{TPM})$ scale, after filtering. Observe, in particular, that the filters that we have applied in the preliminary steps have removed the excess of transcripts near the detection limit.

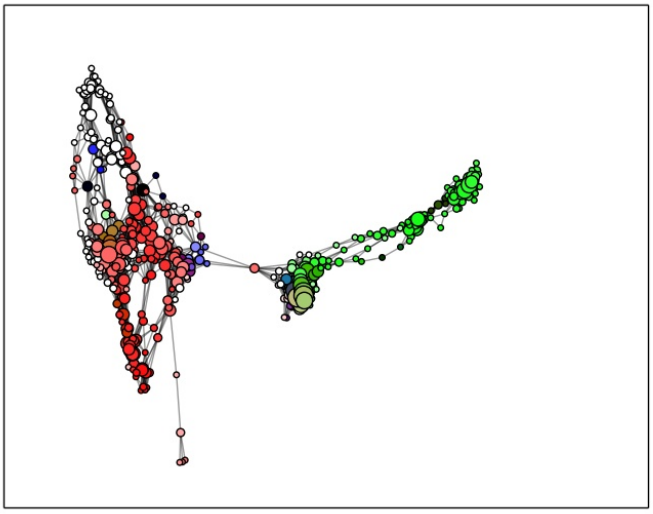
We can use the method `SCTDA.RootedGraph.draw()` to display the topological representation colored by the expression of a given gene:

In [8]:
```
c.draw('Dnmt3b');
```



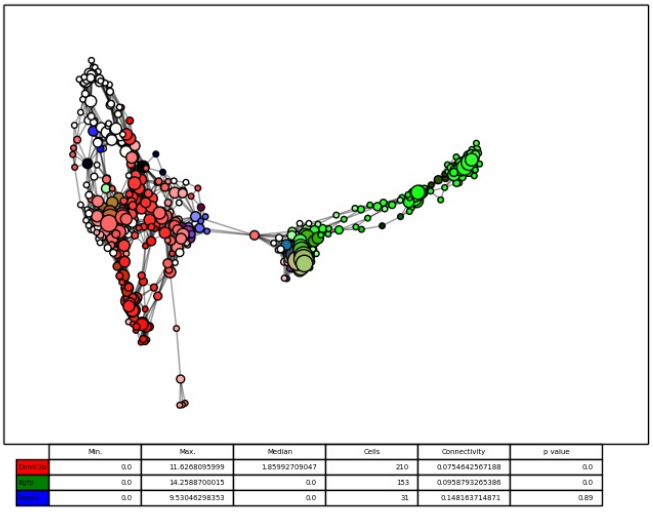
Node sizes are proportional to the number of cells in the node. The method `SCTDA.RootedGraph.draw()` also allows to map a gene or list of genes to red, green and blue channels:

In [9]:
```
c.draw(['Dnmt3b','Egfp', 'Foxa1']);
```



With the option `table=True`, `SCTDA.RootedGraph.draw()` will also display some statistics (see below for a more detailed description of each quantity):

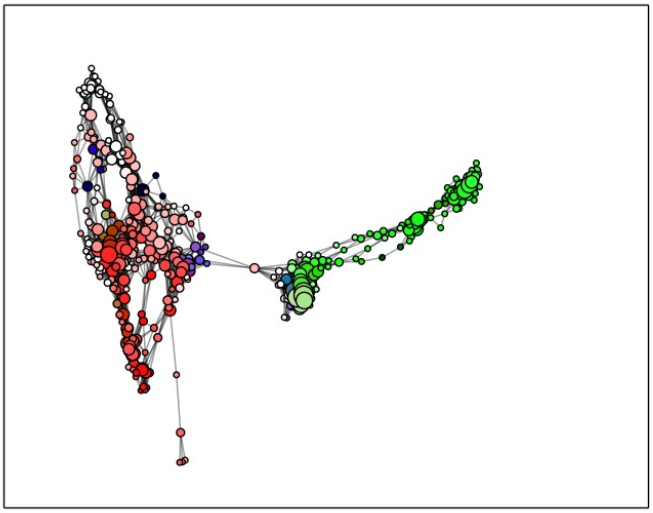
```
In [10]: c.draw(['Dnmt3b','Egfp', 'Foxa1'], table=True);
```



	Min.	Max.	Median	Cells	Connectivity	p value
Dnmt3b	0.0	11.6268095999	1.85992709047	210	0.0754642567188	0.0
Egfp	0.0	14.2588700015	0.0	153	0.0958793265386	0.0
Foxa1	0.0	9.53046298353	0.0	31	0.148163714871	0.89

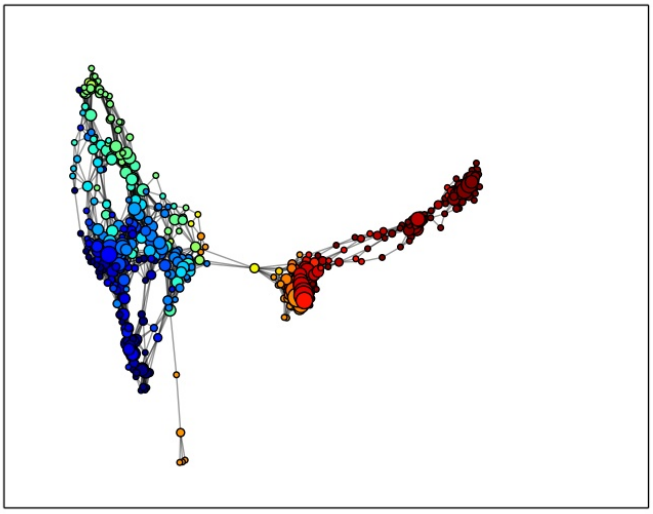
`SCTDA.RootedGraph.draw()` is not limited to single genes, it can also map lists of genes to a channel:

```
In [11]: c.draw([['Dnmt3b', 'Dppa2', 'Dnmt3l'],'Egfp', 'Foxa1']);
```



`SCTDA.RootedGraph.draw()` also accepts some special keywords. The keyword `'timepoint'` can be used to color according to the sampling time point:

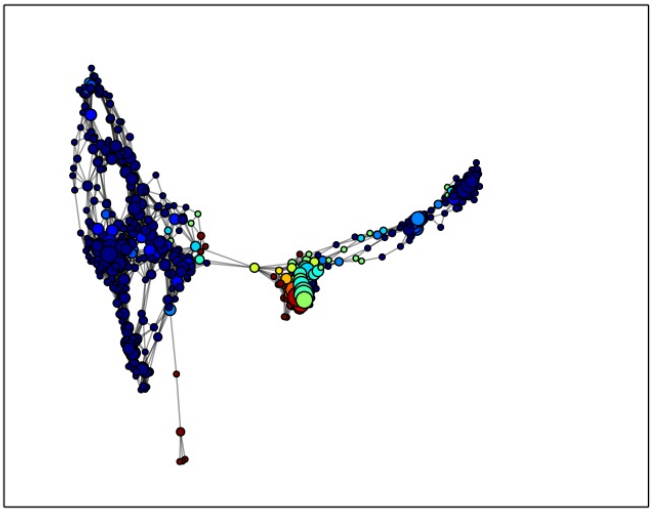
In [12]:
```
c.draw('timepoint');
```



A remarkable feature of the topological representation, based just on expression data, is that it correctly reproduces the differentiation time course, as observed in the above figure.

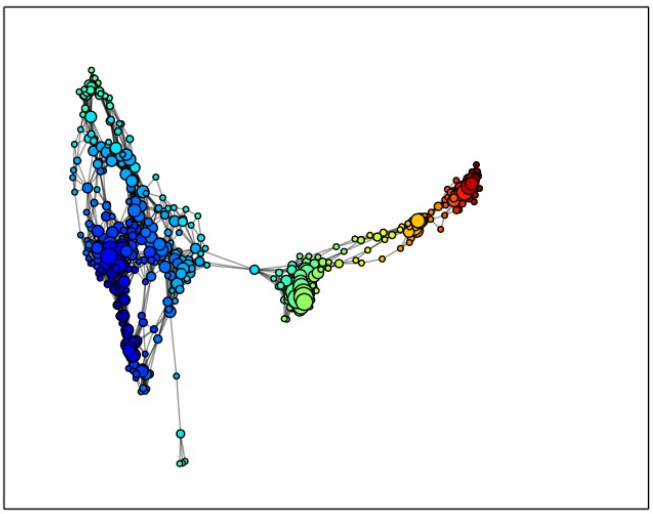
Similarly, the keyword `'timepoint_'` can be used to plot the density of cells of a given time point,

In [13]:
```
c.draw('timepoint_5');
```



SCTDA determines a root node in the topological representation by maximizing the correlation between sampling time points and the graph distance function. The root node corresponds to the less differentiated state. We can color according to the distance to the root node using the keyword `'_dist_root'`,

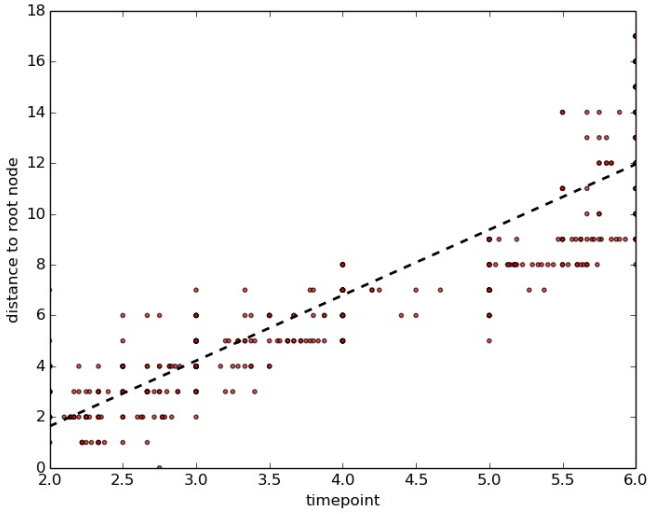
```
In [14]: c.draw('_dist_root');
```



Correlation between sampling time point and distance to root node can be visualized using `SCTDA.RootedGraph.plot_rootlane_correlation()`,

```
In [15]: c.plot_rootlane_correlation()
```

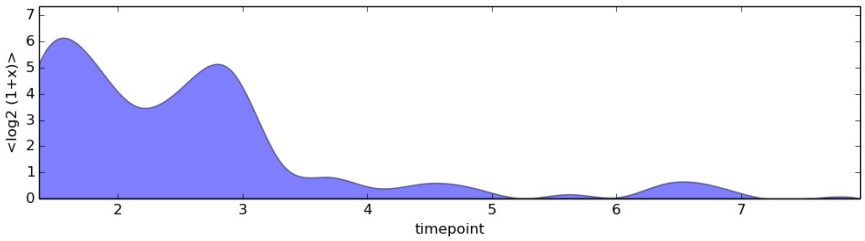
```
Out [15]: (2.579181287149348,
           -3.5195814841484552,
           0.89109027529004081,
           2.2097001444641111e-141,
           0.065190844804728809)
```



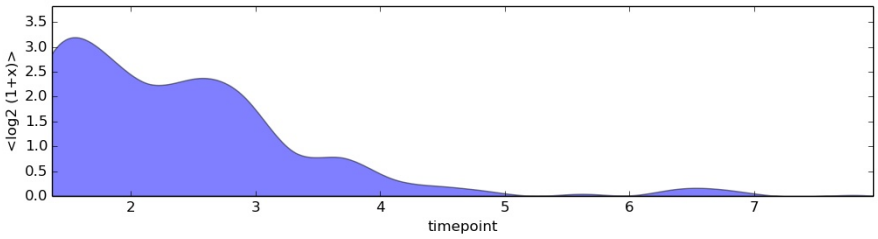
`SCTDA.RootedGraph.plot_rootlane_correlation()` returns the two parameters of the linear fit, Pearson's r, p-value and the standard error.

The method `SCTDA.RootedGraph.draw_expr_timeline()` can be used to draw the expression of a gene or list of genes at different time points, as inferred from the distance to root function,

```
In [16]: c.draw_expr_timeline('Dnmt3b');
```

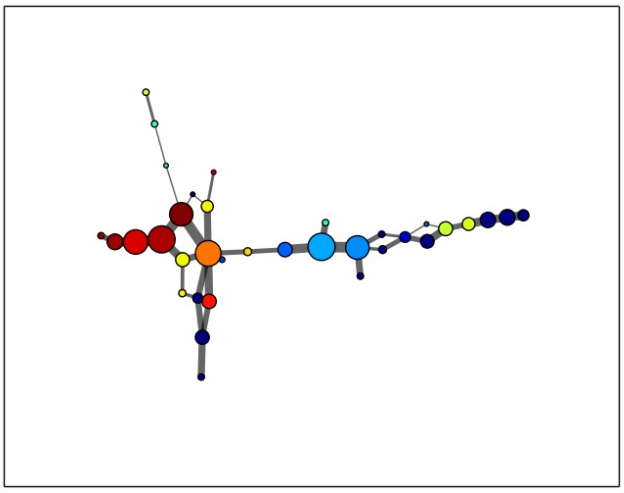



```
In [17]: c.draw_expr_timeline(['Dnmt3b', 'Dppa2', 'Dnmt3l', 'Dppa4']);
```



Finally, we can display a skeleton of the differentiation tree using `SCTDA.RootedGraph.draw_skeleton()`, colored according to a gene or list of genes, where each node in the skeleton corresponds to a set of connected nodes at the same distance of the root node in the topological condensed representation. Node sizes are proportional to the number of cells in the node, whereas edge sizes are proportional to the number of edges in the topological condensed representation connecting each pair of group of nodes.

```
In [18]: c.draw_skeleton('Dnmt3b');
```



2.2 Connectivity, centroid and dispersion

There are several quantities that can be associated to each gene and that allow to identify genes that are specific of particular cell sub-populations or time points. The first quantity that we may consider is the connectivity of the expression of a gene in the topological representation. This is defined as,

$$S(g) = \frac{N-1}{N} \sum_{i,j} p_i(g) w_{ij} p_j(g)$$

where the sum runs over all nodes in the topological condensed representation, w is the adjacency matrix of the topological condensed representation, N the total number of nodes and $p_i(g)$ the average expression of gene g on node i, normalized as a probability, namely

$$\sum_i p_i = 1$$

Connectivity allows to identify genes whose expression appears as highly connected in the topological representation. The connectivity of a gene can be computed using the method `SCTDA.RootedGraph.connectivity()`,

```
In [19]: c.connectivity('Lhx3')
```

```
Out [19]: 0.12959353154921011
```

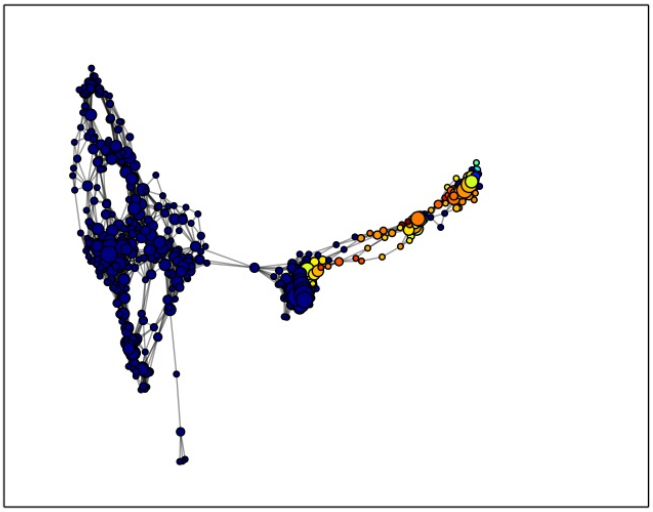
The statistical significance of a particular connectivity score can be assessed by performing a permutation test, where cell labels are randomly permuted. This is implemented in the method `sctda.RottedGraph.connectivity_pvalue()`,

```
In [20]: c.connectivity_pvalue('Lhx3', n=5000)
```

```
Out [20]: 0.0
```

where n specifies the number of permutations. The idea is that genes with a statistically significant connectivity, like the above example, are biologically relevant at a particular stage or for a particular cell sub-population,

```
In [21]: c.draw('Lhx3');
```

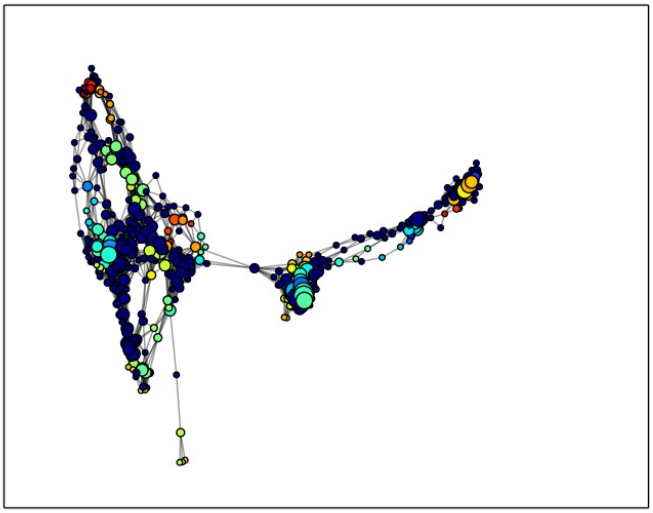


On the contrary, genes with a non-significant connectivity in the condensed topological representation are unspecific of any stage or cell sub-population. For instance,

```
In [22]: c.connectivity_pvalue('St6gal1', n=5000)
```

```
Out [22]: 0.99880000000000002
```

```
In [23]: c.draw('St6gal1');
```



Other interesting quantities are the centroid of a gene with respect to the root node and its dispersion. These are respectively defined as

$$C(g) = \sum_i d_i p_i(g)$$

$$D(g) = \sqrt{\sum_i (d_i - C(g))^2 p_i(g)}$$

where d_i is the distance of node i to the root node in the topological representation.

The centroid and dispersion of a gene or list of genes can be computed with the method `SCTDA.RootedGraph.centroid()`,

```
In [24]: c.centroid('Lhx3')
```

```
Out [24]: [12.60030777275247, 2.5465951516563292]
```

Other useful methods are `SCTDA.RootedGraph.expr()`, that returns the number of cells on which expression of a gene or list of genes is detected,

```
In [25]: c.expr('Lhx3')
```

```
Out [25]: 93
```

and `SCTDA.RootedGraph.delta()`, that returns the mean, minimum and maximum expression values of a gene or list of genes,

```
In [26]: c.delta('Lhx3')
```

```
Out [26]: (1.5655427871802245, 0.0, 11.110091334383007)
```

The methods `SCTDA.RootedGraph.expr()` and `SCTDA.RootedGraph.delta()` can be restricted to specific groups of nodes. Groups of nodes are specified in the file `name.groups.json` and stored in the dictionary `SCTDA.RootedGraph.dicgroups`. For instance, in the example of this tutorial we have defined three groups of nodes in the file `analysis1.groups.json`. These can be accessed through

```
In [27]: c.dicgroups.keys()
```

```
Out [27]: [u'Group_3', u'Group_2', u'Group_1']
```

We can restrict the commands `SCTDA.RootedGraph.expr()` and `SCTDA.RootedGraph.delta()` to any of these groups by using the argument `group=`. For instance,

```
In [28]: c.expr('Lhx3', group='Group_2')
```

```
Out [28]: 22
```

```
In [29]: c.delta('Lhx3', group='Group_2')
```

```
Out [29]: (1.8189051999362191, 0.0, 8.0067929633825727)
```

The method `SCTDA.RootedGraph.save()` can be used to create a file called `name.genes.tsv` containing a tab separated table with all the above quantities for all genes in the dataset, including also Bejamini-Holchberg adjusted p-values for the connectivity. This method allows to filter genes in that are expressed in more than `filtercells` cells, and whose maximum expression value is above `filterexp` in $\log_2(1 + \text{TPM})$ units. In addition, it allows to annotate genes according to one or more lists of genes.

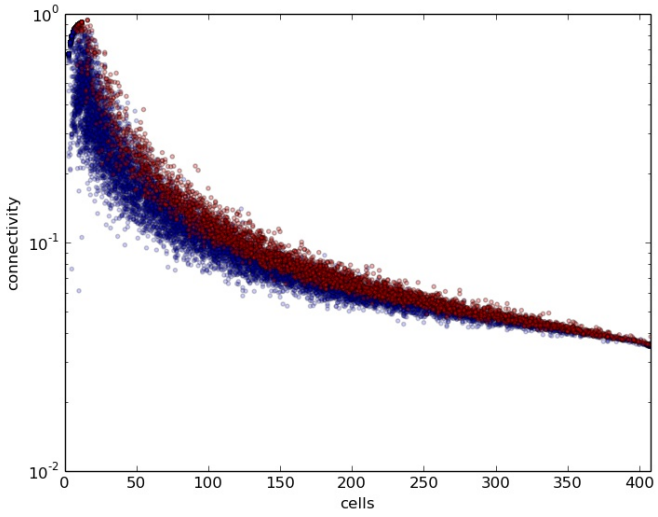
In the following example, we have downloaded from EMBL-EBI QuickGO two tables listing the members of gene ontologies *RNA splicing* and *Poly-(A) RNA binding*. We can parse the genes in those tables into a dictionary that will be used by the command `SCTDA.RootedGraph.save()` to annotate the genes:

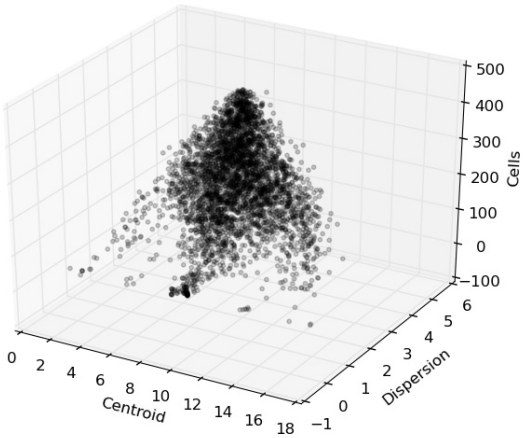
```
In [30]: splic = []
         rna = []

         f = open('GO0008380_RNA_spliciing.tsv', 'r')
         for nb, line in enumerate(f):
             if nb > 0:
                 sp = line.split('\t')
                 splic.append(sp[2])
         f.close()
         splic = list(set(splic))

         f = open('GO0044822_polyA_RNA_binding.tsv', 'r')
```

```
for nb, line in enumerate(f):
    if nb > 0:
        sp = line.split('\t')
        rna.append(sp[2])
f.close()
rna = list(set(rna))

annotations = {'Splicing': splic, 'RNA_binding': rna}
```

We use the method `SCTDA.RootedGraph.save()` to create the file `analysis1.genes.tsv`, where we only consider genes that are expressed in at least 3 cells and statistical significance is assessed using 500 permutations,

In [31]:
```
c.save(n=500, filtercells=2, filterexp=0.0, annotation=annotations);
```





`SCTDA.RootedGraph.save()` produces two plots. The first plot presents connectivity against the number of cells with non-zero expression for each gene. Genes with statistically significant connectivity, after Benjamini-Holchberg adjustment for multiple testing, are displayed in red. The second plot displays the number of cells with non-zero expression against the centroid and dispersion, for genes with statistically significant connectivity.

In [32]:
```
!head analysis1.genes.txt
```

```
      Gene     Cells     Mean    Min     Max  Connectivity p_value BH p-value Centroid Dispersion RNA_binding Splicing
0610005C13Rik   14      0.1986   0.0   7.69687   0.4918     0.348    0.4982    2.1365    0.3433        N         N
0610007N19Rik   49      0.3235   0.0   6.67145   0.2319     0.008    0.0349    7.7496    1.5264        N         N
0610007P14Rik   224     2.7506   0.0   7.98458   0.0597     0.0      0.0       6.9506    3.6452        N         N
0610009B22Rik   215     2.9831   0.0   9.45939   0.0620     0.016    0.0590    6.3790    3.5972        N         N
0610009D07Rik   266     4.1988   0.0   10.2019   0.0469     0.004    0.0202    7.3379    4.0085        N         N
0610009L18Rik   8       0.0907   0.0   5.09394   0.8762     0.284    0.4361    8.6537    0.4757        N         N
0610009O20Rik   137     1.0194   0.0   7.18446   0.0896     0.072    0.1735    7.2715    4.0472        N         N
0610010B08Rik   32      0.0802   0.0   4.10256   0.2968     0.15     0.2865    7.1221    2.0711        N         N
0610010F05Rik   192     2.6658   0.0   10.0240   0.0632     0.31     0.4629    7.7714    4.2535        N         N
```

2.3 Adjacency and cell subpopulations

A natural extension of the concept of connectivity to pairs of genes is that of adjacency, defined as

$$A(g,h) = \frac{N-1}{N} \sum_{i,j} p_i(g) w_{ij} p_j(h)$$

Adjacency takes high values when the expression of genes g and h are adjacent to each other in the topological representation. Note that when $g = h$, adjacency becomes connectivity.

The adjacency matrix of a list of genes can be computed in SCTDA using the method,

In [33]:
```
a = c.adjacency_matrix(['Dnmt3b', 'Dppa2', 'Dnmt3l', 'Dppa4', 'Lhx3']);
print a
```

```
[[ 0.07546426  0.07197224  0.07238027  0.07201561  0.00710653]
 [ 0.07197224  0.09104796  0.08555381  0.07007939  0.        ]
 [ 0.07238027  0.08555381  0.11159461  0.07611521  0.        ]
 [ 0.07201561  0.07007939  0.07611521  0.08636352  0.0004148 ]
 [ 0.00710653  0.          0.          0.0004148   0.12959353]]
```

SCTDA makes use of adjacency and Jensen-Shannon distance to look for potential cell sub-populations,

$$J(g,h) = \left[\frac{1}{2} \sum_i \left(p_i(g) \log p_i(g) + p_i(h) \log p_i(h) - (p_i(g) + p_i(h)) \log \frac{p_i(g) + p_i(h)}{2} \right) \right]^{\frac{1}{2}}$$

Jensen-Shannon distance is implemented in SCTDA through the following method,

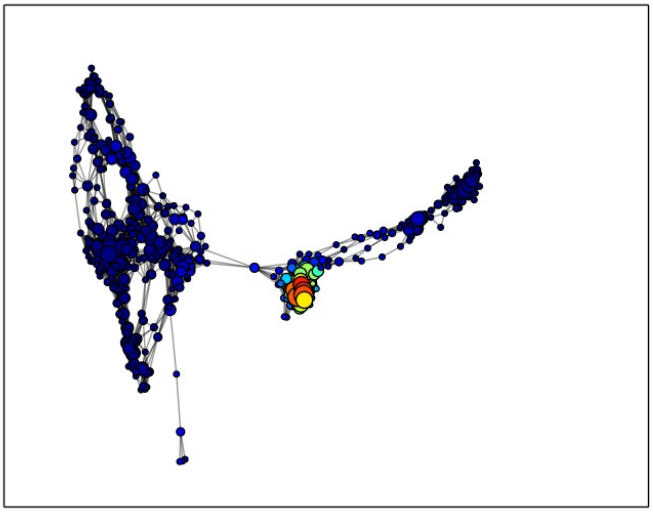
In [34]:
```
b = c.JSD_matrix(['Dnmt3b', 'Dppa2', 'Dnmt3l', 'Dppa4', 'Lhx3'])
print b
```

```
[[ 0.          0.55105197  0.62788196  0.55417842  0.95451214]
 [ 0.55105197  0.          0.63299325  0.60159545  1.        ]
 [ 0.62788196  0.63299325  0.          0.66150927  1.        ]
 [ 0.55417842  0.60159545  0.66150927  0.          0.99758009]
 [ 0.95451214  1.          1.          0.99758009  0.        ]]
```

Adjacency and Jensen-Shannon distance are used by `SCTDA.RootedGraph.candidate_subpopulations()` to look for potential cell sub-populations in a rather unsupervised way. We end this tutorial by illustrating the use of this method. The 3-dimensional plot produced by `SCTDA.RootedGraph.save()` for our particular example showed three "flares" corresponding to three sets of genes with low dispersion and statistically significant connectivity corresponding to centroids in three different ranges. Let us consider the second group (corresponding to progenitor motor neurons),

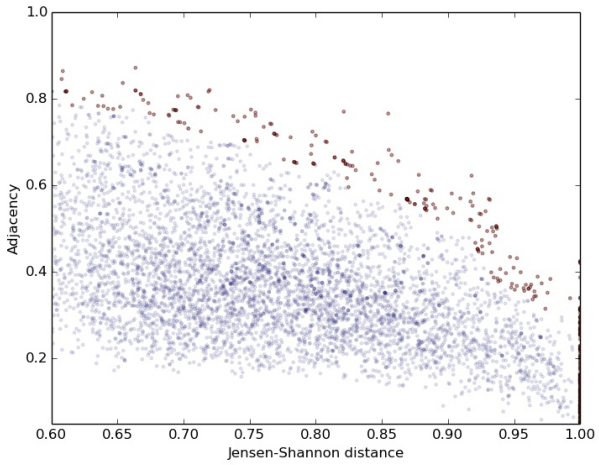
In [35]:
```
group2 = []
f = open('analysis1.genes.txt', 'r')
for n, line in enumerate(f):
    if n>0:
        sp = line[:-1].split('\t')
        if float(sp[7]) <= 0.05 and 7.5 < float(sp[8]) < 8.8
                                        and float(sp[9]) < 1.9:
            group2.append(sp[0])
f.close()
```

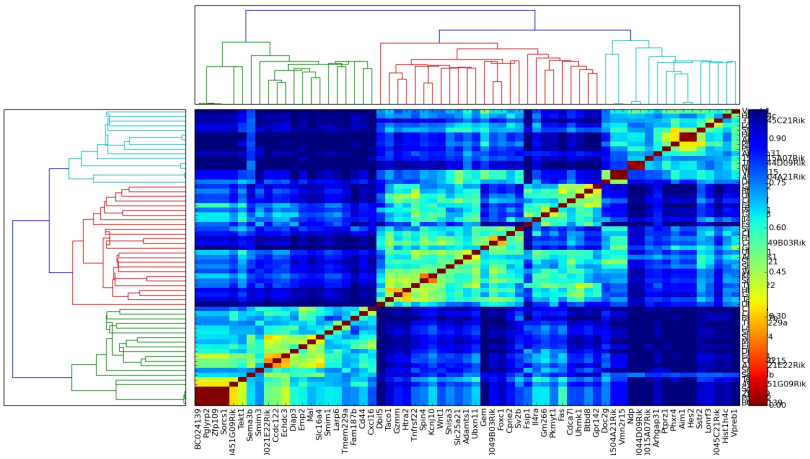
In [36]:
```
c.draw([group2]);
```



A list of potential cell sub-populations based on these genes can be produced using the method `SCTDA.RootedGraph.candidate_supopoulations()`,

In [37]:
```
u = c.candidate_subpopulations(group2, thres=0.05);
```



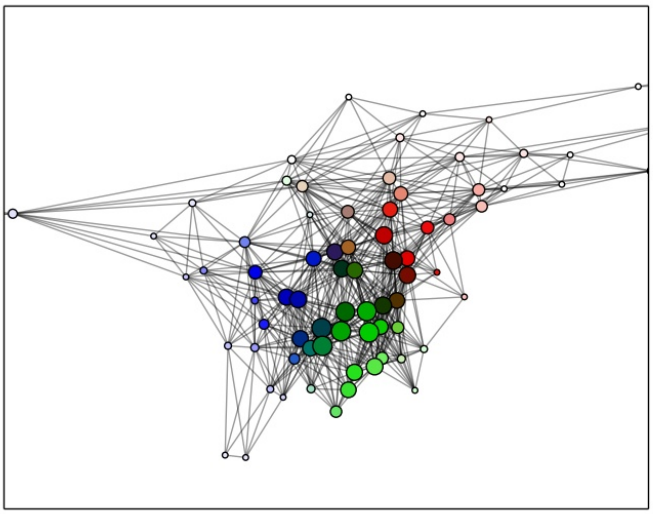


`SCTDA.RootedGraph.candidate_supopoulations()` computes the adjacency and Jensen-Shannon matrices of the genes in the list and looks for pairs of genes that have both high adjacency and high Jensen-Shannon distance. A threshold in this 2-dimensional space is set using the option `thres`, taking values between 0 and 1. The first plot produced by `SCTDA.RootedGraph.candidate_supopoulations()` displays the distribution is this 2-dimensional space of all considered pairs of genes, with pairs passing the threshold marked in red. Genes that appear in pairs of genes that pass the threshold are then clustered using hierarchical clustering and Jensen-Shannon distance. In our particular example, three potential types/stages of progenitor motor neurons are detected. At the end of this process, `SCTDA.RootedGraph.candidate_supopoulations()` returns a list with the genes in each of the clusters defining potential cell sub-populations,

```
In [38]: print u
```

```
[['4921504A21Rik', 'Vmn2r15', 'Doc2g', 'Ndp', '1810044D09Rik', 'Aim1', 'Hes2',
'Phxr4', 'Ptprz1', 'Lonrf3', '3110045C21Rik', 'Hist1h4c', 'Vpreb1', 'Sstr2',
'Arhgap31', '1500015A07Rik'], ['Gzmm', 'Htra2', 'Taco1', 'Spin4', 'Kcnj10',
'Adamts1', 'Ubxn11', 'Slc25a21', 'Shisa3', 'Wnt1', 'Tnfrsf22', 'C430049B03Rik',
'Foxc1', 'Gem', 'Cpne2', 'Sv2b', 'Gm266', 'Pkmyt1', 'Btbd8', 'Gpr142', 'Uhmk1',
'Cdca7l', 'Fas', 'Il4ra', 'Fsip1', 'Dbil5'], ['Zfp109', 'Sorcs1', 'Pglyrp2',
'BC024139', '4930451G09Rik', 'Tekt1', 'Ccdc122', 'Echdc3','A330021E22Rik',
'Mal', 'Slc16a4', 'Emp2', 'Diap3', 'Larp6', 'Tmem229a', 'Smim1', 'Cd44',
'Cxcl16', 'Fam187b', 'Smim3', 'Sema3b']]
```

In [39]:
```
c.draw(u);
```



References

[1] Rizvi, A., Camara, P. G., Kandror, E., Rabadan, R. and Maniatis, T., *Title of paper*. In preparation.