Katholieke
Universiteit
Leuven

MSc in Bioinformatics
Comparative Genomics
[I0U29a]

COMPARATIVE GENOMICS STUDY BETWEEN TWO SPECIES:

Latimeria chalumnae & Danio rerio

Antoine Ruzette (r0829308)

Academic year 2021-2022

Contents

1 Introduction

In the present report are presented the results of a comparative genomics study between two species, Danio rerio and Latimeria chalumnae. On one hand, Danio rerio, known as zebra-fish, is a bony fish in the family Danionidae and has been widely used as a model organism in biology for over 30 years. On the other hand, Latimeria chalumnae, known as coelacanth, is a bony fish in the family of Coelachanthidae. In the early 1940s, scientists though coelacanth have become extinct in the Late Cretaceous. That was before the discovery of the aforementioned species, which is of one the two remaining extant species of coelacanths. It has has been a hot topic during the past few decades as it was considered a 'living fossil'. It is thought that its overall final phenotype arose from approximately 400 million years ago. Since its discovery, a major discussion appeared whether coelacanths or lungfishes, two groups of the lobe-finned fishes, are the closest living relatives of land vertebrates, namely the tetrapodes [1].

However, in the present report, we switch gear and investigate the phylogenetic relation between Latimeria chalumnae, a lobe-fin fish and Danio rerio, a ray-fin fish, both part of the bony-fish clade. To do so, we first generated orthologous pairs, after assessing the similarity between the two whole-genome protein sequences against one each other, using Best Bidirectional Hits (BBH). Then, we further investigated an identified co-ortholog case with a phylogenetic tree approach. Lastly, we identified functional (i.e. conserved) regions for the aforementioned co-orthology case.

The developed methodology has been further described in the following section, for the stake of reproducability.

2 Methods

In the present section are described the details of the methodology applied. All the computations have been performed on the Vlaamse Super Computer (VSC) HPC[1] after having requested credits for this specific project.

2.1 Orthology inference using BBH

2.1.1 Retrieve the whole-genome protein sequences from NCBI

For the two species, the whole-genome protein sequences were retrieved from the NCBI's website as protein FASTA files on the 04.01.2022. Note the large difference in size between the two genomes.

Latimeria chalumnae: LatCha1, RefSeq GCF_000225785.1 (Reference genome from 2011), 1.373 Mbp. [3]
Danio rerio: GRCz11, RefSeq GFC_000002035.6 (Reference genome from 2017), 2.861 Mbp. [4]

2.1.2 Local sequence similarity analysis: BLASTp

We used BLASTp to identify the most similar sequences between the whole-genome protein sequences of the two species. The output of BLASTp contains, for every query sequence from one species, the most similar target sequences from the other species, ordered according to their alignment score. Similarity corresponds to sequence homology. Subsequently, two metrics are associated with these pairs of query-target sequence, that is the score and the expected-value (E-value) of each alignment. As we will define orthology with Bidirectional Best Hits, similarity search has been performed for the four following cases:

1. Query: Latimeria chalumnae vs Target: Danio rerio
2. Query: Danio rerio vs Target: Latimeria chalumnae
3. Query: Danio rerio vs Target: Danio rerio
4. Query: Latimeria chalumnae vs Target: Latimeria chalumnae

Note that this step is computationally intractable on a local computer. Indeed, the Danio rerio genome accounts for about 52.000 protein-coding sequences and the Latimeria chalumnae one for about 34.000 protein-coding sequences. On the HPC, each BLASTp run took between 14 and 24 hours, depending on the number of protein-coding sequences.

[1] https://www.vscentrum.be/, on the 30.01.2022

An example of the bash script (.pbs format) used to send a BLASTp job on the VSC is illustrated in the sample code below:

```
#!/bin/bash

#PBS -l nodes=1:ppn=36
#PBS -l walltime=12:00:00
#PBS -l pmem=5gb
#PBS -A default_project
#PBS -N alignment_blast_danio_latimeria

cd /scratch/leuven/339/vsc33918

module purge
module restore blast_env

makeblastdb -in danio_rerio.faa -dbtype prot
blastp -query latimeria_chalumnae.faa -db danio_rerio.faa -out
    danio_latimeria-2-alignment.txt
```

2.1.3 Compute orthologs using Best Bidirectional Hits

As seen in the Comparative Genomics classes [2], Bidirectional Best Hits (BBH) is a method to infer orthology. It identifies the pairs of genes in two genomes that are more similar to each other than either is to any gene in the other genome. In order to compute such pairs of genes, a python script can be found in the appendix, named *co_orthology.py*. In that script is contained three main parts, namely the extraction of the genes and scores from the BLASTp output files, the generation of the full set of orthologous pairs using BBH and the generation of the in-paralogs for each species.

Extraction of the query and aligned genes, with their scores, from the BLASTp output files

To do so, a function called *extractGenes()* was created, which performs that exact task. It appears that the RefSeq gene IDs are always of the form illustrated in Figure 1. Thus, that pattern can be retrieved using the *re* python package (regular expression, similar to grep).

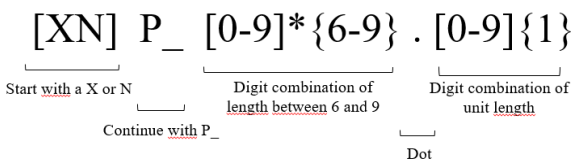

Figure 1: RefSeq gene ID pattern

Actually, the undermentioned script screens through a BLASTp output text file. When it meets a query gene, it retrieves its ID (using the pattern from Figure 1) and the ID of the aligned gene with the highest alignment score (i.e. the best hit). For storage efficiency, each extraction is stored in a dictionary that has been previously initialized.

```
def extractQueryandAlignedgenes(filename, dict_align):

    with open(filename, 'r') as align1:
        for ln in align1:
            #get the query gene name
            if ln.startswith("Query="):
                ID_query = re.search('[XN]P_\d{5,}\.\d{1}', ln).group()
                (dict_align[list(dict_align.keys())[0]]).append(ID_query)

            #get the best (top) target gene name & score
                test = align1.readline()
                while (not re.search('[XN]P_\d{5,}\.\d{1}', test)):
                    test = align1.readline()#continue to read
                ID_target = re.search('[XN]P_\d{5,}\.\d{1}', test).group()
                (dict_align[list(dict_align.keys())[1]]).append(ID_target)
                (dict_align[list(dict_align.keys())[2]]).append(test[70:74]
    return dict_align;
```

.

Generation of the full set of orthologs using BBH

Once the IDs are extracted, we compute the orthologous sequences using the Bidirectional Best Hits method. Imagine a query sequence, named **Q1**, having as best aligned sequence **A1** (associated with a score S1). Assume another query sequence, named **Q2**, having as best aligned sequence **A2** (associated with a score S2). If, and only if, Q1 == A2 **and** Q2 == A1, then Q1 and Q2 are identified as an orthologous pair. The developed python script implements the latter reasoning to compute BBH. Practically, it uses a brute force approach to iterate over all query genes to look for BBH between the two species and store them in a nested list called *early_match*. Note that this is the full set of orthologs, including the in-paralogs. It will be refined in the following section.

```
def orthology(dict_align1, dict_align2):
    query_gene2 = (dict_align2[list(dict_align2.keys())[0]])
    query_gene1 = (dict_align1[list(dict_align1.keys())[0]])
    alignment_gene2 = (dict_align2[list(dict_align2.keys())[1]])
    alignment_gene1 = (dict_align1[list(dict_align1.keys())[1]])
    score2 = (dict_align2[list(dict_align2.keys())[2]])
    score1 = (dict_align1[list(dict_align1.keys())[2]])

    for j in range(len(query_gene1)) :
        for i in range(len(query_gene2)) :

            if ( query_gene2[i] == alignment_gene1[j] and alignment_gene2[i]
               == query_gene1[j] ) :
                early_match.append([query_gene2[i], score2[i], query_gene1[j],
                    score1[j]])
                #query_gene2 -> Danio
                #alignment_gene2 -> Latimeria
                #query_gene1 -> Latimeria
                #alignment_gene1 -> Danio
```

Generation of the in-paralogs for each species

In-paralogs occur when a speciation event is followed by a duplication event, and result in a pair of genes of the same genome that are more similar to each other than either is to genes from the other genomes under study. The extraction of the sequence IDs and scores is similarly performed as described above, with the difference that the best (i.e. with the highest score) aligned sequence for each query is obviously the query sequence itself (as we are locally aligning a genome againt itself). Thus, the top aligned sequence is discarded and the second best aligned sequence is considered. Query sequence IDs and second best aligned sequence IDs, with their scores, are retrieved. For each early match between two different genomes, the score of the previously identified early match sequence is compared with the score of the best alignment within the same genome of that same query sequence. If the score of an early match is smaller than the score of the best alignment, then that early match is considered an in-paralogous sequence. On the contrary, if an early match possesses a score larger than the one when aligned against the same genome, then it is considered part of an orthologous pair.

```
def in_paralogy(filename, dict_align, toremove):

    with open(filename,"r") as align_self:
        for ln in align_self:
            #get the query gene name
            if ln.startswith("Query="):
                ID_query = re.search('[XN]P_\d{5,}\.\d{1}', ln).group()
                (dict_align[list(dict_align.keys())[0]]).append(ID_query)
            #get the best alignment gene name & score
                test = align_self.readline()

                while (not re.search('[XN]P_\d{5,}\.\d{1}', test)) :
                    test = align_self.readline()#continue to read
                tmp = align_self.readline()#goes to second best alignment
                if (re.search('[XN]P_\d{5,}\.\d{1}', tmp)) :
                    ID_target = re.search('[XN]P_\d{5,}\.\d{1}', tmp).group()
                    (dict_align[list(dict_align.keys())[1]]).append(ID_target)
                    (dict_align[list(dict_align.keys())[2]]).append(tmp[70:74])
                    for z in range(len(early_match)) :
                        if (dict_align == dict_align3): #for latimeria_self
                            if (early_match[z][2] == ID_query and tmp[70:74] >
                                early_match[z][3]) :
                                toremove.append(ID_query)
                        else:                                #for danio_self
                            if (early_match[z][0] == ID_query and tmp[70:74] >
                                early_match[z][1]) :
                                toremove.append(ID_query)
    return toremove;
```

.

Finally, two text files are saved, that is *full_orthology.txt* and *paralogs.txt*. From these two files, orthology, in-paralogy and co-orthology cases can be identified.

2.2 Check orthology with a phylogenetic tree

Subsequently, orthology and paralogy inference is performed through a phylogenetic tree for a co-orthology case, which was identified using the previously implemented BBH method. The sequence from the co-orthology case is then analysed using the online BLASTp implementation[2] to look for homologous sequences from 25 different species. Then, the 25 homologous sequences, that were retrieved in a sole FASTA file, were aligned

[2]https://blast.ncbi.nlm.nih.gov/Blast.cgi, on the 28.01.2022

using an implementation of MUSCLE[3] in the SeaView software[4]. A PhyML phylogenetic tree was inferred from that multiple sequences alignment, using the following PhyML drivers: LG model (for amino-acids), bootstrap with 100 replicates, NNI branch swapping and a BioNJ starting tree.

In order to investigate the speciation and duplication events, a species tree is required. A species tree is a phylogeny depicting the evolutionary relationships of a set of species. It is usually constructed using highly conserved sequences due to their intrinsic essential and universal characteristics. We saw some methods during the classes but none of them can straightly be applied to the whole 25 fish species. Practically, an intuitive method would be to search for 18s rRNA sequences as marker genes but the data are not available, neither on the SILVA database nor on the Nucleotide NCBI database for all the species under study. However, two other methods exists and have been used for fish species. The first consists of using the cytochrome oxidase COI gene, which was a gold standard marker proposed a while ago. The second consists in using the 16s rRNA from the mitochondrial genomes of eukaryotes. The latter will be used. At first sight, one may question the use of 16s rRNA in eukaryotic species. However, according to the endosymbyotic theory, mitochondria has bacterial origins and explains why mitochondria, even contained in an eukaryote, carry prokaryotic small sub-unit 16s rRNA. *Singh AK et al.* [6] showed, in 2015, that mitochondrial 16s rRNA is an efficient conserved marker in phylogenetic inference for fish species. Thus, the mitochondrial 16s rRNA was retrieved from the NCBI's Nucleotide database for each of the 25 species and stored in a single FASTA file. Following the same methodology as for the 25 homologous sequences, the sequences were aligned with MUSCLE and a phylogeny was inferred using the following PhyML drivers: GTR model, bootstrap with 100 replicates, NNI branch swapping, optimized across site rate variation and a BioNJ starting tree. Then, species tree reconciliation was performed.

Both resulting trees were visualized using the FigTree freeware.

2.3 Identification of functional regions

In this last section is described the methodology used to identify highly conserved regions in a multiple sequences alignment. Throughout evolution, if a region is highly conserved, it usually implies that the region must have a particularly functional role. Indeed, genetic regions having a needful function tend to be conserved through evolutionary times. Using the latter reasoning, we can calculate the Shannon entropy [7], a measure of the uncertainty of a process, for each site. For the case of protein sequences, we can interpret the Shannon entropy as a measure of diversity of a site. A high entropy is associated with stochastic signals, that is poorly conserved sites while a low entropy is associated with deterministic signals, that is highly conserved sites.

$$H(i) = -\sum p(i) \log p(i) \tag{1}$$

The calculation of the Shannon entropy, as stated in the equation (1), has been implement in a python script for a whole sequence. The central function of that script is illustrated below.
Firstly, a pre-processing step needs to be performed as follow:

1. Extraction of the sequence IDs and sequences themselves from the *.clw* ClustalW file using the **re** package
2. Merge together the sequences from different lines
3. Split each sequence into its individual residues and store it for further calculation

Secondly, the *shannon()* function is applied to each site of the multiple sequence alignment.
Lastly, the calculated Shannon entropy's are visualized for each site in the multiple sequence alignment.

[3] https://www.ebi.ac.uk/Tools/msa/muscle/, on the 28.01.2022
[4] https://www.softpedia.com/get/Programming/Other-Programming-Files/SeaView.shtml, version 5.0.5

```
def shannon(msa_list):
#compute the shannon entropy of a list of residues from a same site
    score=0
    count = Counter(msa_list)
    freq = []

    for key in count:
        p = count[key]
        p = p/len(msa_list)
        freq.append(p)

    for m in range(len(freq)):
        new_score = freq[m]*math.log2(freq[m])
        shannon = (score + new_score)*-1

    return shannon
```

3 Results

3.1 Orthologs and in-paralogs identification

Once the sequences were processed as described in the *Methods/Orthology inference using BBH*, the two following files can be analyzed: *full_orthology.txt* and *paralogs.txt*. The former contains the full set of identified orthologous pairs while the latter contains the set of in-paralogs sequences. From those two sets, we retrieve one (of the many!) identified case of co-orthology by looking at the in-paralogs of a sequence of an orthologous pair. Since we only needed one, the identification of the co-orthology cases has not been automated through a python script.

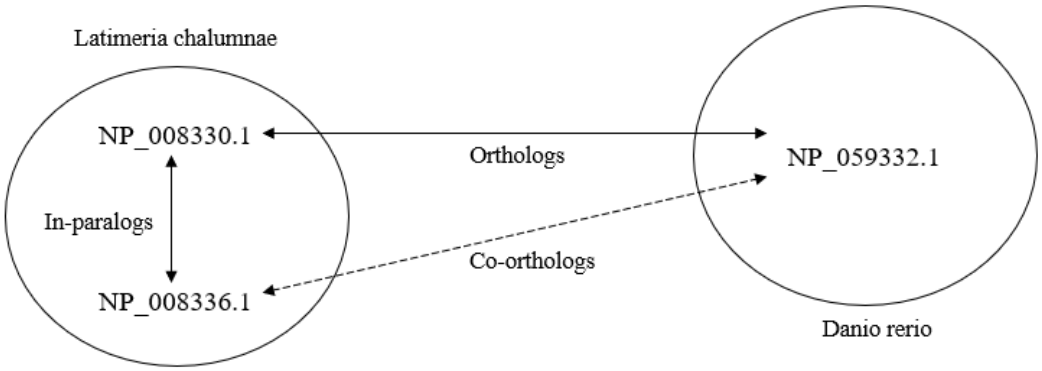

Figure 2: Co-orthology illustration between NP_008336.1 (from Latimeria chalumnae) and NP_059332.1 (from Danio rerio)

We observe that NP_059332.1 is a gene coding for a NADH dehydrogenase subunit 2 (mitochondrion) [Danio rerio], NP_008330.1 for a NADH dehydrogenase subunit 2 (mitochondrion) [Latimeria chalumnae] and NP_008336.1 for a NADH dehydrogenase subunit 3 (mitochondrion) [Latimeria chalumnae].
Overall, the Danio rerio genome contains 52 829 protein coding genes while the Latimeria chalumnae one contains only 28 251. Table 1 illustrates the number of orthologous pairs and the in-paralogs sequences, compared to the total number of protein coding sequences for the two genomes under study.

Table 1: Orthologous pairs and in-paralogous sequences identified from Latimeria chalumnae and Danio rerio genomes.

	# of sequences	# of orthologous pairs	# of in-paralogs
Latimeria chalumnae	34 251	12 246	5770
Danio rerio	52 829		4415
Total	87 080	12 246	10 185

3.2 Orthology with a phylogenetic tree

In the present section, we discuss the results of the methodology described in the corresponding section and we identify the speciation and duplication events.

After retrieving a load of sequences homologous to NP_00336.1 using MUSCLE, we align them and generate the phylogenetic tree in Figure 3.

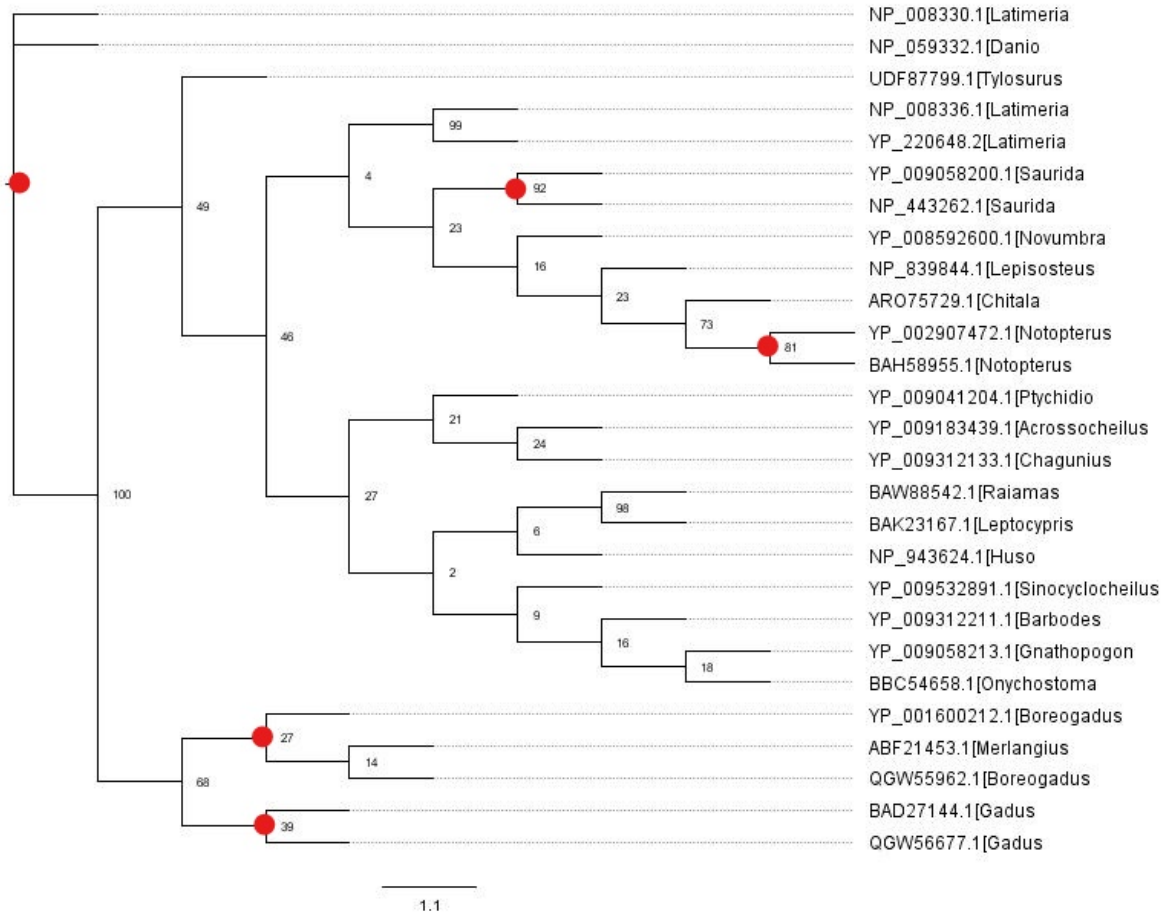

Figure 3: **PhyML homology phylogeny:** GL model, 100 bootstrap replicates, NNI branch swapping. Red dot indicates duplication events, the rest indicates speciation events.

Comparing the homology tree and the species tree, a process named species tree reconciliation, enables us to characterize a node as either a speciation or duplication event. The species tree is illustrated in Figure 4. Duplication nodes are characterized by identifying the overlapping of species in the homology phylogeny that does not occur in the species phylogeny.

Thus, the red-highlighted nodes have been identified as duplication events. The rest of the nodes corresponds to speciation events.

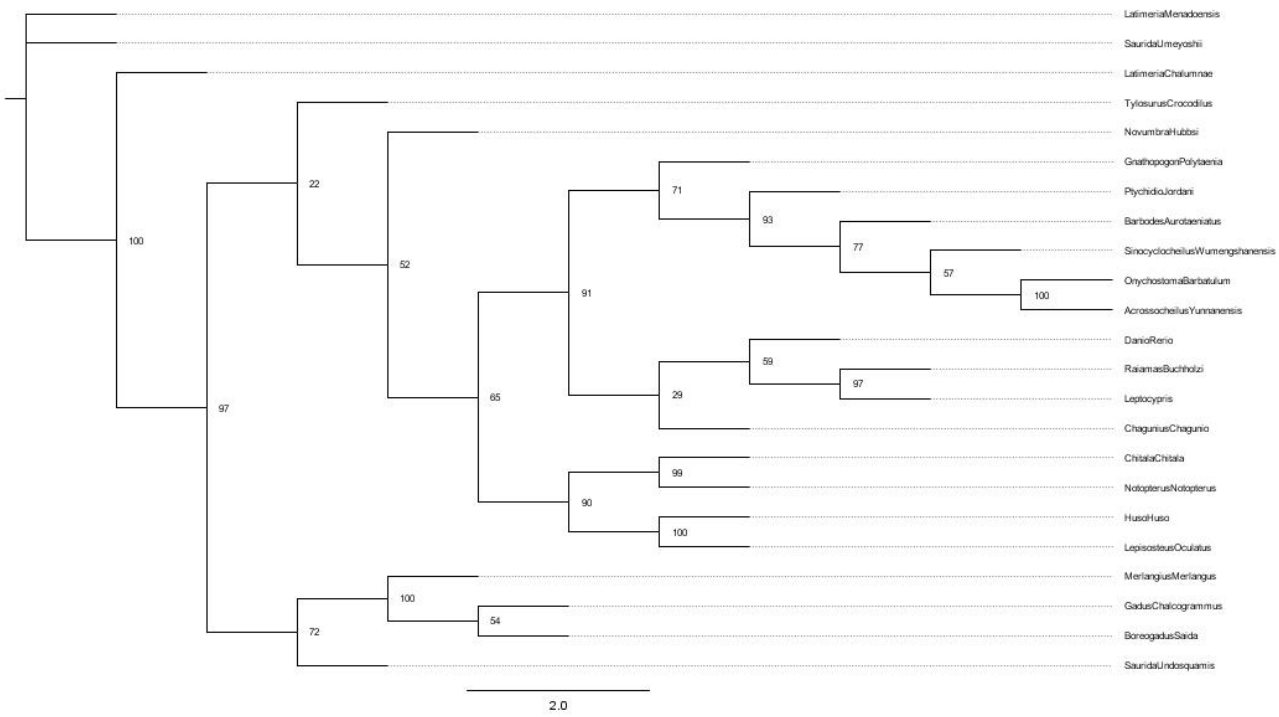

Figure 4: **PhyML species phylogeny:** GTR model, 100 bootstrap replicates, NNI branch swapping - based on mitochondrial 16s rRNA sequences.

Figure 5: Plot of each site in the multiple sequence alignment with its Shannon entropy value.

3.3 Conserved regions identification

The corresponding methodology described in section 2.3 was applied to the multiple sequence alignment of the homologous sequences used in the previous section. For simplicity, note that two sequences were discarded as they had significantly different lengths compared to the others. Thus, we did not have to deal with gaps as all sequences have the same length and are highly similar. From Figure 5, we identify two main conserved regions, that is regions mostly containing low entropy sites (note that a null entropy indicates a site where a single amino acid was observed among all the aligned sequences):

1. from position 32 to 82
2. from approx. position 98 to 117

Based on this, a deeper analysis can be performed to unravel the functional role of these conserved regions. We already observed that all these sequences encode for the NADH dehydrogenase subunit 3, that is, according to NCBI, a core subunit of the mitochondrial membrane respiratory chain NADH dehydrogenase (Complex I) that couples the transfer of electrons from NADH to quinone with the translocation of protons across the membrane. Actually, the NCBI's conserved domains database indicates that the whole sequence is a conserved protein domain under the accession number MTH00136[5] and is part of a domain family called Oxidored_q4 (under the accession number cl00535 [6]).

4 Discussion and conclusion

The present report is intended as an introductory task in the vast and complex subject that comparative genomics is, illustrated by the comparison case between two bony fish species, namely Latimeria chalumnae and Danio rerio.

Firstly, we identified orthologous pairs and in-paralogs for the two species under study thanks to a python script that implements the Best Bidirectional Hits (BBH) method. The in-paralogs are then identified by comparing the scores of the orthologous pairs between species to the scores of the orthologous pairs within species. Out of the 87 080 protein coding genes (from the genomes of the two species), we were able to identify 12 246 orthologous pairs, 5770 and 4415 in-paralogs combinations within, respectively, Latimeria chalumnae and Danio rerio.

Secondly, we investigated orthology relationships between the two species using a phylogenetic approach, especially for a co-orthology case, that is the case of the protein under accession number NP_00336.1, the third sub unit of a mitochondrial NADH dehydrogenase (ND3). We then retrieved a load of homologous sequences from different species. We then inferred a PhyML phylogeny from the latter set of sequences. In order to identify the speciation and duplication events, species tree reconciliation was performed. The corresponding species tree was created using mitochondrial 16s rRNA sequences (retrieved for each species under study from NCBI's Nucleotide database).

Lastly, through a python script, we computed the Shannon entropy at each site of the multiple sequence alignment as a measure of conservation. It allows us to identify conserved regions among the ND3 homologs. The whole sequence was globally highly conserved. Confirmed by NCBI's conserved domains database, the whole sequence has been previously identified as a conserved region.

Needless to say that this report still possesses a huge margin of improvement and is far from perfect. The following tasks merit to be enhanced:

1. Due to a problem in one of the BLASTp run (job timed out on the HPC), the total number of orthologous pair is slightly underestimated.
2. In-paralogs were identify solely based on the second best match of an alignment between the genomes of the same species. It covers only a part of the whole definition of in-paralogs.
3. The methodology for phylogenetic inference and for conservation assessment should be applied on the whole set of identified orthologs, instead of arbitrary choosing a specific co-orthology case of study.
4. Implementation of a conservation metric taking into account the adjacent residue to fully characterize regions instead of assessing the entropy independently, site by site.

[5] https://www.ncbi.nlm.nih.gov/Structure/cdd/cddsrv.cgi?ascbin=8&maxaln=10&seltype=2&uid=MTH00136, on the 29.01.2022

[6] https://www.ncbi.nlm.nih.gov/Structure/cdd/cl00535, on the 29.01.2022

5 Data and code accession

The full code and a part of the data is publicly accessible at: https://github.com/AntoineRuzy/comparative-genomics. Note that the full code has also been appended in the present report (see Appendix). BLASTp ouput files and protein FASTA files were too large to sit in the aforementioned repository.

References

[1] Meyer A, Dolven SI. *Molecules, fossils, and the origin of tetrapods.* J Mol Evol. 1992 Aug;35(2):102-13. doi: 10.1007/BF00183221. PMID: 1501250.

[2] Van Noort V. *Comparative and Regulatory Genomics course [I0U29a].* KU Leuven, MSc in Bioinformatics, Academic year 2021-2022.

[3] NCBI. *Whole-genome sequences retrieval for Latimeria chalumnae.* https://www.ncbi.nlm.nih.gov/data-hub/taxonomy/7897/, consulted on the 04.01.2022.

[4] NCBI. *Whole-genome sequences retrieval for Danio rerio.* https://www.ncbi.nlm.nih.gov/labs/data-hub/taxonomy/7955/, consulted on the 04.01.2022.

[5] Dalquen DA, Dessimoz C. *Bidirectional best hits miss many orthologs in duplication-rich clades such as plants and animals.* Genome Biol Evol. 2013;5(10):1800-6. doi: 10.1093/gbe/evt132. PMID: 24013106; PMCID: PMC3814191.

[6] Singh AK, Kumar R, Singh M, Mishra AK, Chauhan UK, Baisvar VS, Verma R, Nagpure NS, Kushwaha B. *Mitochondrial 16S rRNA gene-based evolutionary divergence and molecular phylogeny of Barilius spp.* Mitochondrial DNA. 2015 Feb;26(1):41-7. doi: 10.3109/19401736.2013.815168. Epub 2013 Jul 24. PMID: 23883181.

[7] Schmitt AO, Herzel H. *Estimating the entropy of DNA sequences.* J Theor Biol. 1997 Oct 7;188(3):369-77. doi: 10.1006/jtbi.1997.0493. PMID: 9344742.

6 Appendix

6.0.1 Co-orthology full python script

```
"""
Created on Wed Jan 26 18:30:42 2022

@author: Antoine Ruzette
Comparative Genomics, MSc in Bioinformatics, KU Leuven, 2021-2022
"""
import pandas as pd
import re

##DATA FRAME INITIALIZATION
early_match = []
toremove3 = []
toremove4 = []

def initializationDataFrame():
        query_gene = []
        alignment_gene = []
        score = []

        return query_gene, alignment_gene, score

##EXTRACT QUERY AND ALIGNES GENE NAMES AND SCORES FROM THE BLAST FILES
def extractGenes(filename, dict_align):
```

```
    with open(filename, 'r') as align1:
        for ln in align1:
            #get the query gene name
            if ln.startswith("Query="):
                ID_query = re.search('[XN]P_\d{5,}\.\d{1}', ln).group()
                (dict_align[list(dict_align.keys())[0]]).append(ID_query)#7:21
                    corresponds to the query gene ID
            #get the best target gene name & score
                test = align1.readline()

                while (not re.search('[XN]P_\d{5,}\.\d{1}', test)):
                    test = align1.readline()#continue to read
                ID_target = re.search('[XN]P_\d{5,}\.\d{1}', test).group()
                (dict_align[list(dict_align.keys())[1]]).append(ID_target)
                (dict_align[list(dict_align.keys())[2]]).append(test[70:74])
    return dict_align;


##DEFINE ORTHOLOGY
def orthology(dict_align1, dict_align2):
    query_gene2 = (dict_align2[list(dict_align2.keys())[0]])
    query_gene1 = (dict_align1[list(dict_align1.keys())[0]])
    alignment_gene2 = (dict_align2[list(dict_align2.keys())[1]])
    alignment_gene1 = (dict_align1[list(dict_align1.keys())[1]])
    score2 = (dict_align2[list(dict_align2.keys())[2]])
    score1 = (dict_align1[list(dict_align1.keys())[2]])

    for j in range(len(query_gene1)) :
        for i in range(len(query_gene2)) :

            if ( query_gene2[i] == alignment_gene1[j] and alignment_gene2[i] ==
                query_gene1[j] ) :
                early_match.append([query_gene2[i], score2[i], query_gene1[j],
                    score1[j]])#Order: Danio, Latimeria
                #query_gene2 -> Danio
                #alignment_gene2 -> Latimeria
                #query_gene1 -> Latimeria
                #alignment_gene1 -> Danio

def in_paralogy(filename, dict_align, toremove):

    with open(filename,"r") as align_self:
        for ln in align_self:
            #get the query gene name
            if ln.startswith("Query="):
                ID_query = re.search('[XN]P_\d{5,}\.\d{1}', ln).group()
                (dict_align[list(dict_align.keys())[0]]).append(ID_query)
            #get the best alignment gene name & score
                test = align_self.readline()

                while (not re.search('[XN]P_\d{5,}\.\d{1}', test)) :
                    test = align_self.readline()#continue to read

                tmp = align_self.readline()#goes to second best alignment
                if (re.search('[XN]P_\d{5,}\.\d{1}', tmp)) :
                    ID_target = re.search('[XN]P_\d{5,}\.\d{1}', tmp).group()
                    (dict_align[list(dict_align.keys())[1]]).append(ID_target)
```

```
                    (dict_align[list(dict_align.keys())[2]]).append(tmp[70:74])
                    for z in range(len(early_match)) :
                        if (dict_align == dict_align3): #for latimeria_self
                            if (early_match[z][2] == ID_query and tmp[70:74] >
                                early_match[z][3]) :#for query_gene1 = latimeria
                                toremove.append(ID_query)
                        else:                                   #for danio_self
                            if (early_match[z][0] == ID_query and tmp[70:74] >
                                early_match[z][1]) :#for query_gene2 = danio
                                toremove.append(ID_query)
    return toremove;


def co_orthology():
    #TO BE DEFINED, as a pair of in-paralogous sequences that are both
        orthologous to
    return

#data frame initialization
for i in 1,2,3,4:
    globals()['dict_align' + str(i)] = {('query_gene' + str(i)) :
        initializationDataFrame()[0],
                                        ('alignment_gene' + str(i)) :
                                            initializationDataFrame()[1],
                                        ('score' + str(i)) :
                                            initializationDataFrame()[2]}
    i += 1

#query and aligned genes extraction

#extractGenes("alignment_test.txt", dict(dict_align1))
#extractGenes("alignment_test.txt", dict(dict_align2))

extractGenes("danio_latimeria-2-alignment.txt", dict(dict_align1))#DB: danio
extractGenes("latimera_rerio-2-alignment.txt", dict(dict_align2))#DB: latimera


#compute orthologous genes according to BBH
orthology(dict_align1, dict_align2)

#save results of orthology to file
column_names = ["Gene1 (Latimeria)", "Score1", "Gene2 (Danio)", "Score2"]
full_ortho = pd.DataFrame(columns = column_names)
for i in range(len(early_match)):
    full_ortho = full_ortho.append({"Gene1 (Latimeria)": early_match[i][2], "
        Score1": early_match[i][3], "Gene2 (Danio)":early_match[i][0], "Score2":
        early_match[i][1]}, ignore_index=True)
#send it to a text file
full_ortho.to_csv(r'full_ortho.txt', header = True, sep = '\t')


in_paralogy("latimeria_self-alignment.txt", dict(dict_align3), toremove3)#DB
    and query: latimeria
in_paralogy("danio_self-alignment.txt", dict(dict_align4), toremove4)#DB and
    query: danio
#in_paralogy("alignment_test.txt", dict(dict_align3), toremove3)#DB and query:
    latimeria
```

```
#save results of in-paralogy to file
column_paralogs = ["Paralogs"]
paralogs = pd.DataFrame(columns = column_paralogs)
for i in range(len(toremove3)):
    paralogs = paralogs.append({"Paralogs": toremove3[i]}, ignore_index=True)
for i in range(len(toremove4)):
    paralogs = paralogs.append({"Paralogs": toremove4[i]}, ignore_index=True)
paralogs.to_csv(r'paralogs.txt', header = True, sep = '\t')\\\\
```

6.0.2 Conservation full python script

```
"""
Created on Fri Jan 28 21:52:58 2022

@author: Antoine Ruzette
Comparative Genomics, MSc in Bioinformatics, KU Leuven, 2021-2022
"""

import re
import pandas as pd
import numpy as np
import math
from collections import Counter
import matplotlib.pyplot as plt


def split(word): #split a string into its characters
    return [char for char in word]


def shannon(msa_list): #compute the shannon entropy of a list
    score=0
    count = Counter(msa_list)
    freq = []

    for key in count:
        p = count[key]
        p = p/len(msa_list)
        freq.append(p)

    for m in range(len(freq)):
        new_score = freq[m]*math.log2(freq[m])
        shannon = (score + new_score)*-1

    return shannon


msa = []
msa_tmp = []

#retrieve IDs and sequences from the .clw file
file_in = open("Original_MSA_Muscle.clw")
for line in file_in:
    if line.startswith('CLUSTAL'):
        continue #skip the header of the .clw file
```

```
    elif re.match(r'\s', line):
            continue #skip empty lines
    else:
        msa.append(line[0:31])#IDs
        msa_tmp.append(line[37:98])#protein sequences

msa_list = list(dict.fromkeys(msa))#get the IDs, remove duplicates

#merge the sequences from different lines
master = []
for i in range(len(msa_list)):
    master.append(''.join([msa_tmp[i], msa_tmp[i+len(msa_list)]]))


output = []
for grp in master: #split each sequence into its residues and store it in a
    nested list
    output.append(split(''.join(grp)))
msa_table = pd.DataFrame(data=output, index= msa_list)


shannon_list = []
for i in range(len(output[0])):
    shannon_tmp = shannon(msa_table[i])
    shannon_list.append(shannon_tmp)


#plot 1D list of shannon entropy of each site
x = np.array(range(0, len(shannon_list)))
x_tick = np.arange(0, len(shannon_list), 10)
y = shannon_list

plt.scatter(x, y, color = "darkblue", label = "residue's␣shannon␣entropy", s =
    8, alpha = 1)
plt.xlabel("Residue␣position␣in␣the␣MSA")
plt.ylabel("Shannon␣entropy")
plt.title("Shannon␣entropy␣of␣each␣position␣in␣the␣MSA")
```