Evolution of Usher Syndrome

Hayden Speck

March 23, 2017

Introduction

Usher Syndrome (USH), a genetic sensory disorder, is the most common cause of combined blindness-deafness in humans. The genes associated with Usher syndrome play key structural and functional roles in ciliated sensory cells of the vertebrate retina and inner ear. Usher genes form interciliary links and their anchoring complexes in photoreceptors and the mechanosensory hair cell (Kremer *et al.* 2006). When a mutation occurs in one of these genes, mechanotransduction is abolished and the retina degenerates, resulting in blindness, deafness and impaired vestibular function.

Given the key role these genes play in vertebrate sensory structures, it is concievable that these genes may serve similar sensory functions in other Metazoan groups. Previously thought to be confined to vertebrates, USH homologs have been identifed within the genome of the Echinoderm *Strongylocentrotus* (Sodergren *et al.* 2006). Recently, USH homologs have been shown to be upregulated in the choanocytes of the sponge *Ephydatia*, hinting that these genes may play a conserved role in the evolution of ciliated sensory structures of the Metazoa (Pena *et al.* 2016). The morphological similarity of verteberate inner ear hair cells and the ciliated structure of choanoflagelates has long been noted, and lends itself to the notion that these structures may be related by common descent (Fritzsch & Straka 2014). By investigating the evolutionary history of the genes involved in Usher syndrome, this project can better determine how the suite of genes involved with Usher syndrome were assembled within the Metazoa and its close relatives, and what role these genes might have played in the sensory evolution of early animals. This project begins to address these questions by finding likely Usher Syndrome homologs from animals and their close relatives.

Methods

Overview

Genetic sequences representing the longest known isoforms of the proteins were downloaded from NCBI (Benson *et al.* 2004). Biopython's NCBIWWW module was used to remotely querry the NCBI refseq_protein database using BLASTp to discover likely homologs within various Metazoa and closely related organisms (Altschul *et al.* 1990; Cock *et al.* 2009). BLAST and other sequence searching programs find similar sequences to the querry sequence and calculate an expected value ("e-value") evaluating the likelihood of the match occuring purely by chance. This e-value is based on the size of the database searched and size of the sequence used to querry the database. An "excess of similarity" (i.e. low e-value) is a strong indication for homology between two sequences (Pearson 2014). As protein based searches are better able to detect distant homologs than nucleotide based searches, an e-value cutoff of 0.00001 was used, although only the top hit or top several hits were used. For almost all cases, the e-values of the sequences used to build the trees were far below that of the cutoff used in the search.

Blast results were parsed by the Biopthon NCBIXML module, and likely homologs discovered by the BLAST search were retrieved from NCBI's Entrez databases using Biopython's Entrez module (Cock *et al.* 2009). Multiple sequence alignments were then created using MUSCLE, run through Biopython's MUSCLE wrapper (Edgar 2004, Cock *et al.* (2009)). Alignmets were used to generate maximum likelihood gene trees using RAxML (Stamatakis 2014). Gene tree files were visualized in R using the ape library, and annotation files for the trees were generated using a mix of python and bash commands (Paradis *et al.* 2004).

Data Description

Data consisted of FASTA formated protein sequences retrieved from remote databases. Protein sequences were used as they offered a greater chance of detecting distant homologs than nucleotide sequences, due to the greater selective pressure acting upon them (Pearson 2014). The increased sensitivity offered by protein sequences is particularly imortant as several of the taxa of searched in this report shared a common ancestor with vertebrates more than 500 million years ago (Erwin *et al.* 2011). Organism database were selected to widely sample within major Metazoan lineages, as well as Opisthokont groups more closely related to Animals than they are to Fungi. Unfortunately, not all lineages have a large number of taxa full genomes and complete protein repetoires in NCBI's sequence database, so some lineages were represented by a single taxa, and other groups of interest, such as Ctenophores were excluded from the analysis. In lineages where many organism databases were available, preference was given to well-characterized model systems (e.g. *Drosophila* within the Protostomes). The list of taxa used for this report and the major lineages they belong to are as follows: Porifera: *Amphimedon queenslandica*; Cnidaria: *Hydra vulgaris*, *Nematostella vectensis*; Placozoa *Trichoplax adhaerens*; Deuterostomes: *Homo sapiens*, *Strongylocentrotus purpuratus*; Protostomes: *Octopus bimaculoides*, *Lingula anatina*, *Caenorhabditis elegans*, *Drosophila melanogaster*; Choanoflagellate: *Salpingoeca rosetta* ; Holozoa: *Capsaspora owczarki.* Data from other taxa was also downloaded, but ultimately left for later analysis. A full list of the taxa searched and their taxa ids can be found in the document "list_of_org_IDs_long.csv" found **here**

Code

Below are the scripts and functions used throughout the workflow of the project.

Remote BLAST search

This function searches for Usher syndrome associated genes with BLAST in taxa specified by a list of taxa IDs

```
def search_taxa_all_gene_delay(list_of_taxa):
    # blasts sequences in a file against a list of taxa
    # loop through the list and run blast for each one
    # will save each result to a separate xml file

    from Bio.Blast import NCBIWWW
 # imports the NCBIWWW module to allow remote Searching

    import time
 # delays imputs to the NCBI servers and get kicked off

    with open("USH_Search_seq.fasta", "r") as fasta_file:
        sequences = fasta_file.read()
        fasta_file.close()
        #reads in sequences we will be searching

    for i in list_of_taxa:

        result_handle = NCBIWWW.qblast("blastp",
# specifies the program for a protein-protein search
                                       "refseq_protein",
#  database of protein sequences
                                       sequences,
# our list of sequences we read in
                                       alignments = 100,
# asks for 100 best hits
                                       descriptions = 100,
                                       expect = 0.00001,
# specifies max E-value(likelihood of a random match for our query)
                                       entrez_query = str(i))
# specifies the taxa as we loop through it


        file_name=str("USH_Search_"+str(i)+".xml")
#this creates a name for the file

        save_file=open(file_name, "w")
#we are opening a file that does not yet exist to write to it

        save_file.write(result_handle.read())
```

```
#writing the result of our blast search to local file

        save_file.close()
#closing it to allow the file to actually write it

        result_handle.close()
#close the results handle


        print("created "+ file_name)
#this is just a nice way to track the progress of the program


        time.sleep(60)
# 1 minute delay writing the output and requests to the ncbi server

# NCBI is a shared resource, shouldn't monopolize computer time
```

The list of taxa used in this study and the function itself can be found **here**.

Parsing the Files

This function Breaks apart the BLAST search result and gives relevant summary statistics that can be easily viewed by humans (similar to NCBI's website BLAST summary page) and can be used to create file for easier data manipulation. Such manipulations include retrieving gene ID numbers and parsing the sequence's annotation for use in tree creation and annotation.

```
def parse_and_summarize(blast_output_xml):
# goes through the output of a BLAST xml file
# finds the relevant stats to summarize the search
    from Bio.Blast import NCBIXML
    from Bio.SeqRecord import SeqRecord
    #import the required libraries

    for file_name in blast_output_xml:
        result_handle = open(str(file_name), "r")
# sets the result handle

        blast_records = NCBIXML.parse(result_handle)
        #need to use parse if it has multiple records in it

        for blast_record in blast_records:
            org_desig=file_name.split("_")[2]\
```

```
.split("[")[0].replace(" ", "_")
 #properly formats the taxa id so to name things

            homo_sapiens = "[Homo sapiens]"
            blast_query=blast_record.query
            if homo_sapiens in blast_query:
                gene_name=blast_record.query.split("|")[4]\
.split("[")[0]\.replace(" ", "_").replace("_protein", "")\
.replace("_isoform_b3", "")
                formated_gene_name = gene_name[1:-1]
            else:
                formated_gene_name=blast_query.split("|")\
[4].split("_")[0]
#this conditional formats the gene name
#switches between two formats in spliting the name
```

This function can be found in the document "Workshop_assemble_list_of_gene_ids_all_orgs.ipynb" found **here**.

Downloading Sequences from NCBI database

This function takes a list of gene IDs and posts them to NCBI. The sequences are then be retrieved from NCBI at a measured rate. One very important note is that NCBI does not want users to retrieve more than 100 sequences at a time during peak US hours. Therefore, if one intends to retrieve lots of data from NCBI, this program should only be used during the weekend or late at night. Otherwise, the administrators will ban the user from accessing the system. This is why it is important to specify your email in the program, as NCBI will try to contact you before banning you. The input sequences to download can be generated using bash commands to extract the first column of the summary file generated by the parsing program

```
def download_seqs_from_list_and_autoname(in_filename):
    with open(in_filename, "r") as query_file:
        query_ids = query_file.read().splitlines()
    #open the file, grab the ids

    from Bio import Entrez
    Entrez.email = "hspeck@ucla.edu"
    # import Bio Entrez, let NCBI know who I am

    search_results = Entrez.read(Entrez.epost(db ="protein",
                                             id = ",".join(query_ids))
                                             # needs the id's to be as
                                             # a list separated by commas
```

```
#upload the IDs to NCBI
webenv_return = search_results["WebEnv"]
query_key_return = search_results["QueryKey"]
#grab the relevant variables to call on the stuff we posted

count = len(query_ids)
#assigns the count based on the number of sequences we searched for

from urllib.error import HTTPError
# load required library for the try and except conditions

batch_size = 20
#this determines how many things we retrieve and write to the file
#can safely be larger in the future

out_filename = str(in_filename[:-7]+".fasta")
#attempting to rename the file based on the input of the original file
out_handle = open(out_filename, "w")
#open file to write to

for start in range(0, count, batch_size):
    end = min(count, start+batch_size)
    print("Going to download record %i to %i" % (start+1, end))
    attempt = 0
    while attempt < 3:
        attempt += 1
        try:
            fetch_handle = Entrez.efetch(db="protein",
            #says which db
                                         rettype="fasta",
                                         #says what format
                                         #the data should be in
                                         retmode="text",
                                         #what the output should be
                                         retstart = start,
                                         #say what range of
                                         #results you want returned
                                         retmax = batch_size,
                                         #say end of range of
                                         #results want returned
                                         webenv = webenv_return,
                                         #specify the info we uploaded
                                         # with ePost
                                         query_key = query_key_return)
```

```
            except HTTPError as err:
                if 500 <= err.code <= 599:
                    print("Recieved error from server %s" % err)
                    print("Attempt %i of 3" % attempt)
                    time.sleep
                else:
                    raise
        data = fetch_handle.read()
        fetch_handle.close()
        out_handle.write(data)
    out_handle.close()
```

This program and the sets of sequences downloaded can be found **here** and for the gene trees using only the top hit, the sequences can be found **here**.

Align the sequences using MUSCLE

This uses the Biopython wrapper to run MUSCLE on a file of sequences. It can be combined with the above function to produce a script that downloads and aligns sequences with one input(seen below).

```
from Bio.Align.Applications import MuscleCommandline
muscle_cline = MuscleCommandline(input = Sequences_to_align_file_name,
                                 out = multiple_alignment_file_name)
stdout, stder = muscle_cline()
```

Combined Download and alignment

This function downloads and immediately aligns the resulting sequence files, using a list of Gene IDs provided to it

```
def download_seqs_from_list_autoname_align(in_filename):
    with open(in_filename, "r") as query_file:
        query_ids = query_file.read().splitlines()
    #open the file, grab the ids

    from Bio import Entrez
    Entrez.email = "hspeck@ucla.edu"
    # import Bio Entrez, let NCBI know who I am

    search_results = Entrez.read(Entrez.epost(db ="protein",
                                              id = ",".join(query_ids)))
                                              # needs the id's to be
                                              #as a list separated by commas
```

```
#upload the IDs to NCBI
webenv_return = search_results["WebEnv"]
query_key_return = search_results["QueryKey"]
#grab the relevant variables to call on the stuff we posted

count = len(query_ids)
#assigns the count based on the number of sequences we searched for

from urllib.error import HTTPError
# load required library for the try and except conditions

batch_size = 20
#this determines how many things we retrieve and write to the file

out_filename = str(in_filename[:-7]+".fasta")
#let's alter the file name a bit so we can easily move on

#attempting to rename the file based on the input of the original file
out_handle = open(out_filename, "w")
#open file to write to

for start in range(0, count, batch_size):
    end = min(count, start+batch_size)
    print("Going to download record %i to %i" % (start+1, end))
    attempt = 0
    while attempt < 3:
        attempt += 1
        try:
            fetch_handle = Entrez.efetch(db="protein",
            #says which db
                                         rettype="fasta",
                                         #says what format
                                         #the data should be in
                                         retmode="text",
                                         #what the output should be
                                         retstart = start,
                                         #say what range of
                                         #results you want returned
                                         retmax = batch_size,
                                         #say end of range of
                                         #results want returned
                                         webenv = webenv_return,
                                         #specify the info
                                         #we uploaded with ePost
```

```
                                                 query_key = query_key_return)
            except HTTPError as err:
                if 500 <= err.code <= 599:
                    print("Recieved error from server %s" % err)
                    print("Attempt %i of 3" % attempt)
                    time.sleep
                else:
                    raise
        data = fetch_handle.read()
        fetch_handle.close()
        out_handle.write(data)
    out_handle.close()



    #now to run MUSCLE on it
    raw_seq_input = out_filename
    intermediate_name = in_filename.split("_")[1:-1]
    #grabs the core of the name
    muscle_align_name = "muscle_align_" +
                        "_".join(intermediate_name) + ".txt"
    # generates a named file for the muscle alginment output

    from Bio.Align.Applications import MuscleCommandline
    #imports the relevant biopython module for python
    muscle_cline = MuscleCommandline(input = raw_seq_input,
                                     out = muscle_align_name)
    stdout, stder = muscle_cline()
```

This function is found the document "best_hit_generation_tree_building.ipynb" **here**.

Generate the Tree using RAxML

Call this command in a directory containing the RAxML program in order to generate a tree. The options used specify the following: -p gives a numerical seed so that the random results can be repofduced consistently, -# specifies the number of times the program attemtps to fit a model to the data, and -m specifies the model of substitution to be used (in this case amino acid substitution).

```
./raxmlHPC -m PROTGAMMAWAG -p 12345 -s Path/to/alignment/aligned_file
-# 5 -n output_file_name
```

Generate Tree Annotations with Python

This function will generate annotations based on the summary csv files generated by the parsing step. Bash scripts can be used to gather a list of all appropriate files and run them through the annotation building program.

```
def annotations_for_treebuilding(file_input):
    with open(file_input, "r") as input_doc:
        summary_file = input_doc.readlines()
    #opens the file given to the program

    annotations_name_component = file_input.split("_")[1:-3]
    annotations_file_name = str("annotations_tree_"
                            + "_".join(annotations_name_component)
                            +".txt")
    annotations_file = open(annotations_file_name, "w")
    #opens the file to write that we tell it to

    taxa_Dict={"Amp_que": "Porifera",
               "Sal_ros": "Choanoflagellate",
               "Cae_ele": "Protostome",
               "Dro_mel": "Protostome",
               "Lin_ana": "Protostome",
               "Cap_owc": "Holozoa",
               "Hyd_vul": "Cnidaria",
               "Nem_vec": "Cnidaria",
               "Oct_bim": "Protostome",
               "Tri_adh": "Placozoa",
               "Hom_sap": "Deuterostome",
               "Str_pur": "Deuterostome"}
        #defines a dictionary we can use
        #to classify the organisms into broader clades

    for line in summary_file:
        #can't split it apart before hand because
        #there are commas included in some of the gene names

        gene_ID = line.split(",")[1]
        #pulls out the gene ID number

        formated_org = line.split('[')[1].split("]")[0]
        #gets the genus and species of the organism
        genus = formated_org.split(" ")[0]
        #takes only the genus
```

```
        genus_code = formated_org.split(" ")[0][:3]
        #takes only first 3 letters of genus
        species_code = formated_org.split(" ")[1][:3]
        #takes only the first 3 letters of species

        final_org_name = genus_code + "_" + species_code
        #gives us an abbreviated species

        taxa_group = taxa_Dict[final_org_name]
        #assigns the species to a broad taxonomic division

        gene_annotation_slice = line.split("[")[0].split(",")[2:]
        if "PREDICTED:" in gene_annotation_slice:
            if "-like" in gene_annotation_slice:
                gene_annotation = gene_annotation_slice[0].replace(" ", "")
            else:
                gene_annotation = gene_annotation_slice[1:3]
        elif "Drosophila" in gene_annotation_slice:
            gene_annotation = gene_annotation_slice[1:3]
        else:
            gene_annotation = gene_annotation_slice[0:1]
        gene_annotation_final = "_".join(gene_annotation).replace(",", "")
        #cuts down the gene annoatation names to make them more manageable
        #problematic as the gene names do not conform
        #to a single format that makes them easily parsable
        #have to switch between formats

        combined_name = '"' +final_org_name
                       + " " + gene_annotation_final + '"'
        genus_and_gene = '"' +genus + " " + gene_annotation_final + '"'

        annotations_file.write(gene_ID+","+
                    final_org_name+","+
                    gene_annotation_final+","+
                    taxa_group+ ","+
                    combined_name+ "," +
                    genus_and_gene+'\n')
        #Writes it to the file

    annotations_file.close()
```

This script is found in the “renaming_seqs_tree_annotations.ipynb” document **here**.

Generate the Tree Figures

Using the tree files generated by RAxML this R function creates PDFs of the annotated trees. The annotation function gives several columns of data that could be used to name the tips of the trees. In the example below, I've opted to use the column that includes the genus of the organism the gene was found in, as well as the sequence annotation.

```
make_gene_tree_pdf <- function(gene_tree_file,
                               gene_annotation_file,
                               title,
                               output_file_name){
  library(ape)
  gene_tree <- read.tree(gene_tree_file)
  #read in the tree file

  gene_annotations <- read.csv(gene_annotation_file,
                               header =FALSE,
                               stringsAsFactors = FALSE)
  names(gene_annotations) <- c("GeneID",
                               "OrgName",
                               "Annotation",
                               "Group",
                               "CombinedName",
                               "Genus_Gene")
  #read in the gene annotations
  rownames(gene_annotations) <- gene_annotations$GeneID
  #set the names of the rows of the data frame to the ID number

  order <- match(gene_tree$tip.label, rownames(gene_annotations))
  gene_annotations <- gene_annotations[,][order,]
  #use the matching row names to reorder the data frame
  #to match the tree file

  gene_tree$tip.label <- gene_annotations$Genus_Gene
  # Rename tips of Tree to increase Readability

  pdf(output_file_name, width = 9, height = 5)
  #create a pdf output for the graph
  par(mar=c(5.1, 4.1, 4.1, 8), xpd =TRUE)
  #this gives extra space to the right to add a legend

  plot(gene_tree, cex = 0.5, label.offset = 0.1)
  # decrease size of labels a bit

  title(title)
```

```r
  #add a title
  add.scale.bar(x = 0, y = -0.1, lwd = 2, lcol = "black")
  #add a scale bar! this adds it in the margin outside the figure

  # Color the tips to indicate which lineage the organism belongs to

  gene_annotations$Group <- as.factor(gene_annotations$Group)

  #start by setting the Group to factor
  lineages <- unique(gene_annotations$Group)
  cols <- rainbow(n = length(lineages))
  #color vector for legend
  colvec <- cols[gene_annotations$Group]
  #color vector for tree
  tiplabels(pch = 19, col = colvec)


  # Add a legend!
  legend(x = "bottomright",
         inset= c(-.1, 0), #this offsets the legend
         #lwd = 0,
         pch = 19 ,
         legend = levels(lineages),
         col = cols,
         cex = 0.5) # resize the key
  dev.off()
}
```

This function is found in the “Best_hit_treebuiding.R” document in **this directory**.

Example Tree - Whirlin-Harmonin-PDZD7 family tree

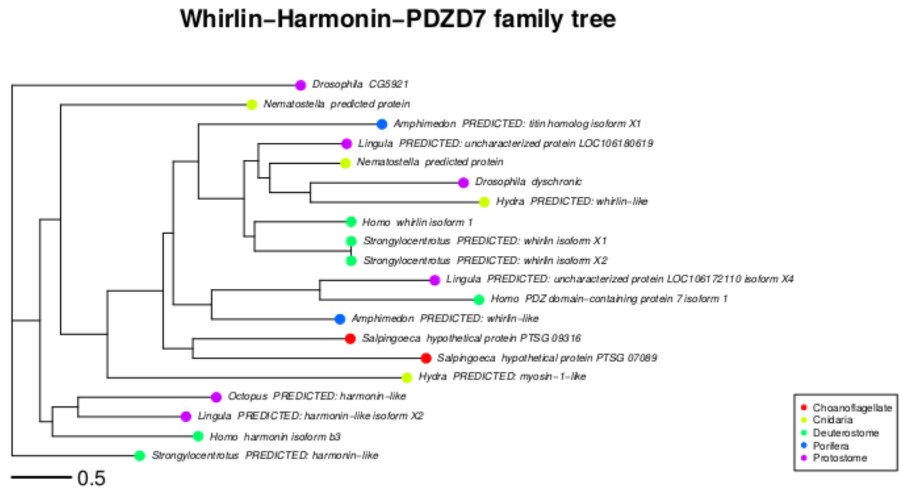

Figure 1: Gene tree for the PDZ scaffold proteins involved in Usher Syndrome: Harmonin, Whirlin, and PDZD7. Sequences from organisms where previous searches had failed to find closely related genes (*Trichoplax*, *Caenorhabditis*, etc.) were excluded from the alignment. Top BLAST hits from each gene searched were concatenated and redundant sequences were removed prior to alignment.

The workflow and files used to generate this tree can be found **here**.

Gene trees for the top search results for each gene can be found in Appendix I, while the code and files used to generate them can be found **here**.

Results

Homologs for several Usher syndrome associated genes were found not only in the Sponge *Amphimedon queenslandica*, but also in the Choanoflagellate *Salpingoeca rosetta*, supporting findings from Pena et al 2016 (Pena *et al.* 2016). This supports the idea that these genes may have played a structural role in the microvillar and ciliated cells of the most recent Metazoan common ancestor. Examination of the gene trees for the PDZ scaffold proteins Harmonin, Whirlin and PDZD7 showed these genes to be closely related to one another, prompting the assembly of a Harmonin-Whirlin-PDZD7 combined tree (seen above). This tree shows PDZD7 and Whirlin diverging from each other following their divergence from Harmonin, with PDZD7 relatively restricted in distribution in the taxa investigated. As Harmonin and Whirlin mostly associate with different protein complexes and links in the hair cell created by different Usher Syndrome proteins, the timing of the initial divergence betwen the two genes may indicate the timing of the origin of these different complexes (Kremer *et al.* 2006). Investigation the best hit trees shows that the closest homolog of many Usher syndrome genes present in *Caenorhabditis elegans* are only distantly related to Usher Syndrome homologs found in closely related taxa. This is a likely indication of loss of these genes in *C. elegans* or nematodes more generally, as other Protostome lineages, including the Ecdysozoan *Drosophila melanogaster*, posses homologs of most of these genes. Loss of a number of Usher Syndrome associated genes also seems likely in *Trichoplax adhaerens*, given the presence of many of these genes in the Choanoflagellate *S. rosetta*, which shared a common ancestor with the vertebrates less recently than *T. adhaerens* given *T. adhaerens'* status as an animal. Further work will be needed to determine whether Usher syndrome homologs play functional and structural roles in the taxa searched here similar to the ones they play in the vertebrate eye and ear, and how the domain structure of these genes evolved within certain lineages.

Github Repository

The full project repository can be found at the following link:

https://github.com/hspeck/project

Appendix I - Best hit trees for Usher syndrome genes and candidate genes

Genes are divided by which version of Usher Syndrome they are associated with. USH1 causes the most severe effects on hearing and vision, while USH3 is the least deleterious, and USH2 is intermediate between the other two forms (Kremer *et al.* 2006).

Usher Syndrome 1 associated genes: USH1B-D, USH1F-G, USH1J.

Usher Syndrome 2 associated genes, USH2A-B, USH2D

Usher Syndrome 3 associated gene, USH3

Usher Syndrome gene of unknown association: PDZD7

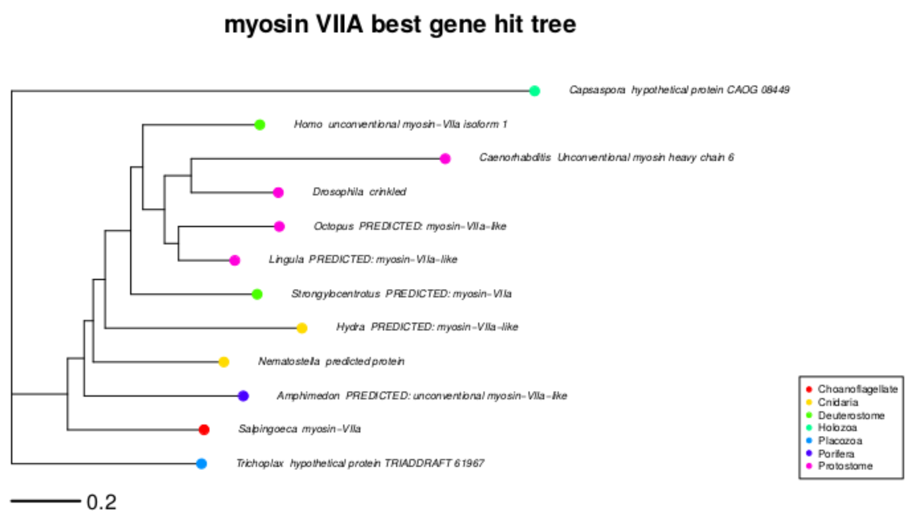

Figure 2: Gene Tree for Mysoin VIIA, (USH1B)

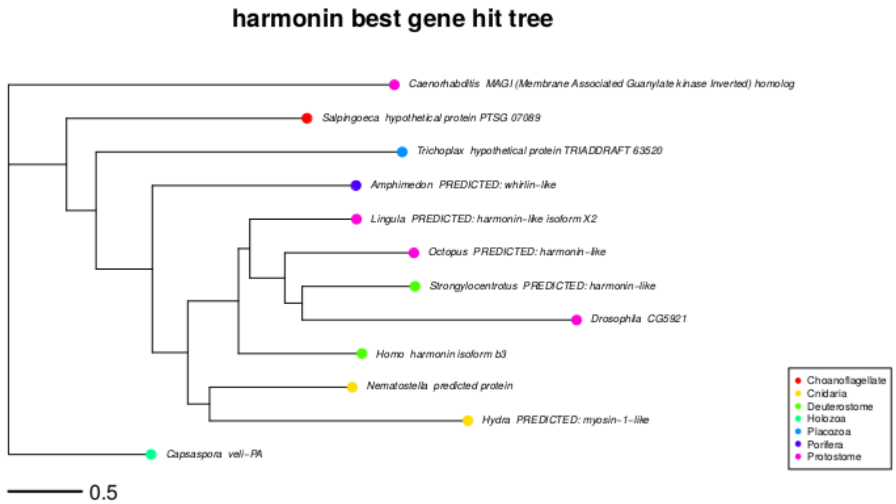

Figure 3: Gene Tree for harmonin, (USH1C)

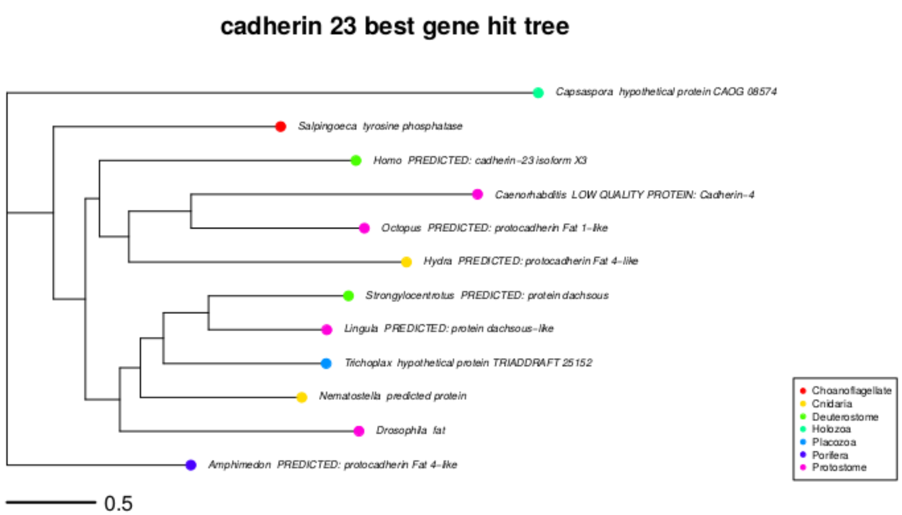

Figure 4: Gene Tree for Cadherin 23, (USH1D)

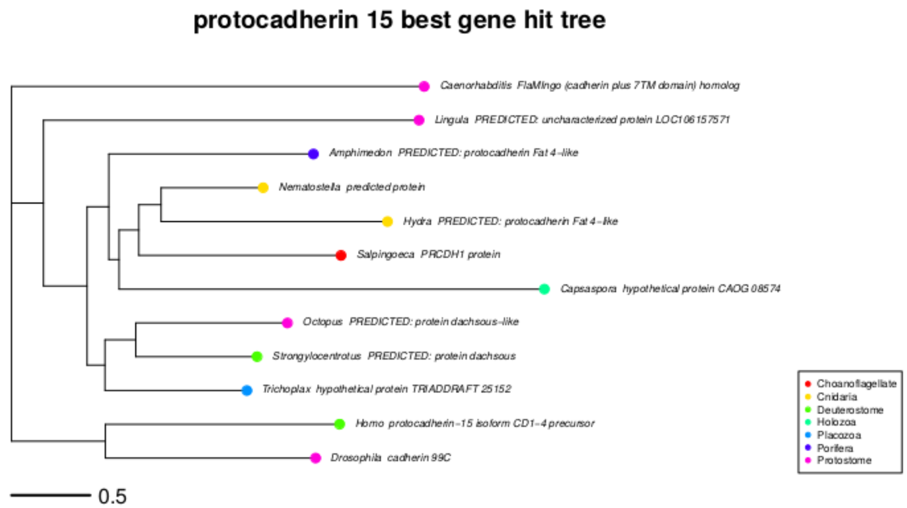

Figure 5: Gene Tree for Protocadherin 15, (USH1F)

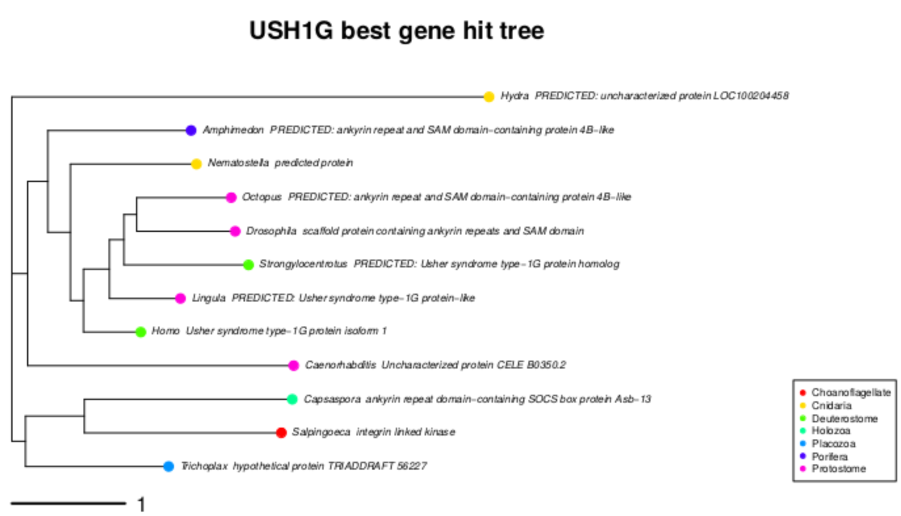

Figure 6: Gene Tree for SANS (USH1G)

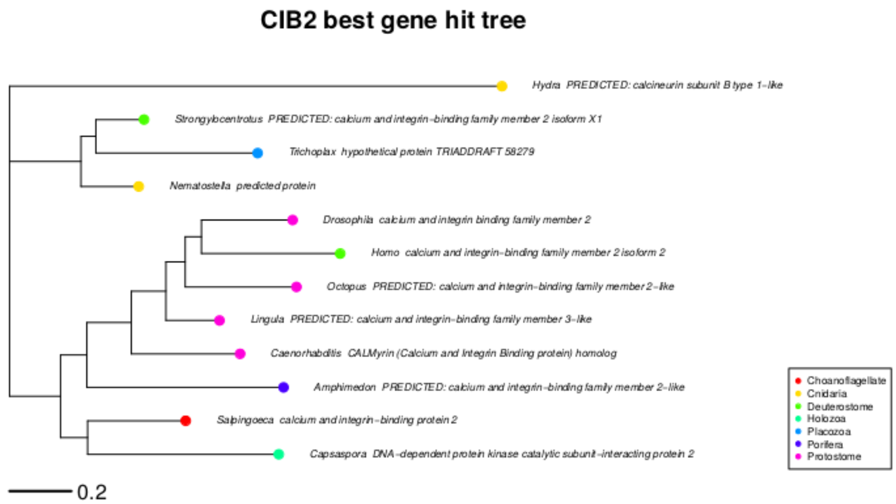

Figure 7: Gene Tree for CIB2, (USH1J candidate)

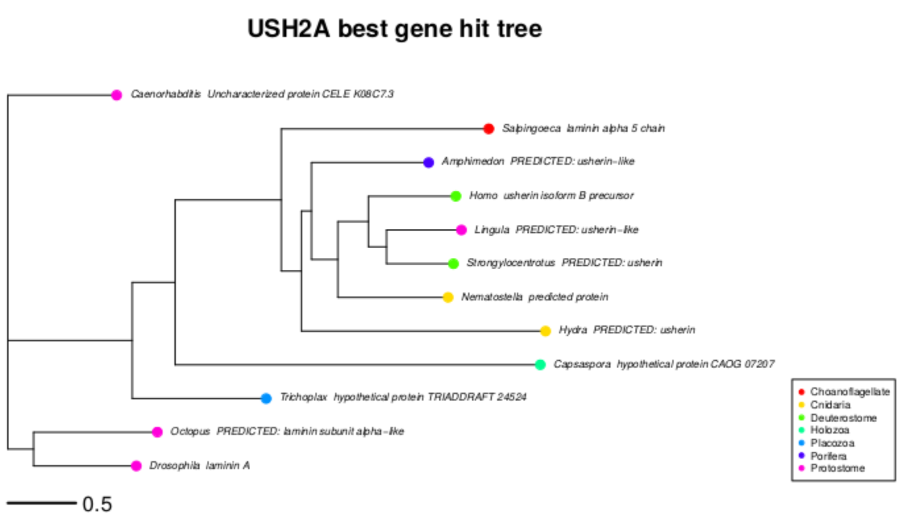

Figure 8: Gene Tree for Usherin, (USH2A)

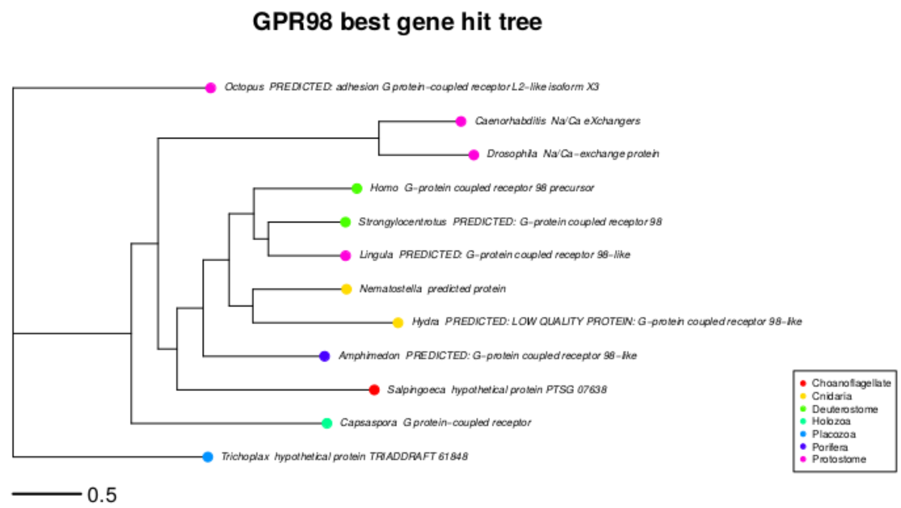

Figure 9: Gene Tree for GPR98, also known as VLGR1 (USH2B)

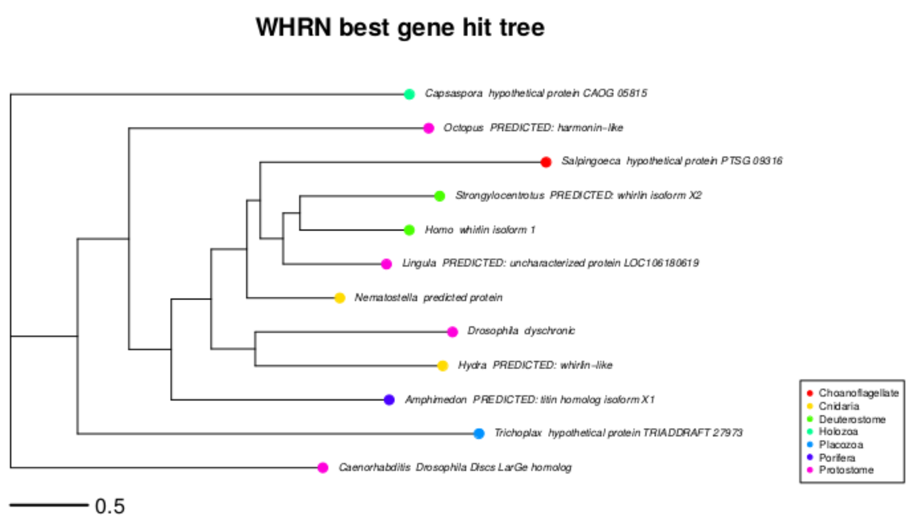

Figure 10: Gene Tree for Whirlin, (USH2D)

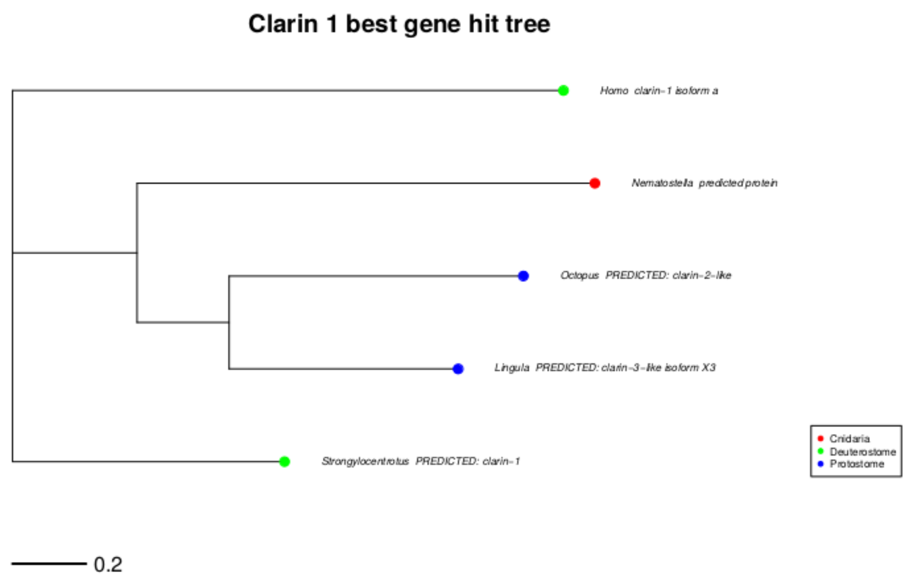

Figure 11: Gene Tree for Clarin 1, (USH3 candidate)

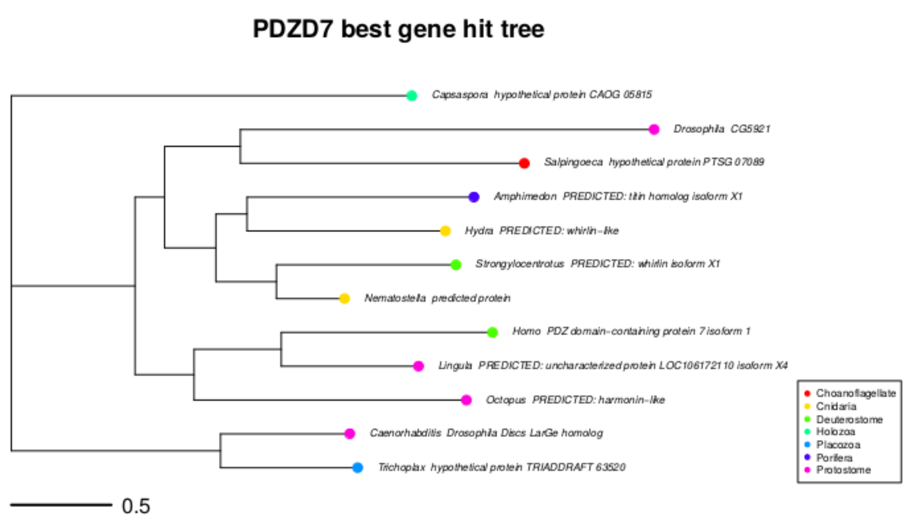

Figure 12: Gene Tree for PDZD7, (USH gene candidate)

References

Altschul, S., Gish, W., Miller, W., Myers, E. & Lipman, D. (1990). Basic local alignment search tool. *Journal of Molecular Biology.*

Benson, D., Karsch-Mizrachi, I., Lipman, D., Ostell, J. & Wheeler, D. (2004). GenBank. *Nucleic Acids Research*, **33**.

Cock, P., Antao, T., Chang, J., Chapman, B., Cox, C., Dalke, A., Friedberg, I., Hamelryck, T., Kauff, F., Wilczynski, B. & Hoon, M. de. (2009). Biopython: Freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, **25**.

Edgar, R.C. (2004). MUSCLE: Multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, **32.5**, 1792–1797.

Erwin, D., Laflamme, M., Tweedt, S., Sperling, E., Pisani, D. & Peterson, K. (2011). The cambrian conundrum: Early divergence and later ecological success in the early history of animals. *Science*, **334**, 1091–1097.

Fritzsch, B. & Straka, H. (2014). Evolution of vertebrate mechanosensory hair cells and inner ears: Toward identifying stimuli that select mutation driven altered morphologies. *Journal of Comparative Physiology A*, **200**, 5–18.

Kremer, H., Wijk, E. van, Märker, T., Wolfrum, U. & Roepman, R. (2006). Usher syndrome: Molecular links of pathogenesis, proteins and pathways. *Human molecular genetics*, **15**, 262–270.

Paradis, E., Claude, J. & Strimmer, K. (2004). APE: Analyses of phylogenetics and evolution in R language. *Bioinformatics*, **20**, 289–290.

Pearson, R. (2014). An introduction to sequence similarity ('homology') searching. *Current protocols in bioinformatics.*

Pena, J., Alie, A., Richter, D., Wang, L., Funayama, N. & Nichols, S. (2016). Conserved expression of vertebrate microvillar gene homologs in choanocytes of freshwater sponges. *EvoDevo*, **7(1)**.

Sodergren, E., Angerer, R., Angerer, L., Arnone, M., Burgess, D., Burke, R. & Coffman, J. (2006). The genome of the sea urchin strongylocentrotus purpuratus. *Science*, **314**, 941–952.

Stamatakis, A. (2014). RAxML version 8: A tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, **30**, 1312–1313.