Bioinformatics pipeline draft
Jackson Lee v.2/8/15

This document details the processing pipeline of a meta-omics sequencing project from Elkhorn Slough (Moss Landing, CA) microbial mat experiments conducted in 2011, with metagenomic and metatranscriptomic sequencing completed in 2012-2014 by the Joint Genome Institute (Walnut Creek, CA). Sequencing data from these runs were subjected to a computing intensive bioinformatics pipeline based on DOE NERSC cluster computing resources that consisted of quality control of raw reads, Co-assembly of metagenomic reads, binning of metagenomic assemblies, mapping of transcripts to binned metagenomic assemblies, and annotation of reads to reference databases. This information was then used to build a transcriptomic-based analysis of the systemic response of microbes in mats to the day night cycle and to molybdate addition.

Programming Libraries used:
Python 2.7.9 - Numpy 1.9.1, Matplotlib 1.3.1, Biopython 1.64, Pandas 0.15.1,
Perl 5.16.3
R 3.1.1 - RStudio 0.98.1049, vegan 2.2-1, RColorBrewer 1.1-2, alphahull 1.0, e1071 1.6-4, fpc 2.1-9, rgl 0.93.1098, fields 7.1
Bash shell scripting (NERSC compatible scripts included) - scp, tar, cat, cut, head, tail, paste, sed, unique, grep, vim
NOTE: syntax will vary depending on platform and version. If you're on a mac, gsed could be more useful than the Darwin sed and is used in this guide.
Samtools 0.1.19-44428cd
Bamtools 2.20.1-4-gb877b35
Prodigal 2.6.1
Hmmsearch - HMMR 3.1b1

Fonts and formatting legend:
`Script and software commands are written in fixed width font.`
Red 'greater than' sign indicates execution at the command line in Unix/Linux, i.e.:

```
>split_interleaved_fastq.py –i ElkSloSequencingFile.fastq
```

Blue 'greater than' sign indicates execution in R or RStudio, i.e.:

```
>filtered_df <- Coassembly_coverage[Coassembly_coverage$Reference.length > 1500,]
```

Red 'colon' indicates commands in VIM, i.e.:

```
:%s/^M//g
```

Table of Contents

1 Raw sequence processing and metagenome assembly with Ray

Raw files consisted of ~80 GB fastq Illumina HiSeq files of metatranscriptomes, and ~20 GB fastq Illumina HiSeq files of metagenomes in interleaved format. These were split into mate-pair files using the following header format denoting forward and reverse reads:

```
@HISEQ06:204:C06F3ACXX:5:1101:1475:2407 1:N:0
@HISEQ06:204:C06F3ACXX:5:1101:1475:2407 2:N:0
```

```
>split_interleaved_fastq.py -i ElkSloSequencingFile.fastq
ElkSloSequencingFile.1.fastq
ElkSloSequencingFile.2.fastq
```

These were combined with cat to form co-assembly raw reads:

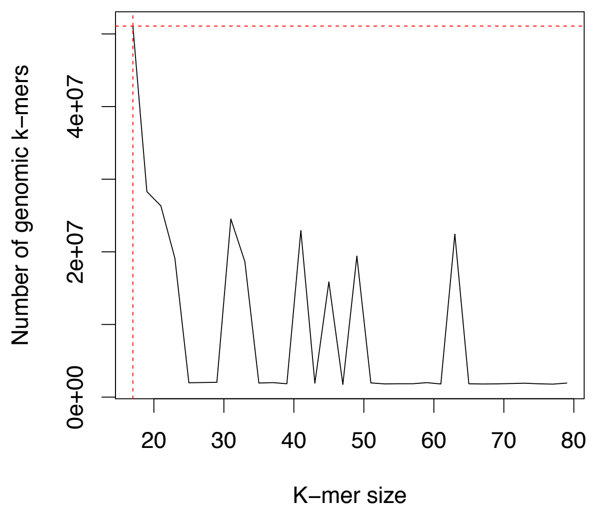
```
>cat ElkSloSequencingFile.A.1.fastq ElkSloSequencingFile.B.1.fastq ElkSloSequencingFile.C.1.fastq >
combined.ElkSloSequencingFile.1.fastq
>cat ElkSloSequencingFile.A.2.fastq ElkSloSequencingFile.B.2.fastq ElkSloSequencingFile.C.2.fastq >
combined.ElkSloSequencingFile.2.fastq
```

This co-assembly source fastq file was assembled with Ray (http://denovoassembler.sourceforge.net/) [1] on the National Energy Research Scientific Computing Center (NERSC) Edison cluster using control scripts for Ray produced by Rob Egan (rsegan@lnl.gov) at NERSC. Fastq format files are detected from the local directory by the script:

```
>module load Ray
>qsub -l mppwidth=480,walltime=12:00:00 -q regular -v K=29
/usr/common/jgi/assemblers/Ray/2.3.0/runRayEdisonScaffold.sh
```

Ray produces a contigs file, a scaffolds file (both fasta) and a number of statistics files (including a ScaffoldLengths.txt file and N/L50 statistics to std_out). From this contigs file KmerGenie (http://kmergenie.bx.psu.edu/) [2] was run to estimate the needed kmer word sizes for assembly. Peaks from these graphs were selected as assembly word sizes (below, 29, 45, 63).

```
>kmergenie Contigs.fasta -k 79 -l 15 -s 2 -t 24 -o kmer_dist_k15_121
```



Ray was rerun for each assembly word size:

```
>qsub -l mppwidth=480,walltime=12:00:00 -q regular -v K=63
/usr/common/jgi/assemblers/Ray/2.3.0/runRayEdisonScaffold.sh
>qsub -l mppwidth=480,walltime=12:00:00 -q regular -v K=49
/usr/common/jgi/assemblers/Ray/2.3.0/runRayEdisonScaffold.sh
```

2 Bowtie2 mapping of assembled scaffolds

In order to generate coverages for assembled scaffolds to serve as statistics for binning of each assembly, Bowtie2 (http://bowtie-bio.sourceforge.net/bowtie2/index.shtml) [3] was used to map the raw reads of each metagenome library to the co-assembly. A bowtie2 shell script on the NERSC Edison cluster was used to map each of the original metagenome libraries to the co-assembly scaffold fasta file. When using these scripts, the appropriate libraries need to be setup first. This includes paths to executables, python libraries (PYTHONPATH) and so forth and are environment specific and not shown here except for NERSC-supplied modules. When using cluster resources, node and walltime parameters need to be determined based on number of libraries and expected running time, respectively. Bowtie2 was run in a 1 file to 1 node format (24 cores / node). GFF files were used as a scaffold reference and was generated from ScaffoldLengths.txt from the Ray run.

```
>module load bowtie2
>sed -e 's/\(^\S*\)\t\([0-9]*\)/\1\tRay\tcontig\t1\t\2\t\.\t+\t\./' ScaffoldLengths.txt >
k29.ScaffoldLengths.gff
>bowtie2-build Scaffolds.fasta k29.scaffods.bowtie2.index
>qsub -l mppwidth=312,walltime=12:00:00 -q regular bowtie2_script.sh
```

```
bowtie2_script.sh
#####
PAIREDFILES=./MT_source/*.1.fastq

for i in $PAIREDFILES
do
  aprun -b -n 1 bowtie2 -x ./k29.scaffolds.bowtie2.index -1 ./$i -2 ./${i%.1.fastq}.2.fastq -S
./${i%.1.fastq}.k29.scaffolds.mapping.sam &
#####
```

Bowtie2 produces *.sam format files which must be summarized into count files using the cluster qsub script:

```
>qsub -l mppwidth=312,walltime=24:00:00 -q regular process_sams_batch.sh
```

To process this in batch with a script:

```
process_sams.sh
#####
echo "Running Samtools, convert sam to bam and sort"
module load samtools

SAMFILES=./*.sam
for i in $SAMFILES
do
  samtools view -bS $i > ${i%.sam}.bam &
done
wait

BAMFILES=./*.bam
for j in $BAMFILES
do
  samtools sort $j ${j%.bam}.sorted.bam &
done
wait

echo "Running bedtools on sorted bam alignments with gff"
SORTEDFILES=./*sorted.bam.bam
for k in $SORTEDFILES
do
  outname=$(basename $k)
  bedtools genomecov -d -ibam $k > k29.Scaffolds.${outname%.sorted.bam.bam}.coverage.txt &
done
wait

echo "Running coverage summary script"
COVFILES=./*coverage.txt
for i in $COVFILES
```

```
do
  outname=$(basename $i)
  contig_coverage_from_perbase_gff.py -i $i -t -o ${outname%.coverage.txt}.Counts.txt &
done
wait
```

To manually process this file locally (for each library (e.g. A, B,C below)):

```
>samtools view -bS ElkSloSequencingFile.A.k29.scaffolds.mapping.sam >
ElkSloSequencingFile.A.k29.scaffolds.mapping.bam
>samtools sort ElkSloSequencingFile.A.k29.scaffolds.mapping.bam
ElkSloSequencingFile.A.k29.scaffolds.mapping.sorted.bam
>sed -e 's/\(^\S*\)\t\([0-9]*\)/\1\tRay\tcontig\t1\t\2\t\.\t+\t\./' ScaffoldLengths.txt >
k29.ScaffoldLengths.gff
>bedtools coverage -d abam ElkSloSequencingFile.A.k29.scaffolds.mapping.sorted.bam -b
k29.ScaffoldLengths.gff > ElkSloSequencingFile.A.k29.scaffolds.mapping.sorted.bam
ElkSloSequencingFile.A.k29.scaffolds.mapping.coverage.txt
>contig_coverage_from_perbase_gff.py -i ElkSloSequencingFile.A.k29.scaffolds.mapping.coverage.txt -
t -o ElkSloSequencingFile.A.k29.scaffolds.mapping.Counts.txt
```

Once each *.Count file is completed, they need to be joined together into one file in a tab-delimited format with scaffold name, length, counts for each library. Using paste may require reformatting to remove extraneous control characters (e.g. ^M)

```
>paste ElkSloSequencingFile.A.k29.scaffolds.mapping.Counts.txt
ElkSloSequencingFile.B.k29.scaffolds.mapping.Counts.txt
ElkSloSequencingFile.C.k29.scaffolds.mapping.Counts.txt
ElkSloSequencingFile.D.k29.scaffolds.mapping.Counts.txt | cut -d$'\t' -f1,2,3,6,9,12 >
k29.Scaffolds.All.Counts.txt
```

Add in a header line to this final file like 'header bplen   MG_A MG_B MG_C' separated by tabs using a text editor (e.g. VIM). This file is now ready to be copied off the cluster for desktop or server processing.

3 Binning preparation & annotation using MG-RAST

The code base of Albertsen et al. (2013) [4] was used when available to prepare data for the binning process.

All scaffolds less than 800 bp were discarded.

```
>filter_fasta_by_len.py -i Scaffolds.fasta -l 800
```

This produced a file Scaffolds.fasta.filtered.greaterthan.800.fasta. Starting from Step 6 of Albertsen et al. we use the %GC, tetranucleotide frequency, and consensus taxonomy scripts to generate input data for R analysis. These steps also identify the open reading frames (ORFs) and the putative essential single-copy gene (based on [4]) using Prodigal (http://prodigal.ornl.gov/algorithm.html) [5] and HMMER [6], respectively.

Step 6:

```
>perl ~/multi-metagenome/R.data.generation/calc.kmerfreq.pl -i
Scaffolds.filtered.greaterthan.800.fasta -o Scaffolds.800.kmer.tab
```

Step 7:

```
>perl ~/multi-metagenome/R.data.generation/calc.gc.pl -i Scaffolds.filtered.greaterthan.800.fasta -
o Scaffolds.800.gc.tab
```

Step 8:

```
>prodigal -a scaffolds.temp.800.orfs.faa -i Scaffolds.filtered.greaterthan.800.fasta -m -o
prodigal.scaffolds.gff -p meta -q -f gff -d scaffolds.assembly.800.orfs.fa
>cut -f1 -d " " scaffolds.temp.800.orfs.faa > scaffolds.assembly.800.orfs.faa
>hmmsearch --tblout scaffolds.assembly.800.orfs.hmm.txt --cut_tc --notextw ~/multi-
metagenome/R.data.generation/essential.hmm scaffolds.assembly.800.orfs.hmm.faa
>tail -n+4 scaffolds.assembly.800.orfs.hmm.txt | gsed 's/ * / /g' | cut -f1,4 -d " " | gsed 's/_/
/g' > scaffolds.assembly.800.orfs.hmm.id.txt
>tail -n+4 scaffolds.assembly.800.orfs.hmm.txt | cut -f1 -d " " > list.of.positive.orfs.txt
```

3.1 MG-RAST annotation

This pipeline uses the MG-RAST [7] annotation system. This requires that each prodigal ORF fasta file be uploaded into the MG-RAST website. Default paramaters were used but no quality control and no removal or assembly of sequences was performed. Once processing was completed, the clusterfile, BLAT amino acid 90% similarity results file, and each of the reference and md5 files were downloaded. Annotation files were downloaded from each project's download section, and the reference files were obtained from the MG-RAST FTP server (ftp://ftp.metagenomics.anl.gov/). These files were combined to extract the top hits for each gene annotation using python scripts.

Files needed for upload:
```
assembly.800.orfs.faa
```

Files to download:
```
mgmNNNNNNN.3.550.cluster.aa90.mapping   #list of genes that clustered at 90% e.g.
                                        'mgm4599995.3.550.cluster.aa90.mapping'
mgmNNNNNNN.3.650.protein.sims           #database of matches, quality scores, and md5s
```

Reference files from MG-RAST FTP server:
```
md5_lca_map.gz      #md5 to taxonomy matches
md5_protein_map.gz  #md5 to function, source, and organism codes
md5_ontology_map.gz #md5 to ontology, source and function code
organism_map.gz     #organism code to organism name
function_map.gz     #function code to function name
ontology_map.gz     #ontology code to ontology name
source_map          #source database code to source database names
```

Read the DocString and default options for query_mg_rast_md5_annotation.py for script functions and database options. This script will generate protein and ontology files based on the *.sims file headers named on the -o option prefix.

Table of sources and source codes from MG-RAST reference files (as of 2/9/15).

Source code	Database name	Type
2	GenBank	protein
3	IMG	protein
4	KEGG	protein
5	PATRIC	protein
6	RefSeq	protein
7	KBase	protein
8	SwissProt	protein
9	TrEMBL	protein
20	M5NR	protein
21	InterPro	protein
22	Phantome	protein
24	SEED	protein
10	COG	ontology
11	GO	ontology
12	KO	ontology
14	Subsystems	ontology
13	NOG	ontology
15	Greengenes	rna
16	RDP	rna
17	LSU	rna
18	SSU	rna
19	M5RNA	rna
23	ITS	rna

Unzip all *.gz files in a single directory (e.g. ~/mg-rast_db/) and keep the original names.
Use these mapping files, clustering files, and selecting a confidence cutoff (i.e. e-val 1e-10, %ID 70):
```
>query_mgrast_md5_annotation.py -i mgm4599995.3.650.protein.sims -d ~/mg-rast_db/ -o
mgm4599995.3.ordered -e 1e-10 -t 70 -s 21,2,6,4,3,5,24,22,12,15,17,18,23 -c
mgm4599995.3.550.cluster.aa90.mapping
```

Use query_mg_rast_md5_taxonomy.py to determine lineages, first filtering on top hits:

```
>pick_top_mgrast_sims_md5.py -i mgm4599995.3.650.protein.sims -o mgm4599995.3.top.md5.tab -e 1e-10
>query_mgrast_md5_taxonomy.py -i mgm4599995.3.top.md5.tab -t ~/mg-rast_db/md5_lca_map -o
mgm4599995.3.top.taxonomy.tab
```

And save as scaffolds.assembly.800.orfs.hmm.blat.tax.txt. Using text editors and shell commands, process this file into a format suitable for the Albertsen et al. (2013) pipeline.

```
>filter_line_by_list.py -f list.of.positive.orfs.txt -i
scaffolds.assembly.800.orfs.hmm.blat.tax.txt
>mv scaffolds.assembly.800.orfs.hmm.blat.tax.filtered.txt
scaffolds.assembly.800.orfs.hmm.blat.tax.modified.txt
```

Modify scaffolds.assembly.800.orfs.hmm.blast.tax.txt to get proper tax ranges and values where needed (e.g. for Proteobacteria class to show up as phylum) using VIM:

```
:%s/Proteobacteria;//
```

Take this file and filter and format for the Albertsen et al. (2013) list of positive taxonomy.

```
>gsed 's/\t/;/' scaffolds.assembly.800.orfs.hmm.blat.tax.modified.txt | cut -f1,4 -d ";" | gsed
's/;/\t/' | gsed 's/_/\t/' > scaffolds.assembly.800.orfs.hmm.blat.tax.tab
```

Modify scaffolds.assembly.800.orfs.hmm.blat.tax.tab for bad formatting using VIM where needed, replace empty strings with 'None' so it will be in the format for the Albertsen et al. (2013) scripts:

```
:%s/^M//g
:%s/\(\d$\)/\1\tNone/
```

Step 9:

```
>perl ~/Desktop/huge_scripts/multi-metagenome/R.data.generation/hmm.majority.vote.pl -i
scaffolds.assembly.800.orfs.hmm.blat.tax.modified.txt -o scaffolds.assembly.800.tax.consensus.txt –
n
```

4 Binning of scaffolds in R using DBSCAN clustering followed by SVM training and classification

The binning procedure follows the code base and methodology of Albertsen et al. (2013), but also tries to use machine learning and data reduction methods to lessen the data burden from large datasets. Coverage statistics are first log transformed and reduced by principal component analysis. The top components are used with DBSCAN clustering (from fpc library) to generate a list of density-based clusters of the longest scaffolds. This cluster library is used as the training dataset for a support vector machine (SVM) classification method (from e1071 library) to bin the remaining scaffolds. A number of visualization and data-handling libraries are also used (see required libraries at the beginning of the manual).

Several annotation files need to be set in the R script:

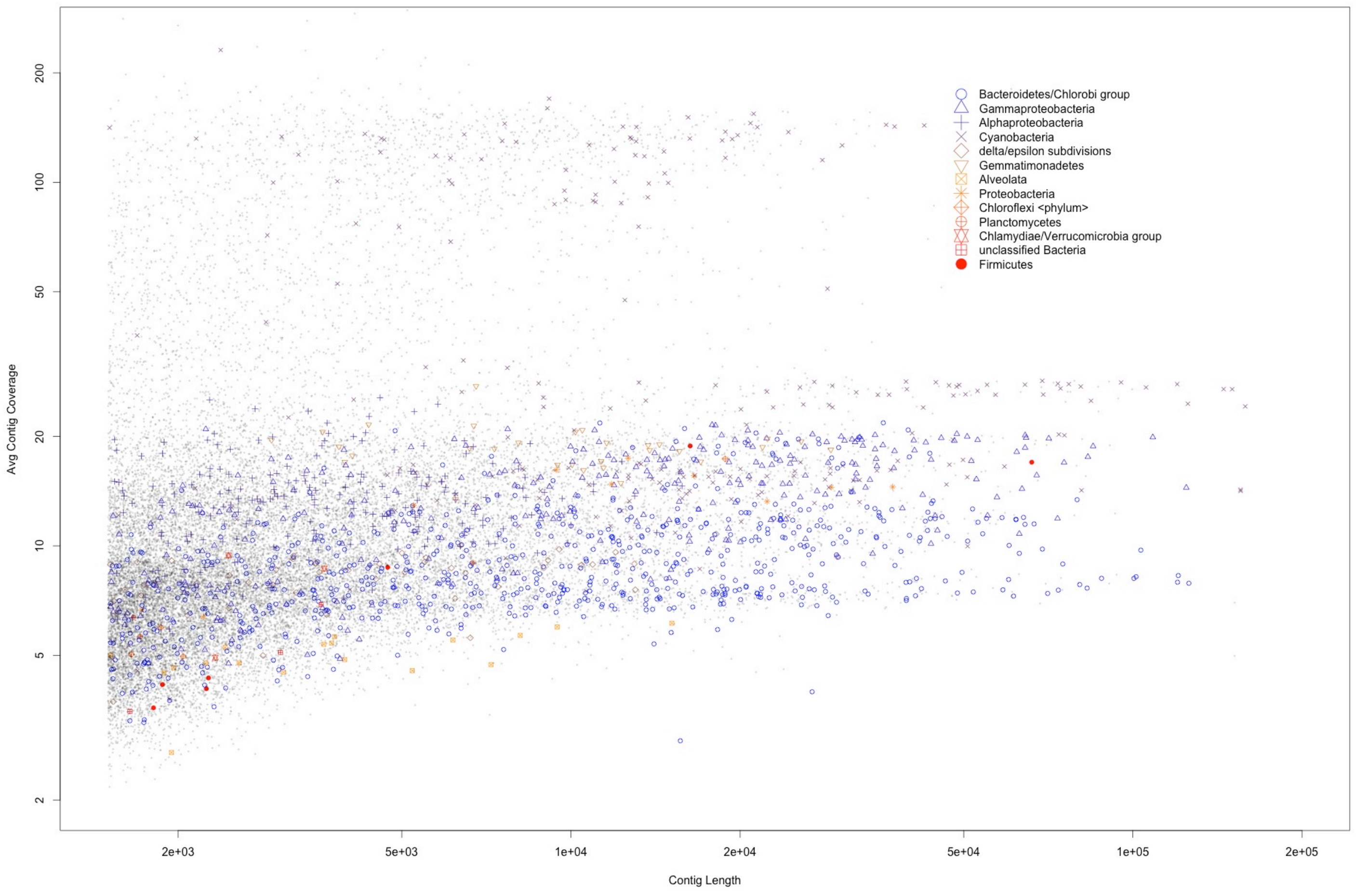
```
k29.Scaffolds.All.Counts.txt                 #the MG coverage file with scaffold length
Scaffolds.800.kmer.txt                       #kmer text file of all kmers for each scaffold
Scaffolds.800.gc.txt                         #%GC content for each scaffold
scaffolds.assembly.800.tax.consensus.txt     #hmm essential genes consensus taxonomy and code
scaffolds.assembly.800.orfs.hmm.blat.tax.tab #full taxonomy of hmm genes
scaffolds.assembly.800.orfs.hmm.id.txt
```

```
>Coassembly_coverage <- read.csv("k29.Scaffolds.All.Counts.txt", header = F, sep = "\t", quote =
"")
>colnames(Coassembly_coverage) =
c("Contig.Name","Reference.length","Coverage_1","Coverage_2","Coverage_3","Coverage_4")
>gc <- read.delim("Scaffolds.800.gc.tab", header = T)
>kmer <- read.delim("Scaffolds.800.kmer.tab", header = T)
>colnames(kmer)[1] = "name"
>ess <- read.table("scaffolds.assembly.800.orfs.hmm.id.txt", header = F)
>colnames(ess) = c("name","orf","hmm.id")
>ess.tax <- read.delim("scaffolds.assembly.800.orfs.hmm.blat.tax.tab", header = F)
>colnames(ess.tax) = c("name","orf","phylum")
>cons.tax <- read.delim("scaffolds.assembly.800.tax.consensus.txt", header = T)
>colnames(cons.tax) = c("name","phylum","tax.color","all.assignments")
```

Using svm.binning.workflow.R read in all files and generate a preliminary map of scaffold length, taxonomy, and coverage distributions.

```
>pre_d <- merge(d[c("name","tax.color","phylum","length")], Coassembly_coverage[c("Contig.Name",
"Coverage_1","Coverage_2","Coverage_3","Coverage_4")], by.x="name",by.y="Contig.Name")
>pre_d$avg <- rowMeans(pre_d[c("Coverage_1","Coverage_2","Coverage_3","Coverage_4")])

>byor<-colorRampPalette(c("blue","orange","red"))
>phspan <- length(unique(pre_d$tax.color))-1
>byor_key <- arrange(pre_d[!duplicated(factor(pre_d$tax.color)),],tax.color)[1:phspan,"tax.color"]
>palette(adjustcolor(byor(phspan), alpha.f = 1))

>plot(x = pre_d$length,
     y = pre_d$avg,
     ylim = c(2, 250),
     xlim = c(1500,200000),
     log = "yx",
     cex = .8,
     pch=pre_d$tax.color,
     col=match(pre_d$tax.color,byor_key),
     xlab = "Contig Length",
     ylab = "Avg Contig Coverage"
)

>points(x = pre_d$length[is.na(pre_d$tax.color)],
       y = pre_d$avg[is.na(pre_d$tax.color)],
       cex = .4,
       pch=20,
       col=rgb(0,0,0,0.1),
)
```

Scaffold length vs. coverage for mapped reads for k=63 with taxonomies labeled where available.



Filter down the dataset into a working data frame (e.g. >1500 bp scaffolds). Reduce the data axes using the $\log_{10}$ of coverage with PCA on the working data frame.

```
>filtered_df <- Coassembly_coverage[Coassembly_coverage$Reference.length > 1500,]
>filtered_df <- merge(filtered_df, gc, by.x ="Contig.Name", by.y="contig")

#PCA based analysis
#log coverage PCA
>data <- cbind(log10(filtered_df[c("Coverage_1","Coverage_2","Coverage_3","Coverage_4")]+1))
>pca_mat <- cbind(t(data))

>pca_results <- prcomp(pca_mat)
>print(pca_results)
>summary(pca_results)
>plot(pca_results)
>pca_results

>PC1 <- data.frame(filtered_df$Contig.Name,
pca_results$rotation[,1]+1,filtered_df$Reference.length)
>PC2  <- data.frame(filtered_df$Contig.Name,
pca_results$rotation[,2]+1,filtered_df$Reference.length)
>PC3  <- data.frame(filtered_df$Contig.Name,
pca_results$rotation[,3]+1,filtered_df$Reference.length)
>PC4  <- data.frame(filtered_df$Contig.Name,
>pca_results$rotation[,4]+1,filtered_df$Reference.length)
>colnames(PC1) = c("Name","Loading","Reference.length")
>colnames(PC2) = c("Name","Loading","Reference.length")
>colnames(PC3) = c("Name","Loading","Reference.length")
>colnames(PC4) = c("Name","Loading","Reference.length")

# Combine all data to one data matrix "d"
>d <- data.frame(PC1$Name,
                PC1$Reference.length,
                filtered_df$gc,
                PC1$Loading*100,
                PC2$Loading*100,
                PC3$Loading*100
                )

>colnames(d) = c("name",
               "length",
               "gc",
               "HPminus",
               "HPplus",
               "HP3"
                )

>d <- merge(d,cons.tax, by = "name", all.x = T)
# Combine data on essential genes
>ess<- merge(ess, d, by = "name", all.x = T)
>ess<- merge(ess, ess.tax, by = c("name","orf"), all.x = T)
```

From this data select the DBSCAN training parameters (training length cutoff (>20k bp contigs), minimum points (>4), EPS (.35) were determined empirically) and visualize.

```
>cutoff = 10000
>d2 <- subset(d,length > cutoff)
>ess2 = subset(ess,length > cutoff)

#DBSCAN analysis
>filtered_data <- data.frame(PC1$Loading*1000, PC2$Loading*1000, PC3$Loading*1000, d$gc*.1,
d$length)
>colnames(filtered_data) <- c("PC1","PC2","PC3","gc","length")
>filtered_data = subset(filtered_data, length > cutoff)
>filtered_data$length = NULL

>db_results <- dbscan(filtered_data, eps=.35, MinPts=4)
>db_results
```

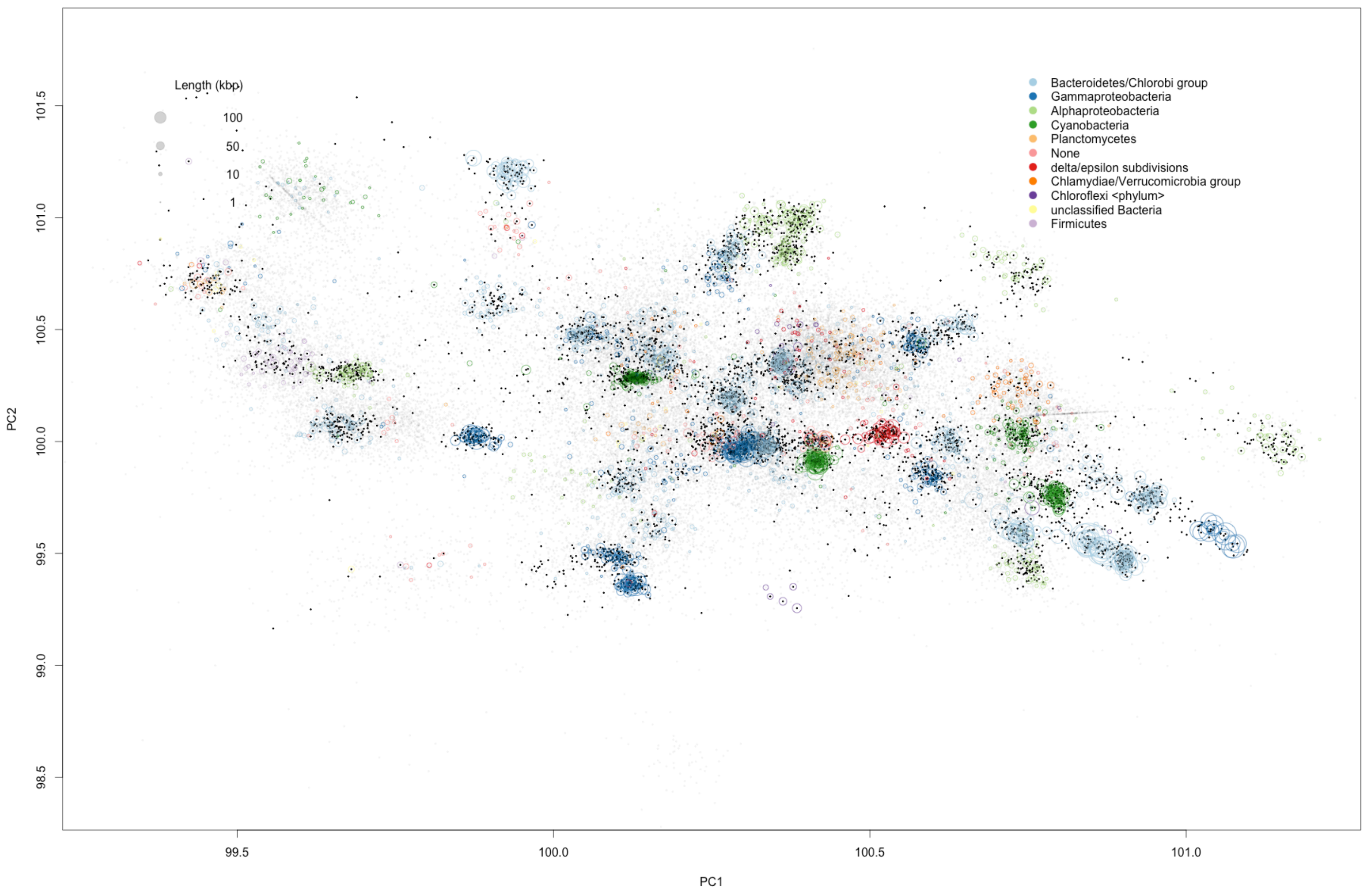
```
>plot(db_results, filtered_data)

>d2$db_cluster = db_results$cluster
>ess2 <- merge(ess2, d2[c("db_cluster","name")], by="name",All.x=T)
>unique_cluster = unique(d2$db_cluster)

>dev.off()
# Define a good color palette and make it transparrent
>gbr<-colorRampPalette(c("green","blue","orange","red"))
>gcspan <- round(max(d$gc)-min(d$gc)+1)
>palette(adjustcolor(gbr(70), alpha.f = 0.1))

>dbspan <- length(unique_cluster)
>palette(adjustcolor(rainbow(dbspan), alpha.f = 1))

>plot(x = d2$HPminus,
     y = d2$HPplus,
     xlim = c(99.3, 101.2),
     ylim = c(98.4, 101.8),
     cex = .8,
     pch=20,
     col=d2$db_cluster,
     xlab = "PC1",
     ylab = "PC2"
)

>borderpts <- subset(d2, db_cluster == 0)
>points(x = borderpts$HPminus,
        y = borderpts$HPplus,
        pch=20,
        cex = sqrt(d_prot$length)/100,
        col=rgb(0,0,0,1)
        lwd=.8
)

# Add a label in the center of each extracted bin
>for (i in 1:length(unique_cluster)) {
       text(mean(d2$HPminus[d2$db_cluster ==unique_cluster[i]]),
              mean(d2$HPplus[d2$db_cluster == unique_cluster[i]]),
              labels = as.character(unique_cluster[i]),
              cex=1,
              font=1
       )
}
```

Code is provided to filter, split, join, and extract bins manually. In practice, most small bins tended to be incomplete even with manual quality control and the majority of bins were distinctly partitioned using 3 principal component axes.

Visualize the complete ‘galaxy plot’:

```
#set colors
>freqtab2 <- as.data.frame(margin.table(table(d3[c("tax.color","phylum")]),1))
>colorlist <- freqtab[with(freqtab,order(-Freq)),]
>topcolors <- colorlist$tax.color[1:11]
>freqtab <- as.data.frame(margin.table(table(d3[c("tax.color","phylum")]),2))
>phylumlist <- freqtab[with(freqtab,order(-Freq)),]
>topphylum <- phylumlist$phylum[1:11]

>plot(x = d2$HPminus,
     y = d2$HPplus,
     xlim = c(99.3, 101.2),
     ylim = c(98.4, 101.8),
     cex = .3,
     pch=20,
     xlab = "PC1",
     ylab = "PC2"
)

>palette(brewer.pal(11,"Paired"))
>points(x = d$HPminus[d$tax.color %in% topcolors],
       y = d$HPplus[d$tax.color %in% topcolors],
       cex = sqrt(d$length[d$tax.color %in% topcolors])/100,
       col=d$tax.color[d$tax.color %in% topcolors],
       lwd=.8
)
>points(x = d$HPminus,
       y = d$HPplus,
       cex = .2,
       col=rgb(0,0,0,0.05),
       lwd=1,
)


# Add scaffold size legend
>t<-as.character(c(100,50,10,1))
>legend(#x = 99,
       #y = 101,
       x = 99.35,
       y = 101.7,
       t,
       bty="n",
       pch=20,
       col=rgb(0,0,0,0.2),
       pt.cex=sqrt(as.integer(t)*1000)/100,
       x.intersp=4,
       y.intersp=1,
       adj=c(0.5,0.5),
       title="Length (kbp)"
)

# Add phylum affiliation legend
>legend(x = 101.3,
       y = 101.7,
       #x = 102.2,
       #y = 101,
       topphylum,
       bty="n",
       pch=20,
       col=topcolors,
       pt.cex=2,
       xjust=1,
       y.intersp=.5
)
```

The final 'galaxy' plot and the distribution of bins and taxonomy in $\log_{10}$ PCA plots. Similar openGL 3D plots were created using the plot3D command of the rgl library.



Run SVM training on this dataset, as well as SVM optimization. Compute statistics on all putative bins and manually manipulate SVM bins where needed. In this study bins >80% complete bins were selected for downstream analysis.

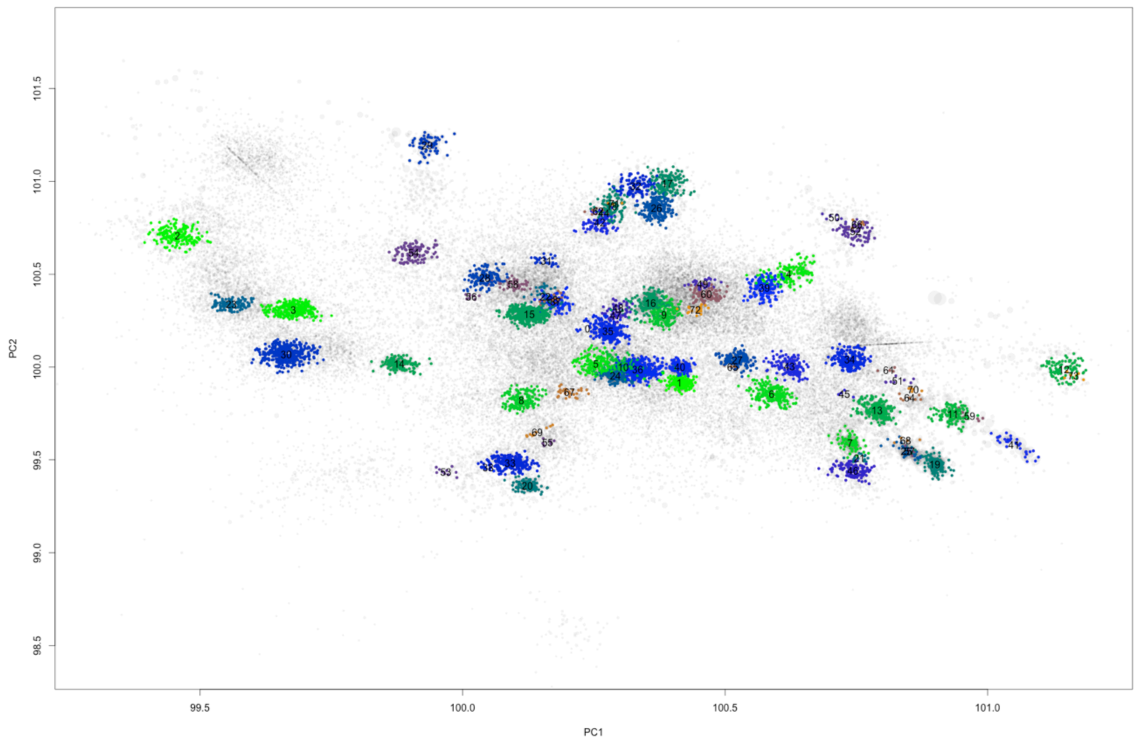
```
>trainset <- d2[c("name","HPplus","HPminus","HP3","gc","db_cluster")]
>trainset <- rename(trainset, replace=c("db_cluster" = "Category"))
>trainset <- subset(trainset, Category >= 0)
>index <- 1:nrow(trainset)
>testindex <- sample(index, trunc(length(index)/3))
>testset <- trainset[testindex,]

## svm testing, only run once
>svm.model <- svm(Category~., data = trainset[2:6], gamma = 1, cost = 10, type = "C-
classification")
>svm.pred <- predict(svm.model, testset[2:5])
#plot(svm.model,trainset[2:7])
>tab <- table(pred = svm.pred, true = testset[,7])
>tab
>classAgreement(tab)
>drops <- c("row.names","Contig.Name")
>tuned <- tune.svm(Category~., data = trainset[2:6], gamma = 10^(0:1), cost = 10^(3:4))
>summary(tuned)

# full svm run
>runset <- d[c("name","HPplus","HPminus","HP3","gc")]
>rownames(runset) <- runset$name
>svm.model <- svm(Category~., data = trainset[2:6], gamma = 100, cost = 10, type = "C-
classification")
>svm.pred <- predict(svm.model, runset[2:5])
>svmpred <- as.data.frame(svm.pred)
>d3 <- merge(d,svmpred, by.x="name",by.y="row.names",All.x=T)
>d3$svm.pred <- as.numeric(as.character(d3$svm.pred))
>e3 <- merge(ess,svmpred, by.x="name",by.y="row.names",All.x=T)
```

```
>e3$svm.pred <- as.numeric(as.character(e3$svm.pred))

#completeness stats

>summary.out <- {}
>for (i in 1:length(unique_cluster)) {
running.hmmid <- e3[e3$svm.pred == unique_cluster[i],]
running.out <- running.hmmid[which(duplicated(running.hmmid$hmm.id) |
               duplicated(running.hmmid$hmm.id, fromLast=TRUE)),]
summary.out <- rbind(summary.out,
             c(unique_cluster[i],
               length(d3$length[d3$svm.pred == unique_cluster[i]]),
               sum(d3$length[d3$svm.pred == unique_cluster[i]]),
               length(unique(running.hmmid$hmm.id)),
               nrow(running.out),
               length(unique(running.out$hmm.id)),
               mean(d3$length[d3$svm.pred == unique_cluster[i]]),
               max(d3$length[d3$svm.pred == unique_cluster[i]]),
               mean(colMeans(subset(Coassembly_coverage, Contig.Name %in% d3$name[d3$svm.pred ==
               unique_cluster[i]])[,3:6])),
               length(unique(d3$phylum[d3$svm.pred == unique_cluster[i]]))-1))
}

>summary.out <- as.data.frame(summary.out)
>colnames(summary.out) = c("bin","total_contigs", "total_bp", "completeness", "total_duplicates",
"unique_duplicates", "avg_len","max_len","mean_cov","est_phyla")
>summary.out
```

And finally visualize selected bins and export svm bin lists.

```
#show bins
>dev.off()
>palette(adjustcolor(gbr(dbspan+1), alpha.f = 1))
>plot(x = d3$HPminus,
     y = d3$HPplus,
     xlim = c(99.3, 101.2),
     ylim = c(98.4, 101.8),
     cex = sqrt(d$length)/100,
     pch=20,
     col=rgb(0,0,0,0.05),
     xlab = "PC1",
     ylab = "PC2"
)

>points(x = d3$HPminus[d3$svm.pred > 0],
     y = d3$HPplus[d3$svm.pred > 0],
     cex = .8,
     pch=20,
     col=d3$svm.pred[d3$svm.pred > 0]
)

# Add a label in the center of each extracted bin

>for (i in 1:length(unique_cluster)) {
  text(mean(d3$HPminus[d3$svm.pred ==unique_cluster[i]]),
       mean(d3$HPplus[d3$svm.pred == unique_cluster[i]]),
       labels = as.character(unique_cluster[i]),
       cex=1,
       font=1
  )
}

write.table(d3[,c("name","svm.pred")],file="svm_bin.txt",quote=F,row.names=F,col.names=F)
```

The final PC1 and PC2 SVM bin visualization from the same data as the galaxy plot with bins highlighted.



5 Transcript mapping of a single assembly

In order to generate coverages for scaffolds to serve as counts for RPKM measurements of each gene, bowtie2 was used to map the raw reads of each metatranscriptome library to the genes of the co-assembly and proceeded similarly to scaffold read mapping.

```
>split_interleaved_fastq.py -i ElkSloSequencingTranscript.A.fastq
```

etc… for each library. Followed by bowtie2 database generation and execution. Downstream scripts will read the entire fasta header line as the index, so don't forget to strip the extra long headers down to just the scaffold name from the orf file to sync all headers together (i.e. using VIM).

```
>module load bowtie2
>bowtie2-build k29.scaffolds.assembly.800.orfs.prodigal_headers_removed.fa k29.MT.bowtie2.index
>qsub -l mppwidth=48,walltime=12:00:00 -q regular $GSCRATCH/bins/bowtie2_MT_script.sh
```

*.sam files from this mapping were processed into *.Count.* libraries in the same way as for scaffold mapping. One notable exception is that a *.gff file needs to be generated for the ORF fasta file as reference for counting scripts.

```
>generate_contig_len.py -i scaffolds.assembly.800.orfs.fa -o
scaffolds.assembly.800.contig.lengths.fa.txt
>gsed -e 's/\(^\S*\)\t\([0-9]*\)/\1\tprodigal\tCDS\t1\t\2\t\.\t+\t\./'
scaffolds.assembly.800.contig.lengths.txt > k45.MT.faa.gff
```

These *.Count.* libraries were then combined by extracting their name and length once, and then merging all count columns from each file:

```
>paste k29.MT.CR1BM.All.Counts.txt k29.MT.CR2B.All.Counts.txt k29.MT.CR2AM.All.Counts.txt
k29.MT.MR2A.All.Counts.txt k29.MT.MR2BM.All.Counts.txt k29.MT.CR3A.All.Counts.txt
k29.MT.CR3B.All.Counts.txt k29.MT.CR5B.All.Counts.txt k29.MT.MR5A.All.Counts.txt
k29.MT.CR6AM.All.Counts.txt k29.MT.CR6B.All.Counts.txt k29.MT.MR6A.All.Counts.txt
k29.MT.CR8ABM.All.Counts.txt | cut -d$'\t' -f1,2,3,6,9,12,15,18,21,24,27,30,33,36,39 >
k29.MT.All.Counts.txt
```

with the corresponding header line added in:

```
header	bplen	CR1BM		CR2B	CR2AM		MR2A	MR2BM		CR3A	CR3B	CR5B	MR5A
	CR6AM		CR6B	MR6A	CR8ABM
```

6 Integrating annotation and count datasets together and generating initial statistics

Once coverage, taxonomy, bin, and annotation files were generated for a single co-assembly (e.g. k=29, k29) they were combined by the following script into two files:

```
>Combine_counts_bins_func_ont_tax.py -c k29.MT.All.Counts.txt -b 10k_EPS_0_35_all_svm_bin.txt -f
mgm4599995.3.ordered.protein_rna.annotations.txt -y mgm4599995.3.ordered.ontology.annotations.txt -
t mgm4599995.3.top.taxonomy.tab -o k29.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered
```

Summary and data files:

```
#tab-delimited database of all annotations & RPKM
k29.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt

#tab-delimited database summarized and combined by ontology
k29.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.summary.txt
```

One notable difference from scaffold mapping is that duplicate gene sequences are common in gene-based mapping and should be marked and removed here (mark_duplicates.py). Additionally this output can be used to combine entries in the data table produced from Combine_counts_bins_func_ont_tax.py (sum_duplicate_mappings.py).

```
>mark_duplicates.py -i scaffolds.assembly.800.prodigal_headers_removed.fa -o
k29.combined.duplicated.txt
>sum_duplicate_mappings.py -i k29.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt -o
k29.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.depreplicated.data.txt
```

KGML files can be generated from KEGG using the Biopython or KGML renderer by Leighton Prichard (James Hutton Institue, UK, https://github.com/widdowquinn). This script will produce one map for each sample for each bin for each pathway and chart the relative expression level of KO annotated genes from summary files to produce a high-level overview of organism functioning. Subset the *.summary.* file to the bins and annotations desired using Excel and save as k29.combined.GN.KO.kgml.txt. The KEGG KGML KO reference maps and images desired for use as a basis will first need to be queried and placed in 1 directory (e.g. ~/KEGG).

```
>generate_kgml_graphics.py -i k29.combined.GN.KO.kgml.txt -d output_directory -K ~/KEGG/
```

Determine the simple majority consensus taxonomy of each bin and calculate historgrams and statistics for bins:

```
>consensus_taxonomy.py -i mgm4599995.3.top.taxonomy.filtered.tab -m 10k_EPS_0_35_all_svm_bin.txt -o
scaffolds.assembly.800.orfs.hmm.blat.tax.consensus.txt
>calculate_bin_stats.py -i k29.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt -o
k29.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.stats.txt
```

This section is repeated for each coassembly kmer word size (in this study k=29,k=45,k-63). At this point these data tables can be exported or subset to extract bins or samples for downstream analysis in Excel or MeV (http://sourceforge.net/projects/mev-tm4/). To subset bin k29.27 and select only the first time point (time points start at the 7th column) and then get the corresponding fasta gene sequences:

```
>grep -e '\tk29\.27\t' k45.2000733.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt | cut –f1-7 >
k29.27.bin.annotations.txt
>cut –f1 k29.27.bin.annotations.txt > k29.27.bin.list.txt
>filter_fasta_by_header –f k29.27.bin.list.txt -i scaffolds.assembly.800.orfs.hmm.fa
```

7 Combining and generating hybrid mapping reference files

In this study the k=29 (k29) coassembly indicated the largest number of complete bins but each coassembly had a different mix of bin quality for corresponding bins. Therefore, the most complete and lowest duplicate bins from each coassembly run was extracted and combined with the k29 coassembly. Annotations and scaffolds for bins from k45 and k63 were extracted, and their corresponding bin removed from k29. Bins from k29 were extracted, and remaining incomplete bins and background sequences were annotated as a background reference for transcript mapping. Additionally, closely related reference genomes were selected from RAST and their genes added to the references.

7.1 Selection of scaffold references

Selecting and combining bins from different assemblies:
e.g. pick bins 7,8,10,12,14,15,18 from k63, extract the headers, then filter the source fa file for bin sequences:

```
>grep -e '\t7\t\|\t8\t\|\t10\t\|\t12\t\|\t14\t\|\t15\t\|\t18\t'
k63.1373669.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt | gsed -e 's/^\(.*\)/k63\.\1/' >
k63.1373669.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.txt
>cut -f1 k63.1373669.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.txt >
k63.Scaffold.picked.list.txt
>filter_fasta_by_header.py -f k63.Scaffold.picked.list.txt -i scaffolds.assembly.800.orfs.fa
>mv scaffolds.assembly.800.orfs.filtered.fa k63.scaffold.800.orfs.picked.fa
```

Add a prefix to all scaffolds with VIM.

```
:%s/^>/>k63\./
```

Pick bins 6,27 from k45 (fasta filtering as above, not shown):

```
>grep -e '\t6\t\|\t27\t' k45.2000733.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt | gsed -e
's/^\(.*\)/k45\.\1/' > k45.2000733.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.txt
```

Pick bins 25,14,16,36,1,19,13,10,7,27,41 from k29 and set rest as background (fasta filtering not shown):

```
>grep -v -e '\t25\t\|\t14\t\|\t16\t\|\t36\t\|\t1\t\|\t19\t\|\t13\t\|\t10\t\|\t7\t\|\t27\t\|\t41\t'
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt | gsed 's/^\(.*\)/bk\.\1/' >
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.background.txt
>grep -e '\t25\t\|\t14\t\|\t16\t\|\t36\t\|\t1\t\|\t19\t\|\t13\t\|\t10\t\|\t7\t\|\t27\t\|\t41\t'
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt | gsed -e 's/^\(.*\)/k29\.\1/' >
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.txt
```

Remove duplicate bins 20,24,15,35,29,3,4,40,11 from k29:

```
>grep -v -e '\t20\|\t24\|\t15\|\t35\|\t29\|\t3\|\t4\|\t40\|\t11'
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.background.txt >
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.filtered.background.txt
```

Mark all entries with background using VIM in the *.background.txt.

```
:%s/\tk29\./\tbk\./
```

Then combine all annotation files and fasta files as below:

```
>cat ../1373669_files/k63.1373669.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.txt
../2000733_files/k45.2000733.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.txt
../1131940_files/k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.txt
../1131940_files/k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.filtered.background.txt >
full.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt
>cat ../1373669_files/k63.scaffold.800.orfs.picked.fa
../2000733_files/k45.scaffold.800.orfs.picked.fa ../1131940_files/k29.scaffold.800.orfs.picked.fa
../1131940_files/k29.scaffold.800.orfs.background.fa >
full.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.fa
```

For the annotation file modify for header to first line and remove header duplicates within the file, replace bin numbers with assembly prefix (e.g. k63. etc.) using VIM. The last step is to split the annotations out again for recombination as a new set of annotation datasets.

```
>cut -f1,2 full.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt > full.combined.bins.txt
>cut -f1,3 full.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt > full.combined.function.txt
>cut -f1,4 full.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt > full.combined.ontology.txt
>cut -f1,6 full.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt > full.combined.taxonomy.txt
```

7.2 Generating annotations for RAST references

Output from consensus_taxonomy.py (above) can be used to estimate which genomes to include in the mapping pipeline. The genes and annotations (as *.gff files) were downloaded from RAST. Note that there is inconsistency in genes, gene counts, and rna gene counts between the genomes and annotations from RAST and must be manually curated such that all genes have a corresponding annotation. Additionally, the KEGG orthology numbers given in RAST are the best gene match which happen to contain EC numbers and are inconsistently matched to KEGG EC reference pathways, and were intended for the KEGG EC reference pathway maps and not the KO reference system. A KO orthology matching REST API asynchronous http script was produced to match annotations to their respective KEGG orthology number (i.e.

K02586 - nifD; nitrogenase molybdenum-iron protein alpha chain [EC:1.18.6.1]). The complete list of genes from *.gff files was extracted:

```
>cat *.gff > all.gff
>cut -f8 all.gff | gsed -e 's/ID=//' | gsed -e 's/;Name=/\t/' | gsed -e 's/;Ontology=/\t/' >
kegg.genomes.annotation.txt
>cut -f2 kegg.genomes.annotation.txt | sort | uniq > unique.kegg.annotation.txt
>query_ko_from_rast_annotations.py -i unique.kegg.annotation.txt -o kegg.ko.annotations.txt -d
kegg_annot
```

This file was then parsed in excel and manual matches made for remaining unmatched genes to generate a final list of unique KEGG ontology assignments such that the final tab-delimited *.ontology.* file had the form: Gene_name\tmatched_ko|ID=%of_matches. Similarly, the gene function table was parsed to match function files produced from the MG-RAST pipeline.

Similarly, single term bin and taxonomy files for each gene of each reference genome was created:

```
>cut -f8 genome.A.gff | gsed -e 's/ID=\(.*\);Name=/\1\tgenome.A.name/' > genome.A.taxonomy.txt
>cat genome.*.taxonomy.txt genome.All.taxonomy.txt
>cp genome.All.taxonomy.txt genome.All.bin.txt
```

The metagenome bin, function, ontology annotations were merged with the RAST annotated reference genomes.

```
>cat genome.All.bin.txt full.combined.bins.txt > full.bins.txt
>cat genome.All.taxonomy.txt full.combined.taxonomy.txt > full.taxonomy.txt
>cat genome.All.function.txt full.combined.function.txt > full.function.txt
>cat genome.All.ontology.txt full.combined.ontology.txt > full.ontology.txt
>cat genomes.All.fa full.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.fa > full.hybrid.fa
```

8 Bowtie2 mapping of transcripts to selected hybrid metagenome reference

Similarly to section 7, metatranscripome libraries were mapped with bowtie2 to this hybrid reference. In brief:

```
>split_interleaved_fastq.py -i ElkSloSequencingTranscript.A.fastq
```

etc… for each library. Followed by bowtie2 database generation and execution.

```
>module load bowtie2
>bowtie2-build full.hybrid.fa full.hybrid.MT.bowtie2.index
>paste full.hybrid.MT.CR1BM.All.Counts.txt full.hybrid.MT.CR2B.All.Counts.txt
full.hybrid.MT.CR2AM.All.Counts.txt full.hybrid.MT.MR2A.All.Counts.txt
full.hybrid.MT.MR2BM.All.Counts.txt full.hybrid.MT.CR3A.All.Counts.txt
full.hybrid.MT.CR3B.All.Counts.txt full.hybrid.MT.CR5B.All.Counts.txt
full.hybrid.MT.MR5A.All.Counts.txt full.hybrid.MT.CR6AM.All.Counts.txt
full.hybrid.MT.CR6B.All.Counts.txt full.hybrid.MT.MR6A.All.Counts.txt
full.hybrid.MT.CR8ABM.All.Counts.txt | cut -d$'\t' -f1,2,3,6,9,12,15,18,21,24,27,30,33,36,39 >
full.hybrid.MT.All.Counts.txt
```

with the corresponding header line added in:
header bplen CR1BM CR2B CR2AM MR2A MR2BM CR3A CR3B CR5B MR5A CR6AM CR6B MR6A CR8ABM

```
>Combine_counts_bins_func_ont_tax.py -c full.hybrid.MT.All.Counts.txt -b
10k_EPS_0_35_all_svm_bin.txt -f mgm4599995.3.ordered.protein_rna.annotations.txt -y
mgm4599995.3.ordered.ontology.annotations.txt -t mgm4599995.3.top.taxonomy.tab -o
full.hybrid.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered
>mark_duplicates.py -i scaffolds.assembly.800.prodigal_headers_removed.fa -o
full.hybrid.combined.duplicated.txt
>sum_duplicate_mappings.py -i full.hybrid.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt -o
full.hybrid.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.depreplicated.data.txt
```

Subset the *.summary.* file to the bins and annotations desired using Excel and save as full.hybrid.combined.GN.KO.kgml.txt.

```
>generate_kgml_graphics.py -i full.hybrid.combined.GN.KO.kgml.txt -d output_directory -K ~/KEGG/
```

```
>consensus_taxonomy.py -i mgm4599995.3.top.taxonomy.filtered.tab -m 10k_EPS_0_35_all_svm_bin.txt -o scaffolds.assembly.800.orfs.hmm.blat.tax.consensus.txt
>calculate_bin_stats.py -i full.hybrid.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt -o full.hybrid.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.stats.txt
```

8.1 Searching the data file

A simple search script was written to query this database for specific search terms representing functional genes that can serve as an overview placeholder for metabolic pathways. This script creates large heatmap table summaries that can be used for figure generation in Excel or Illustrator.

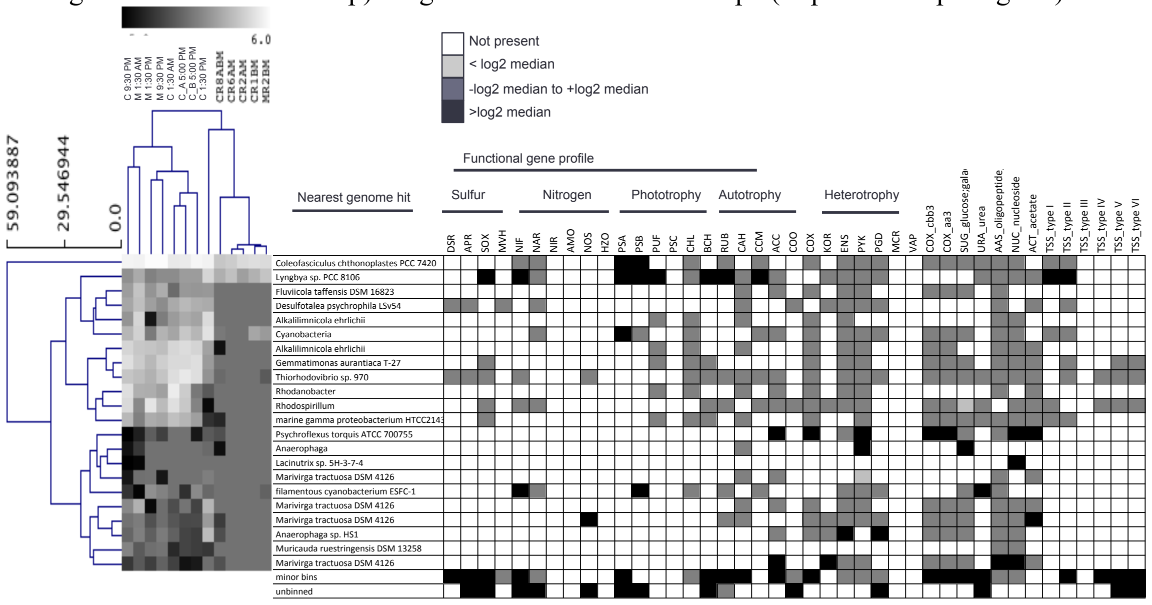
The search table format (in tab-delimited table) is:

abbreviation	name	chain or subunit	code	subunit	EC	exclude	nested
DSR	dissimilatory sulfite reductase|sulfite reductase, dissimilatory-type	alpha|beta |A|B	dsr	AB	1.8.99.1|1.8.99.2		
NIF	nitrogenase|dinitrogenase reductase|dinitrogenase	alpha|beta |H|D|K	nif	HDK	1.18.6.1	DRAT	
AMO	ammonia monooxygenase	alpha|beta |gamma|A |B|C	amo	ABC	1.14.99.39		
ENS	enolase|phosphopyruvate hydratase		ens		4.2.1.11		
PYK	pyruvate kinase		pyk		2.7.1.40		
PGD	6-phosphoglucanate dehydrogenase		pgd		1.1.1.44		
SUG	transporter|cotransporter|symporter|permease						glucose;galactose;maltose;arabinose;fucose;xylose;mannose

'Abbreviation' is the label used in tables. The search algorithm first searches for EC label matches (and subunit matches if present), then searches remaining for code and subunit matches (if present) and finally for name matches (with chain or subunit matches where present) for remaining annotations. At all steps terms with exclude values are dropped. After name searches, a nested subsearch can be used to break out name search terms. A new data column is created for each nested search term. The search uses the python regex library system and regular expression matching is possible. '|' indicates 'or' searches, ';' indicates unique searches.

```
>search_combined_data.py -i full.hybrid.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.txt -o full.hybrid.MT.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.searches.txt -q search.table.txt
```

Example figure showing a clustergram of the median bin transcript response (RPKM) for each bin over different samples (generated from *stats.txt file exported to Excel then to MeV for clustergrams) and showing the nearest genome match (top hit from consensus_taxonomy.py) and indicating abundance and taxonomic relatedness among bin expression profiles in this study. (right) Heat map of functional gene profiles as a $\log_2$ ratio to bin medians over all samples indicating putative metabolic lifestyle of putative bin (generated from search_combined_data.py exported to excel and generated into a heatmap). Figure was created in InkScape (https://inkscape.org/en/).



9 Create a metaproteomics reference database

Similarly to the hybrid mapping reference dataset, a hybrid metaproteomics reference database (without reference RAST genomes) was generated for proteomics studies.

```
>cut -f1 k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.txt >
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.list.txt
>cut -f1 k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.filtered.background.txt >
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.background.list.txt
>cut -f1,2,3,4,5,6 k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.txt >
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.annotations.txt
```

Replace bin numbers with k29 or bk prefix using VIM in the annotations data file:

```
:%s/\(\S*\.\)\(scaffold-\d*_\d*\t\)\(\d*\)\t/\1\2\1\3\t/
```

Followed by filtering the picked sequences from respective assemblies, adding in annotations, and joining them together in 1 fasta file.

```
>filter_fasta_by_header.py -i scaffolds.assembly.800.orfs.faa -f
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.list.txt
>add_annotations_to_fasta.py -i scaffolds.assembly.800.orfs.filtered.k29.faa -a
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.annotations.txt -o
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.annotations.faa
>cat k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.annotations.faa
../2000733_files/k45.2000733.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.annotations.faa
../1373669_fasta/k63.1373669.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.annotations.faa
k29.1131940.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.background.annotations.faa >
k63_45_29.combined.IP.GN.RS.IMG.PA.SE.KO.ordered.data.picked.annotations.faa
```

This file combined with the annotations file can then be formatted to the desired custom sequence database for proteomics interrogation software (not documented).

References

[1] S. Boisvert, F. Raymond, É. Godzaridis, F. Laviolette, and J. Corbeil, “Ray Meta: scalable de novo metagenome assembly and profiling,” *Genome Biol.*, vol. 13, no. 12, p. R122, Dec. 2012.

[2] R. Chikhi and P. Medvedev, “Informed and automated k-mer size selection for genome assembly,” *Bioinformatics*, vol. 30, no. 1, pp. 31–37, Jan. 2014.

[3] B. Langmead and S. L. Salzberg, “Fast gapped-read alignment with Bowtie 2,” *Nat. Methods*, vol. 9, no. 4, pp. 357–359, Apr. 2012.

[4] M. Albertsen, P. Hugenholtz, A. Skarshewski, K. L. Nielsen, G. W. Tyson, and P. H. Nielsen, “Genome sequences of rare, uncultured bacteria obtained by differential coverage binning of multiple metagenomes,” *Nat. Biotechnol.*, vol. 31, no. 6, pp. 533–538, Jun. 2013.

[5] D. Hyatt, G.-L. Chen, P. F. LoCascio, M. L. Land, F. W. Larimer, and L. J. Hauser, “Prodigal: prokaryotic gene recognition and translation initiation site identification,” *BMC Bioinformatics*, vol. 11, no. 1, p. 119, Mar. 2010.

[6] S. R. Eddy, “Accelerated Profile HMM Searches,” *PLoS Comput Biol*, vol. 7, no. 10, p. e1002195, Oct. 2011.

[7] F. Meyer, D. Paarmann, M. D’Souza, R. Olson, E. M. Glass, M. Kubal, T. Paczian, A. Rodriguez, R. Stevens, A. Wilke, J. Wilkening, and R. A. Edwards, “The metagenomics RAST server – a public resource for the automatic phylogenetic and functional analysis of metagenomes,” *BMC Bioinformatics*, vol. 9, no. 1, p. 386, Sep. 2008.

List of scripts and files

```
Albertsen scripts                       # https://github.com/MadsAlbertsen/multi-metagenome

bowtie2_script.sh                       #qsub script for running bowtie2 on NERSC Edison
process_sams.sh                         #script to convert bowtie2 *.sam files to *.Count files
process_sams_batch.sh                   #qsub script to convert bowtie2 *.sam files to *.Count
                                        files
svm.binning.workflow.R                  #script binning, analysis, and export for RStudio

add_annotations_to_fasta.py             #merge an annotations file into fasta headers
calculate_bin_stats.py                  #compute histograms, medians, and normalize bin RPKMs
Combine_counts_bins_func_ont_tax.py     #integrate all datasets into one tab-delimited file
consensus_taxonomy.py                   #determine the simple majority consensus at species level
contig_coverage_from_perbase_gff.py     #count coverage average per base stats for entire contigs
filter_fasta_by_len.py                  #filter sequences based on threshold above or below
filter_fasta_by_header.py               #filter sequences based on reference list
filter_line_by_list.py                  #filter tab-delimited file based on reference list
generate_contig_len.py                  #calculate the length of contigs and write to tab-delimited
generate_kgml_graphics.py               #make KEGG pathway maps for each bin and time point
mark_duplicates.py                      #output a list of duplicate sequences in a fasta file
pick_top_mgrast_sims_md5.py             #select the top mgrast annotation for all entries
query_ko_from_rast_annotations.py       #query KEGG API for RAST ontology matches
query_mgrast_md5_annotation.py          #find the best protein and ontology match for all entries
query_mgrast_md5_taxonomy.py            #find the taxonomy match for all entries
search_combined_data.py                 #simple search algorithm for creating functional gene table
split_interleaved_fastq.py              #create F/R fastq files from an Illumina interleaved fastq
sum_duplicate_mappings.py               #combine duplicates in coverage/annotation database
```

Acknowledgements

DOE Scientific Focus Area – A systems biology approach to energy flow in biofuel-producing microbial communities
Funding was provided by the U.S. Department of Energy (DOE) **Genomic Science Program** under contract SCW1039.
This work was performed under the auspices of the U.S. Department of Energy by **Lawrence Livermore National Laboratory** under Contract DE-AC52-07NA27344.
This research used resources of the **National Energy Research Scientific Computing Center**, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.