Homology search

Lauri Mesilaakso

2019-09-26

Contents

Introduction

The approach for finding putative homologous wing development genes from *Drosophila melanogaster* listed in Table 1 from taxa listed in Table 2 is the following:

Table 1: An ordinal number and the name of the gene *Drosophila melanogaster* gene with its gene symbol in parentheses and FlyBase gene id of the gene ID.

No	Name	FlyBase gene ID
1	Crustacean cardioactive peptide (CCAP)	FBgn0039007
2	Eclosion hormone (Eh)	FBgn0000564
3	Bursicon (Burs)	FBgn0038901
4	Ecdysone receptor (EcR)	FBgn0000546
5	ultraspiracle (usp)	FBgn0003964
6	Imitation SWI (Iswi)	FBgn0011604
7	broad (br)	FBgn0283451
8	ftz transcription factor 1 (ftz-f1)	FBgn0001078
9	Ecdysone-induced protein 74EF (Eip74EF)	FBgn0000567
10	Death-associated APAF1-related killer (Dark)	FBgn0263864
11	Death related ICE-like caspase (Drice)	FBgn0019972
12	wingless (wg)	FBgn0284084
13	Distal-less (Dll)	FBgn0000157
14	engrailed (en)	FBgn0000577
15	Ultrabithorax (Ubx)	FBgn0003944
16	extradenticle (exd)	FBgn0000611
17	scalloped (sd)	FBgn0003345
18	spalt major (salm)	FBgn0261648
19	spalt-adjacent (sala)	FBgn0003313
20	spalt-related (salr)	FBgn0000287
21	Insulin-like receptor (InR)	FBgn0283499

Table 2: An ordinal number, the species from which putative homolous genes are searched from and wing morphology of the species.

No	Species	Wing morphology
1	C hookeri (smooth stick-insect)	Apterous
2	C lectularius (bed bug)	Apterous
3	M extradentata (Vietnamese walking stick)	Apterous
4	T cristinae	Apterous
5	D melanogaster (fruit fly)	Macropterous
6	D simulans (fruit fly)	Macropterous
7	A pisum (pea aphid)	Polyphenic
8	F exsecta (narrow-headed ant)	Polyphenic
9	G buenoi (water strider)	Polyphenic
10	N lugens (brown planthopper)	Polyphenic

1. Retrieve from various sources published genome assemblies (`.fna.gz`), annotation files (`.gff.gz`), annotated multifasta protein files (`pep.fa.gz`) and for *Drosophila melanogaster* lexicographically first annotated translated alternatively spliced wing development gene isoforms. In Table 3 is listed what is retrieved from where.

Table 3: Name and type of source downloaded for the project, the actual source where the downloads were from and accession and version or notes regarding the downloaded data.

No	Name and type	Accession.version or notes
1	*A pisum* genome annotation	Annotation of assembly: GCF_005508785.1
2	*C hookeri* annotated protein sequences	Annotated proteins of assembly: GCA_002778355.1
3	*C hookeri* genome annotation	Annotation of assembly: GCA_002778355.1
4	*C hookeri* genome assembly	GCA_002778355.1
5	*C lectularius* annotated protein sequences	Annotated proteins of assembly: GCF_000648675.2
6	*C lectularius* genome annotation	Annotatation of assembly: GCF_000648675.2
7	*D melanogaster* genome annotation	Annotatation of assembly: GCF_000001215.4
8	*D melanogaster* wing protein sequences	See Table 1 for which genes
9	*D simulans* annotated protein sequences	Annotated proteins of assembly: GCA_000754195.3

No	Name and type	Accession.version or notes
10	*D simulans* genome annotation	Annotation of assembly: GCF_000754195.2
11	*F exsecta* genome annotation	Annotation of assembly: GCF_003651465.1
12	*G buenoi* annotated protein sequences	Annotated proteins of official gene set version 1.1
13	*G buenoi* genome annotation	Annotation of official gene set version 1.1.1
14	*M extradentata* annotated protein sequences	Annotated proteins of assembly: GCA_003012365.1
15	*M extradentata* genome annotation	Annotation of assembly: GCA_003012365.1
16	*M extradentata* genome assembly	GCA_003012365.1
17	*N lugens* genome annotation	Annotation of assembly: GCA_000757685.1
18	*T cristinae* genome annotation	Annotation of assembly: GCA_002928295.1

2. Search for homologous protein sequences of Table 1 listed *Drosophila melanogaster* wing development genes in taxa listed in Table 2 using the following methods (in order):

    1. Find gene models with same names in annotation files 1, 3, 6, 7, 10, 11, 13, 15, 17 and 18 in Table 3 by using gene names listed in Table 1 as search terms (which were in practice regular expressions).

    2. If there weren't any name matches found for certain genes in certain species and in addition to a `.gff` annotation file, there could be found annotated multifasta polypeptide file (2, 5, 9, 12 and 14 in Table 3) for the taxon, a search with exonerate v. 2.4.0 (Slater and Birney 2005) was executed against these multifasta polypeptide files using *Drosophila melanogaster* translated wing development genes (8 in Table 3).

    3. If there weren't any alignments which seemed to fit (see step 4 further on) with the other putative *D melanogaster* homologues when searching with exonerate v. 2.4.0 against multifasta polypeptide files of certain taxa and certain genes, another search with exonerate was executed using the same protein sequences against the whole genome assemblies (4 and 16 in Table 3).

3. Create multiple protein alignments (MPAs) of putative homologous protein sequences which had alignments with highest exonerate raw scores, query coverages and alignment lengths and which were found in most taxa listed in Table 2. For each species was picked out 2-5 best alignments. The MPAs were executed with MAFFT v 7.407 (Katoh and Standley 2013).

4. Refine alignments produced in step 3 by removing not so well aligned proteins with eye-balling the alignments. After the removal a new alignment with MAFFT v 7.407 was executed. This was repeated until all protein sequences seemed to fit better with each other.
5. Create approximately-maximum-likelihood phylogenetic trees from final alignments of step 4 using FastTree v. 2.1.10 (Price, Dehal, and Arkin 2010).

Let's go through the steps above:

Chapter 1

Retrieve input data

In this section the annotation files (`.gff`) and checksum files available for these annotations are downloaded. Then the md5checksums are calculated for each downloaded file and search in the checksum files. If each checksum is found in the checksum files the download can be confirmed to have been succesful.

```
#CURR_DATE=$(date +%F)

checksums_output="data/2019-08-15/checksums"
annotations_output="data/2019-08-15/annotations"

# Create a directory for annotation files
if [ ! -d $annotations_output ]
then
  mkdir -p $annotations_output
  echo "Creating $annotations_output directory"
else
  echo "$annotations_output directory already exists, skipping..."
fi

# Create a directory for checksum files
if [ ! -d $checksums_output ]
then
  mkdir -p $checksums_output
  echo "Creating $checksums_output directory"
else
  echo "$checksums_output directory already exists, skipping..."
fi

# Define the downloadable data and the organisms in an associative array
declare -A annotations
```

```
annotations["ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/002/778/355/GCA_002778355.1_ASM27783
annotations["ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/648/675/GCF_000648675.2_Clec_2.1
annotations["ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/003/012/365/GCA_003012365.1_ASM30123
annotations["ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/001/215/GCF_000001215.4_Release_
annotations["ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/754/195/GCF_000754195.2_ASM75419
annotations["ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/005/508/785/GCF_005508785.1_pea_aphi
annotations["ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/003/651/465/GCF_003651465.1_ASM36514
annotations["ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/757/685/GCF_000757685.1_NilLug1.
annotations["ftp://ftp.ncbi.nlm.nih.gov/genomes/genbank/invertebrate/Timema_cristinae/latest
annotations["ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/001/010/745/GCA_001010745.2_Gbue_2.0

for record in "${!annotations[@]}";
do
  # Parse the two URLs in the keys
  annotationURL=$(echo "${record}" | awk -F, '{print $1}')
  checksumURL=$(echo "${record}" | awk -F, '{print $2}')
  #echo $checksumURL
  #echo $annotationURL
  # Make it easier to see what is what
  species=$(echo "${annotations[$record]}")
  # Quick and dirty way of extracting the current file extension
  file_extension=$(echo $(basename "$annotationURL") | awk -F "_genomic." '{print $2}')

  # Output files
  checksum_output_file=$checksums_output/$(echo $species)"_checksums.txt"
  annotation_output_file=$annotations_output/$(echo $species)"."$file_extension
  # First download both the annotations files and their checksums
  wget -c -O $checksum_output_file $(echo $checksumURL)
  wget -c -O $annotation_output_file $(echo $annotationURL)
  #echo "${record}" # The keys
  #echo "${annotations[$record]}" # The value
  zgrep "$(md5sum $annotation_output_file | awk '{print $1}')" $checksum_output_file
done
```

1.1 Download missing *C hookeri* data

C hookeri annotations file and polypeptide multifasta files weren't downloaded above so let's do it now:

```
downloads_dir="data/2019-08-28"
wget -c -O $downloads_dir/"C_hookeri.gff3.gz" https://i5k.nal.usda.gov/sites/default/files/d
wget -c -O $downloads_dir/"C_hookeri_pep.fa.gz" https://i5k.nal.usda.gov/sites/default/files
```

1.2 Download *D simulans* polypeptides

Let's download the data:

```
unzip_dir="analyses/2019-09-02/polypeptides"
downloads_dir="data/2019-08-28"

wget -c -O $downloads_dir/"D_simulans_pep.fa.gz" "ftp://ftp.ensemblgenomes.org/pub/metazoa/r
```

1.3 Extract and write protein sequences as fasta files of genomes available in i5k Workspace@NAL

There are also annotations data about the species of interest in i5k Workspace@NAL which have not yet been published in NCBI.[1] By comparing the website data and with the currently downloaded data, the following species have newer annotations which weren't explored in steps above:

- *Cimex lectularius*
- *Gerris buenoi*
- *Medauroidea extradentata*

So let's download the annotation files and multifasta files containing all polypeptides associated with the annotated proteins:

```
annotations_output="data/2019-08-28"

# Create a directory for annotation data
if [ ! -d $annotations_output ]
then
  mkdir -p $annotations_output
  echo "Creating $annotations_output directory"
else
  echo "$annotations_output directory already exists, skipping..."
fi

declare -A annotations
annotations["https://i5k.nal.usda.gov/sites/default/files/data/Arthropoda/cimlec-%28Cimex_le
annotations["https://i5k.nal.usda.gov/sites/default/files/data/Arthropoda/medext-%28Medauro
annotations["https://i5k.nal.usda.gov/sites/default/files/data/Arthropoda/gerbue-%28Gerris_b

declare -A polypeptides
```

[1]In i5k Workspace@NAL there was also annotation which had already been published in NCBI and included in the steps above. It was: Clitarchus hookeri.

```
polypeptides["https://i5k.nal.usda.gov/sites/default/files/data/Arthropoda/cimlec-%28Cimex_l
polypeptides["https://i5k.nal.usda.gov/sites/default/files/data/Arthropoda/medext-%28Medaur
polypeptides["https://i5k.nal.usda.gov/sites/default/files/data/Arthropoda/gerbue-%28Gerris_

# Download annotations
for record in "${!annotations[@]}";
do
  # Parse the two URLs in the keys
  annotationURL=$(echo "${record}")
  #echo $annotationURL
  # Make it easier to see what is what
  species=$(echo "${annotations[$record]}")
  # Quick and dirty way of extracting the current file extension
  file_extension_end=$(echo $(basename "$annotationURL") | awk -F ".gff" '{print $2}')

  # Output files
  annotation_output_file=$annotations_output/$(echo $species)".gff"$file_extension_end
  #echo $annotation_output_file
  # download the annotations files
  wget -c -O $annotation_output_file $(echo $annotationURL)
  #echo "${record}" # The keys
  #echo "${annotations[$record]}" # The value
done

# Download polypeptide multifasta files
for polyp_mf in "${!polypeptides[@]}";
do
  # Parse the two URLs in the keys
  annotationURL=$(echo "${polyp_mf}")
  #echo $annotationURL
  # Make it easier to see what is what
  species=$(echo "${polypeptides[$polyp_mf]}")

  # Quick and dirty way of extracting the current file extension
  file_extension_end=$(echo $(basename "$annotationURL") | awk -F "_pep." '{print $2}')

  # Output files
  annotation_output_file=$annotations_output/$(echo $species)"_pep."$file_extension_end
  #echo $annotation_output_file
  # download the polypeptides files
  wget -c -O $annotation_output_file $(echo $annotationURL)
  #echo "${polyp_mf}" # The keys
  #echo "${polypeptides[$polyp_mf]}" # The value
done
```

Let's now also gzip the files which were not in compressed format when downloaded. It saves space and the files have then also unified extensions.

```
annotations_dir="data/2019-08-28"

file_names=("C_lectularius.gff3" "C_lectularius_pep.fa" "G_buenoi.gff3" "G_buenoi_pep.fa")

# Print the csv body
for file in "${file_names[@]}"; do
  gzip $(echo $annotations_dir/$file)
done
```

Chapter 2

Extract name matches from .gff annotation files

Let's first find the number of mRNAs with the gene names in Table 1 in `.gff` files.

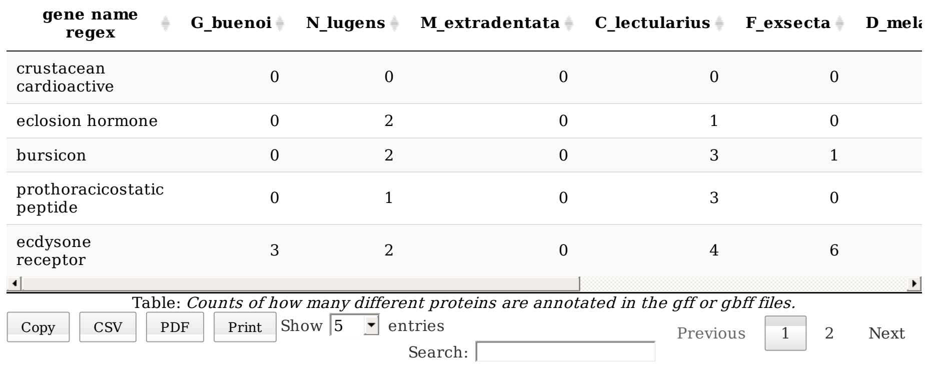
```
annotations="data/annotations"

results="data/annotations"

isoforms=("crustacean cardioactive" "eclosion hormone" "bursicon" "prothoracicostatic peptic

# Remove the previous version of the file so there won't be any dublicated data
rm -f "$results/num_isoforms.csv"

annotation_files_w_path=$( find $annotations -name "*.gz" -and -type f -print0 | xargs -0 ec

read -r -a annotation_files_w_path_array <<< $annotation_files_w_path

let len_annotation_array=${#annotation_files_w_path_array[@]}

# Print summary information of how many features were found for each .gff file

# Print the csv header
printf "gene name regex," >>$results"/num_isoforms.csv"
# Loop through all the organism names an print them
for (( i=0; i<$len_annotation_array; i++ )); do
  annotation="${annotation_files_w_path_array[$i]}"
  annotation_file_name=$(basename $(echo $annotation))
  organism_name=$( echo $annotation_file_name | awk -F "\.g" '{print $1}' )
  if (( $i == $len_annotation_array-1 )); then
    # Last organism so no comma
    printf "%s" "$organism_name" >>$results"/num_isoforms.csv"
  else
    printf "%s," "$organism_name" >>$results"/num_isoforms.csv"
  fi
done
printf "\n" >>$results"/num_isoforms.csv"

# Print the csv body
for isoform in "${isoforms[@]}"; do
  # Print the isoform name regex
  printf "%s," "$isoform" >>$results"/num_isoforms.csv"

  # Let's search trough each annotation with the current isoform regex
  for (( i=0; i<$len_annotation_array; i++ )); do

    annotation="${annotation_files_w_path_array[$i]}"
    # Parse the organism name out of the file names
    annotation_file_name=$(basename $(echo $annotation))

    # Store the organism name of the current file, e.g. N_lugens
    organism_name=$( echo $annotation_file_name | awk -F "\.g" '{print $1}' )

    # Let's search with just the gene names for the four exceptions
    if [ "$organism_name" == "M_extradentata" ]; then
      #echo "Matched: $organism_name"
      match_count=$(zgrep -Eic "$isoform" "$annotation")
    else
      #echo "Not matched: $organism_name"
      match_count=$(zgrep -Eic "mRNA\s\w+.+$isoform" "$annotation")
    fi

    # Don't print a comma to the end of the line
    if (( $i == $len_annotation_array-1 )); then
      # Last annotation so no comma
      printf "%s" "$match_count" >>$results"/num_isoforms.csv"
    else
      printf "%s," "$match_count" >>$results"/num_isoforms.csv"
    fi
  done
  printf "\n" >>$results"/num_isoforms.csv"
done
```

2.1 Summarise extracted name matches

Let's now take a look at how many of the proteins could be found in each `.gff`-file with these text searches:

```
library(tidyverse)
library(DT)

read_csv("data/annotations/num_isoforms.csv") %>%
  datatable(
    rownames = FALSE,
    extensions = c('FixedColumns','Buttons','KeyTable'),
    options=list(
      scrollX=TRUE,
      dom = 'tBlrpf',
      fixedColumns = list(leftColumns = 1),
      buttons = c('copy', 'csv', 'excel', 'pdf', 'print'),
      keys = TRUE,
      pageLength = 5
      ), caption = htmltools::tags$caption(
    style = 'caption-side: bottom; text-align: center;',
    'Table: ', htmltools::em('Counts of how many different proteins are annotated in the gf
    )
```

```
## Warning: 1 parsing failure.
## row col   expected    actual                               file
##   9  -- 10 columns 4 columns 'data/annotations/num_isoforms.csv'
```



gene name regex	G_buenoi	N_lugens	M_extradentata	C_lectularius	F_exsecta	D_mela
crustacean cardioactive	0	0	0	0	0	
eclosion hormone	0	2	0	1	0	
bursicon	0	2	0	3	1	
prothoracicostatic peptide	0	1	0	3	0	
ecdysone receptor	3	2	0	4	6	

Table: *Counts of how many different proteins are annotated in the gff or gbff files.*

count-1.bb

Chapter 3

Search with exonerate against protein multifasta files

As there are too many species (esp. monomorphic apterous) with no name hits in the annotation files, these gaps should be patched in some way. So the plan of action for patching involves:

Use only one (lexicographically first) *D melanogaster* translated isoform of each gene to search the annotated protein data of:

- *Cimex lectularius*
- *Gerris buenoi*
- *Medauroidea extradentata*
- *Clitarchus hookeri*
- *Drosophila simulans*

The search is done with exonerate version 2.4.0 (Slater and Birney 2005).

The differences between isoforms in comparison to differences between orthologous genes (with long evolutionary distances), are so small that just picking one isoform is going to carry essentially enough to align a protein sequence to another orthologous protein sequence.

From these exonerate alignments are picked some subset (maybe 3 or 4) of best hits (with minimum of 30 percent identity between the query and subject sequences) and a tree should be built from them to see which are paralogues among the hits and which of them would be most suitable candidate for multiple protein alignment.

So what we need now is to first retrieve the lexicographically first translated isoforms of each gene.

3.1 Retrieve *D melanogaster* gene sequences

```
library(jsonlite)
library(tidyverse)


path <- "data/D_mel_query_proteins/json/"

file <- dir(path, pattern ="*.json")

# Parse and store the .json data from the files
data <- file %>%
       map_df(~fromJSON(file.path(path, .), flatten = TRUE))

# Hash storing human-readable names of FlyBase ID:s and their human-readable names
geneID_name_human_readable <- list(FBgn0039007="Crustacean cardioactive peptide (CCAP)", FBg

# Initialise an empty data.frame for all the sequences
seq_df <- data.frame()
# Initialise an empty character string for the FlyBase ID
FlyBase_id <- ""

# Loop through the parsed and flattened json files data
for(i in 1:length(data$resultset)){
  # Store the current row in a variable
  current_row <- data$resultset[[i]]
  # Find the http request URL in order to get the FlyBase ID
  if(is.character(current_row)){
    if(str_detect(current_row, "http.+")){
      # Store the current FlyBase_ID and ditch the rest
      FlyBase_id <- sub("http://api.flybase.org/api/v1.0/sequence/id/", "", current_row) %>%
        sub("/FBpp", "",.)
    }
  }
  # If the current row is data.frame, this is where the sequences are, so let's loop through
  if(is.data.frame(current_row)){
    for(j in 1:length(current_row$sequence)){
      # Parse out isoform name from the description
      isoform_description <- current_row$description[[j]] %>%
        strsplit(";")
      isoform_name <- isoform_description[[1]][4] %>%
        sub(" name=", "",.)
      # Store the sequence row data in own data.frame variable
      new_seq_row <- data.frame(GeneID_name_human_readable=geneID_name_human_readable[[FlyBa
                               FlyBase_ID=FlyBase_id,
                               Polypeptides_ID=current_row$id[j],
                               isoform_name=isoform_name,
                               sequences=current_row$sequence[j])
      # Append the sequences to the previously defined data.frame
      seq_df <- rbind(seq_df,new_seq_row)
    }
  }
}

# Filter so that only first translated isoforms are left
filtered <- seq_df %>% filter(grepl("-PA", isoform_name))

multifastaRows <- character(nrow(filtered) * 2)
multifastaRows[c(TRUE, FALSE)] <- paste0(">",filtered$isoform_name,
                                         "_",filtered$Polypeptides_ID,
                                         "_",filtered$FlyBase_ID,
                                         "_",filtered$GeneID_name_human_readable)
multifastaRows[c(FALSE, TRUE)] <- as.character(filtered$sequences)

# Create a multifasta file
writeLines(multifastaRows, "data/D_mel_query_proteins/proteins.fasta")
```

Let's lastly separate the multifasta records to single fasta files with this awk script:

```
## #!/usr/bin/awk -f
## /^>/ {
##     header = $0;
##     split(header, header_fields, "_");
##     # Get rid of ">" in the beginning
##     gsub(">", "", header_fields[1]);
##     filename = header_fields[1];
##     getline; # Move reading to next line
##     sequence = $0;
##     # Check if there is an output folder given, if not give this generic one as such
##     if (length(output_folder) == 0){
##         output_folder = "analyses/"
##     }
##     printf("%s\n%s",header,sequence)  > output_folder""filename".fas"
##     }
##
```

It can be run like this:

```
current_root="data/D_mel_query_proteins"
cat $current_root/"proteins.fasta" | awk -f "code/separate_fasta.awk" -v output_folder=$curr
```

3.2 Run exonerate in order to find *D melanogaster* homologues

Let's find the homologues in these species now:

- *C lectularius*
- *D simulans*
- *C hookeri*
- *G buenoi*
- *M extradentata*

```
results="analyses/exonerate_against_5_pps"
unzip_dir="data/polypeptides/unziped"
protein_data="data/polypeptides/gziped"
D_mel_proteins="data/D_mel_query_proteins/fasta/"

# The protein data from each of the 5 species
pep_files=$( find $protein_data -name "*_pep.fa.gz" -and -type f -print0 | xargs -0 echo )
read -r -a pep_files_array <<< $pep_files
let len_pep_files_array=${#pep_files_array[@]}

D_mel_protein_files=$( find $D_mel_proteins -name "*.fas" -and -type f -print0 | xargs -0 ec
#echo $D_mel_protein_files
read -r -a D_mel_proteins_array <<< $D_mel_protein_files
let len_D_mel_proteins_array=${#D_mel_proteins_array[@]}

# Loop through all the organism names an print them
for (( i=0; i<$len_pep_files_array; i++ )); do
  peptide_file="${pep_files_array[$i]}"
  peptide_file_name=$(basename $(echo $peptide_file))
  organism_name=$( echo $peptide_file_name | awk -F "_pep" '{print $1}' )
  #echo $peptide_file, $peptide_file_name, $organism_name
  unzip_output_fn=$unzip_dir/$organism_name"_pep.fa"
  #echo $unzip_output_fn
  # Unzip the file if necessary
  if [ ! -f $unzip_output_fn ]; then
    zcat $peptide_file > $unzip_output_fn
    echo "Unzipping: $peptide_file"
  else
```

```
    echo "$unzip_output_fn file already exists!"
  fi

  for (( j=0; j<$len_D_mel_proteins_array; j++ )); do
    D_mel_prot_file="${D_mel_proteins_array[$j]}"
    D_mel_prot_file_name=$(basename $(echo $D_mel_prot_file))
    gene_symbol=$( echo $D_mel_prot_file_name | awk -F "\." '{print $1}' )
    exonerate --model affine:local --proteinsubmat pam250 --refine full $D_mel_prot_file $un
  done
done
```

3.3 Scrape best hit data from exonerate output

Let's scrape some essential data from all the exonerate results:

```
exonerate_results="analyses/exonerate_against_5_pps"
prot_fasta="data/D_mel_query_proteins/fasta/"
output_results_directory="analyses/exonerate_against_5_pps/scraped_exonerate_best_hits_for_5

# Define exonerate result filtering criteria
query_coverage_lower_bound="0.10"
alignment_length_lower_boundary="0"
minimum_raw_score="50"
number_of_hits="10"

# Acquire all D melanogaster query protein fasta sequences
pep_files=$( find $prot_fasta -name "*.fas" -and -type f -print0 | xargs -0 echo )
read -r -a pep_files_array <<< $pep_files
let len_pep_files_array=${#pep_files_array[@]}

# Take each protein sequence at a time and scrape the results into several (intermediary) cs
for (( i=0; i<$len_pep_files_array; i++ )); do
  peptide_file="${pep_files_array[$i]}"
  protein_symbol=$(echo $(basename $(echo $peptide_file)) | awk -F "." '{print $1}' )
  protein_length=$(fastalength $peptide_file | awk '{print $1}')
  #echo $peptide_file, $protein_symbol, $protein_length
  curr_prot_exonerate_res=$( find $exonerate_results -name "$protein_symbol*" -and -type f -
  # Create a smaller array with the current protein's results
  read -r -a curr_proteins_array <<< $curr_prot_exonerate_res
  let len_curr_proteins_array=${#curr_proteins_array[@]}

  for (( j=0; j<$len_curr_proteins_array; j++ )); do
    curr_prot_results_file="${curr_proteins_array[$j]}"
    curr_organism=$( echo $(basename $(echo $curr_prot_results_file)) | awk -F "." '{print $
    #echo $curr_prot_results_file, $curr_organism
    grep '^vulgar' $curr_prot_results_file | awk -v len=$protein_length '{ print $2, $6, $4
    cat temp_exonerate.res | awk -v organism=$curr_organism -v output_dir=$output_results_di
    done
done
# Remove temporary results file
rm temp_exonerate.res

# Merge .csv files by appending them all together into one

intermediary_csvs=$( find $output_results_directory"intermediary_csvs/" -name "*.csv" -and -
read -r -a intermediary_csvs_array <<< $intermediary_csvs
let len_intermediary_csvs_array=${#intermediary_csvs_array[@]}

# Create the csv header row
echo "organism,seq_length,query_coverage,raw_score,gene_symbol,flybase_prot_id,flybase_gene_

# Loop and append to the one file
for csv in "${!intermediary_csvs_array[@]}"; do
  cat ${intermediary_csvs_array[$csv]} >> $output_results_directory"exonerate_res.csv"
done
```

3.4 Visualise best exonerate hits against polypeptide files

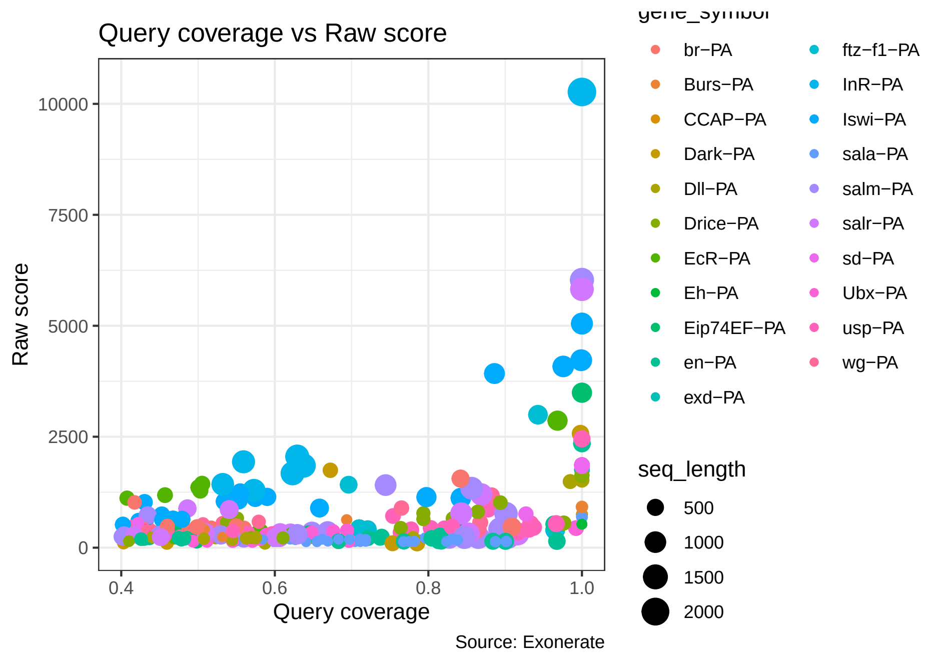
3.4.1 Visualise all matches together

Let's now visualise the exonerate results:

```
# load package and data
options(scipen=999)  # turn-off scientific notation like 1e+48
library(ggplot2)
library(tidyverse)
theme_set(theme_bw())  # pre-set the bw theme.

exonerate_data <- read_csv("analyses/exonerate_against_5_pps/scraped_exonerate_best_hits_for

gg <- ggplot(exonerate_data, aes(x=query_coverage, y=raw_score)) +
  geom_point(aes(col=gene_symbol, size=seq_length)) +
 # geom_smooth(method="loess", se=F) +
  xlim(c(0.4, 1.0)) +
  ylim(c(0, 10500)) +
  labs(y="Raw score",
       x="Query coverage",
       title="Query coverage vs Raw score",
       caption = "Source: Exonerate")
```

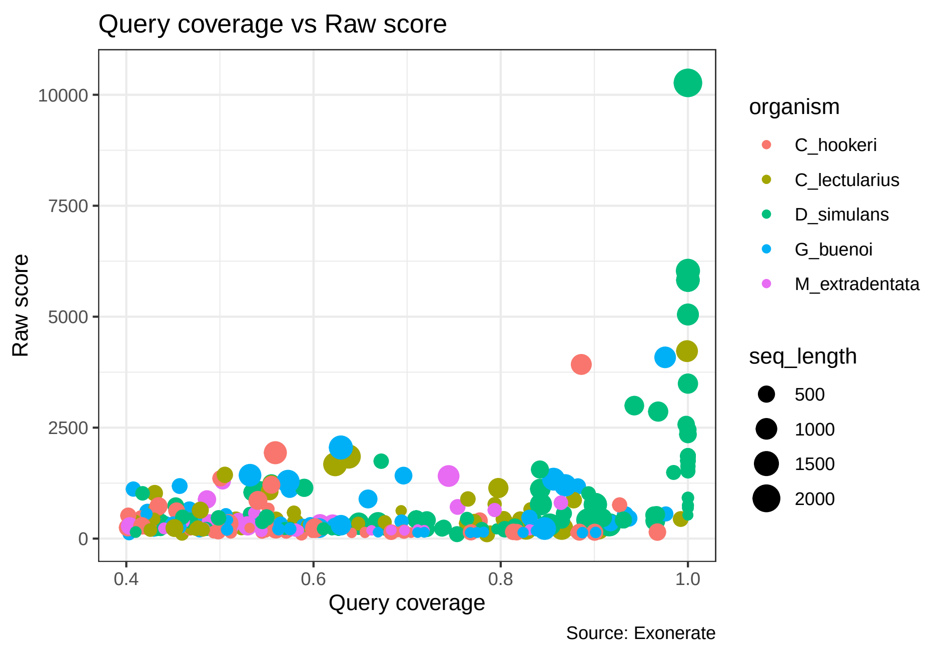
```r
plot(gg)
```

```
## Warning: Removed 428 rows containing missing values (geom_point).
```



```r
gg <- ggplot(exonerate_data, aes(x=query_coverage, y=raw_score)) +
  geom_point(aes(col=organism, size=seq_length)) +
 # geom_smooth(method="loess", se=F) +
  xlim(c(0.4, 1.0)) +
  ylim(c(0, 10500)) +
  labs(y="Raw score",
       x="Query coverage",
       title="Query coverage vs Raw score",
       caption = "Source: Exonerate")

plot(gg)
```

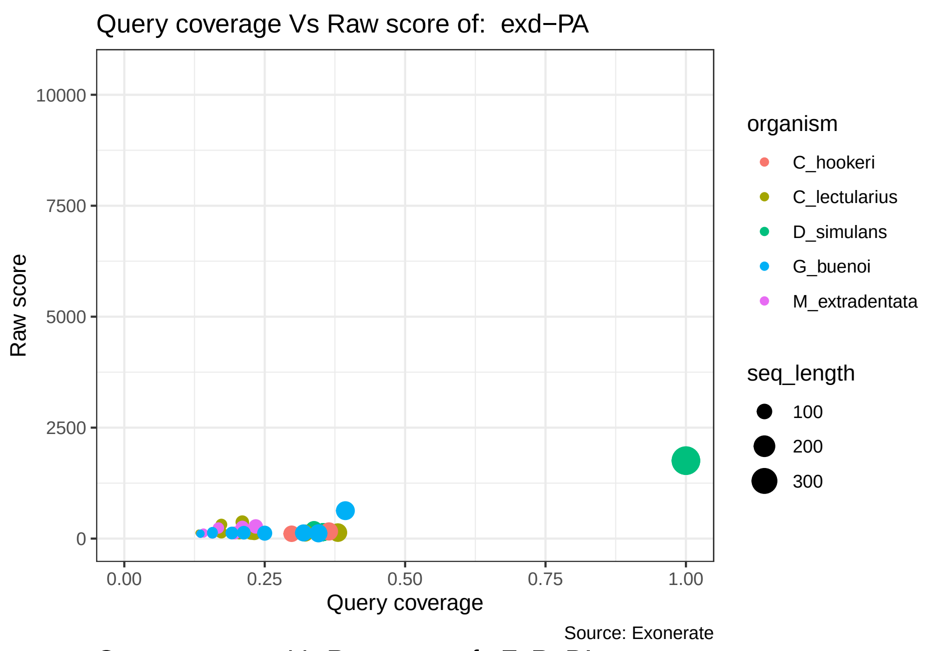
```
## Warning: Removed 428 rows containing missing values (geom_point).
```

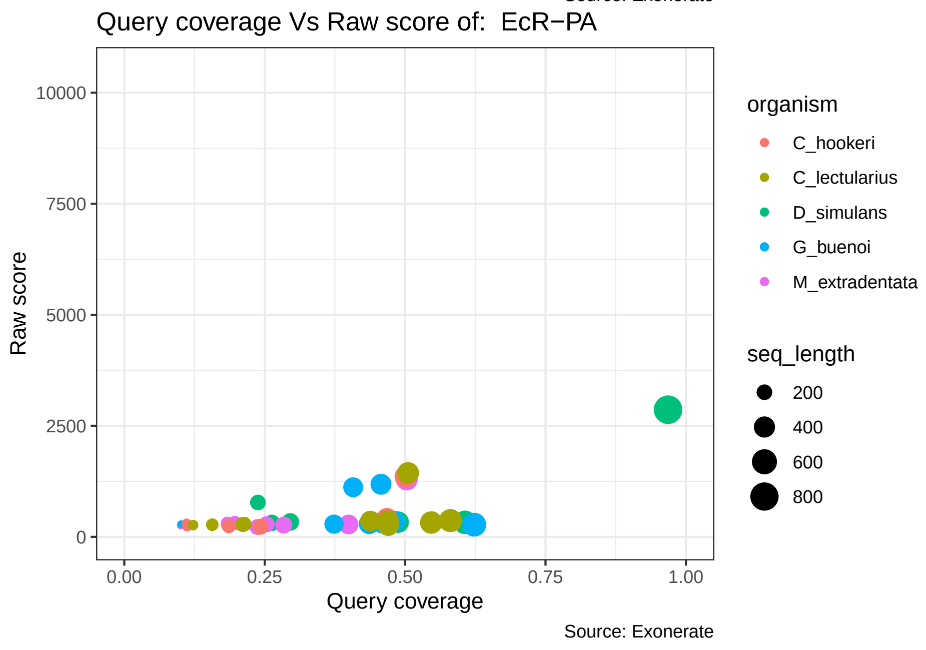


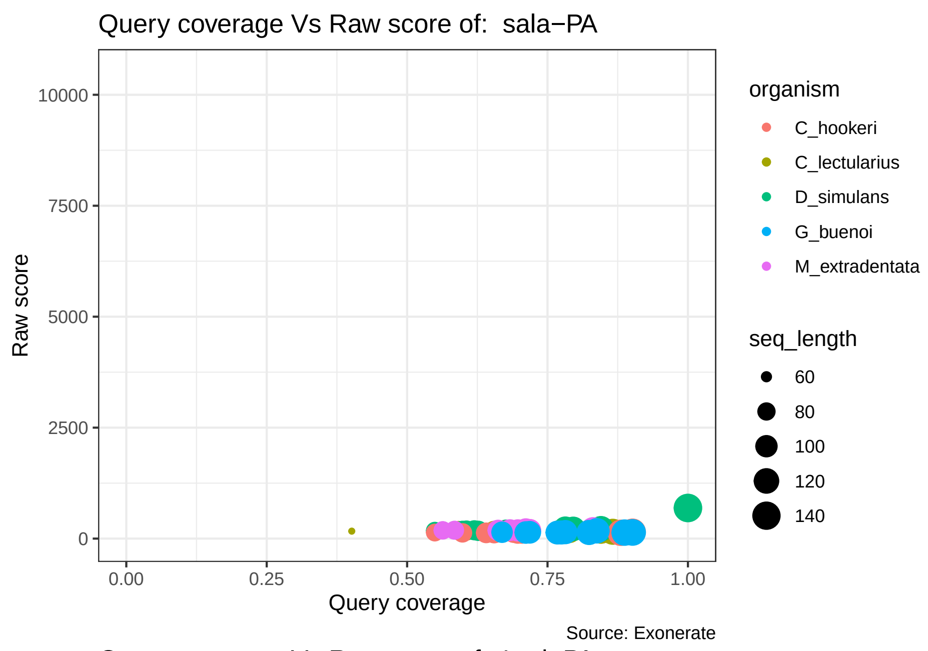
3.4.2 Visualise each gene by itself

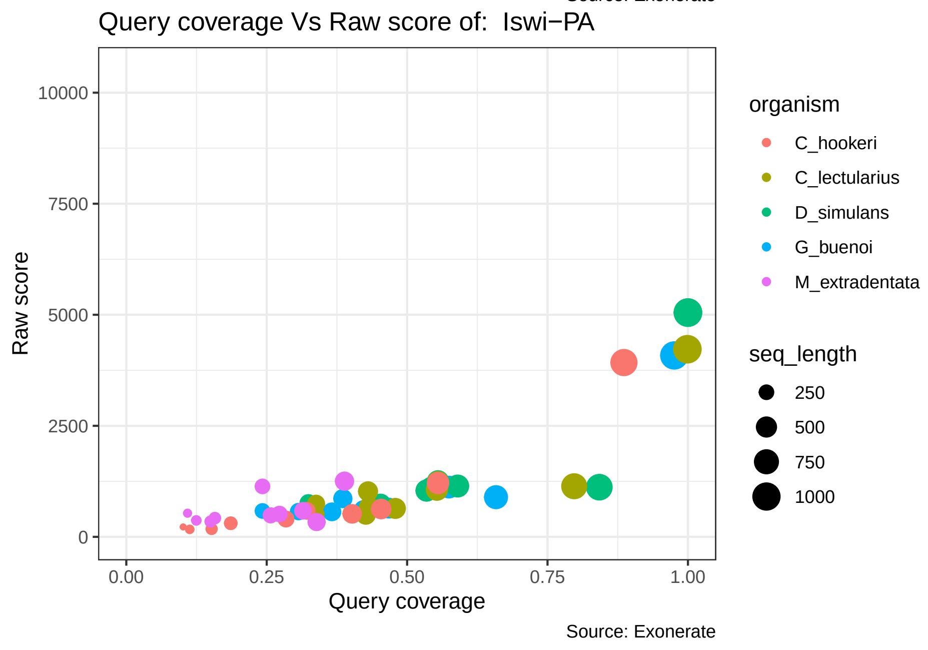
Let's now visualise the maximum of ten best hits for each gene:

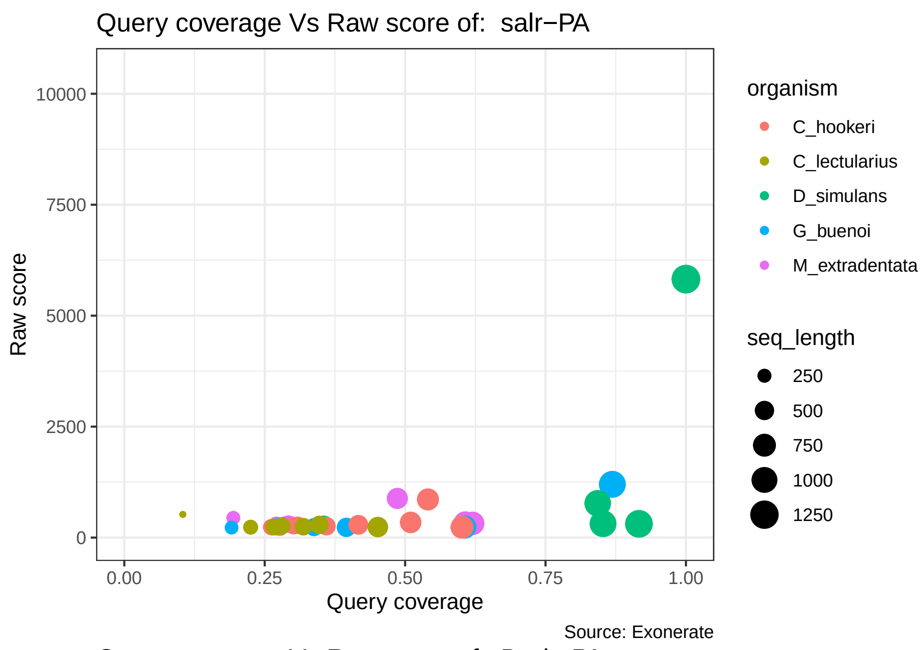
```r
gene_symbols <- exonerate_data %>% distinct(gene_symbol)

for(gene in gene_symbols){
  for(i in gene){
    curr_title = paste("Query coverage Vs Raw score of: ", i)
    filtered <- exonerate_data %>% filter(grepl(i, gene_symbol))
    gg <- ggplot(filtered, aes(x=query_coverage, y=raw_score)) +
    geom_point(aes(col=organism, size=seq_length)) +
    # geom_smooth(method="loess", se=F) +
    xlim(c(0.0, 1.0)) +
    ylim(c(0, 10500)) +
    labs(title=curr_title,
       y="Raw score",
       x="Query coverage",
       caption = "Source: Exonerate")
    plot(gg)
  }
}
```

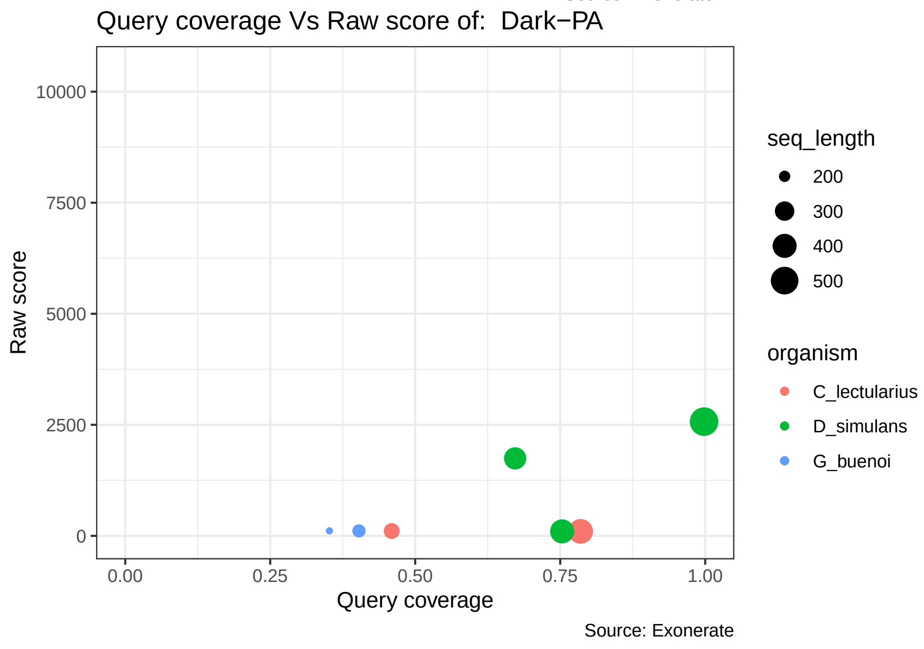

Query coverage Vs Raw score of: exd−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
100
200
300
Source: Exonerate

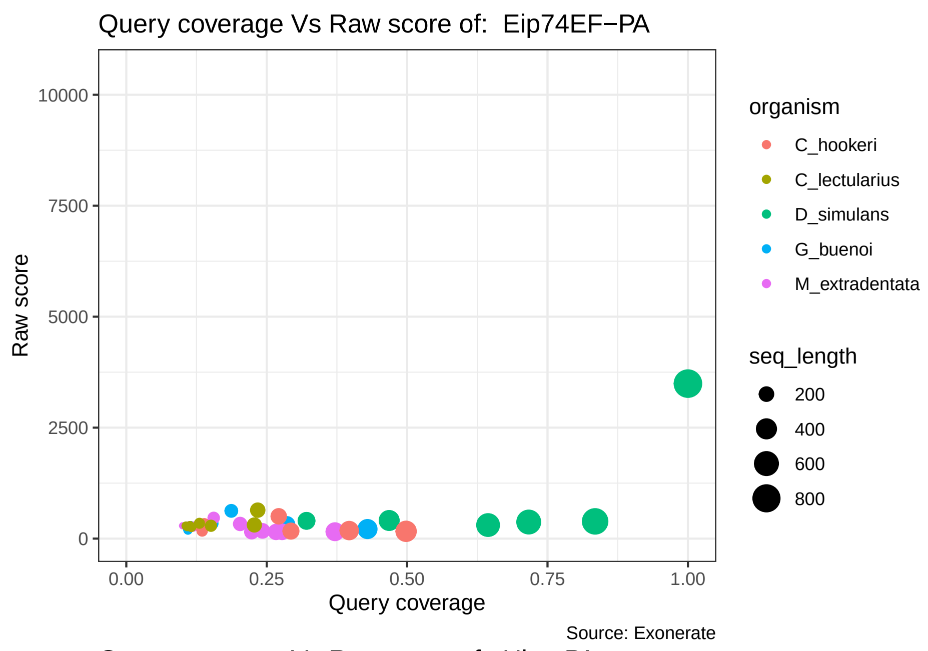

Query coverage Vs Raw score of: EcR−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
200
400
600
800
Source: Exonerate

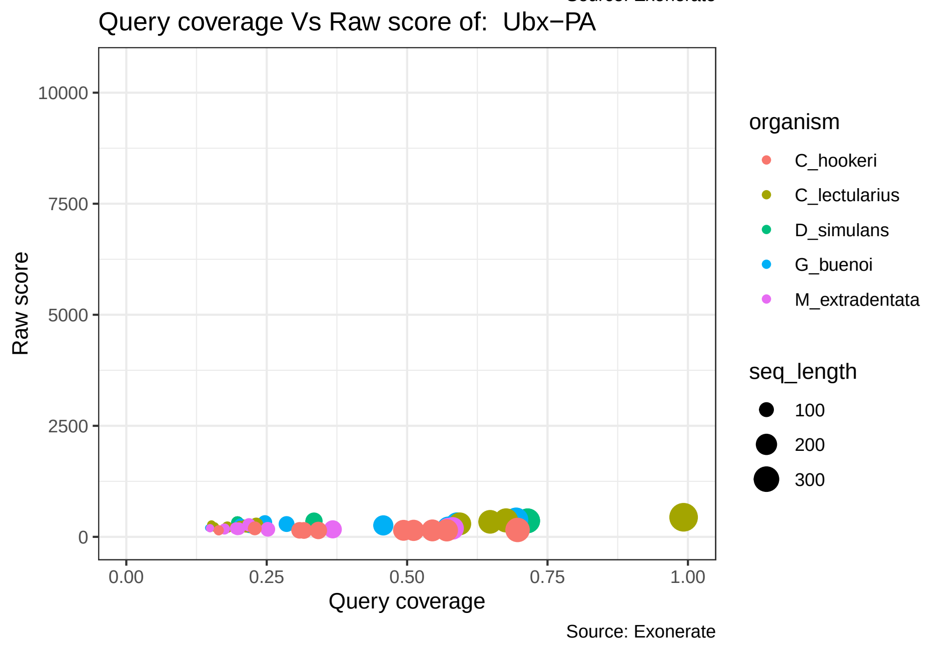

Query coverage Vs Raw score of: sala−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
60
80
100
120
140
Source: Exonerate

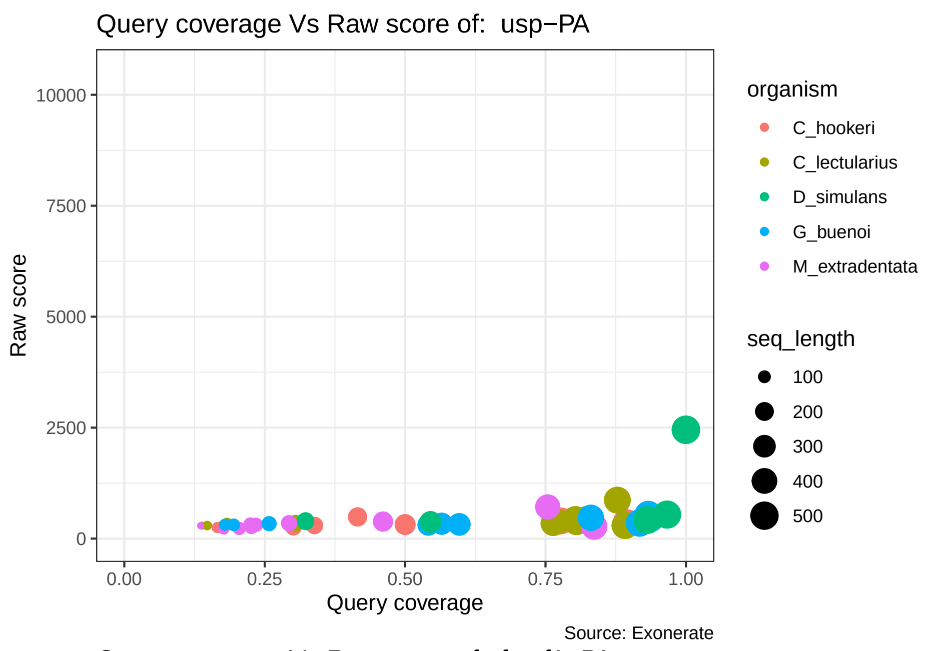

Query coverage Vs Raw score of: Iswi−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
250
500
750
1000
Source: Exonerate

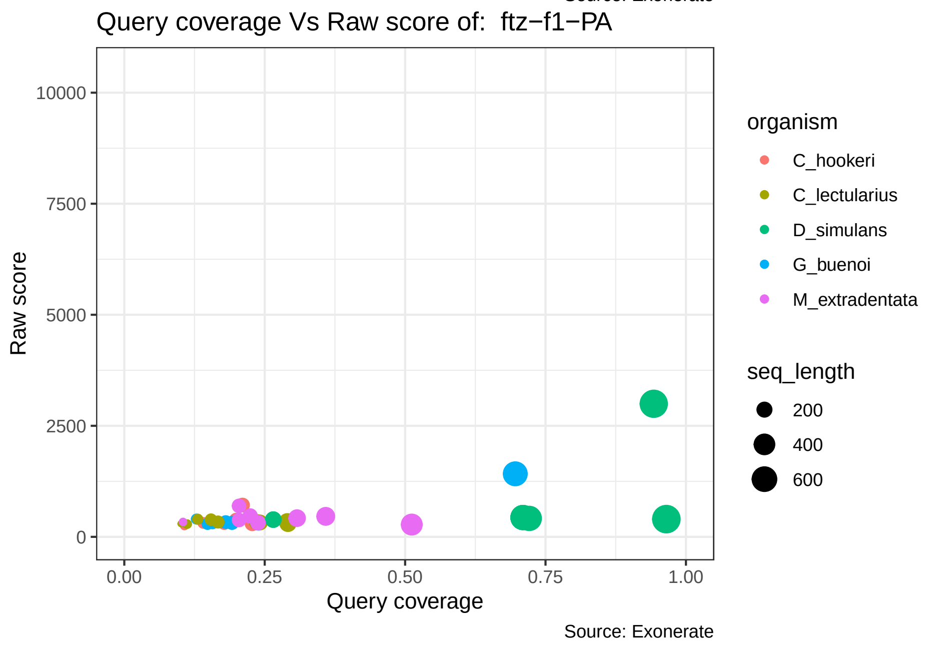

Query coverage Vs Raw score of: salr−PA
Raw score
Query coverage
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
250
500
750
1000
1250
Source: Exonerate

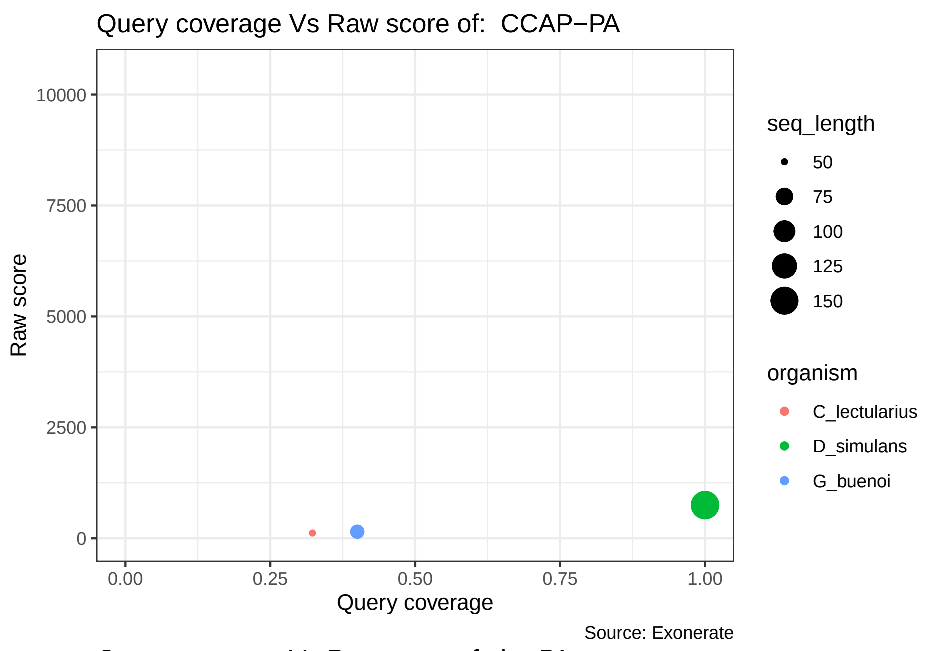

Query coverage Vs Raw score of: Dark−PA
Raw score
Query coverage
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
seq_length
200
300
400
500
organism
C_lectularius
D_simulans
G_buenoi
Source: Exonerate


Query coverage Vs Raw score of: Eip74EF−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
200
400
600
800
Source: Exonerate

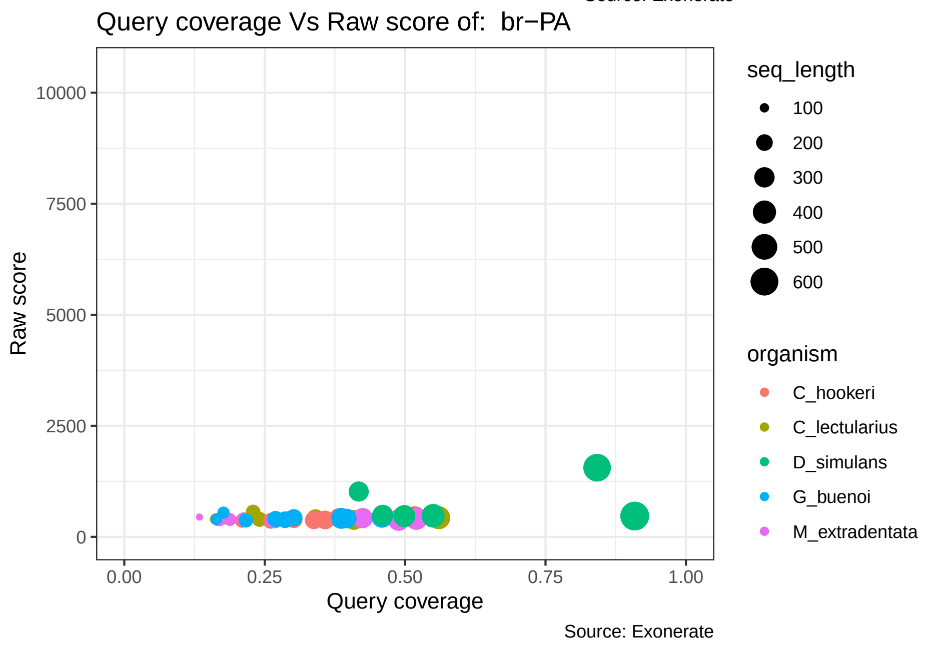

Query coverage Vs Raw score of: Ubx−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
100
200
300
Source: Exonerate


Query coverage Vs Raw score of: usp−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
100
200
300
400
500
Source: Exonerate

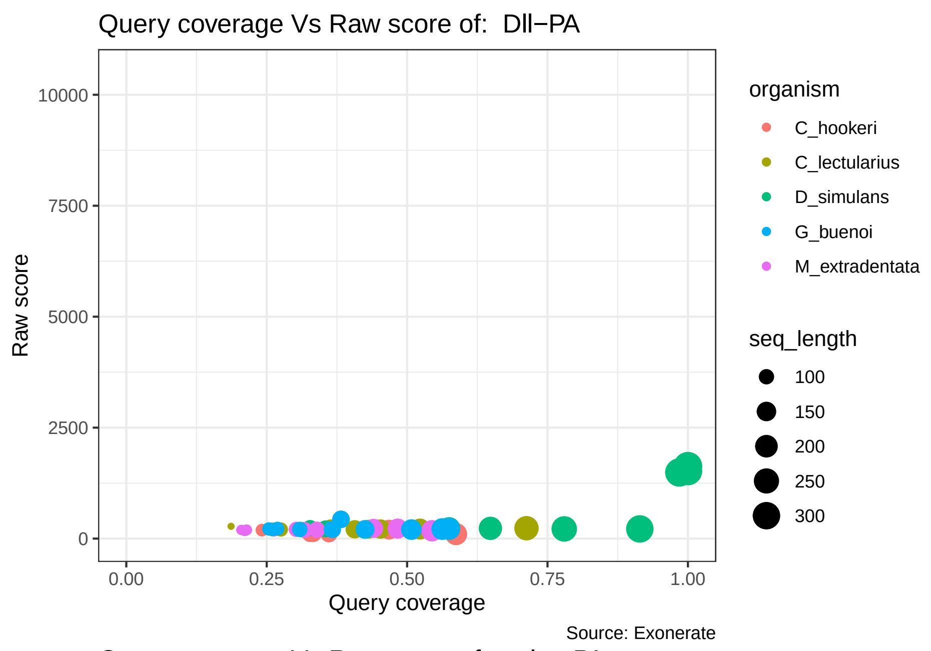

Query coverage Vs Raw score of: ftz−f1−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
200
400
600
Source: Exonerate

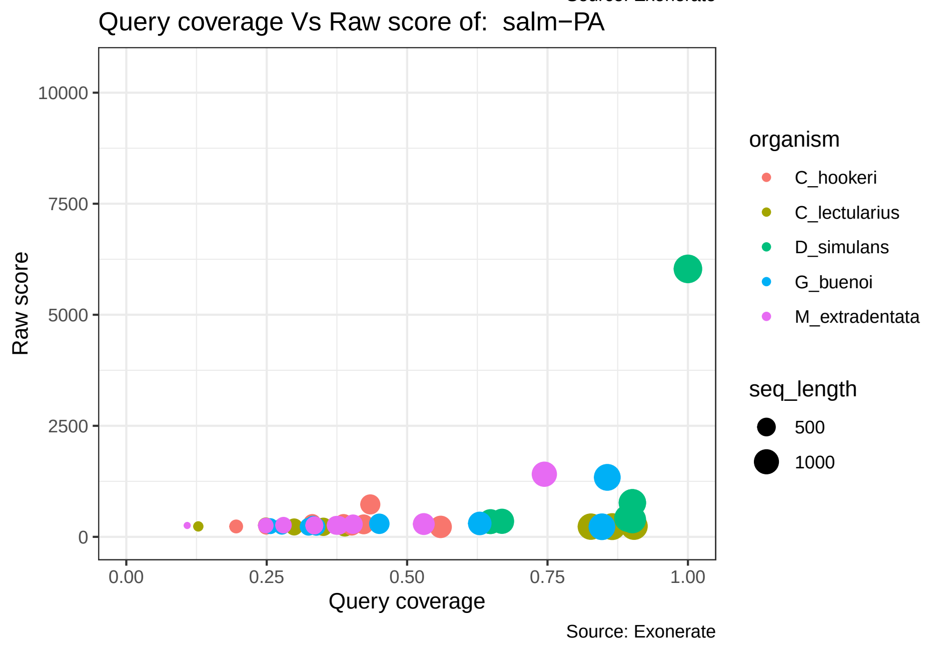


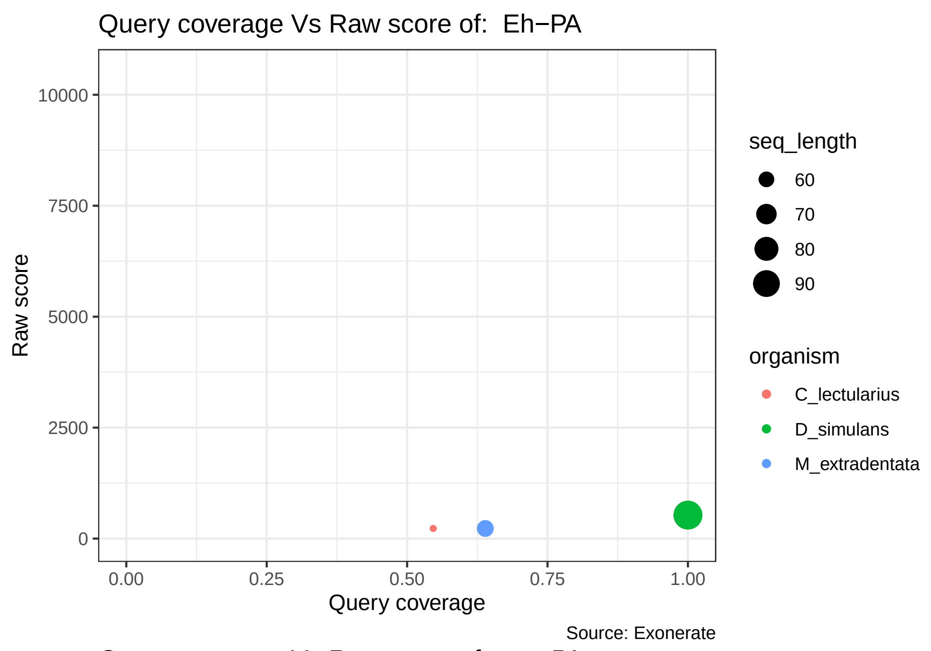
Query coverage Vs Raw score of: CCAP−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
seq_length
50
75
100
125
150
organism
C_lectularius
D_simulans
G_buenoi
Source: Exonerate

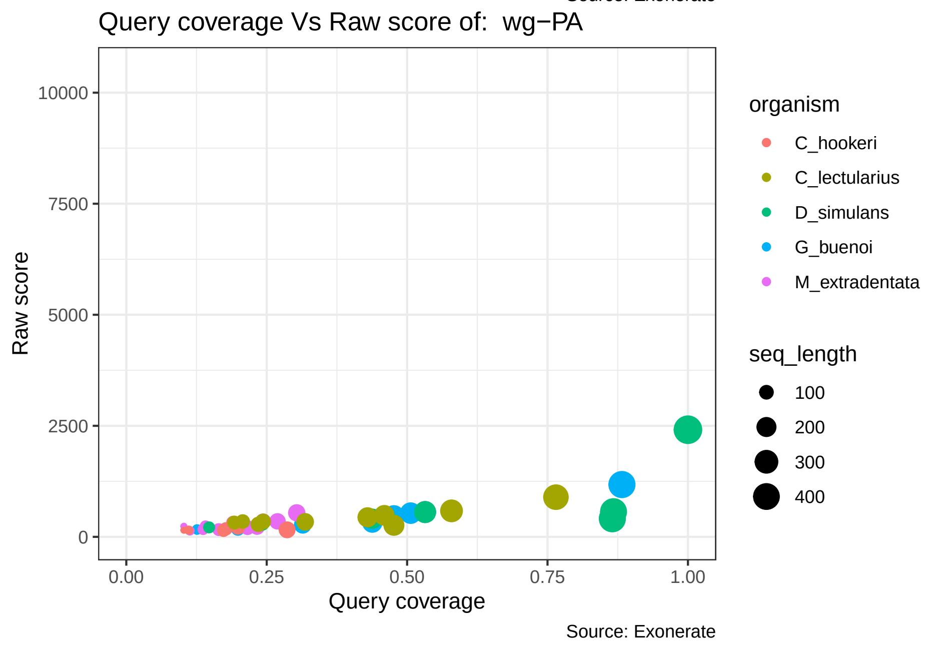


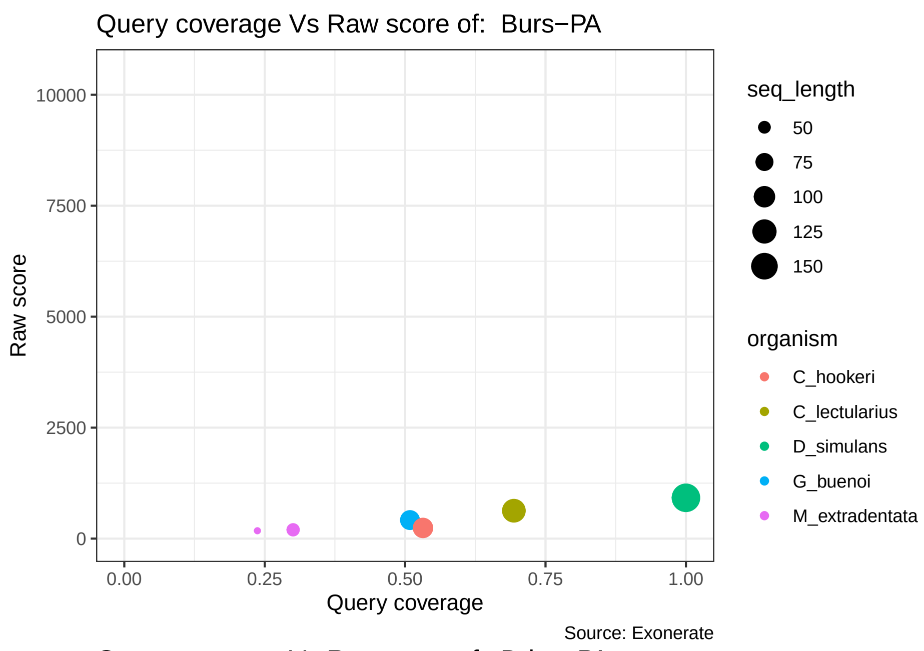
Query coverage Vs Raw score of: br−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
seq_length
100
200
300
400
500
600
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
Source: Exonerate

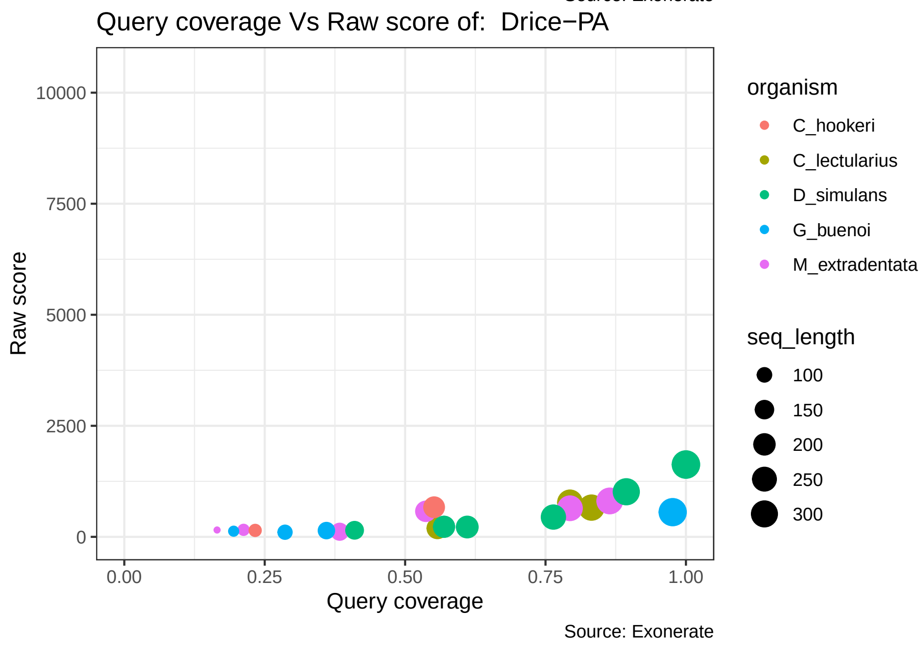

Query coverage Vs Raw score of: Dll−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
100
150
200
250
300
Source: Exonerate

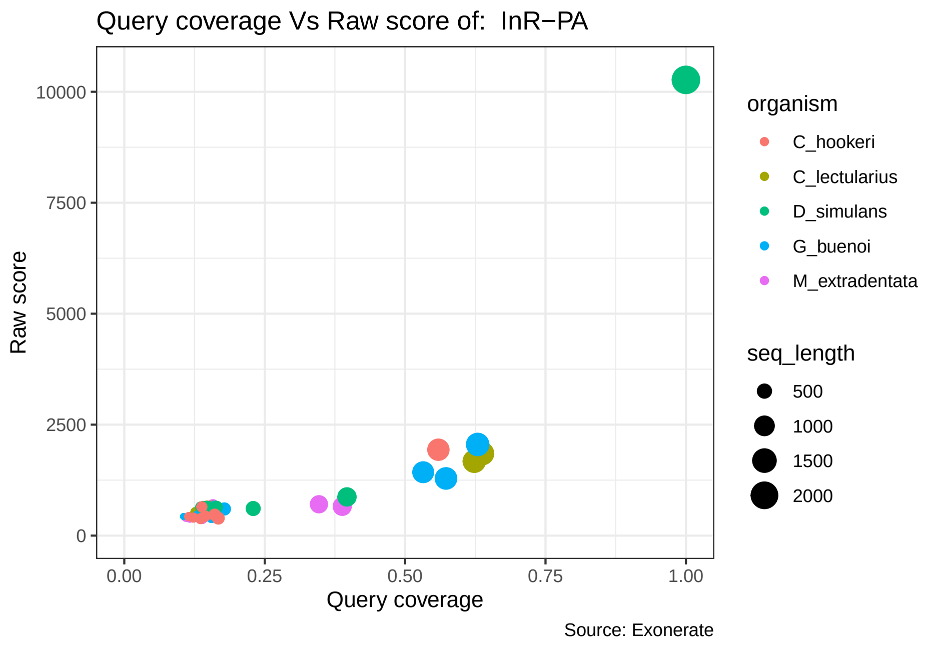

Query coverage Vs Raw score of: salm−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
500
1000
Source: Exonerate


Query coverage Vs Raw score of: Eh−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
seq_length
60
70
80
90
organism
C_lectularius
D_simulans
M_extradentata
Source: Exonerate

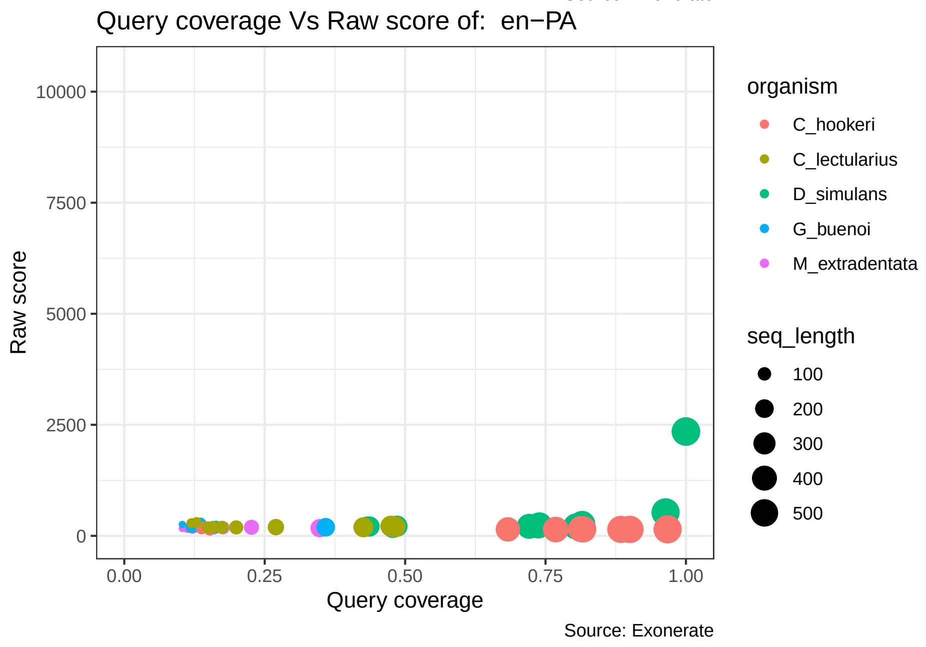

Query coverage Vs Raw score of: wg−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
100
200
300
400
Source: Exonerate


Query coverage Vs Raw score of: Burs−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
seq_length
50
75
100
125
150
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
Source: Exonerate

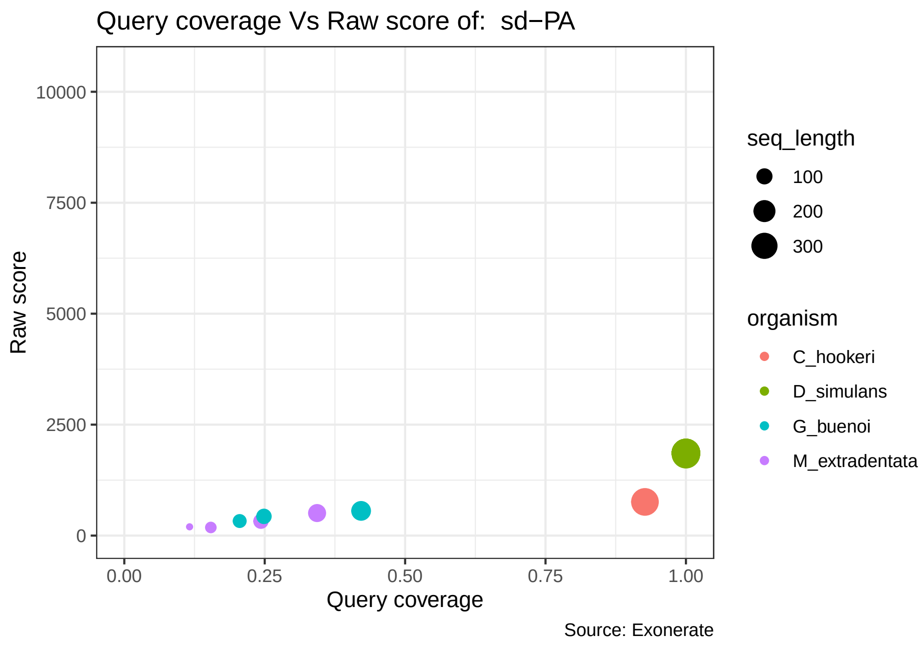

Query coverage Vs Raw score of: Drice−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
100
150
200
250
300
Source: Exonerate



Query coverage Vs Raw score of: InR−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
500
1000
1500
2000
Source: Exonerate



Query coverage Vs Raw score of: en−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
organism
C_hookeri
C_lectularius
D_simulans
G_buenoi
M_extradentata
seq_length
100
200
300
400
500
Source: Exonerate


Query coverage Vs Raw score of: sd−PA
Raw score
10000
7500
5000
2500
0
0.00
0.25
0.50
0.75
1.00
Query coverage
seq_length
100
200
300
organism
C_hookeri
D_simulans
G_buenoi
M_extradentata
Source: Exonerate

Chapter 4

Pick candidate protein sequences of certain genes from each species

By surveying the figures above and the summary csv file in `data/annotations/num_isoforms.csv` we can see that these following genes have putative homological candidates in all nine species:

- broad
- distal-less
- ultrabithorax
- ecd receptor
- engrailed
- InR-like
- Eip74EF
- extradenticle

Next we'll need to start picking out certain amount of candidate protein sequences from each species into a protein multifasta file which then can be aligned with MAFFT v 7.407 (Katoh and Standley 2013).

The hits from exonerate searches to be included in the multifasta file will be chosen by trying to maximise the sequence length and query coverage while at the same time trying to choose matches with the highest raw scores. Quite a lot of relatively subjective decision-making was involved in this.

4.1 Chunk for explorative analyses

```
# THIS CHUNK IS USED FOR EXPLORATIVE ANALYSIS

# The resulting multifasta file is this:
#results="analyses/2019-09-04/"
# Name matched genes in gff files published in NCBI
#prev_name_matches="analyses/2019-08-23"
# The newest versions of annotations (gff+pp:s) for: C hookeri, C lectularius, G buenoi and
#species_annotations="data/2019-08-28"
# D melanogaster 1st translated isoforms of 21 genes of interest
D_mel_proteins="data/D_mel_query_proteins/fasta"
# Results of exonerate searches from 5 species (from annotation pp:s in species_annotations,
exonerate_results="analyses/exonerate_against_5_pps"
# The unzipped versions of annotations pp:s in species_annotations
#species_polypeptides="data/polypeptides/unziped"
# From gff in species_annotations name matched protein sequences of: C lectularius and G bue
#species_name_matched_proteins="analyses/2019-08-29"
# Exonerate searches on M ext and C hook genome assemblies
exonerate_on_wgs="analyses/exonerate_against_2_wgs"

# Some variables and data structures used for exploration
declare -a genes_with_most_matches=("br-PA.fas" "Dll-PA.fas" "Ubx-PA.fas" "EcR-PA.fas" "en-P

declare -a newest_exonerate_search_query_genes=("Ubx-PA.fas" "en-PA.fas" "Eip74EF-PA.fas")

# Alternatives available for the newest exonerate search results
# Eip74EF-PA--to--M_extradentata.res
# Ubx-PA--to--M_extradentata.res
# en-PA--to--C_hookeri.res
# Ubx-PA--to--C_hookeri.res

# Gene to be explored
gene="en"
# How large should query coverage at least be?
lim="0.00"
# How many hits do we want to preview?
number_of_hits="10"
# Organism which exonerate searched in
organism="C_hookeri"

protein_length=$(fastalength $D_mel_proteins/$gene"-PA.fas" | awk '{print $1}')

echo "The length of "$gene" query protein is: " $protein_length
echo "Exonerate search results on genome assembly:"

grep "vulgar:" $exonerate_on_wgs/$gene"-PA--to--"$organism".res" | awk -v len=$protein_lengt

echo
echo "Exonerate search results on annotated polypeptide multifasta:"

#Exonerate against polypeptides
grep "vulgar:" $exonerate_results/$gene"-PA--to--"$organism".res" | awk -v len=$protein_leng
```

```
## The length of en query protein is:  552
## Exonerate search results on genome assembly:
## grep: analyses/exonerate_against_2_wgs/en-PA--to--C_hookeri.res: No such file or directo
##
## Exonerate search results on annotated polypeptide multifasta:
## en-PA_FBpp0087198_FBgn0000577_engrailed maker-scaffold3875-size234203-augustus-gene-1.3-
## en-PA_FBpp0087198_FBgn0000577_engrailed maker-scaffold370-size932106-augustus-gene-7.3-m
## en-PA_FBpp0087198_FBgn0000577_engrailed maker-scaffold370-size932106-augustus-gene-7.3-m
## en-PA_FBpp0087198_FBgn0000577_engrailed maker-scaffold370-size932106-augustus-gene-7.3-m
## en-PA_FBpp0087198_FBgn0000577_engrailed maker-scaffold680-size715670-augustus-gene-4.7-m
## en-PA_FBpp0087198_FBgn0000577_engrailed maker-scaffold167-size1229586-augustus-gene-11.1
## en-PA_FBpp0087198_FBgn0000577_engrailed maker-scaffold370-size932106-augustus-gene-7.3-m
## en-PA_FBpp0087198_FBgn0000577_engrailed maker-scaffold370-size932106-augustus-gene-7.3-m
## en-PA_FBpp0087198_FBgn0000577_engrailed maker-scaffold740-size695741-augustus-gene-1.8-m
## en-PA_FBpp0087198_FBgn0000577_engrailed maker-scaffold5334-size178103-augustus-gene-0.7-
```

4.2 Retrieve protein sequences from NCBI

Maybe the safest way to get the correct translations of the genes is by retrieving the polypeptides from annotated polypeptide files using the accession IDs found in `gff`-files and if there weren't such polypeptide sequences available, then the sequences can be retrieved from NCBI's protein database or FlyBase.

```
import os
from Bio import SeqIO
from Bio import Entrez

Entrez.email = "your.name@student.uu.se"

Dll = {'Dll': ["XP_022188091.1", # N_lug
"XP_029674465.1", # F_exs
"XP_003244593.1", # A_pis
"XP_014245971.1", # C_lec
]}
```

```
Ubx = {'Ubx': ["XP_022206854.1","XP_022183795.1", # N_lug (2nd is partial)
"XP_029663374.1", # F_exs
"XP_008182895.1", # A_pis
"XP_014240207.1" # C_lec
]}

EcR = {'EcR': ["NP_724456.1", # D_mel
"XP_022195205.1","XP_022195205.1", # N_lug
"XP_029677557.1", # F_exs
"NP_001152831.1", # A_pis
"XP_014241436.1" # C_lec
]}

en = {'en': ["XP_022188976.1","XP_022199337.1","XP_022207918.1", # N_lug
"XP_029661733.1","XP_029674962.1", # F_exs
"XP_001949185.2","XP_008178223.1","XP_008189874.1", # A_pis
"XP_014245624.1","XP_014246084.1" # C_lec
]}

InR = {'InR': ["XP_022188928.1", # N_lug
"XP_029679542.1", # F_exs
"XP_008185917.1","XP_003242429.1", # A_pis
"XP_014242610.1","XP_014250978.1","XP_014256336.1" # C_lec
]}

Exd = {'exd': ["XP_022201680.1","XP_022186095.1", # N_lug
"XP_029677537.1", # F_exs
"XP_008182670.1", # A_pis
"XP_024086232.1" # C_lec
]}

Eip74EF = {'Eip74EF': ["NP_730287.1", # D_mel
"XP_022185398.1","XP_022207261.1", # N_lug
"XP_029678074.1", # F_exs
"XP_029347438.1", # A_pis
"XP_014251487.1" # C_lec
]}

broad = {'br': ["NP_726750.1", # D_mel
"XP_022185576.1","XP_022200493.1","XP_022205415.1", # N_lug
"XP_029671050.1", # F_exs
"XP_001942520.2", # A_pis
"XP_016037686.1" # D_sim
]}
# Create a list of dictionaries with a key (gene name) and
# a list (accession ID:s) corresponding to it
all_genes = [Dll, Ubx, EcR, en, InR, Exd, Eip74EF, broad]

file_root = "analyses/multifastas_for_MPAs/NCBI/"

for gene_accessions in all_genes:
  for gene in gene_accessions:
    #print(gene, gene_accessions[gene])
    filename = file_root + gene + ".fasta"
    #print(filename)
    # Write each gene to own file
    out_handle = open(filename, "w+")
    # Iterate through all accession id:s
    for accession in gene_accessions[gene]:
      net_handle = Entrez.efetch(db="protein", id=accession, rettype="fasta", retmode="text'
      # Save the length of the written sequence so no output to STDOUT will happen
      num = out_handle.write(net_handle.read())
      # Close net handle
      net_handle.close()
    # Close file handle
    out_handle.close()
```

4.2.1 Modify NCBI proteins into 2-line fasta

Let's modify these resulting files so that the empty lines are removed and protein sequence takes up only the next line in the file:

```
NCBI_multifastas="analyses/multifastas_for_MPAs/NCBI/"

read -r -a multifastas <<< $( find $NCBI_multifastas -name "*.fasta" -and -type f -print0 |
# Remove empty lines and move the sequences to 1 row
for m_fasta in "${multifastas[@]}"; do
  sed '/^$/d' "$m_fasta" > "temp.fas"
  cat "temp.fas" | awk '/^>/ {printf("\n%s\n",$0);next; } { printf("%s",$0);}' | sed '/^$/d
  printf "\n" >> $m_fasta
done
# Remove temporary intermediary file
rm "temp.fas"
```

Now that there is the multifasta file with sequences from *N lugens*, *F exsecta*, *A pisum*, *D melanogaster* and *D simulans* we can append to it exonerate matches and gff name matches from *G buenoi*, *C lectularius* and *D simulans* as well as exonerate matches from *M extradentata* and *C hookeri*.

4.3 Retrieve sequences from local peptide files

The unique identifiers fro all species were chosen manually by surveying the exonerate search results against:

- *C lectularius*
- *D simulans*
- *C hookeri*
- *G buenoi*
- *M extradentata*

The hits included in the produced multifasta filed were chosen by trying to maximise the sequence length and query coverage while at the same time trying to choose matches with the highest raw scores.

```
# The resulting multifasta files go hear
results="analyses/multifastas_for_MPAs/unreadabilised"
# Results of exonerate searching from 5 species (from annotation pp:s in species_annotations
exonerate_results="analyses/exonerate_against_5_pps"
# The unzipped versions of annotations pp:s in species_annotations
species_polypeptides="data/polypeptides/unziped"

#EcR
declare -A ecdyconeR=([GBUE004915-PA,GBUE013140-PA,GBUE021020-PA,GBUE021385-PA]="G_buenoi"
[CLEC002129-PA,CLEC025114-PA,CLEC001111-PA]="C_lectularius"
[Medex_00015863-RA]="M_extradentata"
[maker-scaffold389-size1115929-augustus-gene-10.2-mRNA-1,maker-scaffold2708-size326647-augus

#Dll
declare -A distal_less=([GBUE021126-PA,GBUE008733-PA,GBUE021125-PA,GBUE007923-PA,GBUE004076-
[CLEC001685-PA,CLEC002970-PA,CLEC009091-PA,CLEC011186-PA,CLEC004176-PA]="C_lectularius"
[Medex_00100153-RA,Medex_00089546-RA,Medex_00043961-RA]="M_extradentata"
[maker-scaffold740-size695741-augustus-gene-1.8-mRNA-1]="C_hookeri")

#UBx
declare -A ultrabithorax=([GBUE021017-PA,GBUE020981-PA,GBUE000143-PA,GBUE007864-PA,GBUE0040З
[CLEC025236-PA,CLEC025113-PA,CLEC025066-PA,CLEC025152-PA]="C_lectularius"
[Medex_00089660-RA,Medex_00021391-RA]="M_extradentata"
[maker-scaffold5131-size174542-augustus-gene-0.3-mRNA-1]="C_hookeri")

#en
declare -A engrailed=([GBUE020981-PA,GBUE021017-PA]="G_buenoi"
[CLEC025152-PA,CLEC025113-PA,CLEC025236-PA]="C_lectularius"
[Medex_00100153-RA,Medex_00096568-RA]="M_extradentata"
[maker-scaffold370-size932106-augustus-gene-7.3-mRNA-1,maker-scaffold167-size1229586-augustu
```

```
#Eip74EF
declare -A Eip74EF=([GBUE008361-PA,GBUE009916-PA,GBUE012840-PA]="G_buenoi"
[CLEC009111-PA,CLEC003449-PA]="C_lectularius"
[Medex_00103380-RA,Medex_00022200-RA,Medex_00059238-RA]="M_extradentata"
[maker-scaffold680-size715670-augustus-gene-4.7-mRNA-1,maker-scaffold212-size1131905-augustu

#exd
declare -A exd=([GBUE010039-PA]="G_buenoi"
[CLEC009091-PA,CLEC009744-PA]="C_lectularius"
[Medex_00068030-RA,Medex_00071840-RA,Medex_00096569-RA]="M_extradentata"
[maker-scaffold845-size653622-augustus-gene-5.4-mRNA-1,maker-scaffold740-size695741-augustus

#InR
declare -A Inr=([GBUE019859-PA,GBUE021108-PA,GBUE020965-PA,GBUE020810-PA,GBUE009720-PA]="G_b
[CLEC010029-PA,CLEC025326-PA]="C_lectularius"
[Medex_00067447-RA,Medex_00078894-RA]="M_extradentata"
[augustus-scaffold862-size931216-processed-gene-3.1-mRNA-1]="C_hookeri")

#Br
declare -A broad=([GBUE021143-PA,GBUE009398-PA,GBUE005702-PA,GBUE001543-PA,GBUE006286-PA,GBU
[CLEC000821-PA,CLEC025389-PA,CLEC025251-PA,CLEC009497-PA]="C_lectularius"
[Medex_00080928-RA,Medex_00060547-RA,Medex_00067579-RA,Medex_00066470-RA,Medex_00045157-RA]=
[maker-scaffold4790-size194163-augustus-gene-0.5-mRNA-1,maker-scaffold325-size972641-augustu

# Check if there are already fasta files in the results dir
# and if yes, remove the old versions of files
# (before creating or appending to any new ones)
num_fastas=$(ls -1 $results/*.fasta 2>/dev/null | wc -l)
if [ $num_fastas != 0 ]; then
  rm $results/*
fi

# Loop through gene data associative array and save multifasta files
# by retrieving the protein sequences from 2-line organism multifastas
# based on the id:s defined in the gene data input
write_multifastas() {
  local -n gene_data=$1
  for ids in "${!gene_data[@]}"; do
    organism=${gene_data[$ids]}
    gene="$1"
    # Parse the key and loop through each id at a time
    for id in $(echo $ids | sed "s/,/ /g"); do
      grep -A 1 "$id" $species_polypeptides/"$organism""_pep.fa" >> $results/"$gene"".fasta'
    done
  done
}

num=0
genes=("ecdyconeR" "distal_less" "ultrabithorax" "engrailed" "Eip74EF" "exd" "Inr" "broad")
for gene in "${genes[@]}"; do
  ((num++))
  write_multifastas $gene
done
```

4.4 Retrieve *D melanogaster* and *D simulans* sequences from FlyBase

Some of the genes which will be included in the multifasta files come from FlyBase. It seems that how one can download sequences from there is in json format.

```
download_dir="analyses/first_multifastas_to_be_used_for_MPAs/drosophila_jsons_downloaded_fro
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0210303/FBpp" -H  "accept: appl
#D melanogaster
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0072286/FBpp" -H  "accept: appl

#UBx
#D simulans
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0314200/FBpp" -H  "accept: appl
#D melanogaster
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0082793/FBpp" -H  "accept: appl

#EcD
#D simulans
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0208736/FBpp" -H  "accept: appl
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0315934/FBpp" -H  "accept: appl

#En
#D simulans
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0224300/FBpp" -H  "accept: appl
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0327509/FBpp" -H  "accept: appl
#D melanogaster
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0087198/FBpp" -H  "accept: appl

#Br
#D simulans
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0325479/FBpp" -H  "accept: appl
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0219998/FBpp" -H  "accept: appl
```

```
#Eip74EF
#D simulans
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0323369/FBpp" -H  "accept: appl

#exd
#D simulans
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0215663/FBpp" -H  "accept: appl
#D melanogaster
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0073940/FBpp" -H  "accept: appl

#InR
#D simulans
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0218373/FBpp" -H  "accept: appl
#D melanogaster
curl -X GET "http://api.flybase.org/api/v1.0/sequence/id/FBpp0083519/FBpp" -H  "accept: appl
```

4.4.1 Parsing downloaded polypeptide json files into csv:s

```
library(jsonlite)
library(tidyverse)

path <- "analyses/first_multifastas_to_be_used_for_MPAs/drosophila_jsons_downloaded_from_Fly

# Initialise an empty data.frame for all the sequences
seq_df <- data.frame()
# Obtain files with the ending: .json
jsons <- dir(path, pattern ="*.json",recursive = TRUE)

for(this_json in jsons){
  #print(this_json)
  gene_name <- strsplit(this_json, "/")[[1]][1]
  file_name <- strsplit(this_json, "/")[[1]][2]
  # Parse and store the .json data from the files
  data <- this_json %>%
    map_df(~fromJSON(file.path(path, .), flatten = TRUE))
  # Loop through the parsed and flattened json files data

  id <- data[8,][[1]][[1]][["id"]]
  description <- data[8,][[1]][[1]][["description"]]
  species <- description %>%
    str_extract("species=.+;") %>% # Find species
    sub("species=","",.) %>% # Get rid of "species="
    sub(";","",.) # Get rid of ";"
  sequence <- data[8,][[1]][[1]][["sequence"]]
  gene_id <- description %>%
    # Find parent gene id
    str_extract(regex("parent=FBgn\\d+", ignore_case = TRUE)) %>%
    sub("parent=","",.) # Get rid or "parent="
  if(str_detect(species,"Dsim")){
    species <- "[Drosophila simulans]"
  }else if(str_detect(species,"Dmel")){
    species <- "[Drosophila melanogaster]"
  }else{
    species <- "NA"
  }

  new_seq_row <- data.frame("Gene_symbol" = gene_name,
                            "FlyBase_gene_ID"=gene_id,
                            "FlyBase_polypeptide_ID"=id,
                            "Species"=species,
                            "Sequence"=sequence,
                            stringsAsFactors = FALSE)
  seq_df <- rbind(seq_df,new_seq_row)
}
seq_tbl <- seq_df %>% as_tibble()

# Write some csv:s
seq_tbl %>% filter(Gene_symbol == "broad") %>%
  write_csv("analyses/Flybase_jsons_parsed_to_csvs/br.csv")
seq_tbl %>% filter(Gene_symbol == "Dll") %>%
  write_csv("analyses/Flybase_jsons_parsed_to_csvs/Dll.csv")
seq_tbl %>% filter(Gene_symbol == "EcR") %>%
  write_csv("analyses/Flybase_jsons_parsed_to_csvs/EcR.csv")
seq_tbl %>% filter(Gene_symbol == "Eip74EF") %>%
  write_csv("analyses/Flybase_jsons_parsed_to_csvs/Eip74EF.csv")
seq_tbl %>% filter(Gene_symbol == "en") %>%
  write_csv("analyses/Flybase_jsons_parsed_to_csvs/en.csv")
seq_tbl %>% filter(Gene_symbol == "Exd") %>%
  write_csv("analyses/Flybase_jsons_parsed_to_csvs/exd.csv")
seq_tbl %>% filter(Gene_symbol == "InR") %>%
  write_csv("analyses/Flybase_jsons_parsed_to_csvs/InR.csv")
seq_tbl %>% filter(Gene_symbol == "Ubx") %>%
  write_csv("analyses/Flybase_jsons_parsed_to_csvs/Ubx.csv")
```

4.4.2 Convert gene csv:s to 2-line fastas

And now that there are some csv:s let's convert them to 2-line fastas.

```
csvs="analyses/Flybase_jsons_parsed_to_csvs/"
multifasta_result="analyses/multifastas_for_MPAs/FlyBase"
read -r -a csvs <<< $( find $csvs -name "*.csv" -and -type f -print0 | xargs -0 echo )

for csv in "${csvs[@]}"; do
  gene=$(echo $(basename "$csv") | awk -F "." '{print $1}')
  #echo "$gene"
  cat $csv | awk -F , 'NR>1 {print ">"$3" "$2" "$1" "$4"\n"$5}' | sed 's/"//g' > $multifasta
done
```

4.5 Concatenate all multifastas from each source

```
multifasta_FlyBase="analyses/multifastas_for_MPAs/FlyBase"
multifasta_NCBI="analyses/multifastas_for_MPAs/NCBI/"
multifasta_local="analyses/multifastas_for_MPAs/unreadabilised/" # These were found by exone
multifasta_concatenated="analyses/multifastas_for_MPAs/concatenated/" # Results go here

read -r -a local_multifastas <<< $( find $multifasta_local -name "*.fasta" -and -type f -pri

for multifasta in "${local_multifastas[@]}"; do
  file=$(echo $(basename "$multifasta")) # e.g. broad.fas
  gene=$(echo $(basename "$multifasta") | awk -F "." '{print $1}')
  # This below is not so beautiful but it works for now...
  if [ "$gene" == "ecdyconeR" ]; then
    cat $multifasta $( find $multifasta_NCBI $multifasta_FlyBase -name "EcR.fasta" -and -typ
  fi
  if [ "$gene" == "Eip74EF" ]; then
    cat $multifasta $( find $multifasta_NCBI $multifasta_FlyBase -name "Eip74EF.fasta" -and
  fi

  if [ "$gene" == "Inr" ]; then
    cat $multifasta $( find $multifasta_NCBI $multifasta_FlyBase -name "InR.fasta" -and -typ
  fi

  if [ "$gene" == "distal_less" ]; then
    cat $multifasta $( find $multifasta_NCBI $multifasta_FlyBase -name "Dll.fasta" -and -typ
  fi

  if [ "$gene" == "exd" ]; then
    cat $multifasta $( find $multifasta_NCBI $multifasta_FlyBase -name "exd.fasta" -and -typ
  fi

  if [ "$gene" == "engrailed" ]; then
    cat $multifasta $( find $multifasta_NCBI $multifasta_FlyBase -name "en.fasta" -and -type
  fi

  if [ "$gene" == "broad" ]; then
    cat $multifasta $( find $multifasta_NCBI $multifasta_FlyBase -name "br.fasta" -and -type
  fi

  if [ "$gene" == "ultrabithorax" ]; then
    cat $multifasta $( find $multifasta_NCBI $multifasta_FlyBase -name "Ubx.fasta" -and -typ
  fi
done
```

4.6 Make fasta headers readable for further analyses

```
readabilising_script="code/readabalise_header.py"
concatenated="analyses/multifastas_for_MPAs/concatenated/"
output="analyses/multifastas_for_MPAs/readabilised/"
summaryfile="analyses/multifastas_for_MPAs/summaries/"

read -r -a genes <<< $( find $concatenated -name "*.fasta" -and -type f -print0 | xargs -0 e

for gene in "${genes[@]}"; do
  file=$(echo $(basename "$gene"))
  gene_name=$(echo $(basename "$gene") | awk -F "." '{print $1}')
  python3 $readabilising_script -i "$gene" -o "$output""$file" > "$summaryfile""$gene_name""
done
```

4.7 Reorder fasta records

It would be good to have the species in order:

1. mono-morphic apterous
2. mono-morphic macropterous
3. polyphenic

This can be done with somewhat ease again with biopython:

```
reordering_script="code/reorder_records.py"
readabilised="analyses/multifastas_for_MPAs/readabilised/"
output="analyses/multifastas_for_MPAs/reordered/"

read -r -a genes <<< $( find $readabilised -name "*.fasta" -and -type f -print0 | xargs -0 e
```

```bash
for gene in "${genes[@]}"; do
  file=$(echo $(basename "$gene"))
  gene_name=$(echo $(basename "$gene") | awk -F "." '{print $1}')
  python3 $reordering_script -i "$gene" -o "$output""$file"
done
```

Chapter 5

Create multiple protein alignments and trees

```
#module load bioinfo-tools
#module load MAFFT/7.407
#module load FastTree/2.1.10

multifastas="analyses/multifastas_for_MPAs/reordered"
results_root="analyses/MPAs_without_wg_exonerate_additions"

#Dll
#1. G_bue5 and G_bue2
mafft_input="$results_root""/Dll/Dll--excluded--G_bue5-G_bue2.fas"
mafft_output="$results_root""/Dll/Dll--excluded--G_bue5-G_bue2.aln.fas"
fast_tree_output="$results_root""/Dll/Dll--excluded--G_bue5-G_bue2.tre"
sed -e '/G_bue5/,+1d' -e '/G_bue2/,+1d' $multifastas/"Dll.fasta" > $mafft_input
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output

#2. G_bue5
mafft_input="$results_root""/Dll/Dll--excluded--G_bue5.fas"
mafft_output="$results_root""/Dll/Dll--excluded--G_bue5.aln.fas"
fast_tree_output="$results_root""/Dll/Dll--excluded--G_bue5.tre"
sed -e '/G_bue5/,+1d' $multifastas/"Dll.fasta"  > $mafft_input
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output

#3. G_bue2
mafft_input="$results_root""/Dll/Dll--excluded--G_bue2.fas"
```

```
mafft_output="$results_root""/Dll/Dll--excluded--G_bue2.aln.fas"
fast_tree_output="$results_root""/Dll/Dll--excluded--G_bue2.tre"
sed -e '/G_bue2/,+1d' $multifastas/"Dll.fasta"  > $mafft_input
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output


#Eip74EF
#1. M_ext3
mafft_input="$results_root""/Eip74EF/Eip74EF--excluded--M_ext3.fas"
mafft_output="$results_root""/Eip74EF/Eip74EF--excluded--M_ext3.aln.fas"
fast_tree_output="$results_root""/Eip74EF/Eip74EF--excluded--M_ext3.tre"
sed -e '/M_ext3/,+1d' $multifastas/"Eip74EF.fasta"  > $mafft_input
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output

#2.M_ext3 and C_hook1:
mafft_input="$results_root""/Eip74EF/Eip74EF--excluded--M_ext3-C_hook1.fas"
mafft_output="$results_root""/Eip74EF/Eip74EF--excluded--M_ext3-C_hook1.aln.fas"
fast_tree_output="$results_root""/Eip74EF/Eip74EF--excluded--M_ext3-C_hook1.tre"
sed -e '/M_ext3/,+1d' -e '/C_hook1/,+1d' $multifastas/"Eip74EF.fasta"  > $mafft_input
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output


#Br
#1. G_bue6 and M_ext5
mafft_input="$results_root""/Br/Br--excluded--G_bue6-M_ext5.fas"
mafft_output="$results_root""/Br/Br--excluded--G_bue6-M_ext5.aln.fas"
fast_tree_output="$results_root""/Br/Br--excluded--G_bue6-M_ext5.tre"
sed -e '/G_bue6/,+1d' -e '/M_ext5/,+1d' $multifastas/"br.fasta"  > $mafft_input
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output

#2. G_bue6, M_ext5 and G_bue5
mafft_input="$results_root""/Br/Br--excluded--G_bue6-G_bue5-M_ext5.fas"
mafft_output="$results_root""/Br/Br--excluded--G_bue6-G_bue5-M_ext5.aln.fas"
fast_tree_output="$results_root""/Br/Br--excluded--G_bue6-G_bue5-M_ext5.tre"
sed -e '/G_bue6/,+1d' -e '/M_ext5/,+1d' -e '/G_bue5/,+1d' $multifastas/"br.fasta"  > $mafft_
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output

#3. G_bue6, M_ext5, G_bue5 and G_bue3
mafft_input="$results_root""/Br/Br--excluded--G_bue6-G_bue5-G_bue3-M_ext5.fas"
mafft_output="$results_root""/Br/Br--excluded--G_bue6-G_bue5-G_bue3-M_ext5.aln.fas"
```

```
fast_tree_output="$results_root""/Br/Br--excluded--G_bue6-G_bue5-G_bue3-M_ext5.tre"
sed -e '/G_bue6/,+1d' -e '/M_ext5/,+1d' -e '/G_bue5/,+1d' -e '/G_bue3/,+1d' $multifastas/"br
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output


#EcR
#1.G_bue4
mafft_input="$results_root""/EcR/EcR--excluded--G_bue4.fas"
mafft_output="$results_root""/EcR/EcR--excluded--G_bue4.aln.fas"
fast_tree_output="$results_root""/EcR/EcR--excluded--G_bue4.tre"
sed -e '/G_bue4/,+1d' $multifastas/"EcR.fasta"  > $mafft_input
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output

#2. G_bue4 and G_bue1
mafft_input="$results_root""/EcR/EcR--excluded--G_bue4-G_bue1.fas"
mafft_output="$results_root""/EcR/EcR--excluded--G_bue4-G_bue1.aln.fas"
fast_tree_output="$results_root""/EcR/EcR--excluded--G_bue4-G_bue1.tre"
sed -e '/G_bue4/,+1d' -e '/G_bue1/,+1d' $multifastas/"EcR.fasta"  > $mafft_input
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output


#en
#1. A_pis2
mafft_input="$results_root""/en/en--excluded--A_pis2.fas"
mafft_output="$results_root""/en/en--excluded--A_pis2.aln.fas"
fast_tree_output="$results_root""/en/en--excluded--A_pis2.tre"
sed -e '/A_pis2/,+1d' $multifastas/"en.fasta"  > $mafft_input
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output

#2. A_pis2 and N_lug2
mafft_input="$results_root""/en/en--excluded--A_pis2-N_lug2.fas"
mafft_output="$results_root""/en/en--excluded--A_pis2-N_lug2.aln.fas"
fast_tree_output="$results_root""/en/en--excluded--A_pis2-N_lug2.tre"
sed -e '/A_pis2/,+1d' -e '/N_lug2/,+1d' $multifastas/"en.fasta"  > $mafft_input
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output

#3. N_lug2
mafft_input="$results_root""/en/en--excluded--N_lug2.fas"
mafft_output="$results_root""/en/en--excluded--N_lug2.aln.fas"
fast_tree_output="$results_root""/en/en--excluded--N_lug2.tre"
```

```
sed -e '/N_lug2/,+1d' $multifastas/"en.fasta"  > $mafft_input
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output


#Exd
#1. N_lug2
mafft_input="$results_root""/Exd/Exd--excluded--N_lug2.fas"
mafft_output="$results_root""/Exd/Exd--excluded--N_lug2.aln.fas"
fast_tree_output="$results_root""/Exd/Exd--excluded--N_lug2.tre"
sed -e '/N_lug2/,+1d' $multifastas/"Exd.fasta"  > $mafft_input
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output

#2. N_lug2 and M_ext1
mafft_input="$results_root""/Exd/Exd--excluded--N_lug2-M_ext1.fas"
mafft_output="$results_root""/Exd/Exd--excluded--N_lug2-M_ext1.aln.fas"
fast_tree_output="$results_root""/Exd/Exd--excluded--N_lug2-M_ext1.tre"
sed -e '/N_lug2/,+1d' -e '/M_ext1/,+1d' $multifastas/"Exd.fasta"  > $mafft_input

#3. N_lug2, M_ext1 and M_ext2
mafft_input="$results_root""/Exd/Exd--excluded--N_lug2-M_ext1-M_ext2.fas"
mafft_output="$results_root""/Exd/Exd--excluded--N_lug2-M_ext1-M_ext2.aln.fas"
fast_tree_output="$results_root""/Exd/Exd--excluded--N_lug2-M_ext1-M_ext2.tre"
sed -e '/N_lug2/,+1d' -e '/M_ext1/,+1d' -e '/M_ext2/,+1d' $multifastas/"Exd.fasta"  > $mafft
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output


#InR
#1. G_bue4, G_bue5 and M_ext2
mafft_input="$results_root""/InR/InR--excluded--G_bue4-G_bue5-M_ext2.fas"
mafft_output="$results_root""/InR/InR--excluded--G_bue4-G_bue5-M_ext2.aln.fas"
fast_tree_output="$results_root""/InR/InR--excluded--G_bue4-G_bue5-M_ext2.tre"
sed -e '/G_bue4/,+1d' -e '/G_bue5/,+1d' -e '/M_ext2/,+1d' $multifastas/"InR.fasta"  > $mafft
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output

#2. G_bue4, G_bue5 and M_ext2 and A_pis2
mafft_input="$results_root""/InR/InR--excluded--G_bue4-G_bue5-M_ext2-A_pis2.fas"
mafft_output="$results_root""/InR/InR--excluded--G_bue4-G_bue5-M_ext2-A_pis2.aln.fas"
fast_tree_output="$results_root""/InR/InR--excluded--G_bue4-G_bue5-M_ext2-A_pis2.tre"
sed -e '/G_bue4/,+1d' -e '/G_bue5/,+1d' -e '/M_ext2/,+1d' -e '/A_pis2/,+1d' $multifastas/"In
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output
```

```
#Ubx
#1. G_bue3, G_bue5, N_lug2, C_lec2 and M_ext1
mafft_input="$results_root""/Ubx/Ubx--excluded--G_bue3-G_bue5-N_lug2-C_lec2-M_ext1.fas"
mafft_output="$results_root""/Ubx/Ubx--excluded--G_bue3-G_bue5-N_lug2-C_lec2-M_ext1.aln.fas'
fast_tree_output="$results_root""/Ubx/Ubx--excluded--G_bue3-G_bue5-N_lug2-C_lec2-M_ext1.tre'
sed -e '/G_bue3/,+1d' -e '/G_bue5/,+1d' -e '/N_lug2/,+1d' -e '/C_lec2/,+1d' -e '/M_ext1/,+1c
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output

#2. G_bue3, G_bue5, N_lug2, C_lec2, M_ext1 and G_bue4
mafft_input="$results_root""/Ubx/Ubx--excluded--G_bue3--G_bue4-G_bue5-N_lug2-C_lec2-M_ext1.f
mafft_output="$results_root""/Ubx/Ubx--excluded--G_bue3--G_bue4-G_bue5-N_lug2-C_lec2-M_ext1.
fast_tree_output="$results_root""/Ubx/Ubx--excluded--G_bue3--G_bue4-G_bue5-N_lug2-C_lec2-M_e
sed -e '/G_bue3/,+1d' -e '/G_bue5/,+1d' -e '/N_lug2/,+1d' -e '/C_lec2/,+1d' -e '/M_ext1/,+1c
mafft --auto --thread 4 $mafft_input > $mafft_output
FastTree $mafft_output > $fast_tree_output
```

5.1 Ecdysone receptor gene in *Gerris buenoi*

The MPA of matched *G buenoi* proteins showed that matches of protein polypeptide accessions "GBUE004915-PA" and "GBUE021385-PA" in annotated proteomes would be one and same protein since when one of the proteins ended the other one started.

The two proteins were both recognised as ecdysone receptor genes[1] in the annotation .gff3 file and "GBUE004915-PA" was also matched with exonerate with score 1184, query coverage 0.457008 and matching query length of 388

When looking at their annotations the genomic coordinates were though in totally different locations: `JHBY02131244.1 OGSv1.0 polypeptide 1087 1329 . + . ID=GBUE021385-PA;Parent=GBUE021385-RA;method=ManualCuration` and `KZ651074.1 OGSv1.0 polypeptide 828414 992184 . + . method=ManualCuration;ID=GBUE004915-PA;Parent=GBUE004915-RA`

However maybe as the names indicate the two polypeptides still should belong together.

[1] GBUE004915-PA had name "ecdysone receptor isoform A" and GBUE021385-PA had name "ecdysone receptor C-term".

Chapter 6

Extract protein sequences of results of exonerate searches on 2 genomes

By looking at the multiple protein alignments of genes en, Ubx and Eip74EF, it was apparent that the putative homologues weren't found in exonerate searches of annotated protein multifasta files of genomes of *C hookeri* and *M extradentata.* A possible way to find a better match (hopefully homologuous proteins) is by searching on the whole genome especially as these polypeptide annotations available for this study in general weren't so useful either.

So first exonerate was ran four times as sbatch scripts:

```
## #!/bin/bash -l
##
## #SBATCH -A snic2019-3-298
## #SBATCH -t 20:00:00
## #SBATCH -p core -n 10
## #SBATCH -o exonerate-%j.out
## #SBATCH -J exonerate-align
## #SBATCH --mail-type=ALL
## #SBATCH --mail-user "your.email@student.uu.se"
##
## QUERY=$1
## TARGET=$2
## RES_ROOT=$3
## QUERY_GENE=$4
## TARGET_SPECIES=$5
##
```

```
## module load bioinfo-tools
## module load exonerate/2.4.0
##
## exonerate --model protein2genome --proteinsubmat pam250 --refine full --cores 10 $QUERY $
```

The calls with arguments were the following:

```
sbatch code/exonerate.sh data/proteins/D_mel_query_proteins/Eip74EF-PA.fas data/genomes/M_ex
sbatch code/exonerate.sh data/proteins/D_mel_query_proteins/Ubx-PA.fas data/genomes/M_extrac
sbatch code/exonerate.sh data/proteins/D_mel_query_proteins/Ubx-PA.fas data/genomes/C_hooker
sbatch code/exonerate.sh data/proteins/D_mel_query_proteins/en-PA.fas data/genomes/C_hookeri
```

These below describe the best exonerate matches:

en -> C_hookeri > en-PA_FBpp0087198_FBgn0000577_engrailed NQII01000084.1 535 0.969203 314 > en-PA_FBpp0087198_FBgn0000577_engrailed NQII01000464.1 547 0.990942 320 > en-PA_FBpp0087198_FBgn0000577_engrailed NQII01001162.1 538 0.974638 320 > en-PA_FBpp0087198_FBgn0000577_engrailed NQII01001533.1 539 0.976449 323 > en-PA_FBpp0087198_FBgn0000577_engrailed NQII01000581.1 535 0.969203 341 > en-PA_FBpp0087198_FBgn0000577_engrailed NQII01000299.1 532 0.963768 604

Ubx -> C_hookeri > Ubx-PA_FBpp0082793_FBgn0003944_Ultrabithorax NQII01000427.1 376 0.966581 305 > Ubx-PA_FBpp0082793_FBgn0003944_Ultrabithorax NQII01000093.1 383 0.984576 312 > Ubx-PA_FBpp0082793_FBgn0003944_Ultrabithorax NQII01001419.1 367 0.943445 354 > Ubx-PA_FBpp0082793_FBgn0003944_Ultrabithorax NQII01001541.1 364 0.935733 363 > Ubx-PA_FBpp0082793_FBgn0003944_Ultrabithorax NQII01000662.1 371 0.953728 406

Ubx -> M_extradentata > Ubx-PA_FBpp0082793_FBgn0003944_Ultrabithorax PNEQ01034244.1 359 0.922879 213 > Ubx-PA_FBpp0082793_FBgn0003944_Ultrabithorax PNEQ01018149.1 313 0.804627 311 > Ubx-PA_FBpp0082793_FBgn0003944_Ultrabithorax PNEQ01076519.1 388 0.997429 473

Eip74EF -> M_extradentata > Eip74EF-PA_FBpp0074965_FBgn0000567_Ecdysone-induced PNEQ01062987.1 545 0.657419 267 > Eip74EF-PA_FBpp0074965_FBgn0000567_Ecdysone-induced PNEQ01084081.1 508 0.612786 274 > Eip74EF-PA_FBpp0074965_FBgn0000567_Ecdysone-induced PNEQ01021588.1 718 0.866104 285 > Eip74EF-PA_FBpp0074965_FBgn0000567_Ecdysone-induced PNEQ01093675.1 640 0.772014 676

The last search (Eip74EF -> M_extradentata) failed with message `Failed Segmentation fault      (core dumped)` but the results got before failure were the above.

The results are somewhat better in both raw scores and query coverages than the previous ones but ultimately new MPAs using the matches could give clearer indications of the "goodness" of the matches.

Let's now extract the protein sequences. One way to do this is by parsing the exonerate output files with BioPython's `SearchIO.read` method but before that can be done there should be a unique way of identifying the the preselected matches. Unfortunately BioPython's parser doesn't manage to uniquely parse something that is different between the hits by just looking at the contents of the hits directly but there is a roundabout way of finding the wished protein sequences by which number from the beginning is the HSP in the BioPython's hit-object. The hit-object's are parsed and stored as QueryResult-objects in same order as they appear in the files so by counting position from the beginning the HSPs appear in the files the right HSP-objects can be pulled out and printed.

So let's first find the locations of the HSPs and write them to a csv-file and just as a measure for safeness write also the whole match records to be picked out from the exonerate output to another file. How the matches can be identifyed in the files is as a combination of the scaffold id and the exonerate raw score (e.g. PNEQ01062987.1 and 267).

6.1 Extract HSPs and write summary information

```
exonerate_results="analyses/exonerate_against_2_wgs"
D_mel_proteins="data/D_mel_query_proteins/fasta"

# Print the csv header
printf "organism,gene name,raw score,matching scaffold id,hsp number,match details\n" > $exo

# en -> C_hookeri
gene="en"
organism="C_hookeri"
protein_length=$(fastalength $D_mel_proteins/"$gene""-PA.fas" | awk '{print $1}')
# Unique match details
declare -A matches

matches["NQII01000084.1"]="314"
matches["NQII01000464.1"]="320"
matches["NQII01001162.1"]="320"
matches["NQII01001533.1"]="323"
matches["NQII01000581.1"]="341"
matches["NQII01000299.1"]="604"

# Print header for human readable match data
head -n 2 $exonerate_results/"$gene""-PA--to--""$organism"".res" > $exonerate_results"/"$gen

for match in "${!matches[@]}"; do
  #echo $match --- ${matches[$match]}
  raw_sc=${matches[$match]}
  # Let's write these to csv too for readability & completeness sake
  printf "%s," "$organism" >> $exonerate_results"/extracted_data_summary.csv"
  printf "%s," "$gene" >> $exonerate_results"/extracted_data_summary.csv"
  printf "%i," "$raw_sc" >> $exonerate_results"/extracted_data_summary.csv"
  printf "%s," "$match" >> $exonerate_results"/extracted_data_summary.csv"
  # Extract hsp number which will be read in a python script
  hsp_no=$(grep -E "vulgar.+""$match" $exonerate_results/"$gene""-PA--to--"$organism".res" |
  printf "%i," "$hsp_no" >> $exonerate_results"/extracted_data_summary.csv"
  # Extract the hsp row details in same format as before just to make sure that hsp number
  match_details=$(grep -E "vulgar.+""$match" $exonerate_results/"$gene""-PA--to--"$organism"
  printf "%s\n" "$match_details" >> $exonerate_results"/extracted_data_summary.csv"
  match_details_human_readable=$(pcre2grep -M -B 4 -A 1 ".+Target: $match.+\$\n.+Model: prot
  #echo $match_details_human_readable
  printf "%s\n" "$match_details_human_readable" >> $exonerate_results"/"$gene"_extracted_dat
done

# Clearing the variable so looping through won't take in unnecessary key/values
unset matches



# Ubx -> C_hookeri
gene="Ubx"
organism="C_hookeri"
protein_length=$(fastalength $D_mel_proteins/"$gene""-PA.fas" | awk '{print $1}')
# Unique match details
declare -A matches

matches["NQII01000427.1"]="305"
matches["NQII01000093.1"]="312"
matches["NQII01001419.1"]="354"
matches["NQII01001541.1"]="363"
matches["NQII01000662.1"]="406"

head -n 2 $exonerate_results/"$gene""-PA--to--""$organism"".res" > $exonerate_results"/"$gen

for match in "${!matches[@]}"; do
  raw_sc=${matches[$match]}
  # Let's write these to csv too for readability & completeness sake
  printf "%s," "$organism" >> $exonerate_results"/extracted_data_summary.csv"
  printf "%s," "$gene" >> $exonerate_results"/extracted_data_summary.csv"
  printf "%i," "$raw_sc" >> $exonerate_results"/extracted_data_summary.csv"
  printf "%s," "$match" >> $exonerate_results"/extracted_data_summary.csv"
  # Extract hsp number which will be read in a python script
  hsp_no=$(grep -E "vulgar.+""$match" $exonerate_results/"$gene""-PA--to--"$organism".res" |
  printf "%i," "$hsp_no" >> $exonerate_results"/extracted_data_summary.csv"
  # Extract the hsp row details in same format as before just to make sure that hsp number 
  match_details=$(grep -E "vulgar.+""$match" $exonerate_results/"$gene""-PA--to--"$organism'
  printf "%s\n" "$match_details" >> $exonerate_results"/extracted_data_summary.csv"
  match_details_human_readable=$(pcre2grep -M -B 4 -A 1 ".+Target: $match.+\$\n.+Model: prot
  #echo $match_details_human_readable
  printf "%s\n" "$match_details_human_readable" >> $exonerate_results"/"$gene"_extracted_dat
done


unset matches



# Ubx -> M_extradentata
gene="Ubx"
organism="M_extradentata"
protein_length=$(fastalength $D_mel_proteins/"$gene""-PA.fas" | awk '{print $1}')

# Unique match details
declare -A matches
matches["PNEQ01034244.1"]="213"
matches["PNEQ01018149.1"]="311"
matches["PNEQ01076519.1"]="473"

head -n 2 $exonerate_results/"$gene""-PA--to--""$organism"".res" > $exonerate_results"/"$gen

for match in "${!matches[@]}"; do
  raw_sc=${matches[$match]}
  # Let's write these to csv too for readability & completeness sake
  printf "%s," "$organism" >> $exonerate_results"/extracted_data_summary.csv"
  printf "%s," "$gene" >> $exonerate_results"/extracted_data_summary.csv"
  printf "%i," "$raw_sc" >> $exonerate_results"/extracted_data_summary.csv"
  printf "%s," "$match" >> $exonerate_results"/extracted_data_summary.csv"
  # Extract hsp number which will be read in a python script
  hsp_no=$(grep -E "vulgar.+""$match" $exonerate_results/"$gene""-PA--to--"$organism".res" 
  printf "%i," "$hsp_no" >> $exonerate_results"/extracted_data_summary.csv"
  # Extract the hsp row details in same format as before just to make sure that hsp number 
  match_details=$(grep -E "vulgar.+""$match" $exonerate_results/"$gene""-PA--to--"$organism'
  printf "%s\n" "$match_details" >> $exonerate_results"/extracted_data_summary.csv"
  match_details_human_readable=$(pcre2grep -M -B 4 -A 1 ".+Target: $match.+\$\n.+Model: prot
  #echo $match_details_human_readable
  printf "%s\n" "$match_details_human_readable" >> $exonerate_results"/"$gene"_extracted_dat
done
```

```
unset matches



# Eip74EF -> M_extradentata
gene="Eip74EF"
organism="M_extradentata"
protein_length=$(fastalength $D_mel_proteins/"$gene""-PA.fas" | awk '{print $1}')

# Unique match details
declare -A matches
matches["PNEQ01062987.1"]="267"
matches["PNEQ01084081.1"]="274"
matches["PNEQ01021588.1"]="285"
matches["PNEQ01093675.1"]="676"

head -n 2 $exonerate_results/"$gene""-PA--to--""$organism"".res" > $exonerate_results"/"$ge

for match in "${!matches[@]}"; do
  raw_sc=${matches[$match]}
  # Let's write these to csv too for readability & completeness sake
  printf "%s," "$organism" >> $exonerate_results"/extracted_data_summary.csv"
  printf "%s," "$gene" >> $exonerate_results"/extracted_data_summary.csv"
  printf "%i," "$raw_sc" >> $exonerate_results"/extracted_data_summary.csv"
  printf "%s," "$match" >> $exonerate_results"/extracted_data_summary.csv"
  # Extract hsp number which will be read in a python script
  hsp_no=$(grep -E "vulgar.+""$match" $exonerate_results/"$gene""-PA--to--"$organism".res"
  printf "%i," "$hsp_no" >> $exonerate_results"/extracted_data_summary.csv"
  # Extract the hsp row details in same format as before just to make sure that hsp number
  match_details=$(grep -E "vulgar.+""$match" $exonerate_results/"$gene""-PA--to--"$organism
  printf "%s\n" "$match_details" >> $exonerate_results"/extracted_data_summary.csv"
  match_details_human_readable=$(pcre2grep -M -B 4 -A 1 ".+Target: $match.+\$\n.+Model: pro
  #echo $match_details_human_readable
  printf "%s\n" "$match_details_human_readable" >> $exonerate_results"/"$gene"_extracted_da
done
```

6.2 Extract protein sequences from shortened exonerate results using BioPython's `SearchIO.read`

6.2.1 Read in best exonerate results

Here the previously (shortened and) extracted data is read into memory

```
from Bio import SearchIO
from Bio import SeqIO
from Bio.Alphabet import generic_protein
from Bio.Seq import Seq
from Bio.SeqRecord import SeqRecord
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC
import re
from re import split as spl

# paths to exonerate results
#paths = ['analyses/exonerate_against_2_wgs/whole_results/Eip74EF-PA--to--M_extradentata.re
#         'analyses/exonerate_against_2_wgs/whole_results/en-PA--to--C_hookeri.res',
#         'analyses/exonerate_against_2_wgs/whole_results/Ubx-PA--to--C_hookeri.res',
#         'analyses/exonerate_against_2_wgs/whole_results/Ubx-PA--to--M_extradentata.res']

paths = ['analyses/exonerate_against_2_wgs/best_hits_only/Eip74EF_extracted_data_M_extradent
         'analyses/exonerate_against_2_wgs/best_hits_only/en_extracted_data_C_hookeri.res',
         'analyses/exonerate_against_2_wgs/best_hits_only/Ubx_extracted_data_C_hookeri.res',
         'analyses/exonerate_against_2_wgs/best_hits_only/Ubx_extracted_data_M_extradentata.

exonerate_results = []

exonerate_results.append(SearchIO.read(paths[0], 'exonerate-text'))
exonerate_results.append(SearchIO.read(paths[1], 'exonerate-text'))
exonerate_results.append(SearchIO.read(paths[2], 'exonerate-text'))
exonerate_results.append(SearchIO.read(paths[3], 'exonerate-text'))

#exonerate_results.append(SearchIO.read(paths[0], 'exonerate-vulgar'))
#exonerate_results.append(SearchIO.read(paths[1], 'exonerate-vulgar'))
#exonerate_results.append(SearchIO.read(paths[2], 'exonerate-vulgar'))
#exonerate_results.append(SearchIO.read(paths[3], 'exonerate-vulgar'))
#print(exonerate_results[0])
```

6.2.2 Read into memory summary data of the best results

Here is read into memory the summary data:

```
summary = []

with open('analyses/exonerate_against_2_wgs/whole_results/extracted_data_summary.csv') as cs
  for row, line in enumerate(csv_file, start=0):
    # Skip header
    if(row>0):
      current_row = spl(r',', line)
      organism = current_row[0]
      gene = current_row[1]
      raw_score = current_row[2]
      scaffold_id = current_row[3]
      hsp_number = current_row[4]
      match_details = current_row[5]
      summary.append([organism,gene,raw_score,scaffold_id,hsp_number,match_details])

#print(summary)
```

6.2.3 Extract *M extradentata* Eip74EF protein sequences

```
cur_res = exonerate_results[0]
ex_out_organism = "M_extradentata"
ex_out_gene = "Eip74EF"
records = []
fasta_out_path = "analyses/exonerate_against_2_wgs/fastas_from_best_hits_using_BioPython/" +

for data in summary:
  organism = data[0]
  gene = data[1]
  raw_score = data[2]
  scaffold_id = data[3]
  hsp_number = int(data[4])
  #type(hsp_number)
  #print(hsp_number)
  match_details = data[5]
  if ex_out_gene == gene and ex_out_organism == organism:
    #print(organism, gene, scaffold_id, raw_score)
    # Find the scaffold where the match happened with scaffold id
    for hit in cur_res.hits:
      if hit.id == scaffold_id:
        #print(hit.id)
        concatenated = ""
        valid = True
        X_count = 0
        for fragment in hit.hsps[0].fragments:
          #print(fragment)
          current_seq = fragment.aln[1].seq
          #print(current_seq)
          for aa in str(current_seq):
            #print(aa)
            if aa != "X":
              concatenated += aa
              #print("found other chars")
              #other_chars = True
            else:
              X_count +=1
              if X_count > 50:
                valid = False
                break
        if not valid:
          print("Invalid sequence")
          break

        print(concatenated)
        records.append(SeqRecord(Seq(concatenated, generic_protein),
                 id=scaffold_id,
                 description=ex_out_organism + " " + ex_out_gene))

SeqIO.write(records, fasta_out_path, "fasta")
```

6.2.4 Extract *C hookeri* en protein sequences

```
cur_res = exonerate_results[1]
ex_out_organism = "C_hookeri"
ex_out_gene = "en"
records = []
fasta_out_path = "analyses/exonerate_against_2_wgs/fastas_from_best_hits_using_BioPython/" +

for data in summary:
  organism = data[0]
  gene = data[1]
  raw_score = data[2]
  scaffold_id = data[3]
  hsp_number = int(data[4])
  #type(hsp_number)
  #print(hsp_number)
  match_details = data[5]
  if ex_out_gene == gene and ex_out_organism == organism:
    #print(organism, gene, scaffold_id, raw_score)
    # Find the scaffold where the match happened with scaffold id
    for hit in cur_res.hits:
      if hit.id == scaffold_id:
        #print(hit.id)
        concatenated = ""
        valid = True
        X_count = 0
        for fragment in hit.hsps[0].fragments:
          #print(fragment)
          current_seq = fragment.aln[1].seq
          #print(current_seq)
          for aa in str(current_seq):
            #print(aa)
            if aa != "X":
              concatenated += aa
              #print("found other chars")
              #other_chars = True
            else:
              X_count +=1
              if X_count > 50:
                valid = False
                break
        if not valid:
          print("Invalid sequence")
          break
        print(concatenated)
        records.append(SeqRecord(Seq(concatenated, generic_protein),
                 id=scaffold_id,
                 description=ex_out_organism + " " + ex_out_gene))

SeqIO.write(records, fasta_out_path, "fasta")
```

6.2.5 Extract *C hookeri* Ubx protein sequences

```
cur_res = exonerate_results[2]
ex_out_organism = "C_hookeri"
ex_out_gene = "Ubx"
records = []
fasta_out_path = "analyses/exonerate_against_2_wgs/fastas_from_best_hits_using_BioPython/" +

for data in summary:
  organism = data[0]
  gene = data[1]
  raw_score = data[2]
  scaffold_id = data[3]
  hsp_number = int(data[4])
  #type(hsp_number)
  #print(hsp_number)
  match_details = data[5]
  if ex_out_gene == gene and ex_out_organism == organism:
    #print(organism, gene, scaffold_id, raw_score)
    # Find the scaffold where the match happened with scaffold id
    for hit in cur_res.hits:
      if hit.id == scaffold_id:
        #print(hit.id)
        concatenated = ""
        valid = True
        X_count = 0
        for fragment in hit.hsps[0].fragments:
          #print(fragment)
          current_seq = fragment.aln[1].seq
          #print(current_seq)
          for aa in str(current_seq):
            #print(aa)
            if aa != "X":
              concatenated += aa
              #print("found other chars")
              #other_chars = True
            else:
              X_count +=1
              if X_count > 50:
                valid = False
                break
        if not valid:
          print("Invalid sequence")
          break
        print(concatenated)
        records.append(SeqRecord(Seq(concatenated, generic_protein),
                 id=scaffold_id,
                 description=ex_out_organism + " " + ex_out_gene))

SeqIO.write(records, fasta_out_path, "fasta")
```

6.2.6 Extract *M extradentata* Ubx protein sequences

```
cur_res = exonerate_results[3]
ex_out_organism = "M_extradentata"
ex_out_gene = "Ubx"
records = []
fasta_out_path = "analyses/exonerate_against_2_wgs/fastas_from_best_hits_using_BioPython/" +

for data in summary:
  organism = data[0]
  gene = data[1]
  raw_score = data[2]
```

```
    scaffold_id = data[3]
    hsp_number = int(data[4])
    #type(hsp_number)
    #print(hsp_number)
    match_details = data[5]
    if ex_out_gene == gene and ex_out_organism == organism:
      #print(organism, gene, scaffold_id, raw_score)
      # Find the scaffold where the match happened with scaffold id
      for hit in cur_res.hits:
        if hit.id == scaffold_id:
          #print(hit.id)
          concatenated = ""
          valid = True
          X_count = 0
          for fragment in hit.hsps[0].fragments:
            #print(fragment)
            current_seq = fragment.aln[1].seq
            #print(current_seq)
            for aa in str(current_seq):
              # Skip all "X":s
              if aa != "X":
                concatenated += aa
                #print("found other chars")
                #other_chars = True
              else:
                X_count +=1
                if X_count > 50:
                  valid = False
                  break
          if not valid:
            print("Invalid sequence")
            break
          print(concatenated)
          records.append(SeqRecord(Seq(concatenated, generic_protein),
                   id=scaffold_id,
                   description=ex_out_organism + " " + ex_out_gene))

SeqIO.write(records, fasta_out_path, "fasta")
```

6.2.7 Make the multifasta files into 2-line multifastas and remove empty lines

Now that we have some protein multifasta or fasta files they can be made into two line fastas.

```
fasta_path="analyses/exonerate_against_2_wgs/fastas_from_best_hits_using_BioPython/"
read -r -a fastas <<< $( find $fasta_path -name "*.faa" -and -type f -print0 | xargs -0 echo

for fasta in "${fastas[@]}"; do
  file=$(echo $(basename "$fasta")) # e.g. broad.fasta
  gene_organism=$(echo $(basename "$fasta") | awk -F "." '{print $1}') # e.g. fasta
  gene=$(echo $gene_organism | awk -F "_" '{print $1}')
  organism=$(echo $gene_organism | awk -F "_" '{print $2"_"$3}')
  cat "$fasta" | awk '/^>/ {printf("\n%s\n",$0);next; } { printf("%s",$0);}' > $organism"_pe
  #echo $fasta, $file, $gene_organism, $gene, $organism
  mv $organism"_pep.fa" "$fasta_path""$gene""_""$organism"".faa"
done
```

6.2.8 Remove blank lines from the fasta files

The blank lines will mess up the readalise python script so they need to be removed:

```
fasta_path="analyses/exonerate_against_2_wgs/fastas_from_best_hits_using_BioPython/"
read -r -a fastas <<< $( find $fasta_path -name "*.faa" -and -type f -print0 | xargs -0 echo

for fasta in "${fastas[@]}"; do
  file=$(echo $(basename "$fasta")) # e.g. broad.fasta
  gene_organism=$(echo $(basename "$fasta") | awk -F "." '{print $1}') # e.g. fasta
  gene=$(echo $gene_organism | awk -F "_" '{print $1}')
  organism=$(echo $gene_organism | awk -F "_" '{print $2"_"$3}')
  sed '/^$/d' $fasta > temp.fa
  # Replace the hold version with the new one
  mv temp.fa $fasta
done
```

6.2.9 Make fasta headers readable for further analyses

```
readabilising_script="code/readabalise_header.py"
output="analyses/exonerate_against_2_wgs/readablilised_fastas_from_best_hits_using_BioPytho
input="analyses/exonerate_against_2_wgs/fastas_from_best_hits_using_BioPython/"
read -r -a fastas <<< $( find $input -name "*.faa" -and -type f -print0 | xargs -0 echo )

for fasta in "${fastas[@]}"; do
  file=$(echo $(basename "$fasta")) # e.g. broad.fasta
  gene_organism=$(echo $(basename "$fasta") | awk -F "." '{print $1}') # e.g. fasta
  gene=$(echo $gene_organism | awk -F "_" '{print $1}')
  organism=$(echo $gene_organism | awk -F "_" '{print $2"_"$3}')
  #echo $file, $gene_organism, $gene, $organism, $fasta
  python3 $readabilising_script -i "$fasta" -o "$output""$gene_organism"".fa" > "$output""$g
  #echo $fasta, $file, $gene_organism #, $readabilising_script, $output
  #mv "$output""$gene_organism"".fa" "$output""$gene_organism"".faa"
done
```

6.3 Rerun proteinsequence extraction with gff & perl script

Because the matches extracted with the above way didn't produce the best exonerate matches, another exonerate search was executed with *D melanogaster* genes just against the scaffolds where the matches were found when searching in the whole genome assemblies. Below are the searches:

6.3.1 Rerun exonerate with gff as additional output format

```
exonerate --model protein2genome --proteinsubmat pam250 --refine full --cores 6 --showtarget
exonerate --model protein2genome --proteinsubmat pam250 --refine full --cores 6 --showtarget
exonerate --model protein2genome --proteinsubmat pam250 --refine full --cores 6 --showtarget
exonerate --model protein2genome --proteinsubmat pam250 --refine full --cores 6 --showtarget
exonerate --model protein2genome --proteinsubmat pam250 --refine full --cores 6 --showtarget
exonerate --model protein2genome --proteinsubmat pam250 --refine full --cores 6 --showtarget
exonerate --model protein2genome --proteinsubmat pam250 --refine full --cores 6 --showtarget
exonerate --model protein2genome --proteinsubmat pam250 --refine full --cores 6 --showtarget
exonerate --model protein2genome --proteinsubmat pam250 --refine full --cores 6 --showtarget
exonerate --model protein2genome --proteinsubmat pam250 --refine full --cores 6 --showtarget
```

As can be seen from the commands above, `gff`-output format is what we're after because that can be parsed and used to obtain the translated sequences of where the queries aligned. The results were obtained and stored in `analyses/exonerate_results/exonerate_matches_against_extracted_scaffolds`. The output files contain in addition to gff output also some human readable match data and in some cases other matches than the one(s) we're interested in. Let's get rid of them:

6.3.2 Extract scaffolds where best exonerate matches were found

The scaffolds against which the searches were run were obtained by first creating BLAST databases from the assemblies with commands:

```
makeblastdb -dbtype nucl -in M_extradentata.fna -parse_seqids
makeblastdb -dbtype nucl -in C_hookeri.fna -parse_seqids
```

and then extracting the scaffolds of interest with commands:

```
# C_hook# or M_ext# are the human readable texts used in MPAs later
# * means that this is a new entry (the best match)
# the last numbers in the comment lines are exonerate raw scores of the best matches that w

DATA="data/genomes"
SCAFFOLDS="data/C_hook_M_ext_scaffolds_where_wg_best_matches_were_found"

#en C_hookeri NQII01000299.1 C_hook2* 604
blastdbcmd -db $DATA/C_hookeri.fna -entry NQII01000299.1 > $SCAFFOLDS/C_hookeri_NQII01000299
#en C_hookeri NQII01000084.1 C_hook1 314
blastdbcmd -db $DATA/C_hookeri.fna -entry NQII01000084.1 > $SCAFFOLDS/C_hookeri_NQII01000084
#Ubx C_hookeri NQII01001419.1 C_hook1 354
blastdbcmd -db $DATA/C_hookeri.fna -entry NQII01001419.1 > $SCAFFOLDS/C_hookeri_NQII01001419
#Ubx C_hookeri NQII01000427.1 C_hook2 305
blastdbcmd -db $DATA/C_hookeri.fna -entry NQII01000427.1 > $SCAFFOLDS/C_hookeri_NQII01000427
#Ubx C_hookeri NQII01000662.1 C_hook4* 406
blastdbcmd -db $DATA/C_hookeri.fna -entry NQII01000662.1 > $SCAFFOLDS/C_hookeri_NQII01000662
#Ubx C_hookeri NQII01000093.1 C_hook3* 312
blastdbcmd -db $DATA/C_hookeri.fna -entry NQII01000093.1 > $SCAFFOLDS/C_hookeri_NQII01000093
#Ubx M_extradentata PNEQ01076519.1 M_ext2* 473
blastdbcmd -db $DATA/M_extradentata.fna -entry PNEQ01076519.1 > $SCAFFOLDS/M_extradentata_PN
#Ubx M_extradentata PNEQ01018149.1 M_ext1 311
blastdbcmd -db $DATA/M_extradentata.fna -entry PNEQ01018149.1 > $SCAFFOLDS/M_extradentata_PN
#Eip74EF M_extradentata PNEQ01062987.1 M_ext1 267
blastdbcmd -db $DATA/M_extradentata.fna -entry PNEQ01062987.1 > $SCAFFOLDS/M_extradentata_PN
#Eip74EF M_extradentata PNEQ01084081.1 M_ext2 274
blastdbcmd -db $DATA/M_extradentata.fna -entry PNEQ01084081.1 > $SCAFFOLDS/M_extradentata_PN
#Eip74EF M_extradentata PNEQ01093675.1 M_ext3* 676
blastdbcmd -db $DATA/M_extradentata.fna -entry PNEQ01093675.1 > $SCAFFOLDS/M_extradentata_PN
```

It happens so that in the case of these scaffolds that each scaffold id contained only one best match so the scaffold id:s in this case worked aswell as ad hoc unique identifiers for the matches. (This hopefully explains the addition of raw scores and human readable protein identifyers.)

6.3.3 Retrieve gff sections from exonerate output files

```
exonerate_output="analyses/exonerate_results_against_scaffolds_with_matches_with_gff_output/
gffs="analyses/exonerate_results_against_scaffolds_with_matches_with_gff_output/gffs_only"

match_details_human_readable=$(pcre2grep -M -B 11 -A 2 "PNEQ01062987.1  exonerate:protein2ge
printf "%s" "$match_details_human_readable" > $gffs/"Eip74EF-PA--to--M_extradentata_PNEQ0106

match_details_human_readable=$(pcre2grep -M -B 11 -A 2 "PNEQ01084081.1  exonerate:protein2ge
printf "%s" "$match_details_human_readable" > $gffs/"Eip74EF-PA--to--M_extradentata_PNEQ0108
```

```
match_details_human_readable=$(pcre2grep -M -B 11 -A 2 "PNEQ01093675.1  exonerate:protein2ge
printf "%s" "$match_details_human_readable" > $gffs/"Eip74EF-PA--to--M_extradentata_PNEQ0109

match_details_human_readable=$(pcre2grep -M -B 11 -A 2 "NQII01000093.1  exonerate:protein2ge
printf "%s" "$match_details_human_readable" > $gffs/"Ubx--to--C_hookeri_NQII01000093.1".gff

match_details_human_readable=$(pcre2grep -M -B 11 -A 2 "NQII01000427.1  exonerate:protein2ge
printf "%s" "$match_details_human_readable" > $gffs/"Ubx-PA--to--C_hookeri_NQII01000427.1".g

match_details_human_readable=$(pcre2grep -M -B 11 -A 2 "NQII01000662.1  exonerate:protein2ge
printf "%s" "$match_details_human_readable" > $gffs/"Ubx-PA--to--C_hookeri_NQII01000662.1".g

match_details_human_readable=$(pcre2grep -M -B 11 -A 2 "NQII01001419.1  exonerate:protein2ge
printf "%s" "$match_details_human_readable" > $gffs/"Ubx-PA--to--C_hookeri_NQII01001419.1".g

match_details_human_readable=$(pcre2grep -M -B 11 -A 2 "PNEQ01018149.1  exonerate:protein2ge
printf "%s" "$match_details_human_readable" > $gffs/"Ubx-PA--to--M_extradentata_PNEQ01018149

match_details_human_readable=$(pcre2grep -M -B 11 -A 2 "PNEQ01076519.1  exonerate:protein2ge
printf "%s" "$match_details_human_readable" > $gffs/"Ubx-PA--to--M_extradentata_PNEQ01076519

match_details_human_readable=$(pcre2grep -M -B 11 -A 2 "NQII01000084\.1 exonerate:protein2ge
printf "%s" "$match_details_human_readable" > $gffs/"en-PA--to--C_hookeri_NQII01000084.1".gf

match_details_human_readable=$(pcre2grep -M -B 11 -A 2 "NQII01000299.1  exonerate:protein2ge
printf "%s" "$match_details_human_readable" > $gffs/"en-PA--to--C_hookeri_NQII01000299.1".gf
```

6.3.4 Create protein and cds sequences from scaffolds and gffs using perl script

Now that the gffs were in place next task is to translate the sequences using them and the scaffold fastas. Here below are the commands for parsing and translating the protein sequences:

```
FASTAS="data/C_hook_M_ext_scaffolds_where_wg_best_matches_were_found"
GFFS="analyses/exonerate_results_against_scaffolds_with_matches_with_gff_output/gffs_only"
code/gff2fasta.pl $FASTAS/C_hookeri_NQII01000299.1.fna $GFFS/en-PA--to--C_hookeri_NQII010002
code/gff2fasta.pl $FASTAS/C_hookeri_NQII01000084.1.fna $GFFS/en-PA--to--C_hookeri_NQII010000
code/gff2fasta.pl $FASTAS/C_hookeri_NQII01001419.1.fna $GFFS/Ubx-PA--to--C_hookeri_NQII01001
code/gff2fasta.pl $FASTAS/C_hookeri_NQII01000427.1.fna $GFFS/Ubx-PA--to--C_hookeri_NQII01000
code/gff2fasta.pl $FASTAS/C_hookeri_NQII01000662.1.fna $GFFS/Ubx-PA--to--C_hookeri_NQII01000
code/gff2fasta.pl $FASTAS/C_hookeri_NQII01000093.1.fna $GFFS/Ubx-PA--to--C_hookeri_NQII01000
code/gff2fasta.pl $FASTAS/M_extradentata_PNEQ01076519.1.fna $GFFS/Ubx-PA--to--M_extradentata
code/gff2fasta.pl $FASTAS/M_extradentata_PNEQ01018149.1.fna $GFFS/Ubx-PA--to--M_extradentata
code/gff2fasta.pl $FASTAS/M_extradentata_PNEQ01062987.1.fna $GFFS/Eip74EF-PA--to--M_extraden
```

```
code/gff2fasta.pl $FASTAS/M_extradentata_PNEQ01084081.1.fna $GFFS/Eip74EF-PA--to--M_extraden
code/gff2fasta.pl $FASTAS/M_extradentata_PNEQ01093675.1.fna $GFFS/Eip74EF-PA--to--M_extraden
```

Katoh, Kazutaka, and Daron M Standley. 2013. “MAFFT Multiple Sequence Alignment Software Version 7: Improvements in Performance and Usability.” *Molecular Biology and Evolution* 30 (4): 772–80. https://doi.org/10.1093/molbev/mst010.

Price, Morgan N., Paramvir S. Dehal, and Adam P. Arkin. 2010. “FastTree 2 – Approximately Maximum-Likelihood Trees for Large Alignments.” Edited by Art F. Y. Poon. *PLoS ONE* 5 (3): e9490. https://doi.org/10.1371/journal.pone.0009490.

Slater, Guy St C., and Ewan Birney. 2005. “Automated generation of heuristics for biological sequence comparison.” *BMC Bioinformatics* 6 (February): 31. https://doi.org/10.1186/1471-2105-6-31.