J. CRAIG VENTER INSTITUTE

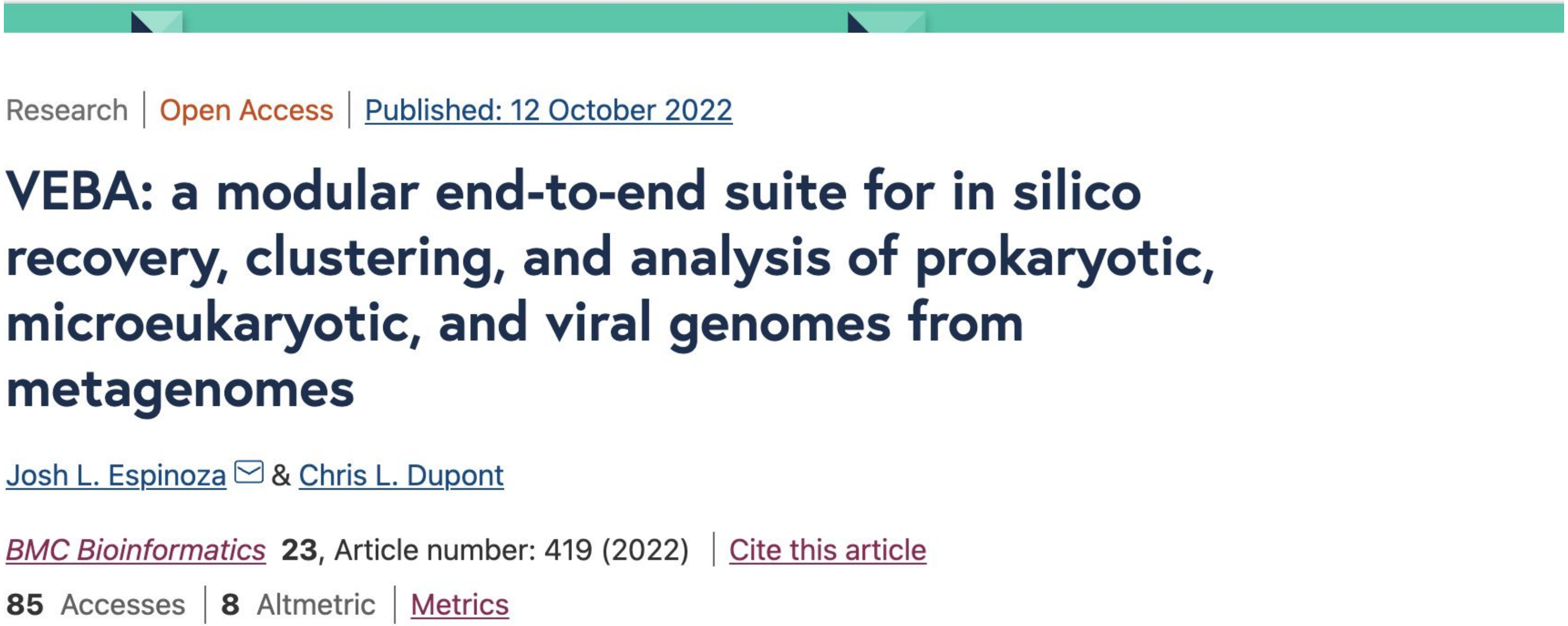
VEBA:

a modular end-to-end suite for *in silico* recovery, clustering, and analysis of prokaryotic, microeukaryotic, and viral genomes from metagenomes

Josh L. Espinoza, Ph.D
Staff Scientist
Dupont Lab

Department of Environment and Sustainability, *J. Craig Venter Institute*
Department of Human Biology and Genomic Medicine, *J. Craig Venter Institute*

Peer-Reviewed Publication



Research | Open Access | Published: 12 October 2022

VEBA: a modular end-to-end suite for in silico recovery, clustering, and analysis of prokaryotic, microeukaryotic, and viral genomes from metagenomes

Josh L. Espinoza ✉ & Chris L. Dupont

BMC Bioinformatics **23**, Article number: 419 (2022) | Cite this article

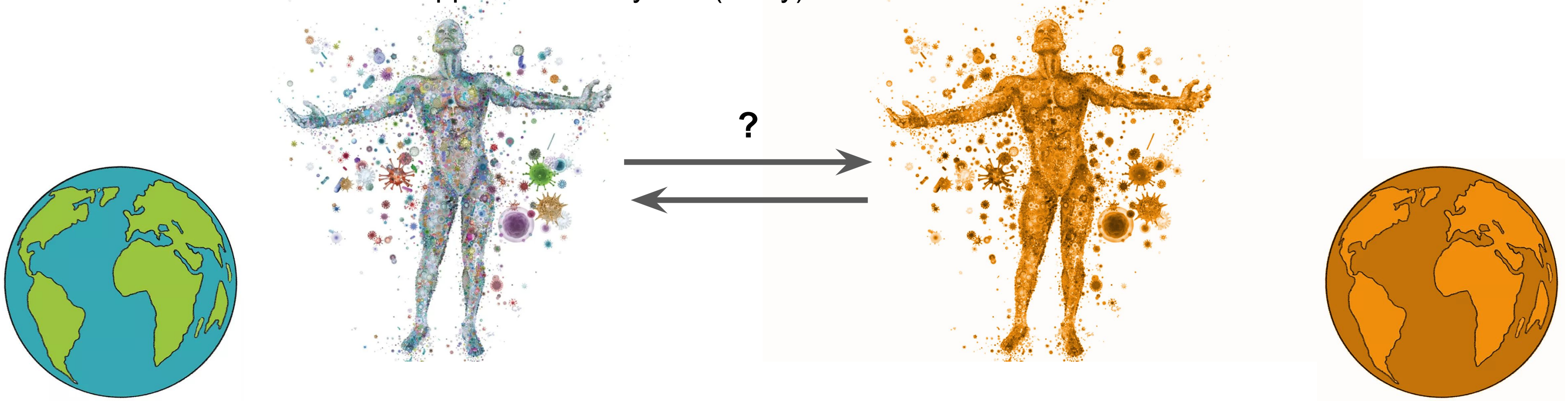
85 Accesses | **8** Altmetric | Metrics

Microbial ecology in the larger context

- Microorganisms provide insight into ecosystem resilience, sustainability, and human health

- Cataloguing and preserving biodiversity is paramount for discovering potential solutions to challenges we face as a growing civilization

- Metagenomics pertains to the *in silico* study of microorganisms within an ecosystem *in situ*
    - Most metagenomics suites have conflicting dependencies
    - Most metagenomics suites only support prokaryotes
        - *Candidate phyla radiation* (CPR) support require manual *post hoc* workflows
        - Limited support for viruses (if any)
        - Limited support for eukaryotes (if any)



Why use *VEBA*?

- Directly recovers, quality assess, and classify prokaryotic, eukaryotic, and viral genomes from metagenomes/metatranscriptomes

- Automated handling of CPR

- Modular to accommodate multiple workflows

- Automates complex tasks in a user-friendly way

- Maximizes information gain from available data

- Can be used with co-assemblies or sample-specific assemblies (+ pseudo-coassemblies)

- Implements clustering at the species and protein level to make sample-specific genomes comparable across multiple samples

- All packages and dependencies are open-sourced (no complicated licensing)

- *VEBA* is installed in one command (i.e., **`bash install_veba.sh`**)

- *VEBA* database is downloaded/configured in one command (i.e., **`bash download_databases.sh /path/to/veba_database`**)

VEBA Modules

- ***preprocess*** – Fastq quality trimming, adapter removal, decontamination, and read statistics calculations
- ***assembly*** – Assemble reads, align reads to assembly, and count mapped reads
- ***coverage*** – Align reads to (concatenated) reference and counts mapped reads

- ***binning-prokaryotic*** – Iterative consensus binning for recovering prokaryotic genomes with lineage-specific quality assessment
- ***binning-eukaryotic*** – Binning for recovering eukaryotic genomes with exon-aware gene modeling and lineage-specific quality assessment
- ***binning-viral*** – Detection of viral genomes and quality assessment

- ***classify-prokaryotic*** – Taxonomic classification and candidate phyla radiation adjusted quality
- ***classify-eukaryotic*** – Taxonomic classification of eukaryotic genomes
- ***classify-viral*** – Taxonomic classification and isolation source of viral genomes
- ***annotate*** – Annotates translated gene calls against NR, Pfam, and KOFAM

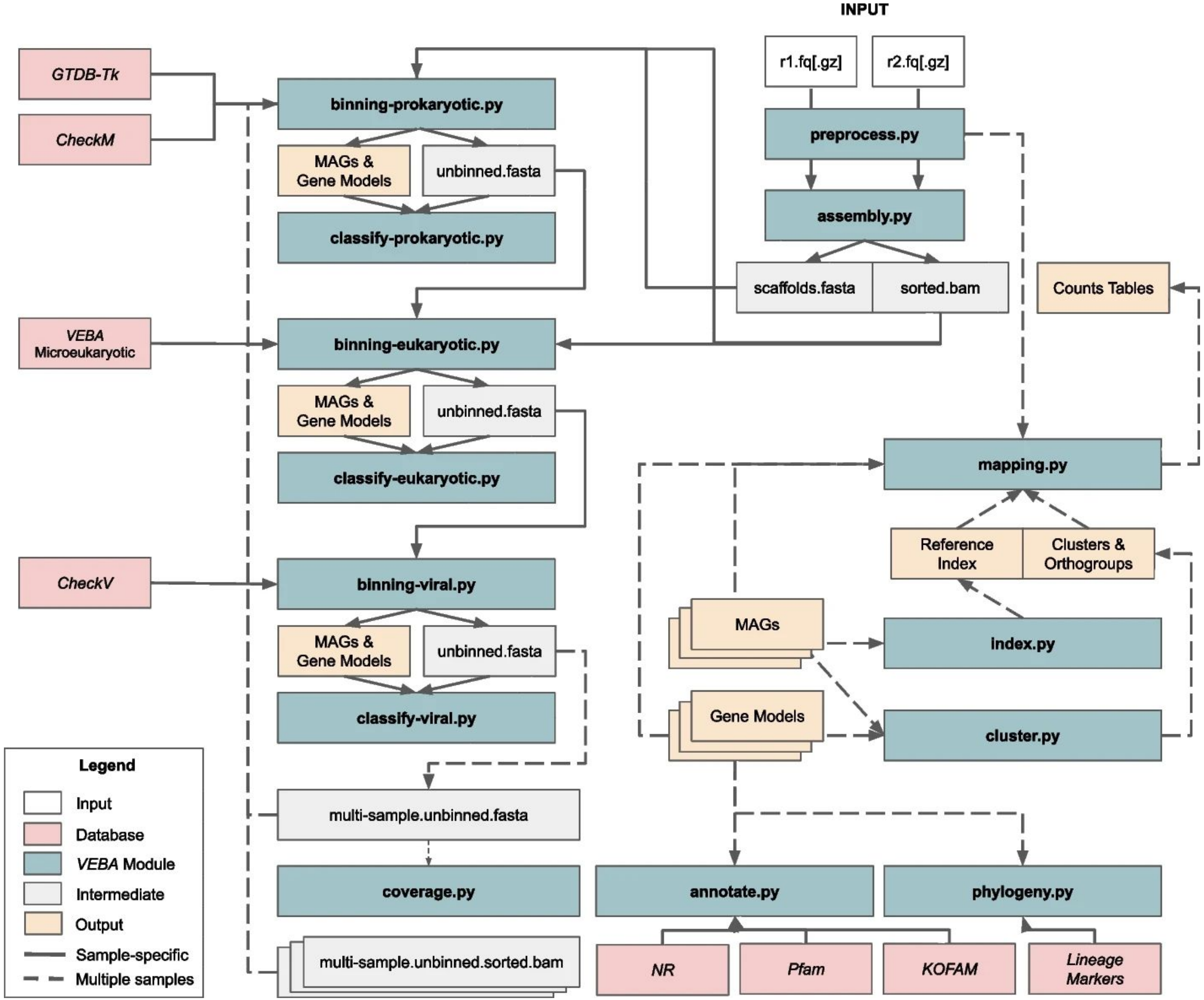
- ***cluster*** – Species-level clustering of genomes and lineage-specific orthogroup detection
- ***phylogeny*** – Constructs phylogenetic trees given a marker set

- ***index*** – Builds local or global index for alignment to genomes
- ***mapping*** – Aligns reads to local or global index of genomes

Legend

□	Preprocessing
🟩	Identification
🟦	Annotation
🟨	Structural
🟧	Quantification

VEBA Modules



INPUT

r1.fq[.gz]

r2.fq[.gz]

preprocess.py

assembly.py

scaffolds.fasta

sorted.bam

Counts Tables

GTDB-Tk

CheckM

binning-prokaryotic.py

MAGs & Gene Models

unbinned.fasta

classify-prokaryotic.py

VEBA Microeukaryotic

binning-eukaryotic.py

MAGs & Gene Models

unbinned.fasta

classify-eukaryotic.py

CheckV

binning-viral.py

MAGs & Gene Models

unbinned.fasta

classify-viral.py

mapping.py

Reference Index

Clusters & Orthogroups

MAGs

index.py

Gene Models

cluster.py

multi-sample.unbinned.fasta

coverage.py

multi-sample.unbinned.sorted.bam

annotate.py

phylogeny.py

NR

Pfam

KOFAM

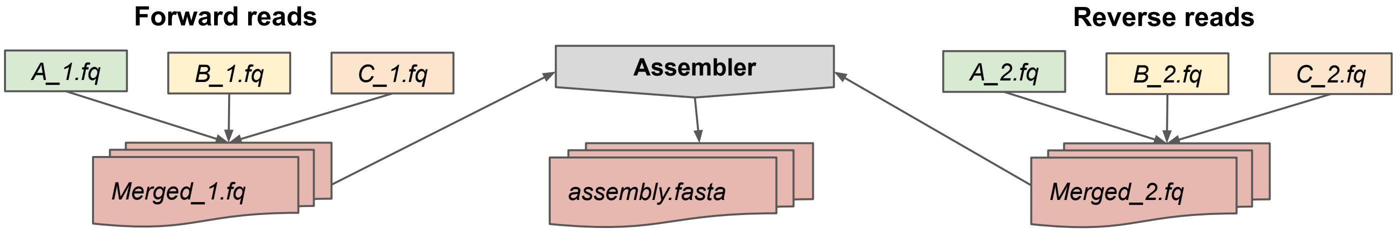
Lineage Markers

Legend

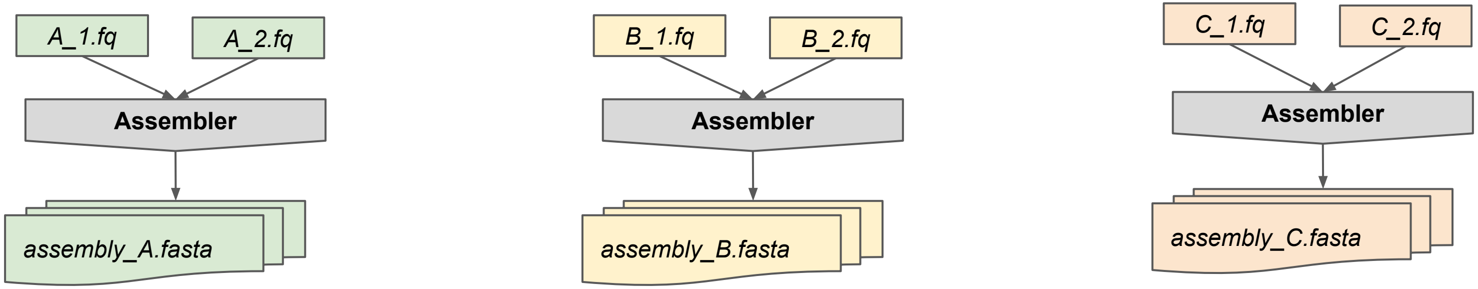
- □ Input
- 🟥 Database
- 🟦 *VEBA* Module
- ⬜ Intermediate
- 🟧 Output
- ― Sample-specific
- - - Multiple samples

What’s the difference between co-assembly and sample-specific metagenomics?

- **Co-assembly**
  - Performing assembly when reads are from multiple samples (i.e., concatenated)



- **Sample-specific**
  - Performing assembly for *N* samples individually resulting in *N* separate assemblies



What’s the difference between co-assembly and sample-specific metagenomics?

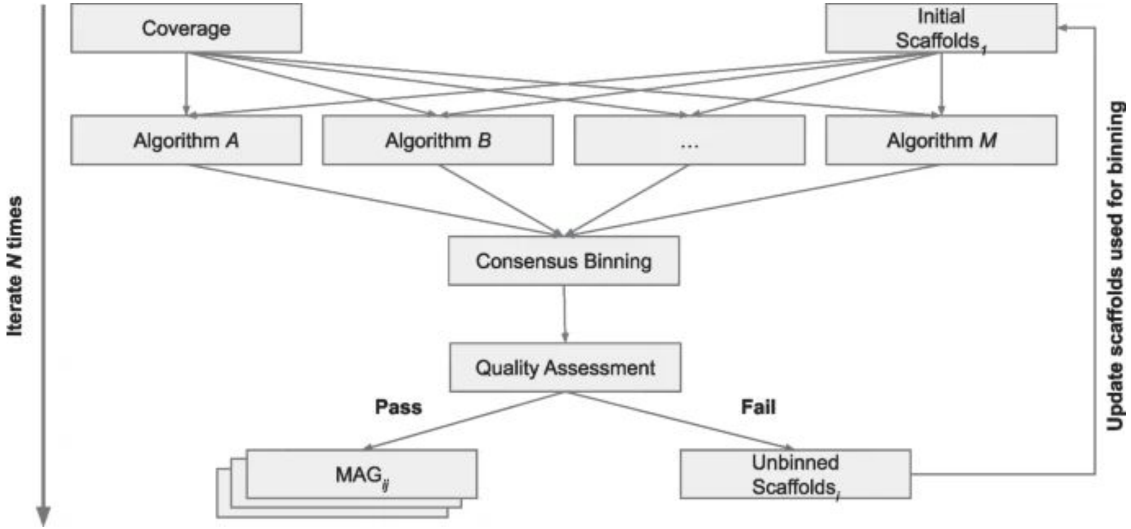
- **Co-assembly**
  - **Pros**:
    - Can increase read depth for low depth samples
    - Allows for direct comparison of features all samples
    - Coverage from multiple samples helps binning
  - **Cons**:
    - Can result in composite genomes that are not biologically accurate
    - Can result in many unbinned contigs
    - Requires much more computational power

- **Sample-specific assemblies**
  - **Pros:**
    - Can recover sample-specific individual strains instead of composites of multiple strains
    - Uses much less compute resources
  - **Cons:**
    - Not as useful in low depth samples
    - Cannot directly compare abundances of biological features

How does *VEBA* maximize available information for genome recovery?

- Uses iterative binning to feed unbinned contigs back into binning (currently, only implemented for prokaryotes)

- [Optional] Uses a “pseudo-coassembly” that makes use of unbinned contigs from multiple samples



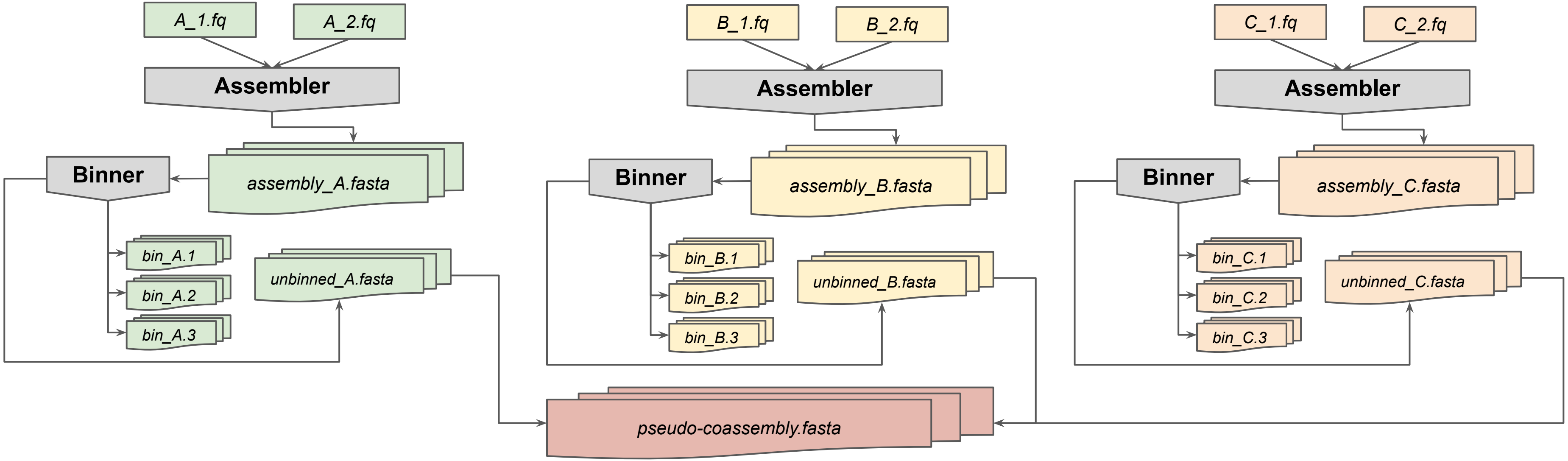
- **Standardized parseable naming scheme**:
  - [SampleID]_[Algorithm]_DomainPrefix.[Iteration]_[Name]
  - SRR17458623_METABAT2_P.1_bin.1

- **P for prokaryotes**
- **E for eukaryotes**

What’s the difference between *bona fide* co-assembly and pseudo-coassembly?

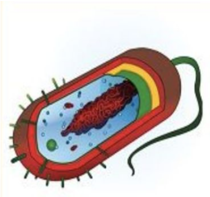
- **Co-assembly**
  - Concatenate all forward reads (i.e., `cat *_1.fastq > concat_1.fastq`)
  - Concatenate all reverse reads (i.e., `cat *_2.fastq > concat_2.fastq`)
  - Assemble concatenated reads (i.e., `coassembly.fasta`)

- **Pseudo-coassembly**
  - No additional assembly involved after sample-specific assembly
  - Concatenate all unbinned contigs from sample-specific binning (i.e., `cat unbinned_*.fasta > pseudo-coassembly.fasta`)



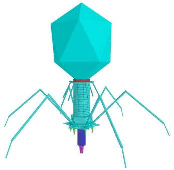
How does *VEBA* recover high-quality genomes from all domains?

- **Prokaryotes**
  - **Recovery** - Iterative consensus binning (*MaxBin2*|*MetaBAT2*|*CONCOCT* → *DAS Tool*)
  - **Gene calls** - *Prodigal* in metagenomics mode
  - **Quality assess** - *CheckM* with automated workflow to handle CPR
  - **Classification** - *GTDBTk*



- **Eukaryotes**
  - **Recovery** - Binning using either *MetaBAT2* or *CONCOCT*
  - **Gene calls** - *MetaEuk* exon-aware gene modeling using custom *VEBA* database
  - **Validation** - *Tiara* to predict if genome is eukaryotic
  - **Quality assess** - *BUSCO*
  - **Classification** - *VEBA* sub-module that uses *MetaEuk* gene targets and bitscores

- **Viruses**
  - **Recovery** - Binning using *VirFinder*
  - **Gene calls** - *Prodigal* in metagenomics mode
  - **Quality assess** - *CheckV*
  - **Classification** - *VEBA* sub-module that uses *CheckV* references



Why does *VEBA* introduce YET ANOTHER eukaryotic database?!?

- **Current eukaryotic databases either do not target microeukaryotes and/or biased towards marine research**
    - ~~Useful for human microbiomes~~
    - ~~Useful for build microbiomes~~

- **Consensus database using microeukaryotic proteins from the following sources:**
    - *MMETSP*
    - *EukZoo*
    - *EukProt*
    - NCBI non-redundant

- **Used for gene modeling and taxonomy classification**

- **Streamlined by removing prokaryotic and higher eukaryotes**

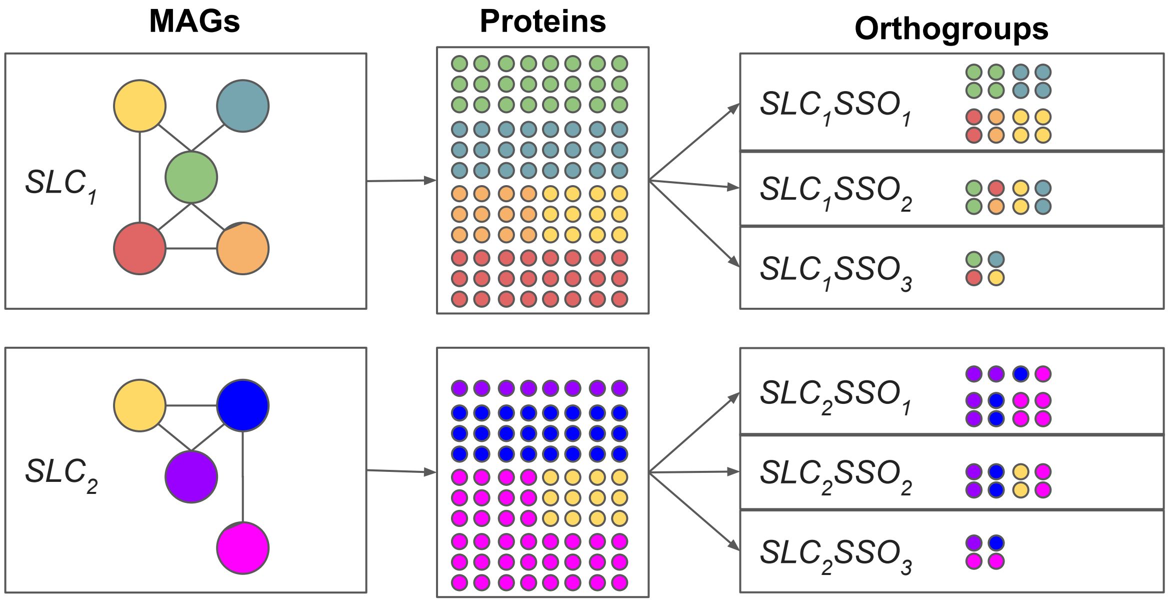
- **Contains 48,006,918 proteins sequences from 42,922 unique species**

- **Available on FigShare (10.21 GB)**
    - https://figshare.com/articles/dataset/Microeukaryotic_Protein_Database/19668855/1

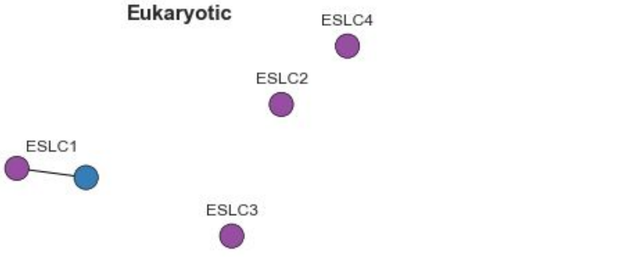
Clustering at the taxonomic and functional level

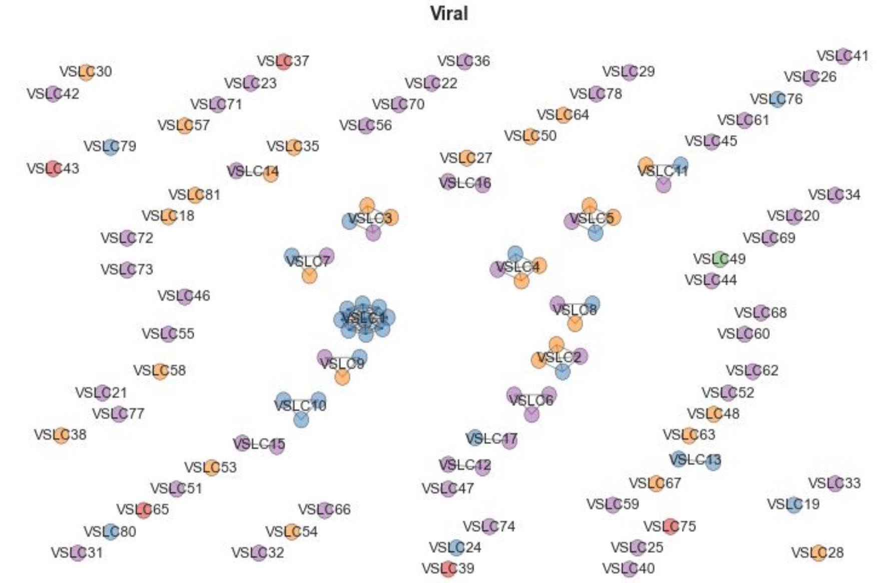
- **Taxonomic**
  - *Metagenome-assemble genomes* (MAG) are clustered by *average nucleotide identity* (ANI)
  - MAGs that cluster at 95% ANI are a *Species-level cluster* (SLC)

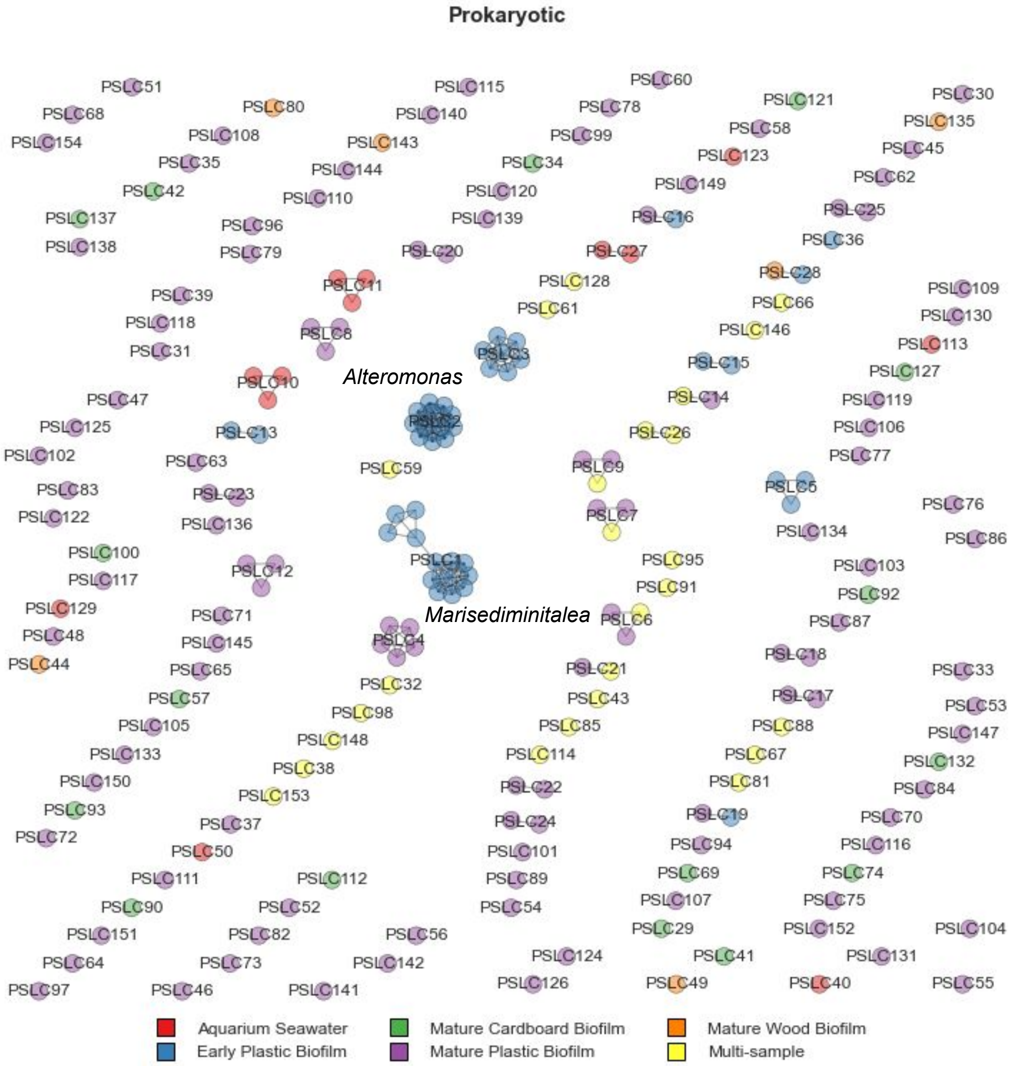
- **Functional**
  - All proteins within a SLC are clustered into *SLC-specific orthogroups* (SSO)



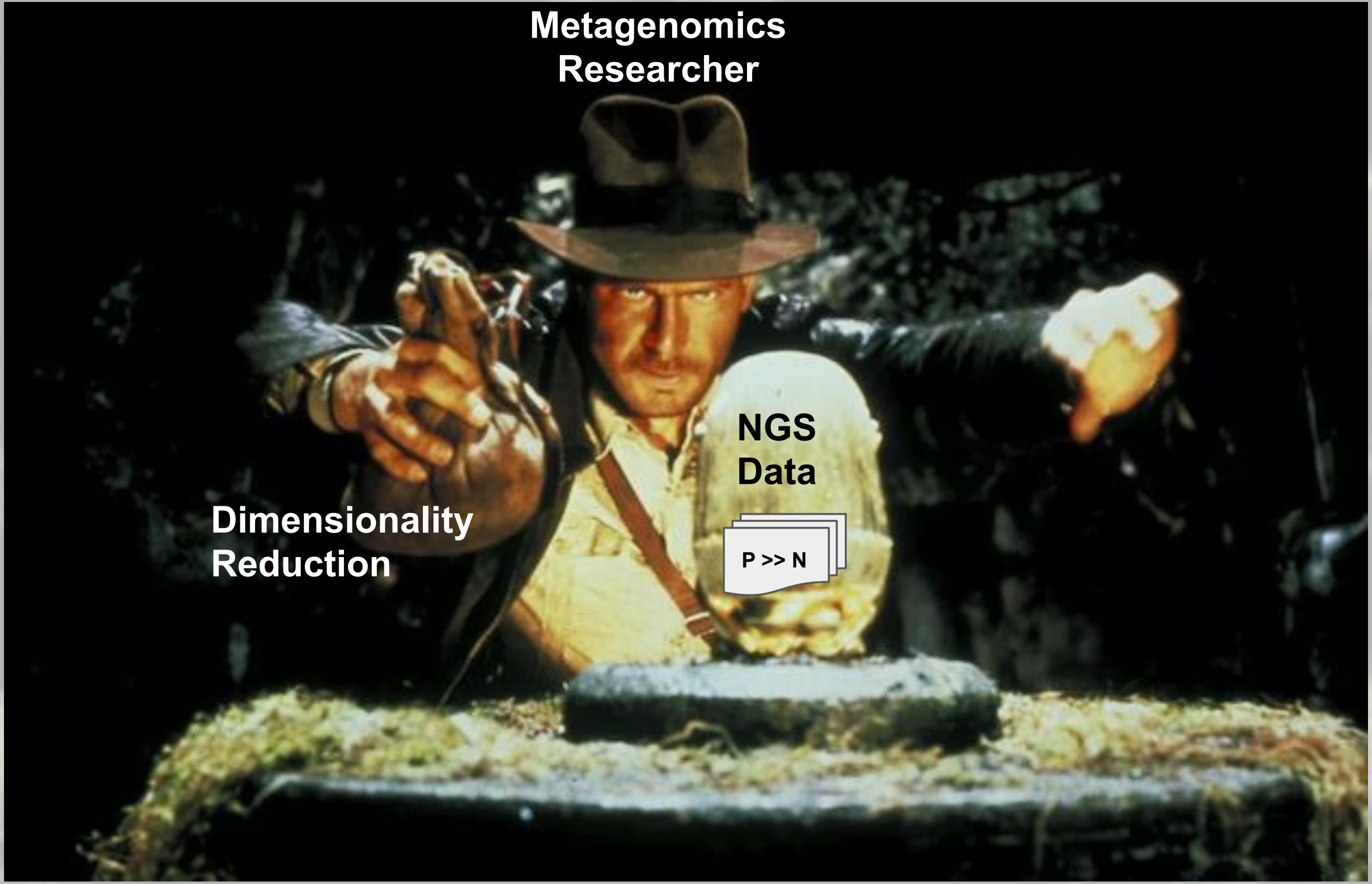
Species-level clustering (IRL)







Curse(s) of dimensionality



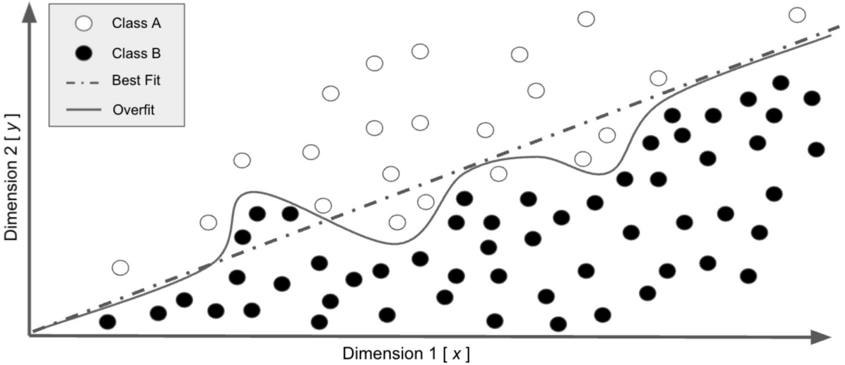
Curse(s) of dimensionality

- The curse of dimensionality refers to various phenomena that arise when analyzing and organizing data in high-dimensional spaces

- "*As the number of features or dimensions grows, the amount of data we need to generalize accurately grows exponentially.*"

  - Charles Isbell, Professor and Senior Associate Dean, School of Interactive Computing, Georgia Tech

- Most NGS datasets (Number of features >> number of observations)
  - Large P, Small N (P >> N)
  - Machine learning algorithms tend to overfit
  - Relationships between observations can be misleading



Using clustering for dimensionality reduction

- MAGs in the same cluster share highly similar genetic segments

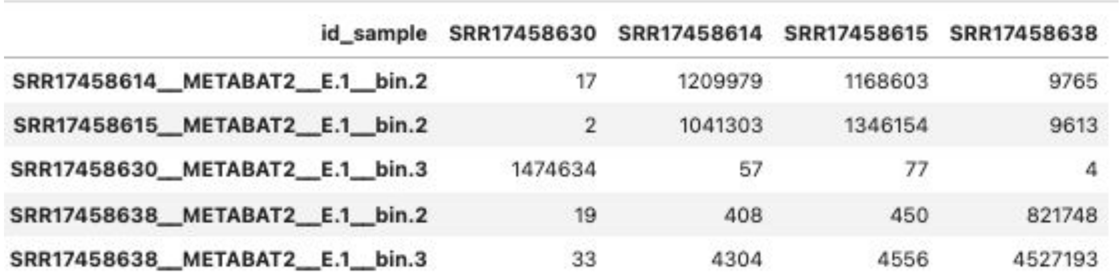
- Mapping reads to MAGs in the same cluster will randomly assign read to one MAG

- Summing MAG counts w.r.t to SLC negates this randomness

- Same for *Genes* → *SSOs*

Samples

MAGs

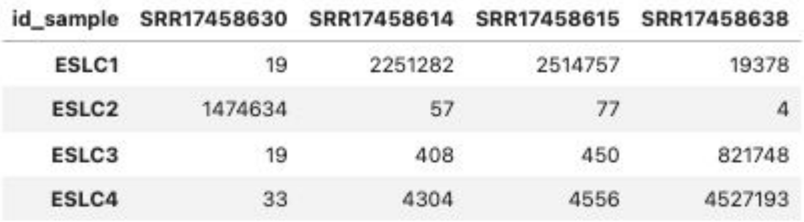


id_sample	SRR17458630	SRR17458614	SRR17458615	SRR17458638
SRR17458614__METABAT2__E.1__bin.2	17	1209979	1168603	9765
SRR17458615__METABAT2__E.1__bin.2	2	1041303	1346154	9613
SRR17458630__METABAT2__E.1__bin.3	1474634	57	77	4
SRR17458638__METABAT2__E.1__bin.2	19	408	450	821748
SRR17458638__METABAT2__E.1__bin.3	33	4304	4556	4527193

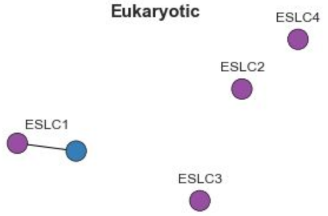
Group MAGs by SLCs and sum counts

Samples

SLCs

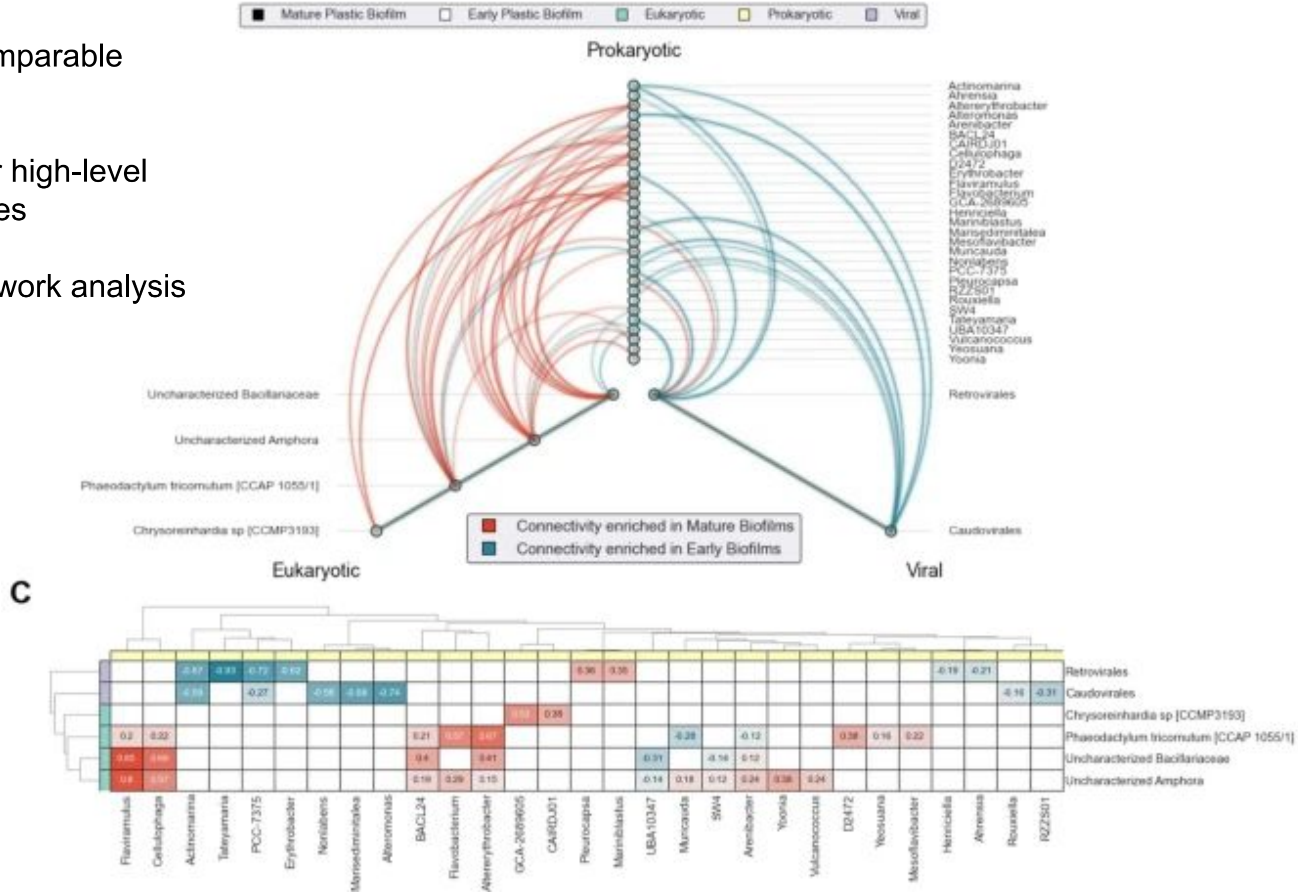


id_sample	SRR17458630	SRR17458614	SRR17458615	SRR17458638
ESLC1	19	2251282	2514757	19378
ESLC2	1474634	57	77	4
ESLC3	19	408	450	821748
ESLC4	33	4304	4556	4527193



How can we put this all together?

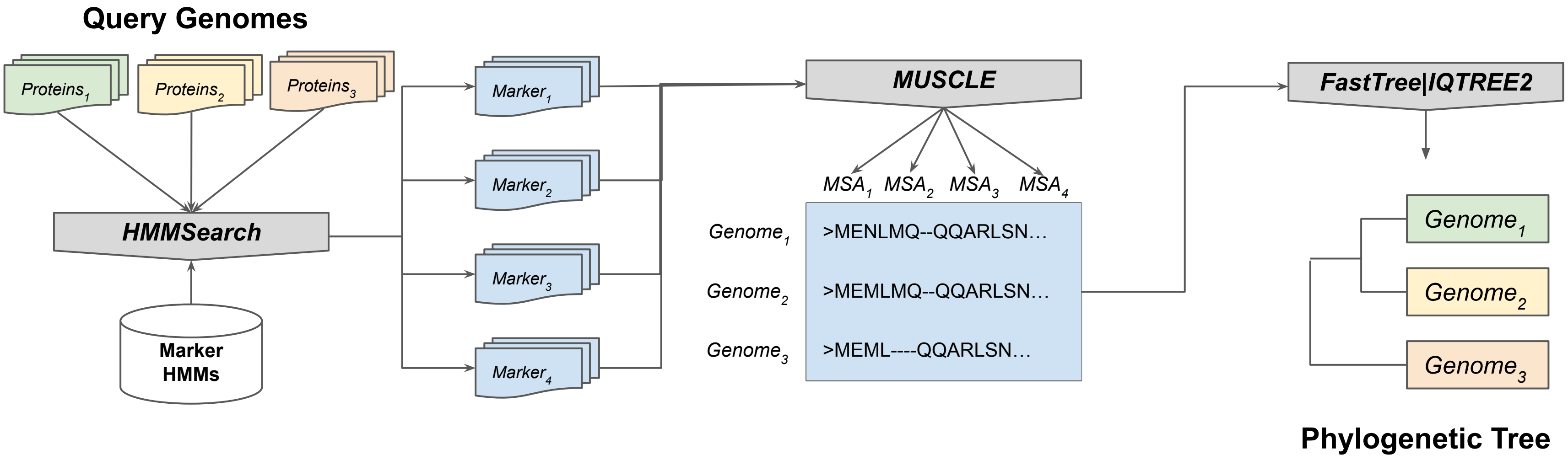
- Abundances/Expression comparable across samples
- Grouping features allows for high-level relationships between classes
- Multi-domain differential network analysis



Phylogenetic inference using concatenated alignments

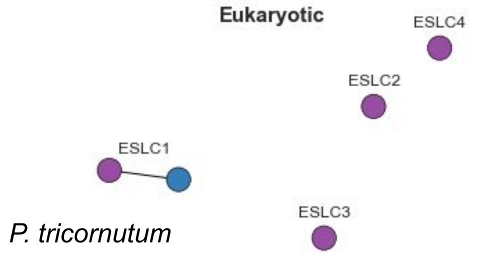
***VEBA* includes the following HMM marker set:**

- Archaea_76.hmm - (*Anvi'o*), Lee
- Bacteria_71.hmm - (*Anvi'o*), Lee
- Protista_83.hmm - (*Anvi'o*), Delmont
- Fungi_593.hmm - (*FGMP*)
- CPR_43.hmm - (*CheckM*)
- eukaryota_odb10 - (*BUSCO*)



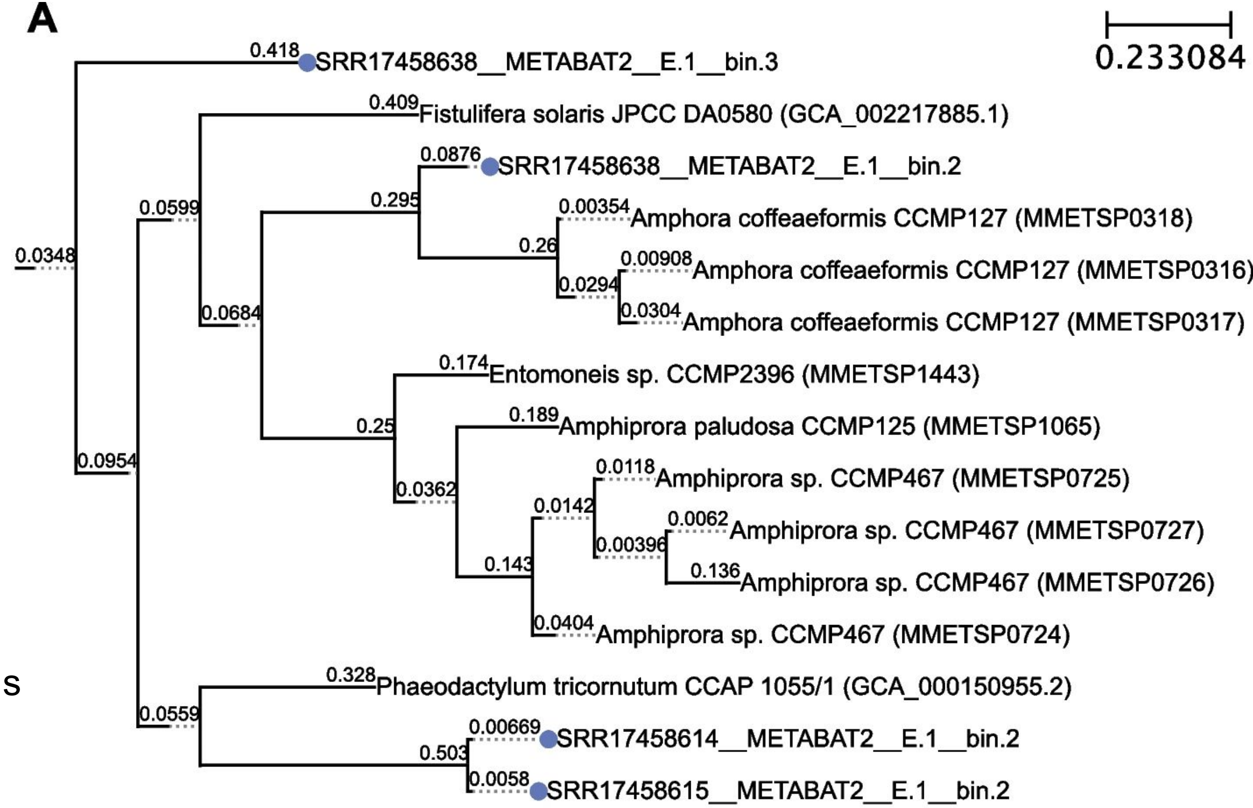
Genome$_1$	>MENLMQ--QQARLSN…
Genome$_2$	>MEMLMQ--QQARLSN…
Genome$_3$	>MEML----QQARLSN…

Phylogenetic inference (IRL)

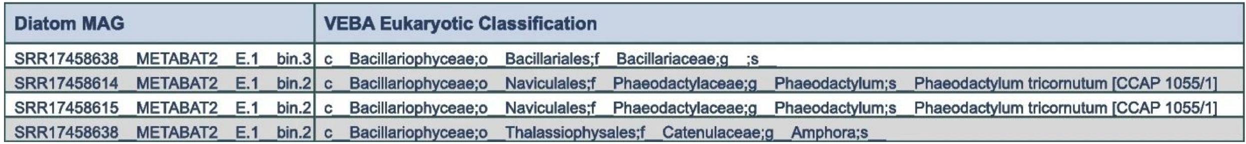


Taxonomic classifications reflect trends seen in phylogenetic inference

A


0.233084
0.418
SRR17458638__METABAT2__E.1__bin.3
0.409
Fistulifera solaris JPCC DA0580 (GCA_002217885.1)
0.0876
SRR17458638__METABAT2__E.1__bin.2
0.295
0.0599
0.00354
Amphora coffeaeformis CCMP127 (MMETSP0318)
0.0348
0.26
0.00908
Amphora coffeaeformis CCMP127 (MMETSP0316)
0.0294
0.0304
Amphora coffeaeformis CCMP127 (MMETSP0317)
0.0684
0.174
Entomoneis sp. CCMP2396 (MMETSP1443)
0.25
0.189
Amphiprora paludosa CCMP125 (MMETSP1065)
0.0954
0.0118
Amphiprora sp. CCMP467 (MMETSP0725)
0.0362
0.0142
0.0062
Amphiprora sp. CCMP467 (MMETSP0727)
0.00396
0.143
0.136
Amphiprora sp. CCMP467 (MMETSP0726)
0.0404
Amphiprora sp. CCMP467 (MMETSP0724)
0.328
Phaeodactylum tricornutum CCAP 1055/1 (GCA_000150955.2)
0.0559
0.00669
SRR17458614__METABAT2__E.1__bin.2
0.503
0.0058
SRR17458615__METABAT2__E.1__bin.2

B



Diatom MAG	VEBA Eukaryotic Classification
SRR17458638__METABAT2__E.1__bin.3	c__Bacillariophyceae;o__Bacillariales;f__Bacillariaceae;g__;s__
SRR17458614__METABAT2__E.1__bin.2	c__Bacillariophyceae;o__Naviculales;f__Phaeodactylaceae;g__Phaeodactylum;s__Phaeodactylum tricornutum [CCAP 1055/1]
SRR17458615__METABAT2__E.1__bin.2	c__Bacillariophyceae;o__Naviculales;f__Phaeodactylaceae;g__Phaeodactylum;s__Phaeodactylum tricornutum [CCAP 1055/1]
SRR17458638__METABAT2__E.1__bin.2	c__Bacillariophyceae;o__Thalassiophysales;f__Catenulaceae;g__Amphora;s__

Tutorial walkthroughs for different meta-omics workflows

- **Downloading and preprocessing fastq files** - Explains how to download reads from NCBI and run *VEBA's* `preprocess.py`module to decontaminate either metagenomic and/or metatranscriptomic reads.
- **Complete end-to-end metagenomics analysis** **-** Goes through assembling metagenomic reads, binning, clustering, classification, and annotation. We also show how to use the unbinned contigs in a pseudo-coassembly with guidelines on when it's a good idea to go this route.
- **Recovering viruses from metatranscriptomics** - Goes through assembling metatranscriptomic reads, viral binning, clustering, and classification.
- **Read mapping and counts tables** - Read mapping and generating counts tables at the contig, MAG, SLC, ORF, and SSO levels.
- **Phylogenetic inference** - Phylogenetic inference of eukaryotic diatoms.
- **Setting up *bona fide* co-assemblies for metagenomics or metatranscriptomics** - In the case where all samples are of low depth, it may be useful to use coassembly instead of sample-specific approaches. This walkthrough goes through concatenating reads, creating a reads table, coassembly of concatenated reads, aligning sample-specific reads to the coassembly for multiple sorted BAM files, and mapping reads for scaffold/transcript-level counts.

Conclusion

- *VEBA* is a user-friendly metagenomics/metatranscriptomics software suite
- *VEBA* is all open-source so you don't have to deal with annoying licenses (e.g., *GeneMark-EP*+)
- *VEBA* can handle:
  - Prokaryotes with direct support for CPR
  - Eukaryotes
  - Viruses
  - Sample-specific assemblies
  - Co-assemblies
  - Genomes from other pipelines/references
- *VEBA* is modular and can be used at many different stages of analysis
- *VEBA* has walkthroughs as step-by-step guides for different common workflows
- *VEBA* makes very complicated workflows extremely easy

Questions?

- **E-mail**
  - jespinoz@jcvi.org

- **LinkedIn:**
  - https://www.linkedin.com/in/jolespin/

- **Soothsayer Ecosystem (GitHub)**
  - ***VEBA*** (https://github.com/jolespin/veba)
    - A modular end-to-end suite for in silico recovery, clustering, and analysis of prokaryotic, microeukaryotic, and viral genomes from metagenomes
  - ***Soothsayer*** (https://github.com/jolespin/soothsayer)
    - High-level analysis package for (bio-)informatics
  - ***Ensemble NetworkX* (**https://github.com/jolespin/ensemble_networkx**)**
    - Ensemble networks in Python
  - ***Hive NetworkX* (**https://github.com/jolespin/hive_networkx**)**
    - Hive plots in Python
  - ***Compositional*** (https://github.com/jolespin/compositional)
    - Compositional data analysis in Python
  - ***GenoPype*** (https://github.com/jolespin/genopype)
    - Architecture for creating bash pipelines, in particular, for bioinformatics

VEBA Module Specifics

preprocess.py — Fastq quality trimming, adapter removal, and decontamination

- **Workflow:**
  - Wrapper around ***fastq_preprocessor*** (A “modernized” reimplementation of ***KneadData***)
  - Automatic quality trimming and adapter removal and with ***FastP***
  - [Optional] Removal/quantification of contamination based if reference provided:
    - ***Bowtie2*** - alignment based (e.g., removing human reads)
    - ***BBDuk*** - *k*-mer based (e.g., removing ribosomal reads)
    - Can quantify but not store read subsets (e.g., count ribosomal hits but don’t save them)
  - Calculate read statistics used ***SeqKit***
- **Input:**
  - Raw paired reads (fastq)
- **Output:**
  - Verified quality trimmed reads (with contamination removed if applicable)
  - Summary statistics for full accounting of reads

assembly.py — Assemble reads, align reads to assembly, and count mapped reads

- **Workflow:**
  - Assembles paired reads using ***SPAdes***-based assemblers (e.g., ***metaSPAdes***, ***rnaSPAdes***)
  - Builds ***Bowtie2*** index and maps reads to assembly to produced sorted BAM file
  - Indexes sorted BAM file
  - Counts reads using ***featureCounts***
  - Calculates summary statistics with ***SeqKit***
- **Input:**
  - Raw paired reads (fastq)
- **Output:**
  - Assembly fasta (and ***Bowtie2*** index)
  - Sorted BAM (and ***Samtools*** index)
  - Summary statistics for assemblies (e.g., total bases, total contigs, N50, etc.)
  - Simplified Annotation Format [SAF] file used for ***featureCounts*** read counting

coverage.py *— Align reads to a reference and count mapped reads*

- **Workflow:**
  - Aligned reads from different samples to a reference using ***Bowtie2***
  - Produces multiple sorted BAM files and indexes
  - Counts reads using ***featureCounts***
  - Calculate read statistics used ***SeqKit***
  - [Optional] Only necessary if doing pseudo-coassembly
- **Input:**
  - Reference fasta
  - A table of read paths `[id_sample]<tab>[path/to/r1.fastq.gz]<tab>[path/to/r2.fastq.gz]`
- **Output:**
  - Multiple sorted BAM (and ***Samtools*** indexes)
  - Summary statistics for full accounting of reads

binning-prokaryotic.py — Iterative consensus binning for prokaryotes

- **Workflow:**
    - Calculated coverage tables needed for binning algorithms using ***CoverM***
    - Models genes using ***Prodigal***
    - **I**terative binning:
        - A,B) ***MaxBin2*** (marker set 40,107); C) ***MetaBAT2***; D) ***CONCOCT***
        - ***DAS Tool*** (A,B,C,D) → Candidate binned genomes
        - Remove eukaryotic genomes classified by ***Tiara*** and genome size filter
        - ***CheckM*** → High quality metagenome assembled genomes [MAG]
    - ***GTDB-Tk*** to classify taxonomy
    - Reevaluated candidate phyla radiation [CPR] using ***CheckM*** CPR marker set
    - Calculate genome statistics using ***SeqKit***
- **Input:**
    - Assembly fasta (i.e., scaffolds.fasta from assembly module)
    - Sorted BAM file
- **Output:**
    - MAG assemblies, cds, protein, gene models
    - Identifier tables (ORF ←→ Contig ←→ MAG)
    - Summary tables (genome statistics, quality metrics, and classifications)
    - ORF-level counts tables
    - Binned/Unbinned lists (useful for grepping) and unbinned fasta file (used for next step)

binning-eukaryotic.py — Binning for recovering eukaryotic genomes

- **Workflow:**
  - Calculated coverage tables needed for binning algorithms using ***CoverM***
  - Bin genomes using either ***MetaBAT2*** or ***CONCOCT*** (can’t use both yet)
  - Remove prokaryotic genomes classified by ***Tiara*** and genome size filter
  - Exon-aware gene modeling using ***MetaEuk*** for candidate eukaryotic genomes
  - Lineage-specific quality assessment using ***BUSCO*** (Remove low quality genomes)
  - Calculate genome statistics using ***SeqKit***
- **Input:**
  - Assembly fasta (i.e., unbinned.fasta from prokaryotic binning module)
  - Sorted BAM file
- **Output:**
  - MAG assemblies, cds, protein, gene models
  - Identifier tables (ORF ←→ Contig ←→ MAG)
  - Summary tables (genome statistics, quality metrics, *MetaEuk* targets, and classifications)
  - ORF-level counts tables
  - Binned/Unbinned lists (useful for grepping) and unbinned fasta file (used for next step)

binning-viral.py — Binning for recovering viral genomes

- **Workflow:**
  - Identify candidate viral genomes using ***VirFinder*** (***geNomad*** coming soon…)
  - Models genes using ***Prodigal***
  - Quality assessment using ***CheckV*** (Remove low quality genomes)
  - Calculate genome statistics using ***SeqKit***
- **Input:**
  - Assembly fasta (i.e., unbinned.fasta from prokaryotic binning module)
- **Output:**
  - MAG assemblies, cds, protein, gene models
  - Identifier tables (ORF ←→ Contig ←→ MAG)
  - Summary tables (genome statistics, quality metrics, isolation source, and classifications)
  - ORF-level counts tables
  - Binned/Unbinned lists (useful for grepping) and unbinned fasta file (useful for pseudo-coassembly)

cluster.py — Species-level clustering of genomes and proteins

- **Workflow:**
  - Cluster MAGs by Average Nucleotide Identity [ANI] using ***FastANI***
  - For each species-level cluster [SLC]:
    - Cluster proteins into lineage-specific orthogroups via ***OrthoFinder***
- **Input:**
  - Scaffolds to bins table
  - List of genome paths
  - List of protein paths
- **Output:**
  - MAG assemblies, cds, protein, gene models
  - Identifier tables (Contig ←→ MAG ←→ SLC) & (ORF ←→ Orthogroup)

classify-prokaryotic.py — Taxonomic classification of prokaryotes

- **Workflow:**
  - Compiles ***GTDB-Tk*** classification files
- **Input:**
  - Prokaryotic binning directory
  - [Optional] Prokaryotic SLC clustering
- **Output:**
  - Taxonomy classifications for each MAG
  - [Optional] Prokaryotic cluster classification

classify-eukaryotic.py — Taxonomic classification of eukaryotes

- **Workflow:**
  - Gets eukaryotic markers using ***HMMER***
  - Gets ***MetaEuk*** targets of eukaryotic markers
  - Classifies eukaryotic taxonomy based on bitscores and lineage
- **Input:**
  - Eukaryotic binning directory
  - [Optional] Eukaryotic SLC clustering
- **Output:**
  - Taxonomy classifications for each MAG
  - Gene source lineage with bitscores for each marker gene used in classification
  - [Optional] Eukaryotic cluster classification

classify-viral.py — Taxonomic classification of viruses

- **Workflow:**
  - Use ***CheckV*** output and database to classify viruses and isolation source
- **Input:**
  - Viral binning directory
  - [Optional] Viral SLC clustering
- **Output:**
  - Taxonomy classifications for each MAG
  - [Optional] Viral cluster classification
  - [Optional] Consensus isolation source

annotate.py — Annotates translated gene calls against NR, Pfam, and KOFAM

- **Workflow:**
  - Align proteins to NCBI’s non-redundant database via ***Diamond***
  - Search for ***Pfam*** protein domains using ***HMMER***
  - Search fo KEGG orthology using **KOFAMSCAN**
  - [Optional] Identifier mapping [id_orf]<tab>[id_contig]
- **Input:**
  - Protein fasta file
- **Output:**
  - Annotation table
  - [Optional] Contig-level annotations based solely on NR

phylogeny.py — Constructs phylogenetic trees given a marker set

- **Workflow:**
  - Identifies marker proteins using ***HMMER*** based on user-provided database
  - [Optional] Remove hits based on marker-score thresholds
  - Protein alignment for each marker identified via ***MUSCLE***
  - Alignments are trimmed using ***ClipKIT***
  - Concatenate alignments
  - Approximately-maximum likelihood phylogenetic inference via ***FastTree2***
  - [Optional] Maximum likelihood phylogenetic inference via ***IQTREE2*** (Takes a long time)
- **Input:**
  - Table of protein fasta files
  - HMM Database
  - [Optional] Table of marker score cutoffs
- **Output:**
  - Newick formatted phylogenetic tree
  - Concatenated multiple sequence alignment
  - Alignment table (*n* genomes, *m* markers, *ij*=fasta alignment)

index.py — Builds index for alignment to genomes

- **Workflow:**
  - Creates reference index for binned genomes via ***Bowtie2***
  - Merges gene models (GFF3)

- **Input:**
  - Reference fasta file[s]
  - Gene model GFF3 file[s]

- **Output:**
  - Concatenated reference fasta
  - Concatenated reference fasta ***Bowtie2*** index
  - Concatenated gene models

mapping.py — Builds index for alignment to genomes

- **Workflow:**
  - Maps reads to ***Bowtie2*** reference index
  - Counts reads for contigs and ORFs
  - [Optional] Aggregates reads for MAG and SLC level
  - [Optional] Calculates spatial coverage for each MAG (i.e., ratio of bases covered in genome)

- **Input:**
  - Paired reads
  - Reference index directory (contains index, fasta, GFF3, and SAF)
  - [Optional] ORF to orthogroup identifier table
  - [Optional] Contigs to MAG identifier table
  - [Optional] Contigs to SLC identifier table

- **Output:**
  - Sorted BAM file
  - Paired unmapped reads
  - ORF-level counts table, contig-level counts table
  - [Optional] MAG-level counts table, SLC-level counts table, Orthogroup-level counts table