Metagenomics Profiling

Taxonomic and functional profiling of metagenomes using a gene-centric approach

Contents

References

Anders, S., Pyl, P. T., Huber, W. 2015. HTSeq-a Python framework to work with high-throughput sequencing data. *Bioinformatics*, 31(2): 166-9. doi: 10.1093/bioinformatics/btu638.

Cury, Jean, Thomas Jové, Marie Touchon, Bertrand Néron, and Eduardo PC Rocha. 2016. 'Identification and Analysis of Integrons and Cassette Arrays in Bacterial Genomes'. *Nucleic Acids Research* 44 (10): 4539–50. https://doi.org/10.1093/nar/gkw319.

Darling, E. A., Jospin, G., Lowe, E., Matsen, F. A., Bik, H. M., Eisen, J. A. 2014. PhyloSift: phylogenetic analysis of genomes and metagenomes. *PeerJ*, 2:e243. doi: 10.7717/peerj.243.

Feldgarden, Michael, Vyacheslav Brover, Daniel H. Haft, Arjun B. Prasad, Douglas J. Slotta, Igor Tolstoy, Gregory H. Tyson, et al. 2019. 'Using the NCBI AMRFinder Tool to Determine Antimicrobial Resistance Genotype-Phenotype Correlations Within a Collection of NARMS Isolates'. Preprint. Microbiology. https://doi.org/10.1101/550707.

Gregor, I., Dröge, J., Schirmer, M., Quince, C., & McHardy, A. C. 2016. PhyloPythiaS+: a self-training method for the rapid reconstruction of low-ranking taxonomic bins from metagenomes. *PeerJ*, *4*, e1603.

Hyatt, Doug, Gwo-Liang Chen, Philip F LoCascio, Miriam L Land, Frank W Larimer, and Loren J Hauser. 2010. 'Prodigal: Prokaryotic Gene Recognition and Translation Initiation Site Identification'. *BMC Bioinformatics* 11 (March): 119. https://doi.org/10.1186/1471-2105-11-119.

Nawrocki, E. P., and S. R. Eddy. 2013. 'Infernal 1.1: 100-Fold Faster RNA Homology Searches'. *Bioinformatics* 29 (22): 2933–35. https://doi.org/10.1093/bioinformatics/btt509.

Ogata, H., S. Goto, K. Sato, W. Fujibuchi, H. Bono, and M. Kanehisa. 1999. 'KEGG: Kyoto Encyclopedia of Genes and Genomes'. *Nucleic Acids Research* 27 (1): 29–34. https://doi.org/10.1093/nar/27.1.29.

Prestat, E., et al. 2014. FOAM (Functional Ontology Assignments for Metagenomes): a Hidden Markov Model (HMM) database with environmental focus. *Nucleic Acids Research*, 42(19): e145. doi: 10.1093/nar/gku702.

Data Preparation

Starting Materials:

- FASTA file of assembled contigs
- High-quality (preprocessed) paired-end short reads in FASTQ format
- BAM file of the high-quality paired-end short reads mapped to the assembly

Initialize PATH variables:

```
project="<project_name>"
sample="<sample_name>"
contigs="${sample}.contigs.renamed.fa"
fq="${sample}.interleaved.atrim.decontam.qtrim.fq.gz"
mapfile="${sample}.mapped.sorted.bam"
dbs="/srv/databases/internal/"
fasta_dir="${dbs}/fasta"
blast_dir="${dbs}/blast"
hmm_dir="${dbs}/hmm"
cm_dir="${dbs}/cm"
diamond_dir="${dbs}/diamond"
json_dir="${dbs}/json"
```

Set universal parameters:

```
cpus=<number>
```

Taxonomic Profiling

Taxonomic profiling of short reads with Phylosift

Use PhyloSift to classify taxonomy of short reads with marker genes:

```
phylosift all --output phylogeny_paired --threads ${cpus} --paired <forward_reads>
<reverse_reads>
phylosift all --output phylogeny_singles --threads ${cpus} <single-end_reads>
```

* preprocessing should already have been performed on the data prior to this point. The input reads will be the same reads used to generate the assembly.
* please **do not** use more than two threads when running phylosift. The limiting steps do not use more than one CPU at a time, so providing phylosift with more than two threads consumes additional resources with very little speedup.

Compare phylosift results from multiple metagenomes in one table (optional) :

```
graphs2table.py -l 6 -o combined.phylosift.tsv <sample1>.html <sample2>.html
```

* the '-l 6' argument causes the table to be generated at taxonomic level 6 = family. Level 5 is order, level 4 is class, etc.

Create an archive of unneeded files:
```
cd phylogeny_<singles_or_paired>
tar -cjf archive.tar.bz2 alignDir treeDir
cd ..
```

Taxonomic profiling of assembled reads with PhyloPythiaS+

Copy /usr/local/etc/ppsp_example.cfg to your local directory and edit with a text editor:
```
cp /usr/local/etc/ppsp_example.cfg ./
nano ppsp_example.cfg
```
* PhyloPythiaS+ uses a configuration file to define the input and output files.

Set the following required parameters in the configuration file:
 # Project pipeline directory for output and temporary data. ***Specify full path.***
 # Fasta file containing contigs/sequences (for the classification)
* Save the file with a new name, such as ppsp_myproject.cfg

Create the output directory that you named in the configuration file:
```
mkdir <NAME-OF-OUTPUT-DIR>
```

Use PhyloPythiaS+ to classify taxonomy of assembled contigs by running the whole PhyloPythiaS+ pipeline:
```
ppsp -c <CONFIG-FILE> -n -g -o s16 mg -t -p c -r -s
```
* useful output files:
summary_train.txt: cumulative sizes and numbers of contigs with each taxonomy
<YOUR-NAME>.PP.pOUT: taxonomy of each contig

Functional Profiling

Feature prediction

Search for putative CRISPR arrays:
```
minced -minNR 3 -gffFull ${contigs} ${sample}.contigs.crispr.gff
```

Search identified CRISPR repeats against the assembly to find additional degraded repeats:
```
makeblastdb -dbtype nucl -max_file_sz 2GB -logfile ${sample}.contigs.indexed.log -
title ${sample} -in ${contigs} -out ${sample}.contigs.index

repeats=$(grep "repeat_unit" ${sample}.contigs.crispr.gff | awk '{print
$1":"$4"-"$5}')
samtools faidx ${contigs} $repeats | blastn -num_threads ${cpus} -evalue ${evalue} -
outfmt 11 -out ${sample}.contigs.crispr.b11 -query stdin -db ${sample}.contigs.index
```

Predict transfer, transfer-messenger, and other non-coding RNAs by searching for homologs to characterized ncRNAs:

```
cmscan --cpu ${cpus} --cut_ga --nohmmonly --rfam --noali --fmt 2 --tblout $
{sample}.contigs.ncRNA.csv -o ${sample}.contigs.ncRNA.log --clanin
${cm_dir}/Rfam.clanin ${cm_dir}/Rfam.cm ${contigs}

awk '$20 != "=" {print $0}' ${sample}.contigs.ncRNA.csv > $
{sample}.contigs.ncRNA.bh.csv  #best hits
awk -v threshold="$evalue" 'NR==1 || $18 < threshold' ${sample}.contigs.ncRNA.bh.csv >
${sample}.contigs.ncRNA.bh.filtered.csv  #filter out hits with e-value > threshold
```

Reformat output of ncRNA prediction as GFF3:

```
ncrna_gff="${sample}.contigs.ncRNA.gff3"
awk '/^#/{next} BEGIN{print "##gff-version 3"; next} {i++}{if($9 < $10)
{strand="+";min=$9;max=$10} else {strand="-";max=$9;min=$10}; print $4"\tInfernal-
<version>\tncRNA\t"min"\t"max"\t"$17"\t"strand"\t0\
tID="i";Alias="$3";Name="$2";Dbxref=Rfam:"$3"}' ${sample}.contigs.ncRNA.bh.filtered.csv
>${ncrna_gff}
```

Use motif detection to locate open reading frames (ORFs) on the contigs and translate the ORFs to protein sequences:

```
cds_faa="${sample}.contigs.cds.faa"
cds_gff="${sample}.contigs.cds.gff3"

prodigal -q -p meta -f gff -i ${contigs} -o ${cds_gff} -a ${cds_faa}
```

Find and remove spurious CDS predictions:

```
hmmsearch --cpu 4 --cut_ga --tblout ${sample}.contigs.cds.bad.csv --out /dev/null
/srv/databases/internal/hmm/AntiFam.hmm ${cds_faa}
```

Remove invalid characters from the protein sequences and simplify headers:

```
clean_cds="${sample}.contigs.cds.clean.faa"
awk '/^>/{split($0, a, " "); print a[1]; next} {gsub(/[-\*_\.]/,""); print}' $
{cds_faa} >${clean_cds}
sed -i 's/\x0//g' ${clean_cds}  # NULL bytes: common undesirable artifact produced by
prodigal for unknown reasons
mv ${clean_cds} ${cds_faa}
```

Link sequence IDs to GFF entries using the GFF's ID attribute:

```
clean_gff="${sample}.contigs.cds.clean.gff3"
awk -F'\t' '/^#/{print $0; next} {sub(/ID\=[^_]*_/, "ID="$1"_", $9); print $1"\t"$2"\
t"$3"\t"$4"\t"$5"\t"$6"\t"$7"\t"$8"\t"$9}' ${cds_gff} >${clean_gff}
mv ${clean_gff} ${cds_gff}
```

* this step is necessary for successful downstream annotation of protein predictions.

Merge predicted features:

```
gff_merged="${sample}.contigs.merged.gff3"
```

```
srun combine_features ${ncrna_gff} ${sample}.contigs.crispr.gff ${cds_gff} >${gff_merged} &
```
* if features overlap, precedence will be determined by input order of the parent GFF3 file. It is recommended that RNA predictions be preferred over ORFs.

Functional annotation (general)

A number of non-specific protein reference databases are available for broad functional annotation. These databases are often used in conjunction with a functional ontology to further describe groups of genes. Both the KEGG (ko) and FOAM ontologies associate KEGG orthologs (KOs) to networks of molecular interactions, reactions, and relationships.

Set program parameters:
```
evalue=<value>
identity=<fraction>
coverage=<length>
```
* `evalue` is the expect value maximum. Hits to a reference with e-value greater than this threshold will be discarded
* `identity` is the fraction of the aligned sequence with positive scoring matches (positions with the same residue in both a query and reference)
* `coverage` is the number of bases with positive scoring matches an alignment must contain to be considered a true match

option 1: FOAM

Assign functional annotation to the translated ORFs by searching for homology to proteins with known function:
```
hmmsearch --cpu 4 -E ${evalue} --noali --tblout ${sample}.contigs.kegg.csv -o /dev/stdout ${hmm_dir}/FOAM.hmm ${cds_faa} | gzip --best 1>${sample}.contigs.kegg.log.gz
```

Reformat output as B6:
```
awk '/^#/{next} {match($3, "K[0-9]+", subject); print $1"\t"subject[0]"\t-\t-\t-\t-\t-\t-\t-\t"$5"\t"$6}' ${sample}.cds.kegg.csv >${sample}.cds.kegg.b6
```

option 2: KEGG

Assign functional annotation to the translated ORFs by searching for homology to proteins with known function:
```
diamond blastp --threads ${cpus} --tmpdir ./ --sensitive --max-target-seqs 1 --evalue ${evalue} --outfmt 6 --out ${sample}.contigs.kegg.b6 --db ${diamond_dir}/KEGG-prokaryotes --query ${cds_faa} 2>${sample}.kegg.log
```

Functional annotation (specialized)

Additional reference databases are available for fine-grained searching of specific features of interest, including mobile elements, integrons, and antimicrobial resistance elements.

Mobile genetic elements

Search for homologs of known mobile genetic elements such as transposons, plasmids, ICEs, and phages:

```
blastp -num_threads ${cpus} -max_hsps 1 -evalue ${evalue} -outfmt 11 -out ${sample}.contigs.mges.b11 -query ${cds_faa} -db ${blast_dir}/ACLAME

blastp -num_threads ${cpus} -max_hsps 1 -evalue ${evalue} -outfmt 11 -out ${sample}.contigs.ices.b11 -query ${cds_faa} -db ${blast_dir}/ICEberg

hmmsearch --cpu ${cpus} --cut_ga --tblout contigs.phages.csv ${hmm_dir}/GyDB.hmm.gz ${cds_faa}
```

Filter hits by alignment length and percent identity and reformat as b6:

```
blast_formatter -archive ${sample}.contigs.mges.b11 -outfmt 6 -max_target_seqs 1 | awk -v identity="$ident_assem" -v coverage="$cov_assem" 'NR==1 || ($3 > identity && $4 > coverage)' >${sample}.contigs.mges.screen.b6

blast_formatter -archive ${sample}.contigs.ices.b11 -outfmt 6 -max_target_seqs 1 | awk -v identity="$ident_assem" -v coverage="$cov_assem" 'NR==1 || ($3 > identity && $4 > coverage)' >${sample}.contigs.ices.screen.b6
```

Integrons

Identify integrons by searching for their primary components - the integron-integrase (intI), attC recombination sites, and cassette promoters (Pc):

```
cmsearch --cpu 4 --noali -E ${evalue} --tblout ${sample}.contigs.attC.csv -o ${sample}.contigs.attC.log ${cm_dir}/attc_4.cm ${contigs}

hmmsearch --cpu 4 --noali -E ${evalue} -o ${sample}.contigs.intI.log --tblout ${sample}.contigs.intI.csv ${hmm_dir}/integron_integrase.hmm ${cds_faa}

blastn -num_threads ${cpus} -evalue ${evalue} -query ${fasta_dir}/variants_Pc_intI1.fst -db ${sample}.contigs.index -out ${sample}.contigs.Pc.b11 -outfmt 11
```

Search contigs containing intIs or attC recombination sites for additional intIs and genes known to reside on gene-cassettes:

```
integrons=$(awk -h '{print substr($1, 1, 14)}' ${sample}.contigs.attC.csv ${sample}.contigs.intI.csv | grep -v "#" | sort -u)

samtools faidx ${contigs} $integrons | srun --cpus-per-task ${cpus} blastn -num_threads ${cpus} -evalue ${evalue} -outfmt 6 -out ${sample}.contigs.integrons.b6 -db ${blast_dir}/INTEGRALL
```

Reformat output of integrase search as B6:

```
awk '/^#/{next} {split($3, subject, "_"); print $1"\t"subject[1]"\t-\t-\t-\t-\t-\t-\t-\t-\t"$5"\t"$6}' ${sample}.contigs.intI.csv >${sample}.contigs.intI.b6
```

Virulence factors

Search for functional homologs of known virulence factors:

```
blastp -num_threads ${cpus} -max_hsps 1 -evalue ${evalue} -outfmt 11 -out ${sample}.contigs.vf.b11 -query ${cds_faa} -db "${blast_dir}/VFDB-setA ${blast_dir}/VFDB-setB"
```

Filter hits by alignment length and percent identity and reformat as b6:

```
blast_formatter -archive ${sample}.contigs.vf.b11 -outfmt 6 -max_target_seqs 1 | awk -v identity="$ident_assem" -v coverage="$cov_assem" 'NR==1 || ($3 > identity && $4 > coverage)' >${sample}.contigs.vf.filtered.b6
```

Resistance genes

Search for functional homologs of genes known to confer resistance to antibiotics:

```
rgi main --num_threads ${cpus} --input_type protein -i ${cds} -o ${sample}.contigs.amr
```

Calculating Coverage

Add feature annotations to GFF:

```
gff_annot="${sample}.cds.merged.annotated.gff"

annotate_features --clear --fields gene,Ontology_term,product,organism,database --conflict [quality|order] --out ${gff_annot} --b6 ${b6_1},${b6_2},... ${gff_merged} 2>${sample}.cds.annotate.log
```

* use the --conflict argument with either order or quality to resolve conflicting annotations by file input order or confidence of the alignment, respectively. In this case, the highest confidence alignment is the one with the largest bitscore.
* a path to one or more reference databases can be supplied as an argument to --mapping to provide additional information about an alignment when annotating the GFF. Any field contained within the provided mapping file(s) can be included as an attribute. These fields should be supplied as a comma-separated list to the --fields argument.
* a default product can be added as an attribute using the --product argument (e.g. --product "hypothetical protein"). Features in the GFF without a hit to a database will be annotated with the value supplied to the argument.

Estimate feature abundances, scale abundance estimates, and map genes to gene families:

```
count_features --buffer 3145728 --mode union --sorting [name|position] --format bam --out ${sample}.cds.abunds ${sample}.mapped.sorted.derep.bam ${sample}.cds.annotated.gff 2>${sample}.count.log
```

* if the output log report indicates that mate records are missing, it may help to increase the buffer size. If the increased buffer does not resolve the issue, the input file should be checked for corruption or incompleteness.
* the alignment file, `${sample}.mapped.sorted.derep.bam`, is generated following assembly in the “MG processing” workflow, and should be pre-sorted either by position in “genome” or name. The default sorting method of `samtools sort` is by position. However, it is almost always better to have the BAM file sorted by name when using count_features, as it drastically saves on

memory. It is possible to pipe the output of `samtools sort -n` directly into `count_features`, simply replace the input BAM file with a dash (-) character.
* a number of normalization methods are available to choose from. The desired methods should be supplied to the --units argument as a comma-separated list. See count_features --help for the available options and a brief description of each.
* one or more relational databases can be supplied as a comma-separated list to the --mapping argument in order to combine features into higher levels of organization, such as genes into gene families. This argument must be used in conjunction with --category, whose value (e.g. Ontology_term) can be any field available in the supplied set of relational databases.

Compare features abundances across samples in a project:

```
abunds=($(find ./ -type f -regex ".*\.feature_coverage\.csv" | sort))

samples=()
for file in ${abunds[@]}; do
  file=$(basename ${file});
  IFS=". " read -a array <<< $file;
  sample=${array[0]};
  samples+=("$sample");
done

compare_features --names $(IFS=", "; shift; echo "${samples[*]}") ${abunds[@]} > $
{project}.feature_coverage.csv
```

Pathways reconstruction

Predict the biological pathways that may be present in the community:

Annotate pathway maps using color to indicate presence/absence of pathway components: