Oral Metapangenomics

Daniel Utter

13 January 2020

Contents

Overview

This is the long-form, narrative version of the metapangenomic methods used for our study **Metapangenomics**. Our goal in writing this is to make the workflow be as transparent and reducible as possible, and also to provide a step-by-step workflow for interested readers to adapt our methods to their own needs. Most of the code here is based off of Tom Delmont and Meren's *Prochlorococcus* metapangenome workflow, and we are deeply indebted to their project and their dedication to sharing a reproducible workflows.

Our project investiated the environmental representation of the various bacterial pangenomes to learn more about the ecology and evolution of microbial biogeography. We were particularly interested in identifying and understanding the population structure of bacteria within the oral cavity, and generating as short as possible lists of candidate genes or functions that may be involved in the observed population structure, using the healthy human mouth as a model system. The mouth is ideally suited to such studies of habit adaptation, as it consists of many distinct sites spatially connected by saliva but each with characteristically distinct community assemblages. We focused on three oral sites (hereafter 'habitats' to avoid ambiguity) - tongue dorsum (TD), buccal mucosa (BM), and supragingival plaque (SUPP). Many resident oral bacterial taxa have many high-quality genomes from cultured isolates, so we collected previously published genomes and combined this information with metagenomes from the Human Microbiome Project to address our goals.

This workflow shows the specific code used for *Haemophilus parainfluenzae*, and changing a few variable names will reproduce the *Rothia* analyses as well.

Organization & Workflow

Each oral habitat (tongue dorsum, TD; buccal mucosa, BM; supragingival plaque, SUPP) was processed independently in parallel. We used batch scripts and job arrays for this. So each oral habitat referred to the same annotated contigs database, to which a different set of metagenomes were mapped and downstream files were kept distinct by appending e.g. `-bm` identifiers to file names. This way, everything happens in the same directory but with minimal duplication of scripts. Ultimately, three independent pangenomes were created, one from each habitat, and then exported to be overlaid onto a single pangenome figure. If this is not clear, it will be later, hopefully.

Side note: We ran all of the anvi'o workflows on Harvard University's Odyssey3 cluster, which uses the Slurm scheduler. We have kept the Slurm header information specifying the cluster parameters requested, since this includes the parallelization information. Also this gives some context for memory and time requirements for replicability.

Step 1 - Collecting and annotating genomes

Getting the genomes from NCBI

Genomes were programmatically downloaded from NCBI RefSeq using the files found at ftp://ftp.ncbi.nlm.nih.gov/genomes/ASSEMBLY_REPORTS/ as of 24 July, 2018. Note that the GenBank accession list should contain all the RefSeq genomes but also additional genomes not thought to be of reference quality by NCBI, and so probably should bear a little more scrutiny in your study (as we did here).

Here is the ad hoc downloader script we wrote to programmatically download these. The NCBI assembly sheet, taxon to search for, and directory into which the genomes are downloaded get specified in the top 3 lines. Please note that this script renames the contig deflines to be A) simple and anvi'o compatible and B) contain the assembly sheet's genus, species, and strain identifier, so that later the originally assigned taxonomy can be recovered. Since these genomes were originally downloaded, anvi'o now has a more elegant workflow for downloading NCBI genomes; we include this code not as a recommendation of our ad hoc method over anvi'o's method, but simply to detail how it was done

```
#!/bin/bash

assemblyFile="assembly_summary_refseq.txt"
taxa="Haemophilus_parainfluenzae"
taxDir="Haemophilus_parainfluenzae"

mkdir ncbi_genomes
mkdir ncbi_genomes/$taxDir

for taxon in $taxa; do

# make sure whitespaces or lack thereof are ok
taxonSafe=$(echo $taxon | tr ' ' '_')
taxonGrep=$(echo $taxon | tr '_' ' ')

# assembly directory on the ncbi ftp site is the 20th column in assembly file
ftpDir=$(grep "$taxonGrep" $assemblyFile | awk -F"\t" '{print $20}')

cd ncbi_genomes/$taxDir

for dirPath in $ftpDir; do

# get just the ID
genomeID=$(echo "$dirPath" | sed 's/^.*\///g')
```

```
# get the path to the genomic fasta file
ftpPath="$dirPath/${genomeID}_genomic.fna.gz"
# make a friendly prefix - 'Gspe' from 'Genus species'
friendlyName=$(grep "$genomeID" ../../$assemblyFile | awk -F"\t" '{print $8}' | awk '{print $1 FS $2}'


# figure out the taxon + strain id from sheet to make deflines; 9 = strain,10=additional info
# this makes it simple to go from anvio contigs db to genome's original strain ID on NCBI
deflineName=$(grep "$genomeID" ../../$assemblyFile | awk -F"\t" '{print $8 " " $9}' | sed 's/strain=//'

genomeID="${friendlyName}_$genomeID"

# download & uncompress
wget $ftpPath -O $genomeID.fa.gz
gunzip $genomeID.fa.gz

# rename the deflines
awk -v defline="$deflineName" '/^>/{print ">" defline "_ctg" (++i)}!/^>/' $genomeID.fa >> temp.txt
mv temp.txt $genomeID.fa

done

done
```

Anvi'o contigs database

Now it is time to concatenate the downloaded raw FASTAs into a single FASTA to start the anvi'o workflow. For us, the code looked like this:

```
cat ncbi_genomes/Haemophilus_parainfluenzae/*fa > Haemophilus-isolates.fa
```

Whole thing

From here, an anvi'o contigs database is then generated, from which we invoked anvi'o's HMM profiler, launched annotation, and made a bowtie2 reference database out of the original FASTA. As part of the `anvi-gen-contigs-db` command, Prodigal was run to call open reading frames (ORFs). We did this with the following script (we called it `01_genContigsDB.sh`). Note that this script submits two other scripts - specifically, `99_geneFunctions.sh` which runs the annotation script, and `02_btmapHMP.sh` which starts the next step.

```
#!/bin/bash
#SBATCH -N 1 #1 node
#SBATCH -n 6 # 1 cores from each
#SBATCH --contiguous
#SBATCH --mem=12G #per node
#SBATCH -t 0-12:00:00
#SBATCH -p shared
#SBATCH --job-name="HaemCont"
#SBATCH -o odyssey_cont.out
#SBATCH -e odyssey_cont.err
#SBATCH --mail-type=END

genus="Haemophilus"

# need to activate venv directly because node receiving job doesn't like bashrc aliases
```

```
source ~/virtual-envs/anvio-dev-venv/bin/activate

# clean up fasta deflines in contig file at the start for smooth downstream
anvi-script-reformat-fasta -o $genus-isolates-CLEAN.fa --simplify-names -r $genus-isolates-contigIDs.txt

mv $genus-isolates-CLEAN.fa $genus-isolates.fa

# make the contig db
anvi-gen-contigs-database -f $genus-isolates.fa -n $genus -o $genus-isolates-CONTIGS.db

# get bacterial single-copy gene, 16S, info from hmm collections
anvi-run-hmms -c $genus-isolates-CONTIGS.db --num-threads 6

# get the functional annotation started
sbatch 99_geneFunctions.sh

# make bt of contigs for mapping later
module load samtools/1.5-fasrc01 bowtie2/2.3.2-fasrc01
bowtie2-build $genus-isolates.fa $genus-isolates

# start next step
sbatch 02_btmapHMP.sh
```

Functional annotation

We used Interproscan to get functional annotation from a variety of databases. The core command consists of

`sh PATH/TO/INTERPRO/interproscan-5.36-75.0/interproscan.sh -i genes.faa -o genes.tsv -f tsv --appl TIGR`

However, for cases like *Rothia* with 69 genomes' worth of genes, a non-parallel workflow took too long, so we made a simple parallelization. Here is the annotation script we used as `99_geneFunctions.sh`:

```
#!/bin/bash

prefix="Haemophilus-isolates"

# activate anvio if not already
source ~/virtual-envs/anvio-dev-venv/bin/activate

# export amino acid sequences
anvi-get-sequences-for-gene-calls --get-aa-sequences --wrap 0 -c $prefix-CONTIGS.db -o $prefix-gene-call

batchDir="$prefix-gene-calls-batches"
mkdir $batchDir

# chop file up into 25k sequences per file
split -l 50000 -d -a 4 $prefix-gene-calls-aa.faa $batchDir/$prefix-gene-calls-aa.faa-

# figure out how many batches we made
hiBatch=$(ls $batchDir/*.faa-* | tail -1 | sed 's/^.*-0*\([1-9]*\)\(.$\)/\1\2/')
numBatches=$(($hiBatch + 1))

# set up the file to concatenate into
echo -e "gene_callers_id\tsource\taccession\tfunction\te_value" > interpro-results-fmt-$prefix.tsv
echo "#!/bin/bash
#SBATCH -N 1 #1 nodes of ram
#SBATCH -n 12 # 12 cores from each
#SBATCH --contiguous
#SBATCH --mem=12G #per node
#SBATCH -t 0-6:00:00
#SBATCH -p shared
#SBATCH --array=0-$hiBatch%$numBatches
#SBATCH --job-name='interpro-%a'
#SBATCH -o odyssey_anviIP_%a.out
#SBATCH -e odyssey_anviIP_%a.err
#SBATCH --mail-type=END

# format job array id to match the split output
taskNum=$(printf %04d $SLURM_ARRAY_TASK_ID)

batch=$batchDir/$prefix-gene-calls.faa-$taskNum

module load jdk/1.8.0_172-fasrc01

./../../interproscan-sh/interproscan-5.36-75.0/interproscan.sh -i $batch -o ${batch}.tsv -f tsv --appl 

# format it manually to be safe
cat ${batch}.tsv | awk -F\"\t\" '{print $1 FS $4 FS $5 FS $6 FS $9}' >> interpro-results-fmt-$prefix.ts

" > ip-$batch.sh && sbatch ip-$batch.sh && rm ip-$batch.sh
```

We invoked `01_genContigsDB.sh`, which read the contig FASTAs into an anvi'o contigs DB and then called the functional annotation script, which wrote the file `interpro-results-fmt-Haemophilus-isolates.tsv` containing the annotation info. After everything here finished, we incorporated the functional annotation into the contigs db with the command

```
anvi-import-functions -c Haemophilus-isolates-CONTIGS.db -i interpro-results-fmt-Haemophilus-isolates.t
```

The resultant contigs database can be found in this FigShare dataset

Step 2 - Mapping

Data acquisition - HMP Metagenomes

Raw short-read metagenomic data from the Human Microbiome Project (HMP) (HMP, 2012; Lloyd-Price et al., 2017) was downloaded for the tongue dorsum (TD, n = 188), buccal mucosa (BM, n = 169), and supragingival plaque (SUPP, n = 194) sites using HMP data portal at https://portal.hmpdacc.org/. Paths for each sample were downloaded via the HMP portal and uploaded to Odyssey3, from which we ran an ad-hoc script to download all of them. We used the S3 method as it was the most reliable. Here is the script we used for downloading the buccal mucosa metagenomes:

```
#!/bin/bash
#SBATCH -N 1 #1 nodes of ram
#SBATCH -n 1 # 1 cores from each
#SBATCH --contiguous
#SBATCH --mem=2G #per node
#SBATCH -t 0-6:00:00
#SBATCH -p shared
#SBATCH --array=0-33%34
#SBATCH --job-name="dlHMP"
```

```
#SBATCH -o odyssey_downloadHuman-%a.out
#SBATCH -e odyssey_downloadHuman-%a.err
#SBATCH --mail-type=NONE

# cat buccal_manifest.txt | split -l 6 -d -a 3 - hmpBatch_

# format to match what we made
taskNum=$(printf %03d $SLURM_ARRAY_TASK_ID)

batch="hmpBatch_$taskNum"

# activate anvio
source ~/virtual-envs/anvio-dev-venv/bin/activate

#loop through each sample in this batch
for hmp in $(sed 's/^.*s3/s3/; s/bz2.*$/bz2/' $batch); do

sh -c 'aws s3 cp "$0" .' $hmp
tar -xjf $(echo "$hmp" | sed 's,^.*/,,g')

done
```

Here, the `buccal_manifest.txt` file was the saved text file of the HMP DACC cart for BM metagenomes, after removing the header line. Prior to running this script, we ran the commented-out line `cat buccal_manifest.txt | split -l 5 -d -a 3 - hmpBatch_` as a oneliner at the command prompt to split into manifest into 5-metagenome batches that the array would then process, and we manually changed the array parameters `--array=0-33%34` to match the number of batches this made.

This same process was performed for tongue dorsum (TD), buccal mucosa (BM), and supragingival plaque (SUPP), each of which in its own directory, e.g., `hmp_all_bm/`.

After downloading and uncompressing, the R1 and R2 reads were in a subdirectory for each metagenome, so we moved the R1/R2 pairs out of these subdirectories leaving the junk singleton reads behind (e.g. `mv */*.1.fastq .`), then deleted the downloaded *bz2 files and the directories.

Recruiting HMP metagenomes to the contigs

After downloading the metagenomes as described above, we competitively recruited the HMP metagenomes onto the exact contigs included in the contigs database using bowtie2 (Langmead & Salzberg, 2012) with default parameters (--sensitive). Each oral habitat (TD, BM, SUPP) were mapped onto separate but identical contigs databases. We did this with one script that made a separate job array for each habitat. Here is our code for tongue dorsum metagenome recruitment:

```
#!/bin/bash

# 20171016 - assign gene fams by interproscan
prefix="Haemophilus-isolates"

# this sets up the site suffixe to iterate through in parallel
sites=(bm td supp)

# iterate through each habitat - this will create a custom script for each habitat that is an array ove
for site in ${sites[*]}; do

mkdir bt_mapped_${prefix}_$site
mkdir batches_$site
```

```
# make batches for each sites samples
ls --color=none hmp_all_$site/*1.fastq | split -l 8 -d -a 3 - batches_$site/$site-

# figure out the upper bound of the array for slurm
hiBatch=$(ls batches_$site/$site-* | tail -1 | sed 's/^.*-0*\([1-9]*\)\(.$\)/\1\2/')
numBatches=$(($hiBatch + 1))

echo "#!/bin/bash
#SBATCH -N 1 #1 nodes of ram
#SBATCH -n 12 # 1 cores from each
#SBATCH --contiguous
#SBATCH --mem=18G #per node
#SBATCH -t 0-06:00:00
#SBATCH -p shared
#SBATCH --array=0-$hiBatch%$numBatches
#SBATCH --job-name='bt-$site'
#SBATCH -o odyssey_bt-$site-%a.out
#SBATCH -e odyssey_bt-$site-%a.err
#SBATCH --mail-type=NONE

# format this array element id to match batch name
taskNum=\$(printf %03d \$SLURM_ARRAY_TASK_ID)

module load samtools/1.5-fasrc02 bowtie2/2.3.2-fasrc02 xz/5.2.2-fasrc01

# set what batch we are on
batch=\"batches_$site/$site-\$taskNum\"

# loop through metagenomes to map in this batch
for FQ in \$(cat \$batch); do

# FQ was the R1 path so extract the R2 path from it
r2=\$(echo \"\$FQ\" | sed 's/.1.fastq/.2.fastq/')

bowtie2 -x $prefix -1 \$FQ -2 \$r2 --no-unal --threads 12 | samtools view -b - | samtools sort -@ 12 - 

# index as next anvio step requires indexed bams
samtools index \$(echo \"\$FQ\" | sed 's/hmp_all/bt_mapped_$prefix/; s/\.1.fastq.*$/.bam/')

done
" > bt-$site-array.sh && sbatch bt-$site-array.sh && rm bt-$site-array.sh # save this file and submit i

sleep 30

done
```

This script is `02_btmapHMP.sh` and was run directly from `01_genContigsDB.sh` (Step 1)

A short statement on competitive mapping and its interpretation

Each sample's short reads were mapped against all genomes for that taxon; thus, the bowtie2 matched each read to the best-matching genomic locus, randomly choosing between multiple loci if they were equally best. Thus, coverage at highly conserved regions is affected by the total population abundance in that sample, while the coverage at variable loci reflects that particular sequence variant's abundance.

Regions of identical nucleotide similarity between reference genomes, relative to the metagenome, then recruit an equivalent number of reads, reflecting their proportion of the total population abundance in that sample. However, regions with polymorphic sites allow for discrimination between genomes, based the variability of sequences in the pangenome / database.

Step 3 - Profiling the metagenome recruitment

After mapping, we profiled each sample's coverage into an anvi'o profile database and then merged them all. We wrote one master script to rule them all, with the same overall architecture as in Step 2.

Individual sample profiling code:

```
#!/bin/bash

# 20171016 - assign gene fams by interproscan
prefix="Haemophilus-isolates"

# need to activate venv directly because node receiving job order doesn't play with bashrc aliases

# habitats to iterate over
sites=(bm td supp)

# for each habitat we generate a custom job array
for site in ${sites[*]}; do

mkdir profs_mapped_${prefix}_$site

# make batches for each sites samples but leaving commented off as the bowtie2 batches work here too
#ls --color=none hmp_all_$site/*1.fastq | split -l 8 -d -a 3 - batches_$site/$site-

# find high number of batches we made before
hiBatch=$(ls batches_$site/$site-* | tail -1 | sed 's/^.*-0*\([1-9]*\)\(.$\)/\1\2/')
numBatches=$(($hiBatch + 1))

echo "#!/bin/bash
#SBATCH -N 1 #1 nodes of ram
#SBATCH -n 12 # 1 cores from each
#SBATCH --contiguous
#SBATCH --mem=18G #per node
#SBATCH -t 0-08:00:00
#SBATCH -p shared
#SBATCH --array=0-$hiBatch%$numBatches
#SBATCH --job-name='pro-$site'
#SBATCH -o odyssey_prof-$site-%a.out
#SBATCH -e odyssey_prof-$site-%a.err
#SBATCH --mail-type=NONE

taskNum=\$(printf %03d \$SLURM_ARRAY_TASK_ID)

source ~/virtual-envs/anvio-dev-venv/bin/activate

batch=\"batches_$site/$site-\$taskNum\"

# for each fastq that was mapped
for FQ in \$(cat \$batch); do
```

```
# recreate the name of the bam from the name of the fastq
bam=\$(echo \"\$FQ\" | sed 's/hmp_all/bt_mapped_$prefix/; s/\.1.fastq.*$/.bam/')

# generate a profile
anvi-profile -i \$bam -c $prefix-CONTIGS.db -W -M 0 -T 12 --write-buffer-size 500 -o \$(echo \"\$bam\"

done
" > prof-$site.sh && sbatch prof-$site.sh && rm prof-$site.sh

sleep 25
done
```

Note that it if reproducing this analysis, retaining the -M flag is highly important. From the anvi'o documentation, `-M` parameter is the

> Minimum length of contigs in a BAM file to analyze . . . we chose the default to be 2500 [nt]

Since a few genomes we encountered contained contigs with fewer than 2500 nt, we elected to maintain all contigs, rather than dropping contigs or genomes. anvi'o sensibly set this default to 2.5 kb for the purpose of meaningful tetranucleotide frequency comparisons for bin refinement; since we are not refining bins but rather working off of existing genomes from nominally axenic isolates, we feel our choice is justified.

Another important factor is that we used the same contigs db for all the profiles. In other words, we made one *-CONTIGS.db at the start, and then mapped hundreds of metagenomes onto it, and now have made over a hundread each of TD, BM, and SUPP metagenome profiles for this one set of contigs.

Step 4 - Pangenome and metapangenome construction

This step incoporation of multiple related substeps. The overall workflow is to merge each habitat's profiles into a single merged profile database, then store the information about which contigs belong to which genomes since that hasn't been done yet. Then, now that we have the information stored about which contigs and genes came from which genome, we can make a pangenome. Once we have this, since we have the 3 profile databases that contain the information about coverage, SNPs, etc in each metagenome sample from each site corresponding to the genes in our CONTIG.db, we can overlay that metagenomic data onto the pangenome to have a metapangenome. This was all accomplished in one script, like so:

```
#!/bin/bash

# 20171016 - assign gene fams by interproscan
prefix="Haemophilus-isolates"

# make a two-column table of which contigs go with which genomes for later, easy since original contigs
sed 's/_ctg[0-9]*$//' $prefix-contigIDs.txt > $prefix-renaming-manual.tsv

# list of habitats to iterate over
sites=(td bm supp)
mcl=10

for site in ${sites[*]}; do

echo "#!/bin/bash
#SBATCH -N 1 #1 node
#SBATCH -n 12 # 12 cores from each
#SBATCH --contiguous
#SBATCH --mem=120G #per node
```

```
#SBATCH -t 1-18:00:00
#SBATCH -p shared
#SBATCH --job-name=\"fin-$site-$prefix\"
#SBATCH -o odyssey_metapan-$site-$prefix.out
#SBATCH -e odyssey_metapan-$site-$prefix.err
#SBATCH --mail-type=END

# activate anvio and load necessary modules
source ~/virtual-envs/anvio-dev-venv/bin/activate
module load mcl/14.137-fasrc01 ncbi-blast/2.6.0+-fasrc01

# merge the individual profiles into a single profile database
anvi-merge profs_mapped_${prefix}_$site/*/PROFILE.db -c $prefix-CONTIGS.db -o $prefix-$site-MERGED -W

# import the data that matches contigs to their genomes
anvi-import-collection -c $prefix-CONTIGS.db -p $prefix-$site-MERGED/PROFILE.db -C Genomes --contigs-mod

# summarize the mapping data so we have all the information nicely for next steps
anvi-summarize -c $prefix-CONTIGS.db -p $prefix-$site-MERGED/PROFILE.db -C Genomes -o $prefix-$site-SUMM

# run custom script to generate a table telling anvio where to find all the information for each genome
./anvi-script-gen-internal-genomes-table.sh $prefix-$site

# correct the output as we share one CONTIGS db with three profiles
sed \"s/$prefix-$site-CONTIGS.db/$prefix-CONTIGS.db/g\" $prefix-$site-internal-genomes-table.txt > tmp
mv tmp $prefix-$site-internal-genomes-table.txt

# pangenome is two steps - store all the gene info, then calculate the pangenome
anvi-gen-genomes-storage -i $prefix-$site-internal-genomes-table.txt -o $prefix-$site-GENOMES.db
anvi-pan-genome --mcl-inflation $mcl -o $prefix-$site-PAN -g $prefix-$site-GENOMES.db -n $prefix-$site -

# then summarize the the metagenomic data on top of pangenome
anvi-meta-pan-genome -i $prefix-$site-internal-genomes-table.txt -g $prefix-$site-GENOMES.db -p $prefix-
" > finish-$site.sh && sbatch finish-$site.sh && rm finish-$site.sh

sleep 30

done
```

At this point, for our *H. parainfluenzae* specific case shown here, we have one directory with a single contigs db called `Haemophilus-isolates-CONTIGS.db` and three merged profile databases like `Haemophilus-isolates-bm-MERGED/PROFILE.db` with coverage and variant data from all the metagenomes for that habitat, and three genome databases like `Haemophilus-isolates-bm-GENOMES.db` and three pangenome databases like `Haemophilus-isolates-bm-10-CONTIGS.db` where the number is the mcl inflation factor used.

The `*GENOMES.db` and `*PAN` databases can be found in this FigShare dataset.

Some background on the methods wrapped by the anvio programs

The pangenome was calculated with the anvi-pan-genome command. We wanted to compare homologous genes between all genomes (here, *H. parainfluenzae*), so we first need to cluster the observed genes (technically, ORFs) into homologous units – what anvi'o and we refer to as **gene clusters**. For more background, please see the step-by-step anvi'o documentation and explanation here. BLASTP (Altschul et al., 1990) computed amino acid-level similarities between all possible ORF pairs after alignment with MUSCLE (Edgar, 2004).

Too-weak matches were culled by employing the –minbit criterion with the default value of 0.5. The Markov Cluster Algorithm (MCL) (van Dongen & Abreu-Goodger, 2012) then used these pairwise identities to group ORFs into gene clusters, putatively homologous gene groups. MCL uses a hyperparameter (inflation, `--mcl-inflation`) to adjust the clustering sensitivity, i.e., the tendency to split clusters. For the nominally single-species *H. parainfluenzae* pangenome we used `--mcl-inflation 10`; for all other pangenomes which were genus-level, we used `--mcl-inflation 6`.

The `anvi-pan-genome` family of commands requires a specially-formatted table to tell anvi'o where to look. This is the helper script `anvi-script-gen-internal-genomes-table.sh` we wrote and reference in the above script to collect all this info:

```
#!/bin/bash

taxonPrefix=$1
collection="Genomes"

echo -e "name\tbin_id\tcollection_id\tprofile_db_path\tcontigs_db_path" > $taxonPrefix-internal-genomes-

for bin in $(ls $taxonPrefix-SUMMARY/bin_by_bin); do

echo -e "$bin\t$bin\t$collection\t$taxonPrefix-MERGED/PROFILE.db\t$taxonPrefix-CONTIGS.db" >> $taxonPre

done
```

This wrote the needed info as `*-internal-genomes-table.txt` that was used for the pangenome steps.

Conceptual assumptions and rationale of metapangenome details

Note that the `--min-detection` parameter in `anvi-meta-pan-genome` is left at the default value of `0.5`. Thus, if half or more of the total nucleotides in a genome cannot be covered by at least one metagenome for that habitat, the program skips trying to calculate anything and returns a value of `NA`. This protects us from cases where we could be looking at curious coverage disparities between a gene and its genome, but it is simply a result of the fact that the genome is not represented in this habitat.

Here, the `--fraction-of-median-coverage 0.25` parameter means that we set the threshold between '**environmental core**' or '**environmental accessory**' for each gene's median coverage across a habitat's metagenomes to be 0.25 of the genome's corresponding coverage across those same samples. For instance, *H. parainfluenzae* T3T1 obtained a median coverage of 12X across the 188 TD metagenomes. Or to look at it a different way, if the median TD metagenome covered T3T1 12X, then if a given T3T1 gene had a median coverage of 1.2X from those same 188 metagenomes, it would be considered environmental accessory in TD. However, if a different T3T1 gene had a median coverage of 4.8X or even 27.6X, then that gene would be environmental core in TD.

By thresholding relative to genomic coverage we can scale our expectation for finding a given gene in a habitat based on the genome. We do this so that when our metric reports that a gene is 'environmental accessory' in a habitat, we know that we're not just seeing every gene from a low-abundance genome but rather a gene that is much less covered in its habitat than in the environment. So, this determination of environmental core/accessory is a proxy for whether a gene sequence is at a notably lower frequency in the environment than we presume its background genome to be (which could be a result of selection).

Relating to the genome's median coverage also helps even out noise in gene-level coverages from non-specific mapping (falsely inflating coverage) or from having highly similar DNA sequences in the bowtie2 reference database competing for the same metagenome reads (lowering coverage).

Combining metapangenomes

We then combined each habitat's metagenome's environmental core/accessory layer onto a single pangenome figure so we could directly compare the environmental representation of each homologous gene across sites.

All of each pangenome's misc data were exported using `anvi-export-misc-data` parameter, e.g.

```
#!/bin/bash

prefix="Haemophilus-isolates"
readsStart=11  # which column in anvio layers table is the start of the metagenomes

sites=(td bm supp)  # important that td is first if that will be the one onto which others are added

for site in ${sites[*]}; do

anvi-export-misc-data -p $prefix-$site-PAN/$prefix-$site-PAN.db -t items -o $prefix-$site-items.txt
anvi-export-misc-data -p $prefix-$site-PAN/$prefix-$site-PAN.db -t layers -o $prefix-$site-layers.txt

# delete the unprocessed columns from TD so don't end up with two sets of TD metagenomes (formatted + u
anvi-delete-misc-data -p $prefix-td-PAN/$prefix-td-PAN.db -t layers --keys-to-remove $(head -1 $prefix-$

# select metapan rings from items data and add oral habitat as prefix
awk -F"\t" -v site=$site 'BEGIN{site=toupper(site)}; NR==1{print $1 FS site"-"$9 FS site"-"$10 FS site"-
# prepend capitalized site ID before each metagenome's coverage of the genomes
cut -d' ' -f1,$readsStart- $prefix-$site-layers.txt | sed "s/SRS/\U$site-SRS/g" > $prefix-$site-layers.t

# trim off junk from SRS filenames in header
sed -i"" 's/_DENOVO_DUPLICATES_MARKED_TRIMMED//g' $prefix-$site-layers.tmp

# import this data back into the td data
anvi-import-misc-data -p $prefix-td-PAN/$prefix-td-PAN.db -t items $prefix-$site-items.tmp
anvi-import-misc-data -p $prefix-td-PAN/$prefix-td-PAN.db -t layers $prefix-$site-layers.tmp

done
```

For the `*layers.txt` file, which contains each genome's per-sample coverage information, we used a custom R script to calculate and export the median coverage for each genome for each site, as well as combine the per-sample coverages for a heatmap:

```
t <- read.csv("Haemophilus-isolates-td-layers.tmp", sep = "\t", header = T)
b <- read.csv("Haemophilus-isolates-bm-layers.tmp", sep = "\t", header = T)
p <- read.csv("Haemophilus-isolates-supp-layers.tmp", sep = "\t", header = T)

# save genome ids for later
genomeIDs <- t[,1]

# get just the coverages
t <- t[,grep("SR", colnames(t))]
b <- b[,grep("SR", colnames(b))]
p <- p[,grep("SR", colnames(p))]

# get the median for each row=genome
t <- apply(t, 1, median)
b <- apply(b, 1, median)
p <- apply(p, 1, median)

# save to import into anvio
write.table(cbind(layers=as.character(genomeIDs), TD_HMP_MEDIAN=t, BM_HMP_MEDIAN=b, SUPP_HMP_MEDIAN=p),
```

which makes a file that looks like:

```
##      layers                                  TD_HMP_MEDIAN
## [1,] "Haemophilus_parainfluenzae_1128_HPAR" "3.50155084141615"
## [2,] "Haemophilus_parainfluenzae_1209_HPAR" "6.815398696077"
## [3,] "Haemophilus_parainfluenzae_137_HINF"  "3.1613550551289"
## [4,] "Haemophilus_parainfluenzae_146_HPAR"  "15.1593914157683"
## [5,] "Haemophilus_parainfluenzae_155_HPAR"  "3.47668191817337"
## [6,] "Haemophilus_parainfluenzae_174_HPAR"  "3.35610746823065"
##      BM_HMP_MEDIAN      SUPP_HMP_MEDIAN
## [1,] "0.657201515386123" "1.06875757236494"
## [2,] "1.08419195652694"  "1.55167011239881"
## [3,] "0.565412049050462" "1.9723758712279"
## [4,] "1.06501435940931"  "1.88690992300501"
## [5,] "0.563448807228174" "0.794885888409524"
## [6,] "0.636198880422302" "2.31254813948498"
```

All these data were then imported into the TD pangenome database (TD chosen arbitrarily) using `anvi-import-misc-data -t layers`.

Anvi'o interactive display choices for metapangenomes

The genome layers in the pangenomes were ordered by gene cluster frequency, and the gene clusters were ordered by frequency in genomes.

Coloring and spacing were set manually through the anvio interactive interface `anvi-display-pan`

Supplemental tables

Supplemental tables of gene clusters and their functions (i.e., SD1, SD3) were generated by summarizing each combined metapangenome with `anvi-summarize`, like

```
anvi-summarize -g Haemophilus-isolates-GENOMES.db -p Haemophilus-isolates-td-PAN/*PAN.db
-C default -o Haemophilus-isolates-td-PAN-SUMMARY
```

Querying the gene cluster annotations for enriched functions

Pangenomes ultimately rely on the ability to cluster genes into 'gene clusters' based on detecting islands in amino-acid sequence space (described above). But, how should we interpret each gene cluster, biologically? In other words, do different gene clusters by and large represent distinct functions, or different sequence groups of the same functions? This has significant implications for how we proceed with interpreting core genomes, etc.

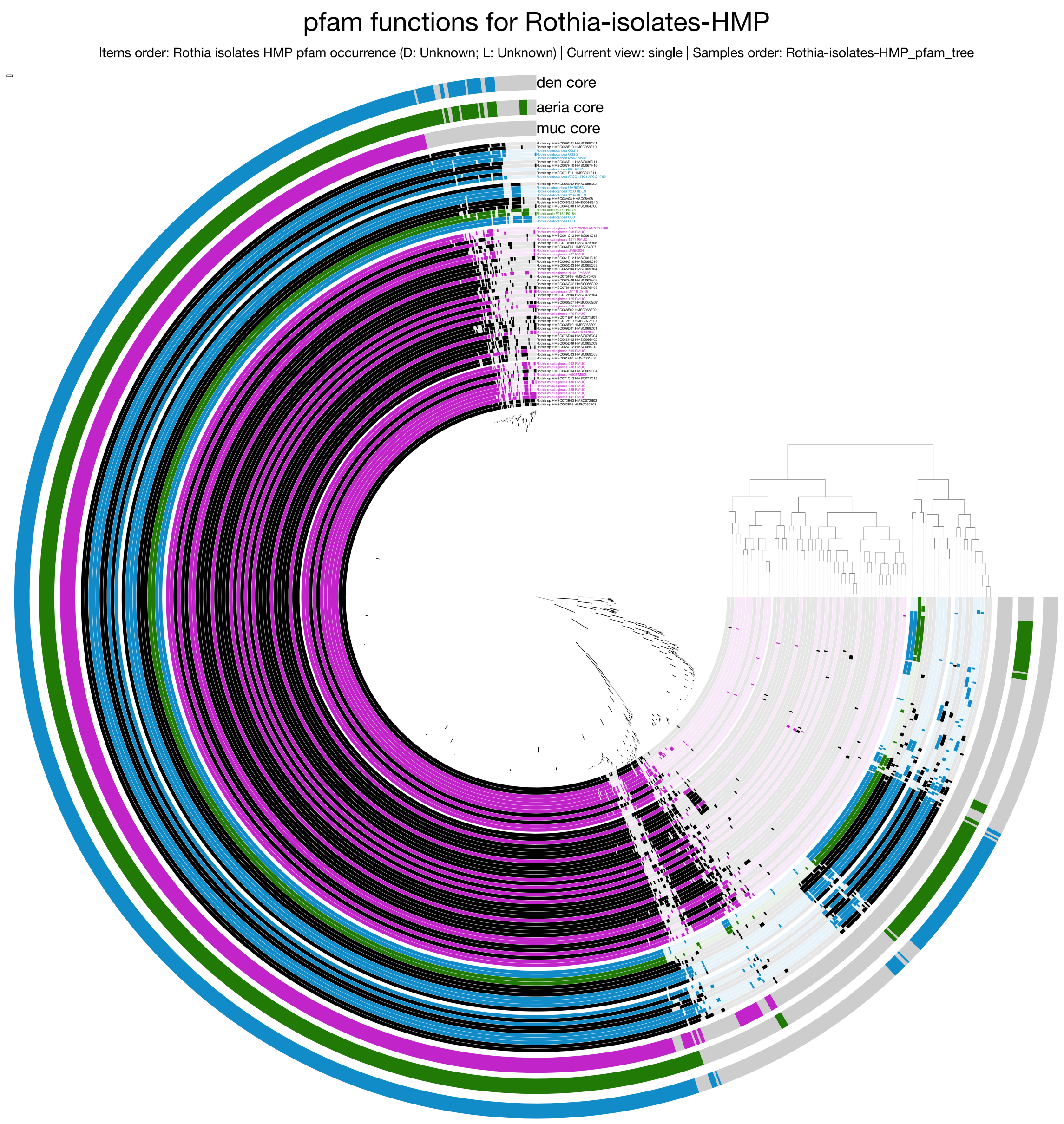
Before the methods, it is probably a good idea to talk about evolution. The fundamental problem of pangenomes is that each gene may be under a different selective regime. For instance, a ribosomal protein evolves under different constraints than a given transcriptional regulater than a membrane transporter. This is essentially why it is so important to select a representative gene(s) for building a phylogeny. An extreme example is the gene encoding the RuBisCO large subunit, all of which fix CO_2_ yet the sequences are divergent except for a tiny window of conservation around the active site. To be clear, such divergence exists on a much, much more broad evolutionary scale than either of the taxa investigated here. On the other hand, the 16S ribosomal RNA gene sequence is conserved across all of Bacteria. And, even a 'small' trunctation can totally abolish enzymatic function (e.g. Sirias et al. 2020 in submission). So when thinking about what evolutionary or functional relationships we want our gene clustering algorithm to recover, there is not a one-size-fits-all approach given the disparity in selective constraints within a genome.

To understand the functional implications of our gene clusters, we took two approaches: 1) Re-create the pangenome, but plot functions on the radial axis vs genomes, instead of gene clusters x genomes (most useful for Rothia). 2) Count up the number of gene clusters per function, and pay special attention to cases where one function is found in all genomes but split into species-specific gene clusters.

To re-create the pangenome, we applied Alon Schaiber's example code to our data. The core program involved is `anvi-get-enriched-functions-per-pan-group`.

Ultimately, this finds functions enriched in each group defined. Here, we defined groups as the three detected subgroups of H. parainfluenzae seen in the pangenome.

Following the steps in Alon Schaiber's guide, this pangenome for Rothia displaying functions was created.



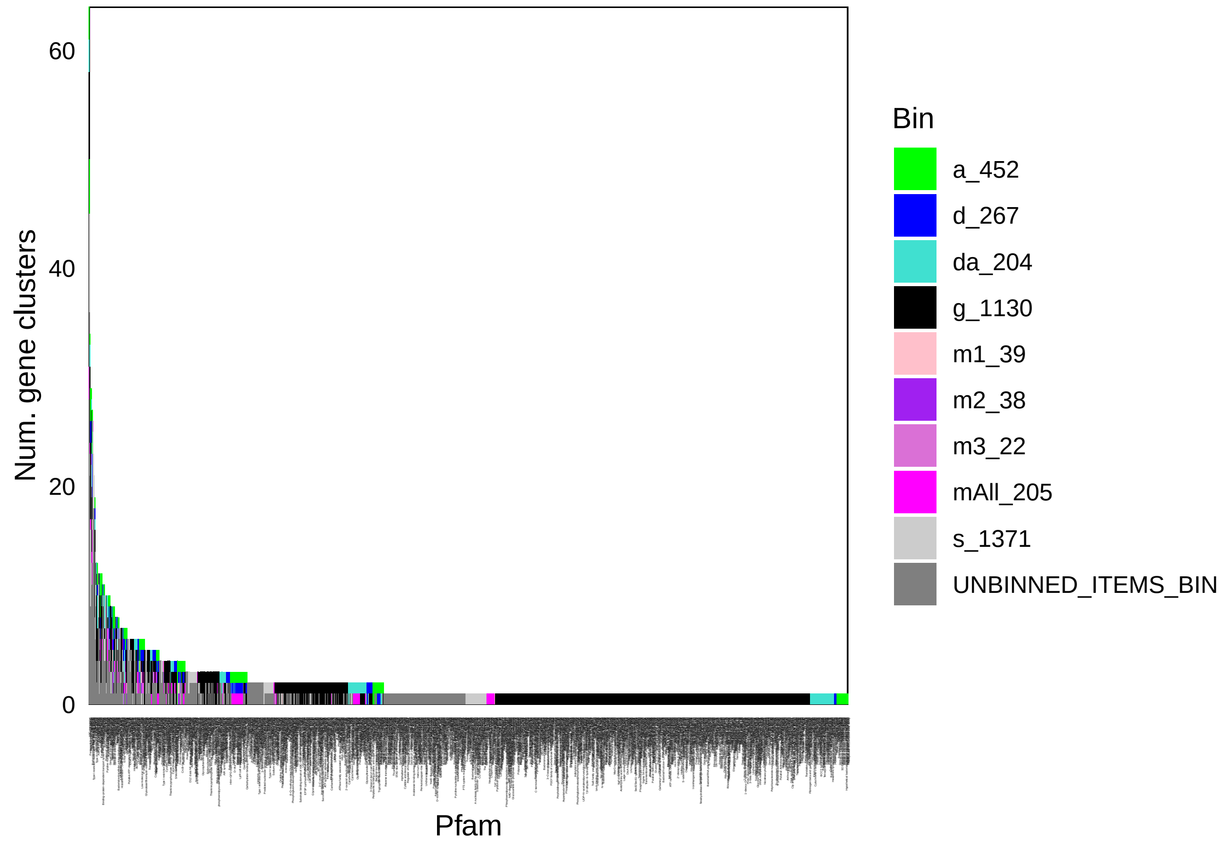
As expected, many of the singleton accessory genome drops off, and the relative representation of the core genome is much larger since core genes are typically more studied and thus presumably better annotated. But, the overall relationship patterns between genomes are visually similar.

To count up the functions by gene clusters, to identify places where the same function was split across multiple gene clusters (particularly if that split matched species groups, e.g., if anvi'o made three dnaK gene clusters, one for each species' dnaK), we branched off from this workflow. When we ran `anvi-summarize` on the pangenomes above, they produced files named like `Haemophilus-isolates-td-PAN-SUMMARY/Haemophilus-isolates-td_gene_clusters_summary.txt` that among other things report the annotation for each gene and which gene cluster to which that gene belongs. From this, we can count up how many times each function was found, by core or accessory set. Here is the Python we used to parse this file:

```
#!/usr/bin/env python3

import pandas as pd

summaryPath = 'Haemophilus-isolates-td-PAN-SUMMARY/Haemophilus-isolates-td_gene_clusters_summary.txt'

summary = pd.read_csv(summaryPath, sep="\t", dtype={'ProDom_ACC': str, 'ProDom': str})

summary = summary[['gene_cluster_id','bin_name','functional_homogeneity_index','geometric_homogeneity_i

summary = summary[pd.notnull(summary['Pfam'])]


funcCounts = []

for pfam in summary['Pfam'].unique():
    subPfam = summary.ix[summary['Pfam'] == pfam]
    for bin in subPfam['bin_name'].unique():
        subPfamsubBin = subPfam.ix[subPfam['bin_name'] == bin]
        gcIDsUnique = list(subPfamsubBin['gene_cluster_id'].unique())
        df = pd.DataFrame({'bin_name': [bin], 'Pfam': [pfam], 'num_uniq_gcs': [len(gcIDsUnique)],
                            'uniq_gc_ids': [','.join(gcIDsUnique)]})
        funcCounts.append(df)

funcCounts = pd.concat(funcCounts, axis=0)
funcCounts.to_csv(summaryPath + "-homogeneity.tsv", sep="\t", index=None)
```

This produced a slimmed-down table with only the relevant information, but it was still difficult to conceptualize all these data. We plotted this info, as well as turned it into a wide-format table, giving a breakdown, for each function, how many gene clusters existed, and what set of core genes to which they belonged (e.g., genus core, aeria + dentocariosa core, singleton accessory genome, etc.).

```
# plot pfam redundancy of GCs for rothia metapangenome

library(ggplot2)
library(reshape2)

funcCounts <- read.csv('Rothia-isolates-HMP_gene_clusters_summary.txt-homogeneity.tsv', sep = "\t")

funcCounts <- funcCounts[,!(colnames(funcCounts) %in% "X")]

coreCols <- setNames(c('green','blue','turquoise','black','pink','purple','orchid','magenta','grey80','

funcCounts$Pfam <- factor(funcCounts$Pfam, levels = unique(as.character(funcCounts$Pfam[order(funcCount
funcCounts$Pfam <- factor(funcCounts$Pfam, levels = names(rev(sort(tapply(funcCounts$num_uniq_gcs, func

ggplot(funcCounts, aes(x = Pfam, y = num_uniq_gcs)) +
    geom_bar(stat = 'identity', position = 'stack', aes(fill = bin_name), width = 1) +
    theme_minimal() + scale_x_discrete(expand = c(0,0)) + scale_y_continuous(expand = c(0,0)) +
    scale_fill_manual(values = coreCols) + labs(y = "Num. gene clusters", fill = 'Bin') +
    theme(panel.grid = element_blank(), panel.background = element_rect(fill = NA, color = 'black'), ax
          axis.text.x = element_text(angle = 90, hjust = 1, size = 1))
```



And here is what the top of the table looks like:

```
funcTable <- dcast(funcCounts, Pfam ~ bin_name, value.var = 'num_uniq_gcs')
funcTable[is.na(funcTable)] <- 0

head(funcTable, 30)
```

```
##                                                        Pfam a_452
## 1                                            ABC transporter     3
## 2                                Pentapeptide repeats (9 copies)     5
## 3                               Major Facilitator Superfamily     1
## 4                                                 AAA domain     3
## 5                                       Helix-turn-helix domain     1
## 6                       Bacterial regulatory proteins, tetR family     1
## 7                                       S-layer homology domain     4
## 8            Type I restriction modification DNA specificity domain     0
## 9                                                 RHS Repeat     0
## 10                                         N-6 DNA Methylase     0
## 11                               Glycosyl transferase family 2     1
## 12                                              AIPR protein     0
## 13                       Type III restriction enzyme, res subunit     0
## 14                                  Zinc-binding dehydrogenase     0
## 15                                              NUDIX domain     0
## 16                              Acetyltransferase (GNAT) family     2
## 17                                                   AAA ATPase domain     0
## 18                                                       DNA methylase     0
## 19                                                  AMP-binding enzyme     0
## 20                                            Methyltransferase domain     0
## 21                                    Excalibur calcium-binding domain     4
## 22                                       Glycosyl transferases group 1     2
## 23                                              Acyltransferase family     2
## 24 Binding-protein-dependent transport system inner membrane component     0
## 25                                                    Helix-turn-helix     0
## 26                                           alpha/beta hydrolase fold     0
## 27                                                      Fic/DOC family     1
## 28                                                         MarR family     1
## 29                                  Protein of unknown function DUF262     0
## 30                                                    HNH endonuclease     0
##    d_267 da_204 g_1130 m1_39 m2_38 m3_22 mAll_205 s_1371 UNBINNED_ITEMS_BIN
## 1      0      3     27     0     0     0        3      0                 28
## 2      0      0      0     0     0     0        0      9                 36
## 3      0      2      7     0     0     0        1      2                 21
## 4      2      0      7     0     0     0        1      7                  9
## 5      0      1      4     0     0     0        1      3                 19
## 6      2      4      6     0     0     0        1      2                 11
## 7      4      0      2     1     0     0        5      0                 11
## 8      0      0      0     0     0     1        0     12                 13
## 9      0      0      0     0     0     0        0      5                 18
## 10     0      0      0     0     0     0        0      3                 18
## 11     1      1      8     0     0     0        0      0                  8
## 12     0      0      0     0     0     0        0      8                  6
## 13     0      0      2     0     0     0        0      2                  9
## 14     0      3      6     0     0     0        0      0                  4
## 15     0      1     10     0     0     0        0      0                  2
## 16     1      3      3     0     0     0        0      0                  4
## 17     0      0      0     0     0     0        0      4                  8
## 18     0      0      1     0     0     0        0      3                  8
## 19     0      0      8     0     0     0        0      3                  1
## 20     0      2      3     0     0     0        0      2                  5
## 21     2      0      0     0     0     0        2      0                  4
## 22     0      0      8     0     0     0        0      0                  2
## 23     0      1      2     0     0     0        0      0                  7
## 24     0      0      5     0     0     0        1      0                  5
## 25     0      1      0     0     0     0        0      3                  7
## 26     1      1      2     0     0     0        0      1                  6
## 27     0      0      0     0     0     0        0      0                 10
## 28     0      3      1     0     0     0        0      2                  4
## 29     0      0      0     0     0     0        0      6                  4
## 30     0      0      0     0     0     0        0      3                  7
```

Granular analysis of mapping results at the per-gene, per-sample level

To get a better idea of what the per-gene coverages looked like along with the metapangenomes, we extracted the per-gene coverage information for each genome. This is what generated Figure 4A (so switching to the Rothia dataset for this example code). This was actually done in the background in `anvi-meta-pan-genome` and can be done ad-hoc with `anvi-interactive --gene-mode`, but we needed more control for formatting and miscellaneous data so we re-gathered the nucleotide-level coverage information per genome, and stored this information individually. We used the following script to collect and organize the data:

```
#!/bin/bash
#SBATCH -N 1 #1 nodes of ram
#SBATCH -n 1 # 1 cores from each
#SBATCH --contiguous
#SBATCH --mem=24G #per node
#SBATCH -t 0-5:00:00
#SBATCH -p shared
#SBATCH --array=0-68%69
#SBATCH --job-name="MP-$target"
#SBATCH -o odyssey_meta-$target.out
#SBATCH -e odyssey_meta-$target.err
#SBATCH --mail-type=NONE

taxPrefix="Rothia-isolates"

### RUN THESE 2 LINES FIRST
#mkdir genomesMapped
#awk '{print $2}' $taxPrefix-renaming-manual.tsv | uniq | split -l 1 -d -a 3 - genomesMapped/genome-

cd genomesMapped

taskNum=$(printf %03d $SLURM_ARRAY_TASK_ID)

# which genome this batch is targeting
target=$(cat genome-$taskNum)

# activate anvio
source ~/virtual-envs/anvio-dev-venv/bin/activate

## for each habitat, make a dir, enter, get this genome's coverage and detection info, then go back up
mkdir $target-TD
cd $target-TD
anvi-script-gen-distribution-of-genes-in-a-bin -c ../../$taxPrefix-CONTIGS.db -p ../../$taxPrefix-td-ME
cd ..

mkdir $target-PQ
cd $target-PQ
anvi-script-gen-distribution-of-genes-in-a-bin -c ../../$taxPrefix-CONTIGS.db -p ../../$taxPrefix-supp-
cd ..

mkdir $target-BM
cd $target-BM
anvi-script-gen-distribution-of-genes-in-a-bin -c ../../$taxPrefix-CONTIGS.db -p ../../$taxPrefix-bm-ME
cd ..
```

This script was run the top level directory, where all the other files have been made. Note that this is an array, so the two commented out lines (`mkdir` and `awk`) were run first as one-liners to set everything up. This script then creates a bunch of batch information inside that genomesMapped directory, which will also have 3 directories for each genome, one for each habitat. The `--fraction-of-median-coverage` parameter was kept at `0.25`, identical to the metapangenome calculation above.

This output of this script made separate directories for each genome for each site. Ideally, we would select the top N samples (top being determined by highest mean or median coverage) from each site for each genome and combine into a single figure, along with the environmental core/accessory designations for each gene from each habitat. So we wrote a helper python script to help with the merging:

```
#!/usr/bin/env python

from optparse import OptionParser
import pandas as pd
import os

parser = OptionParser()
(options, args) = parser.parse_args()

fileRoot=args[0]
keep=int(args[1])

# load coverages
covs_td = pd.read_csv("../"+args[0]+"-TD/"+args[0]+"-GENE-COVs.txt", sep ="\t", index_col=False)
covs_bm = pd.read_csv("../"+args[0]+"-BM/"+args[0]+"-GENE-COVs.txt", sep ="\t", index_col=False)
covs_supp = pd.read_csv("../"+args[0]+"-PQ/"+args[0]+"-GENE-COVs.txt", sep ="\t", index_col=False)

# load detections
det_td = pd.read_csv("../"+args[0]+"-TD/"+args[0]+"-ENV-DETECTION.txt", sep ="\t", index_col=False)
det_bm = pd.read_csv("../"+args[0]+"-BM/"+args[0]+"-ENV-DETECTION.txt", sep ="\t", index_col=False)
det_supp = pd.read_csv("../"+args[0]+"-PQ/"+args[0]+"-ENV-DETECTION.txt", sep ="\t", index_col=False)

det_td.columns.values[1] = 'TD_detection'
det_bm.columns.values[1] = 'BM_detection'
det_supp.columns.values[1] = 'SUPP_detection'

# sort by max median coverage
covs_td = covs_td.reindex_axis(covs_td.median().sort_values(ascending=False).index, axis=1)
covs_bm = covs_bm.reindex_axis(covs_bm.median().sort_values(ascending=False).index, axis=1)
covs_supp = covs_supp.reindex_axis(covs_supp.median().sort_values(ascending=False).index, axis=1)

# add in sites to samples
covs_td.columns.values[1:] = [x+"-TD" for x in covs_td.columns.values[1:]]
covs_bm.columns.values[1:] = [x+"-BM" for x in covs_bm.columns.values[1:]]
covs_supp.columns.values[1:] = [x+"-SUPP" for x in covs_supp.columns.values[1:]]

# make the new coverages
covs_keep = pd.concat([covs_td.iloc[:,0:(keep+1)], covs_bm.iloc[:,1:(keep+1)], covs_supp.iloc[:,1:(keep
det_all = pd.concat([det_td, det_bm.iloc[:,1], det_supp.iloc[:,1]], axis=1)

# save it
covs_keep.to_csv(args[0]+"-GENE-COVs.txt-COMBO-"+str(keep), sep="\t", index=False)
det_all.to_csv(args[0]+"-ENV-DETECTION.txt-COMBO-"+str(keep), sep="\t", index=False)

keys = sorted(covs_keep['key'].tolist(), reverse=True)
with open("synteny.txt", 'w') as f:
        for k in keys:
                f.write(str(k)+'\n')


with open("RUN_ME_FOR_ANVIO", "w") as f:
        f.write("anvi-interactive -P 8081 --manual --title '{0}' -d {0}-GENE-COVs.txt-COMBO-{1} -A {0}-
os.chmod("RUN_ME_FOR_ANVIO", 0o755)
```

We had this script as `indivMetaCombiner.py` in the appropriate directory and ran it like so `./indivMetaCombiner.py Rothia_sp_HMSC061E04_HMSC061E04 30`, where the first argument is the file prefix for the earlier environmental detection output, which also happens to be the name of the genome, and the second argument is the number of samples to include (N), taking the N with the highest median coverage across samples for that habitat. We chose median, since mean could be biased by one outlier gene (e.g. 16S) picking up reads, and are not interested in high coverage by that reason.

Differential function analysis

To investigate potential functional drivers affecting the different tropisms evidenced by *H. parainfluenzae* strains, we assigned each genome into the three genomic groups observed from the metapangenome (Figure 2 in text) and looked for functions enriched in a particular group relative to the other groups.

Here is the file showing each genome's assignments:

```
layer	habitat
Haemophilus_parainfluenzae_C2004002727	Group3
Haemophilus_parainfluenzae_ATCC_9796	Group3
Haemophilus_parainfluenzae_C2005004058	Group3
Haemophilus_parainfluenzae_UMB0748	Group3
Haemophilus_parainfluenzae_C2006002596	Group3
Haemophilus_parainfluenzae_CCUG_58848	Group3
Haemophilus_parainfluenzae_488_HPAR	Group3
Haemophilus_parainfluenzae_ATCC_33392_v1	Group3
Haemophilus_parainfluenzae_ATCC_33392_v2	Group3
Haemophilus_parainfluenzae_174_HPAR	Group3
Haemophilus_parainfluenzae_137_HINF	Group3
Haemophilus_parainfluenzae_HK2019_HK2019	Group3
Haemophilus_parainfluenzae_HK262_HK262	Group3
Haemophilus_parainfluenzae_CCUG_62654	Group3
Haemophilus_parainfluenzae_CCUG_62655	Group3
Haemophilus_parainfluenzae_C2004000280	Group2
Haemophilus_parainfluenzae_C2004002729	Group2
Haemophilus_parainfluenzae_901_HPAR	Group2
Haemophilus_parainfluenzae_432_HPAR	Group2
Haemophilus_parainfluenzae_146_HPAR	Group2
Haemophilus_parainfluenzae_209_HPAR	Group2
Haemophilus_parainfluenzae_T3T1_T3T1	Group2
Haemophilus_parainfluenzae_215035_2_ISO5	Group2
Haemophilus_parainfluenzae_C2008001710	Group2
Haemophilus_parainfluenzae_C2008003258	Group1
Haemophilus_parainfluenzae_C2009038101	Group1
Haemophilus_parainfluenzae_C2011020591	Group1
Haemophilus_parainfluenzae_60884_B_Hi_2	Group1
Haemophilus_parainfluenzae_65114_B_Hi_3	Group1
Haemophilus_parainfluenzae_777_HPAR	Group1
Haemophilus_parainfluenzae_155_HPAR	Group1
Haemophilus_parainfluenzae_1128_HPAR	Group1
Haemophilus_parainfluenzae_1209_HPAR	Group1
```

We called this `habitat_groups.txt` and then loaded it into the anvi'o pangenomes DB with

```
anvi-import-misc-data -p Haemophilus-isolates-td-PAN/Haemophilus-isolates-td-PAN.db -t layers habitat_g
```

We then used the `anvi-get-enriched-functions-per-pan-group` command for both Pfam and TIGRfam annotations in the following way:

```
anvi-get-enriched-functions-per-pan-group -p Haemophilus-isolates-td-PAN/Haemophilus-isolates-td-PAN.db
```

This produces Supplemental Data 2 (TIGRFAM). Note that this snippet is specifically for TIGRFAMs, we ran it also again with `--annotation-source Pfam` to get a table of differential Pfam functions.

Nucleotide level coverage plots (Figure 4A, 4B)

Visually inspecting the nucleotide level coverage over genes of interest is crucial for determining whether the gene or contig is well-behaved, and thus whether any resulting summary coverage values are trustworthy.

Rothia candidate driver of adaptation to BM

After identifying a candidate driver of habitat specificity (see above), we used the interactive anvio interface `anvi-interactive` with `--gene-mode` to identify the contig and split from which the gene came.

For each oral habitat, we exported the coverage using

```
anvi-get-split-coverages -p $prefix-$site-MERGED/PROFILE.db -C Genomes -b Rothia_sp_HMSC061E04_HMSC061E
```

changing the path as needed for each habitat's data.

Then, similarly adjusting the path to specify each habitat, we obtained the sequence variants mapped from the metagenome for each nucleotide position for the contig.

```
anvi-gen-variability-profile -c $prefix-CONTIGS.db -p $prefix-$site-MERGED/PROFILE.db -C Genomes -b Rot
```

To link genes to split-based coverage, we exported the table listing the split-based start/stop positions for each gene with sqlite, like so: `sqlite3 Haemophilus-isolates-CONTIGS.db 'PRAGMA table_info(genes_in_splits)' | awk -F"|" '{print $2}' | tr '\n' '|' | sed "s/|$//" > Rothia-isolates-splits-genes.txt sqlite3 Rothia-isolates-CONTIGS.db 'SELECT * from genes_in_splits' >> Rothia-isolates-splits-genes.txt`

With these inputs, we can plot per-nucleotide coverage from the metagenomes on a window around a focal gene, as in Figure 4b.

```r
library(ggplot2)
library(reshape2)
library(entropy)
library(dplyr)

# read in and pre-process variability information
getVariability <- function(prefix, sites=c('TD','BM','SUPP'), pos.min = 0, pos.max = 999999, min_covera
    all <- data.frame()

    # iteratively load and attach row-wise
    for (site in sites) {
        df <- read.csv(paste0(prefix, "variability-", site, '.tsv'), sep = "\t", header = T)

        # only keep variability info for sites with > 10x coverage
        df <- df[df$coverage >= min_coverage,]

        # get entropy across samples and lose junk
        df <- group_by(df, split_name, pos) %>% summarise(a=sum(A), c=sum(C), t=sum(T), g=sum(G)) %>%
          group_by(split_name,pos) %>% summarise(entropy=entropy.empirical(c(a,c,t,g), unit = 'log2'))

        df <- data.frame(df)
        df$site <- toupper(site)

        all <- rbind(all, df)
    }

    # subset if requested
    all <- all[(all$pos >= pos.min) & (all$pos <= pos.max),]

    all$site <- factor(all$site, levels = c("SUPP","BM","TD"))

    return(all)
}


getCoverages <- function(prefix, sites=c('TD','BM','SUPP'), pos.min = 0, pos.max = 999999) {
    all <- data.frame()

    # iteratively load and attach row-wise
    for (site in sites) {
        df <- read.csv(paste0(prefix, site, '.tsv'), sep = "\t", header = T)#, nrows = 1000000)
        colnames(df) <- c("unique_pos_id","nt_position","split_id","sample_name","coverage")

        df$site <- toupper(site)

        all <- rbind(all, df)
    }

    return(all)
}

# THIS FIGURES OUT WHICH SPLITS NEED COVERAGE - doing this first to reduce filesize of the covs file(s)
findSpecificSplits <- function(geneID="104649", genes) {

  # find what contig the gene is on
  splitName <- genes$split[genes$gene_callers_id == geneID]
  splitNum <- unlist(strsplit(as.character(splitName), "_"))[4]
  splitBody <- unlist(strsplit(as.character(splitName), "_"))[1:3]

  # get adjacent splits too just in case it's on edge of split (same contig, though)
  adjacentSplits <- sprintf(paste0(c(splitBody, "%05s"), collapse = "_"), (as.numeric(splitNum)-1):(as.


  splits <- unique(as.character(genes$split[genes$split %in% adjacentSplits]))

  return(splits)
}

plotCovsAndGenes <- function(covs = getCoverages(), genes = getGeneCalls(), surroundingGenes=10, prefix
                             tax, splits, keep_fraction = 1, target, site="TD", outputSuf=NULL) {

  splitID <- genes$split[genes$gene_callers_id == target]

  workingCov <- covs[grep(splitID, covs$split_id),]
```

```r
# take n genes either side
workingGcs <- genes[(which(genes$gene_callers_id == target) - surroundingGenes):(which(genes$gene_cal

# make sure they're all on same contig
workingGcs <- workingGcs[workingGcs$split == workingGcs$split[workingGcs$gene_callers_id == target],]

# pull out window provided by GeneCaller positions
workingCov <- workingCov[(workingCov$nt_position >= workingGcs$start_in_split[1]) &
                           (workingCov$nt_position <= workingGcs$stop_in_split[length(workingGcs$stop

print(head(workingCov))

genes <- workingGcs
covs <- workingCov

  genes$site <- factor(site)
  genes$gene_callers_id <- factor(genes$gene_callers_id)

  geneColors <- setNames(rep("grey80", nrow(genes)), as.character(genes$gene_callers_id))
  geneColors[target] <- "#BA0000"

  ###############
  # drop variability info not in window, fix NAs to 0 for scaling
  variability <- variability[variability$split_name %in% unique(as.character(covs$split_id)),]

  # sometimes chunk might not have variability and crash; only renormalize if kept any
  if (nrow(variability) > 0){
    print("found variability; proceeding...")
    variability <- variability[(variability$pos >= min(covs$nt_position)) & (variability$pos <= max(c
    variability$site <- factor(variability$site, levels = c("SUPP","BM","TD"))
    variability$entropy[is.na(variability$entropy)] <- 0

    # multiply out coverage to get proportional SNPs
    variability$entropyOrig <- variability$entropy
    variability$entropy <- variability$entropy * (max(covs$coverage)/max(variability$entropy))

    print(max(variability$entropyOrig))
  } else { # make a dummy
    print("no variability in this window; making a dummy")
    variability[1, ] <- NA
    variability$site <- site
    variability$entropyOrig <- 0
  }


#######
  # make the plot
  p <- ggplot(data = covs, aes(x = nt_position, y = coverage)) +
      facet_wrap(~site, ncol = 1, scales = 'fixed', strip.position = 'right', ) + theme_minimal() +
      geom_col(data = variability, stat = 'identity', position = 'identity', aes(x = pos, y = entropy
      geom_line(size = 0.3, alpha = 0.5, aes(color=site, group=sample_name)) +
      geom_hline(data = genes, yintercept = 0.97*max(covs$coverage), color = 'grey20') +
      geom_segment(data = genes, aes(x = start_in_split, xend = stop_in_split, color = gene_callers_i
                          y = 0.99*max(covs$coverage), yend = 0.99*max(covs$coverage), size = 4) +
        labs(title = target) + ylab("Coverage") +
        scale_x_continuous(expand=c(0,0)) + scale_y_continuous(expand=c(0,0), sec.axis = sec_axis(~.*ma
        scale_color_manual(values = c(setNames(c("#2790db", "#b287e8", "#69d173"), c("TD","BM","SUPP"))
        theme(panel.grid = element_blank(), axis.line = element_line(colour = 'black', size = 0.5), axi
              strip.text = element_text(angle=90), axis.ticks = element_line(colour='black', size=0.3))


    # did you give me a suffix to use to save it?
    if (!is.null(outputSuf)){
        pdf(paste0(prefix,outputSuf,".pdf"), width = 7, height = 4)
    }

    print(p)

    if (!is.null(outputSuf)){
        dev.off()
    }
}

prefix <- '~/g3/oral_genomes_analysis/rothia/rot_split_cov-ALL-'

geneCallsPath <- 'Rothia-isolates-splits-genes.txt'

tax <- 'R. muciaginosa E04'

splitPosRelativeToContig <- 203284 # rothia_buccal

geneTarget <- c("104283","103512","104082","103409","103186","103758","103125","104483","104321","10482
                "104322","104458","104363","103468","103939","103201","104081","104464","104323","10338


# get gene call positions
gcs <- read.csv(geneCallsPath, header = T, sep="|")

# then find out what splits needed
splitsNeeded <- c()
for (gene in geneTarget) {
  splitsNeeded <- c(splitsNeeded, findSpecificSplits(geneID = gene, genes = gcs))
}

site='SUPP'

# get coverages
cov <- getCoverages(prefix = prefix, pos.min = 0, pos.max = Inf, sites = site)
# and variability
variability <- getVariability(prefix = prefix, sites = site, min_coverage = 0)


# save each one
for (i in 1:length(geneTarget)) {
  plotCovsAndGenes(covs = cov, prefix=paste0('~/g3/oral_genomes_analysis/rothia/bm_gene_covs/20190826/E
                   genes = gcs, target = geneTarget[i],
                 variability = variability,
                 tax = tax, splits=splitDF, site = site,
                 outputSuf = geneTarget[i])
}
```

Investigating the effect of non-specific mapping

Regions of high nucleotide similarity between reference genomes relative to the metagenome thus recruit an equivalent number of reads reflecting the total population abundance in that sample. However, regions with polymorphic sites allow for discrimination between genomes based on the variability of sequences in the pangenome. By plotting the mean coverage per split (approx. 20kb segments of a contig) of genomes by sample, genomes with similar coverage patterns per sample can be seen, reflecting this pseudorandom assignment or nonspecific mapping, that is, mapping reads originating from non-target taxa in the metagenome onto the target genome(s) (Supplemental Figure S5). Close manual inspection revealed that despite some broad-stroke similarities, genomes with similar recruitment within a sample do recruit differently in other samples, suggesting that both potential mapping concerns do not drown out the biological signals from diverse populations. Inspection of coverage at the per-nucleotide level, e.g., Figure 4B and Supplemental Figure S3, confirmed that non-specific mapping produces a distinct signature of extremely spiky coverage that can be easily differentiated from the underlying signal. Thus, a particular sequence's coverage reflects the representation of that sequence variant in that sample's population.