Exploring the Differentially Expressed Genes in the Life Stages of the Jellyfish *Aurelia*

Clive Lau

Abstract

Cnidarians are a phylogenetically interesting group of organisms when it comes to the study of the evolution of animal complexity. RNA-seq has become a widely used technique to analyze the expression of developmental genes in the ontogeny of an organism. Exploration of RNA-seq data in cnidarians can yield insights and help guide future research directions in the evo-devo of the complex life cycles characteristic of this clade. In this project, a bioinformatic pipeline is developed to generate graphics that could help interpret differential gene expression data from RNA-seq. A demonstration of this pipeline is done using example genes in the jellyfish *Aurelia sp. 1.*

Introduction

The phylum Cnidaria, which includes the corals, sea anemones, box jellies, and true jellies, is the sister clade to the bilaterians. Due to its position on the metazoan tree of life, the cnidarians can offer particular insights to major innovations in the evolution of animal complexity. Recently, the Jacobs lab and collaborators have sequenced the genome and transcriptome of the jellyfish *Aurelia sp. 1. Aurelia* belongs to a clade of cnidarians known as the medusozoans; compared to their sister clade, the anthozoans, the medusozoans exhibit a complicated life cycle characterized by having a mobile adult life stage known as the medusa (Fig. 1). The emergence of this complicated life cycle is poorly understood, and a detailed stage-specific differential expression study is needed to parse out the genetic mechanisms for this evolutionary innovation.

The goal of this project is to develop a programmatic pipeline to perform some preliminary explorations of the *Aurelia* transcriptome, with the aim to find out more about changes in the expression patterns of certain genes across the life cycle of this jellyfish. Previous analyses done by the Jacobs lab have identified eight clusters of genes that are differentially expressed across the life stages of *Aurelia*. Here I present a series of scripts in a pipeline that will enable one to broadly look at the expression characteristics of the genes in each of these clusters, as well as certain, more specific, genes of interest. All scripts are available on GitHub (https://github.com/clflau/eeb-174-final-project).

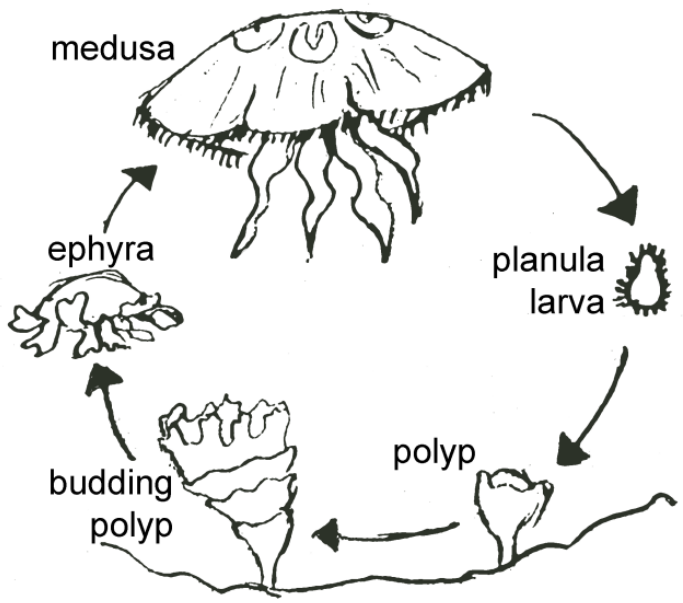


Figure 1: The life cycle of *Aurelia*

Graphically presenting gene expression across stages

The expression counts data of all differentially expressed (DE) genes in the transcriptome is stored in the file "Aurelia_RSEM_10-18-16.TMM.EXPR.100aa.matrix". A convenient way to visualize DE gene data across life stages is to graphically present the expression levels across stages. Here I demonstrate how this can be achieved using the cluster 3 genes as an example. Previous Gene Ontology analysis done by the Jacobs Lab has shown that cluster 3 of the differentially expressed genes are enriched in "eye development" genes. To extract only the cluster 3 genes, the python script `gene_cluster_finder.py`, once executed, will prompt the user to enter the desired cluster (a number from 1 to 8), and outputs a .txt file containing a list evm gene model numbers of the genes within the desired cluster. The content of this python script is reproduced below:

```
#####
# This script will search the file 'Cluster_IDs.txt' with input from
# user for a specific gene cluster, and will output the gene evm model
# IDs of genes within that cluster.
#####

clust_num = input("Enter cluster number (1 to 8): ")

def gene_cluster_finder():
    import re
    cluster = "Clust" + clust_num
    gene_IDs = []
```

```
    with open("Cluster_IDs.txt") as clusterIDs:
        for line in clusterIDs:
            if re.search(cluster, line):
                cluster_search = re.search(cluster + "\t*(.*)\n", line)
                gene_IDs.append(cluster_search.group(1))
    return gene_IDs


gene_IDs = gene_cluster_finder()

ticker = 0
outfile = open("./data/cluster{}_genes.txt".format(str(clust_num)), "w")
outfile.write("# Genes of interest\n")
outfile.close()
outfile = open("./data/cluster{}_genes.txt".format(str(clust_num)), "a")
while ticker < len(gene_IDs):
    outfile.write(gene_IDs[ticker] + "\n")
    ticker = ticker + 1

outfile.close()

print("Results stored in ./data/cluster{}_genes.txt".format(str(clust_num)))
```

Running `gene_cluster_finder.py` to capture the cluster 3 genes will return a file in the `data` folder called `cluster3_genes.txt`. The next step is to obtain the counts data for these genes; this is done by running the bash script `Aurelia_gene_count_finder.sh`:

```
#!/bin/bash/

#####
# This bash script takes 2 arguments. The first argument is the name
# of the file that contains the list of evm model IDs to use as query;
# the second argument is the name of the output file on which the
# results of this script will write.
#####

list_of_genes=$(tail -n +2 ./data/$1 |cut -d "," -f 1 | sort | uniq)

head -n 1 Aurelia_RSEM_10-18-16.TMM.EXPR.100aa.matrix | cut -c 2- > ./data/$2

for gene in $list_of_genes
    do
        echo $gene
        grep -m1 $gene ./Aurelia_RSEM_10-18-16.TMM.EXPR.100aa.matrix >> ./data/$2
done
```

```
echo "Results stored in ./data/$2"
```

Since the gene expression counts data contains biological replicates for each of the life stages, a simple way to aggregate the data is by taking the average of the biological replicates for each life stage. This can be accomplished with the python script `replicate_averager.py`:

```
"""
This python code will take the Aurelia expression count data
and calculate the average expression levels across the biological
replicates of each of the life stages of Aurelia.
"""

# First read in the file with the expression count data
# Data should be structured as tab separated file with 19 columns
# First column is geneID
# Next two columns are for 'early planula'
# Next two columns are for 'late planula'
# Next three columns are for 'polyps'
# Next three columns are for 'early strobila'
# Next three columns are for 'late strobila'
# Next three columns are for 'ephyra'
# Final two columns are for 'juvenile'

import numpy as np
import pandas as pd

file_to_open = input("Enter the name of the input file: ")
with open(file_to_open) as expression_data:
    expression_data.readline()
    data_array = pd.read_csv(file_to_open, sep = "\t", header = 0)


e_plan_mean = data_array.iloc[ : , [0, 1]].mean(axis = 1).round(4)
l_plan_mean = data_array.iloc[ : , [2, 3]].mean(axis = 1).round(4)
polyp_mean = data_array.iloc[ : , [4, 5, 6]].mean(axis = 1).round(4)
e_strob_mean = data_array.iloc[ : , [7, 8, 9]].mean(axis = 1).round(4)
l_strob_mean = data_array.iloc[ : , [10, 11, 12]].mean(axis = 1).round(4)
ephyra_mean = data_array.iloc[ : , [13, 14, 15]].mean(axis = 1).round(4)
juven_mean = data_array.iloc[ : , [16, 17]].mean(axis = 1).round(4)


mean_matrix = pd.concat([e_plan_mean, l_plan_mean, polyp_mean, e_strob_mean, l_strob_mea
mean_matrix.columns = ["early planula", "late planula", "polyp", "early strobila", "late
```

```
file_to_write_to = file_to_open[0:7] +
  "mean_" + file_to_open[7:-3] + "csv"
mean_matrix.to_csv(file_to_write_to, sep = '\t')

print("Results stored in " + file_to_open[0:7] + "mean_" +
        file_to_open[7:-3] + "csv")
```

The output of this `replicate_averager.py` is a `.csv` file containing the average per stage expression counts for each gene. At this point, the data is ready to be visualized graphically by a lineplot. This is done by loading the data into `R`. The function `lineplot_stages()` is stored in the `R` script file `lineplot_stages.R`. Because some genes are much more highly expressed than others, this function log transforms the gene count data and center the data around the gene median using the equation:

$$x_{ij,centered} = log_2(x_{ij} + 1) - log2(m_i) \tag{1}$$

where x_{ij} is the replicate-average gene count for gene i at stage j and m_i is the median of the replicate-average gene count for gene i. Since $log_2(0)$ is undefined, 1 is added to all the data before the log transformation. Since we are only interested in the relative changes of expression across the stages, this procedure does not alter the interpretations of the results. The content of `lineplot_stages()` is reproduced below:

```
source("lineplot_stages.R", echo = TRUE, max.deparse.length = 10000)
```

```
##
## > lineplot_stages <- function(file, gene_group = "",
## +     plot_mean = T) {
## +     library(matrixStats)
## +     library(reshape2)
## +     library(ggplot2)
## +     file_data <- read.csv(file, sep = "\t")
## +     names(file_data) <- c("Gene", "E. Planula", "L. Planula",
## +         "Polyp", "E. Strobila", "L. Strobila", "Ephyra", "Juvenile")
## +     row_median_vector <- rowMedians(as.matrix(file_data[2:8] +
## +         1))
## +     file_data_cent <- log2(file_data[2:8] + 1) - log2(row_median_vector)
## +     file_data[2:8] <- file_data_cent
## +     df1 <- melt(file_data, id = c("Gene"), variable.name = "Stages")
## +     p1 <- ggplot() + geom_line(data = df1, aes(x = Stages, y = value,
## +         group = Gene, color = Gene), alpha = 0.2) + geom_point(data = df1,
## +         alpha = 0.1, aes(x = Stages, y = value, color = Gene,
## +             group = Gene), size = 0.5) + theme(legend.position = "none") +
## +         ggtitle(paste(gene_group, "Gene expression across stages",
## +             sep = " ")) + ylab("Centered log2(TMM+1)")
```

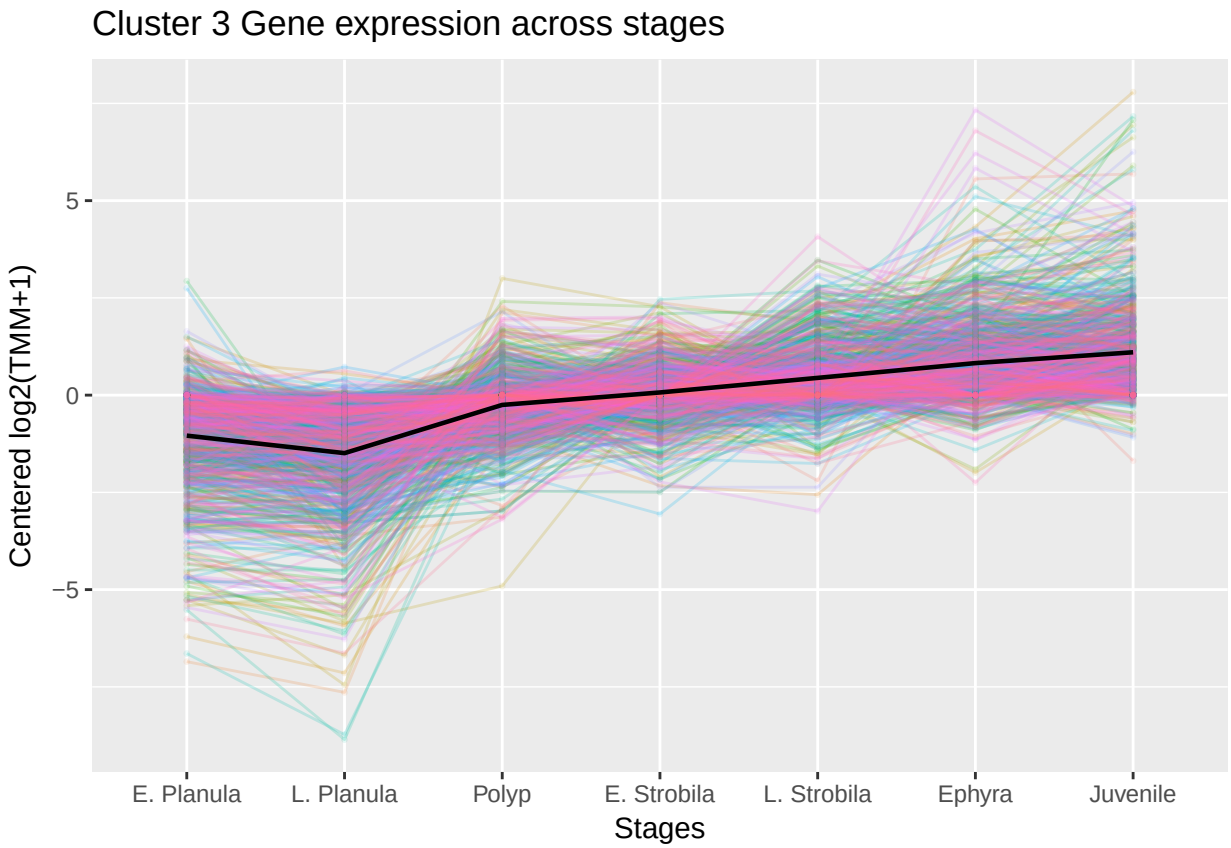
```
## +     if (!plot_mean) {
## +         return(p1)
## +     }
## +     else {
## +         stages <- levels(df1$Stages)
## +         stage_means <- c()
## +         for (stage in stages) {
## +             stage_means <- c(stage_means, mean(df1[which(df1$Stages ==
## +                 stage), 3]))
## +         }
## +         stage_means <- cbind.data.frame(stages, stage_means)
## +         stage_means_max <- stage_means[which(stage_means$stage_means ==
## +             max(stage_means$stage_means)), 1]
## +         stage_means_min <- stage_means[which(stage_means$stage_means ==
## +             min(stage_means$stage_means)), 1]
## +         cat("stage with highest mean expression level:", as.character(stage_means_m
## +             "\n")
## +         cat("stage with lowest mean expression level:", as.character(stage_means_mi
## +             "\n")
## +         p1 <- p1 + geom_line(data = stage_means, aes(x = stages,
## +             y = stage_means, group = 1), color = "black", size = 0.8)
## +         return(p1)
## +     }
## + }
```

Calling `lineplot_stages()` on the cluster 3 genes will produce a line plot for each of the genes. Additionally, the mean expression across stages of all the graphed genes is drawn as a black line:

```
lineplot_stages("./data/mean_cluster3_counts.csv", "Cluster 3")
```

```
## matrixStats v0.51.0 (2016-10-08) successfully loaded. See ?matrixStats for help.
```

```
## stage with highest mean expression level: Juvenile
## stage with lowest mean expression level: L. Planula
```



Line plots for all eight clusters are shown in Fig. 2.

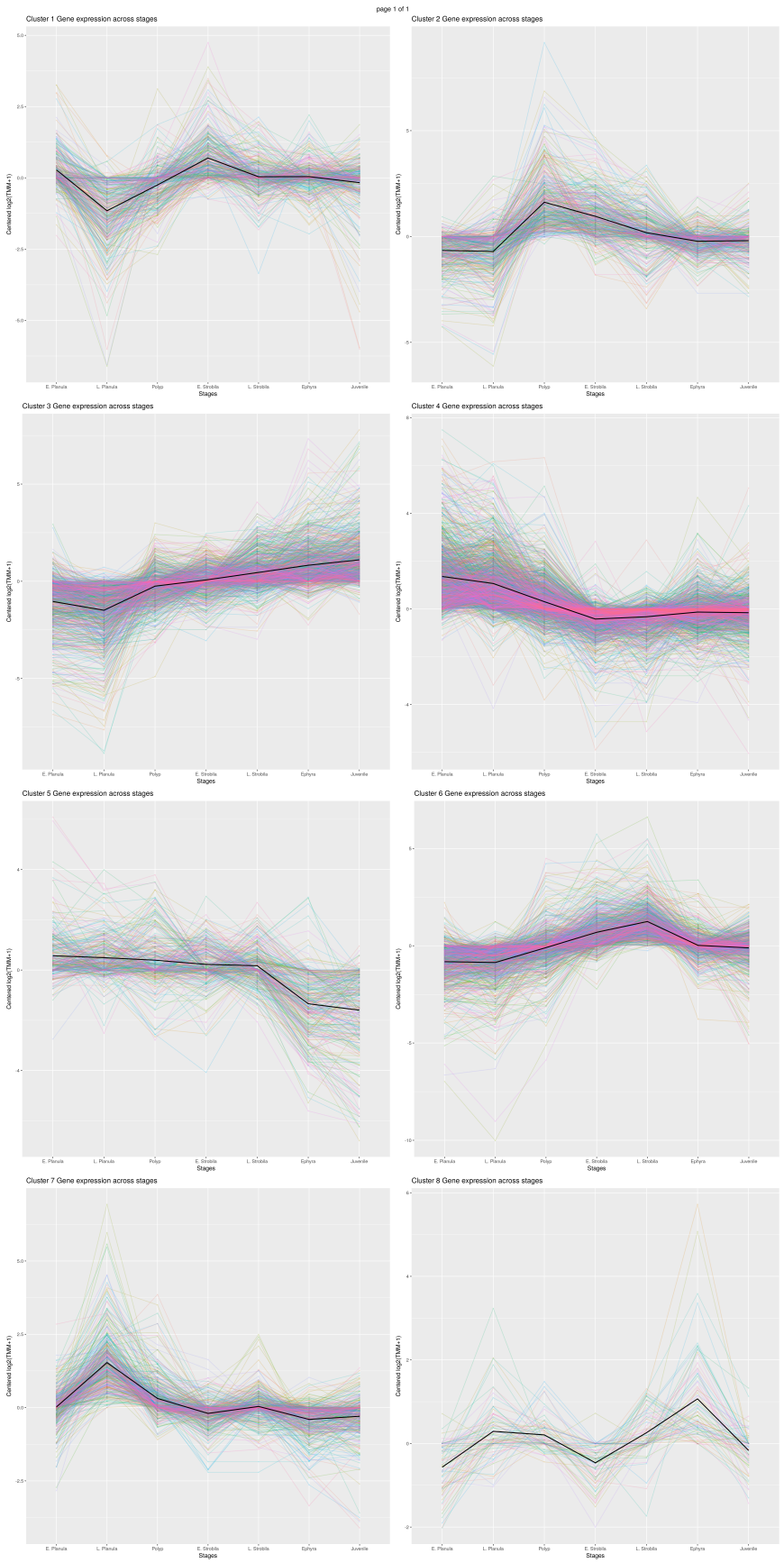


Figure 2: Line plots for all eight clusters

Examining the expression of a specific family of genes

Oftentimes it is interesting to look at a specific family of genes to see how they are potentially associated in the development of an organism. A simple way to explore this is by visualizing using a heatmap. Here I present a workflow/pipeline to achieve this, using the POU gene family as example. The POU gene family is a highly conserved family of transcription factors that contains the POU DNA-binding domain. Many genes in this gene family are critical in the development of the nervous systems and nerve nets (Okamoto *et al.* 1993, Anderson *et al.* (1995)). This gene family is found in many vertebrate and invertebrate taxa, and POU homologs are thought to have evoloved before the last common ancestor of Metazoa (Gold *et al.* 2014). Homologs of two subclasses of POU genes, Pit1 and Brn3, have been discovered to be expressed in the sensory organs of *Aurelia* (Nakanishi *et al.* 2010); it is unclear whether other homologs are expressed in this taxon as well. The following is a demostration of a pipeline to explore this expression of gene family in *Aurelia*.

We start by searching for POU-related genes on the UniProt protein database. This is done by running the script `webscrape_swissprot.sh`:

```
#!/bin/bash/

#' This bash script takes as argument a string to use as the query
#' tothe uniprot website. It returns a tab separated file with
#' unique protein identifiers to be used to search within the
#' annotated Aurelia gene models

query_url="http://www.uniprot.org/uniprot/?query="
query_url+=$1
query_url+="+AND+reviewed:yes&sort=score&columns=id,entry%20name&format=tab"

curl $query_url > ./data/uniprot-$1.txt
echo "results stored in ./data/uniprot-$1.txt"
```

The results file `./data/uniprot-POU.txt` contains two columns; the first contains the unique SwissProt accession IDs of the genes, the second column contains the gene names and the associated organism. To proceed to the next step, we need to extract only the first column of this file; this is done by running `swissprot_id_grabber.sh`, which is a simple cut command.

```
#!/bin/bash/

# This bash script takes the output of the swissprot webscraping
# and grabs only the swissprot ID

tail -n +2 $1 | cut -f 1 > accID.txt
echo "Results stored in ./data/accID.txt"
```

Because the expression count data file identifies genes by the evm gene model

numbers, we have to match the SwissProt IDs with the respective evm numbers using `search_within_trinotate.sh`; this information is stored in the file `trinotate_annotation_report.11_8_16.csv`.

```
#!/bin/bash/

# This bash script will batch search within the annotated
# gene model file for the evm numbers

query=$(cat $1 | sort -g | uniq)

echo "# quarry results using source file: $1" > ./data/outfile.txt

for item in $query
    do
        echo $item
        grep $item trinotate_annotation_report.11_8_16.csv | cut -d "," -f 1 >> ./data/o
done

echo "Results stored in ./data/outfile.txt"
```

At this point, we have a file containing a list of genes identified by their evm numbers. This can then be used as input in the pipeline shown in the previous section starting with the script `Aurelia_gene_count_finder.sh`. After `replicate_averager.py` is run, one addtional optional step is to replace the evm model names by the gene names for the convenience of reading the final results. This is done by running `replace_evm.py`:

```
# This python script replaces gene model names in expression count files with the Swis
# Swissprot annotations are the top blastx hits of the gene model sequences; these are

# The annotation.csv file is created using the following shell pipeline:
# cat trinotate_annotation_report.11_8_16.csv | cut -d "," -f 1,3 | sed 's/\"//g' > an

# This script requires the following files:
# annotation.csv - contains the gene model names and top blastx Swissprot hits
# an expression count file - contains the expression data of genes models


import sys
import re
import csv

with open("annotation.csv", newline = "\n") as annotations:
    annotations.readline()
    annotation_reader = csv.reader(annotations, delimiter = ",")
    # create dictionary of annotations
```

```
    model_name_list = []
    sprot_anno_list = []
    for row in annotation_reader:
        model_name_list.append(row[0])
        sprot_anno_list.append(row[1])
    annotations_dict = {}
    for ii in range(len(model_name_list)):
        annotations_dict[model_name_list[ii]] = sprot_anno_list[ii]


# In[55]:

with open(sys.argv[1], newline = "\n") as count_file: # replace with sys.argv[1] later
    header_line = count_file.readline().strip()
    count_reader = csv.reader(count_file, delimiter='\t')
    # pseudocode
    # for each row in count_reader, use row[0] as key in annotations_dict to find corr
    # if the returning dict value is '.', do nothing, continue to next row
    # if the returning dict value is NOT '.', replace row[0] with that dict value
    new_row_list = []
    for row in count_reader:
        if annotations_dict[row[0]] != '.':
            row[0] = annotations_dict[row[0]]
            new_row_list.append(row)
        else:
            new_row_list.append(row)

# What if multiple gene models Blast matched to the same Swissprot?

first_item = []  # make list of all gene model/Swissprot annotations
for item in new_row_list:
    first_item.append(item[0])


from collections import defaultdict

def list_duplicates(list_with_dups):
    tally = defaultdict(list)
    for i,item in enumerate(list_with_dups):
        tally[item].append(i)
    dup_list = []
    for key, index in tally.items():
        if len(index) > 1:
            dup_list.append(index)
```

```
    return dup_list

dup_list = list_duplicates(first_item)

# for all duplicate Swissprot names, add in a number at the end to make them all diffe
for item in dup_list:
    for ii in item:
        new_row_list[ii][0] = new_row_list[ii][0] + "({})".format(ii)




# Write to file
with open(sys.argv[1][:-4] + "_evm_replaced.csv", "a") as outfile: # replace with sys.a
    outfile.write("\t" + header_line + "\n")
    writer = csv.writer(outfile, delimiter = "\t")
    writer.writerows(new_row_list)

print("Results are stored in " + sys.argv[1][:-4] + "_evm_replaced.csv")
```

Hierarchical clustering on expression data

A commonn way to visualize the relationship between gene expressions is by constructing a heatmap based on hierarchical clustering (Eisen *et al.* 1998). This allows one to discern similar patterns of expression in a set of genes. We can construct a heatmap of the POU genes using the `R` function `heatmap.cordist()` stored in the script `hierarchical_clustering.R`.

```
heatmap.cordist <- function(file){
  file_data <- read.csv(file, sep = "\t")

  names(file_data) <- c("Gene", "E. Planula", "L. Planula", "Polyp",
                        "E. Strobila", "L. Strobila", "Ephyra", "Juvenile")
  rownames(file_data) <- file_data$Gene
  colnames(file_data)
  file_data <- file_data[, -1]

  # sort on max difference
  min <- apply(file_data, 1, min)
  max <- apply(file_data, 1, max)
  s_file_data <- file_data[order(max - min, decreasing = T), ]

  cor_matrix <- cor(t(s_file_data))
  cor_dist_matrix <- 1-cor_matrix
```

```
  plot(cor_dist_matrix, main = "Gene expression pairwise correlation distance",
       xlab = "row", ylab = "column")
  hc <- hclust(as.dist(cor_dist_matrix), method = "ward.D2")
#  plot(hc, cex = 0.6, hang = -1)
par(mai = c())
# cut tree at desired height (h = 0.5)
# gene_partition_assignments <- cutree(hc, h=0.5/100*max(hc$height))

  library(gplots)
  hp <- heatmap(as.matrix(s_file_data), Rowv = as.dendrogram(hc),
                Colv = NA, col = greenred(10), margins = c(6, 6),
                cexRow = 0.5, cexCol = 0.93)
  return(hp)
}
```

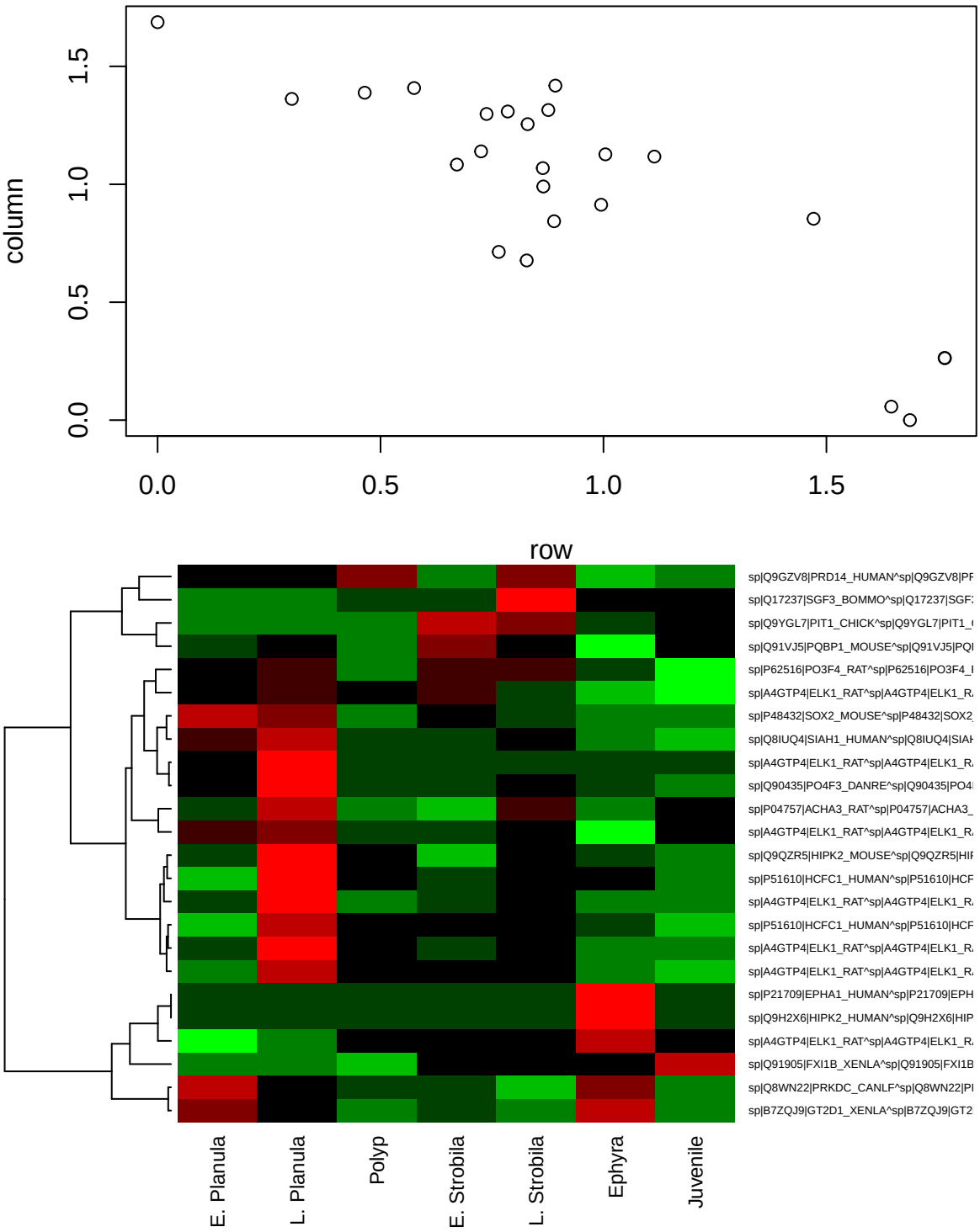
This function is written such that the clustering is done based on the Pearson correlations of gene expressions across the stages, rather than basing on the Euclidean distances. This is done to avoid clustering of genes by the absolute expression values, which would have the undesired effect of segregating highly expressed genes from lowly expressed genes without regard of changes in their patterns.

```
source("hierarchical_clustering.R")

heatmap.cordist("./data/mean_pou_gene_counts_evm_replaced.csv")
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```




```
## $rowInd
##  [1]  2 20 12  4 22 24  9  5 13  7 10 16 11 19  3 21  6 15  8 17  1 18 14
## [24] 23
##
```

```
## $colInd
## [1] 1 2 3 4 5 6 7
##
## $Rowv
## NULL
##
## $Colv
## NULL
```

Conclusion

I have demonstrated how DE genes from RNA-seq data could be visualized by different graphics, enabling one to gain a preliminary assessment of stage-wise expression patterns of genes in *Aurelia*. While the example data set used is specific to the species, the workflow describd above is not limited to any particular taxon, or indeed, any particular gene family. Other gene families such as the Usher Syndrome genes or genes associated with neurogenic placodes could be interesting candidates to explore.

References

Anderson, M.G., Perkins, G.L., Chittick, P., Shrigley, R.J. & Johnson, W.A. (1995). Drifter, a drosophila POU-domain transcription factor, is required for correct differentiation and migration of tracheal cells and midline glia. *Genes & Development*, **9**, 123–137.

Eisen, M.B., Spellman, P.T., Brown, P.O. & Botstein, D. (1998). Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, **95**, 14863–14868.

Gold, D.A., Gates, R.D. & Jacobs, D.K. (2014). The early expansion and evolutionary dynamics of POU class genes. *Molecular Biology and Evolution*, **31**, 3136–3147.

Nakanishi, N., Yuan, D., Hartenstein, V. & Jacobs, D.K. (2010). Evolutionary origin of rhopalia: Insights from cellular-level analyses of otx and POU expression patterns in the developing rhopalial nervous system. *Evolution & Development*, **12**, 404–415.

Okamoto, K., Wakamiya, M., Noji, S., Koyama, E., Taniguchi, S., Takemura, R., Copeland, N.G., Gilbert, D.J., Jenkins, N.A. & Muramatsu, M. (1993). A novel class of murine pou gene predominantly expressed in central nervous system. *Journal of Biological Chemistry*, **268**, 7449–7457.