riboSeed Documentation

Release 0.4.9

Nicholas Waters, Florence Abram, Fiona Brennan, Ashleigh Holm

Nov 01, 2019

Contents

Impatient? See our Quickstart Guide

A brief overview of the theory can be found here.

The manuscript can be found here: Nicholas R Waters, Florence Abram, Fiona Brennan, Ashleigh Holmes, Leighton Pritchard; riboSeed: leveraging prokaryotic genomic architecture to assemble across ribosomal regions, Nucleic Acids Research, Volume 46, Issue 11, 20 June 2018, Pages e68, https://doi.org/10.1093/nar/gky212

CHAPTER 1

Description

riboSeed is an supplemental assembly refinement method to try to address the issue of multiple ribosomal regions in a genome, as these create repeates unresolvable by short read sequencing. It takes advantage of the fact that while each region is identical, the regions flanking are unique, and therefore can potentially be used to seed an assembly in such a way that rDNA regions are bridged.

Contents:

1.1 Description

riboSeed is an supplemental assembly refinement method to try to address the issue of multiple ribosomal regions in a genome, as these create repeates unresolvable by short read sequencing. It takes advantage of the fact that while each region is identical, the regions flanking are unique, and therefore can potentially be used to seed an assembly in such a way that rDNA regions are bridged.

1.1.1 The Problem

As you probably know, repeated regions are difficult to resolve when sequencing with a short read technology. Specifically, if the length of the repeat exceeds the length of the kmers used to construct the de Bruijn graph, the repeat cannot be resolved.

rDNAs, the genomic regions containing the sequences coding for ribosomal RNAs, are often found multiple times in a single genome. The rDNAs are usually around 5kb long, which is much longer than the length of short reads. Because of this, these regions cause breaks in the assembly.

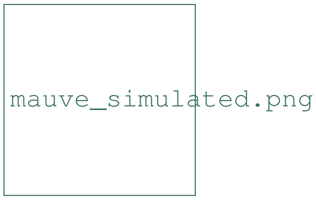
Due to how well rDNA is conserved within a taxa, we hypothesized that if the regions flanking the rDNAs are sufficiently unique within a genome, those regions would be able to locate an rDNA within the genome during assembly.

1.1.2 The Method

We call our method a *de fere novo* assembly (meaning "starting from almost nothing"), as we use a subassembly technique to minimize the bias caused by reference choice. We map the short reads to the reference genome, extract the reads mapping to rDNA (with flanking) regions, and perform subassemblies with SPAdes to reassemble the rDNA and flanking regions from the reads. These "long reads" are concatenated together separated with 5kb of N's. The reads are then mapped to the concatenated sequence and and subassembled for several additional iterations.

1.1.3 The Results

We generated a simulated genome from the 7 rDNA regions with 5kb flanking regions, and then used **'ART (MountRainier-2016-06-05)<https://www.niehs.nih.gov/research/resources/software/biostatistics/art/>'__** to generated simulated MiSeq reads of various depths.



In this **'Mauve<http://darlinglab.org/mauve/mauve.html>'__** visualization, we show (from top to bottom) the reference simulated genome, riboSeed's *de fere novo* assembly, *de novo* assembly, and a negative control *de fere novo* assembly using a *Klebsiella* reference genome. The results show that with riboSeed's *de fere novo* assembly correctly joins six of the seven rDNA regions to reconstruct the simulated genome with only short reads. By contrast, the short reads alone failed to bridge any gaps caused by the repeated rDNAs, and the assembly using a poor reference choice only assembled across a single rDNA region. We have run this successfully on many real datasets with positive results

1.2 Installation

From conda (new and recommended!) —

Conda is a cross-platform, cross-language package management system. If you haven't already installed conda, follow these instructions here, and install the python3.6 version. Once you have that done, add the appropriate channels.

```
conda config --add channels defaults
conda config --add channels conda-forge
conda config --add channels bioconda
```

and then install riboSeed and all of its dependencies with one command:

```
conda install riboseed
```

(Note the lowercase "s")

1.2.1 From Pypi

riboSeed is on Pypi, so you can install with pip, preferably within a virtualenv:

```
virtualenv -p python3.5 venv-riboSeed
source venv-riboSeed/bin/activate
pip3.5 install riboSeed
```

1.2.2 From TestPypi

To install the bleeding-edge version, install from testpypi:

```
virtualenv -p python3.5 venv-riboSeed
source venv-riboSeed/bin/activate
pip install --extra-index-url https://testpypi.python.org/pypi riboSeed
```

1.2.3 From GitHub

You can also clone this repository, create a virtual environment, and run `python3.5 setup.py install`.

1.2.4 Dependencies

1.2.4.1 External Requirements

riboScan.py

- Barrnap (must be 0.7 or above)
- EMBOSS's Seqret

riboSelect.py

- None

riboSnag.py

- PRANK or Mafft
- BLAST+ suite
- Barrnap (must be 0.7 or above)

riboSeed.py

- SPAdes v3.8 or higher
- BWA (tested with 0.7.12-r1039)
- SAMTools (must be 1.3.1 or above)
- QUAST (tested with 4.1)

NOTE: barrnap has certain Perl requirements that may not be included on your machine. Ensure barrnap runs fine before trying riboSeed. Or try python barrnap.

1.3 Choosing an Appropriate Reference

1.3.1 Introduction

Any sort of analysis that involves a reference strain begs the question: which strain should I use? Luckily, riboSeed's *de fere novo* approach gives you a bit of flexiblility here, in that the method is not affected my major genomic restructuring events, as long as they do not occur in the rDNA region. Still, using a close reference maximizes your chance for a successful assembly.

There are three ways that we recommend selecting a reference. The first uses average nucleotide identity to select an optimal reference for a given set of potential genomes – perfect if you are sequencing a popular bug and are spoiled for reference choice. The Kraken method gives great results when nothing is known a priori, and results in a high degree of certainty, but requires a bit of legwork. The last method can be done entirely through your web browser, but is much less robust. For popular bugs, we reccomend ANI; otherwise we recommend Kraken, but if you have already pulled out all of your hair from bioinformatics software installation, reads2Type/cgfind is a nice, painless alternative.

1.3.2 Reference Selection via ANI

With any assembly that uses a reference, the choice of that reference is crucial. Here, we outline a protocol for using Average Nucleotide Identity via pyANI. We call this pipeline Plenty of Bugs, as it will help you find a compatible match for your sequenced isolate.

PlentyOfBugs found here

This is easiest to do with Docker (or Singularity)

The following script will identify 25 random E. coli genomes, download then, build the pyani database, do a quick assembly of your isolate, and find the closest reference to your isolate.

```
docker run --rm -t -v  ${PWD}:/data/ nickp60/plentyofbugs:0.87 -f /data/tests/
↪references/toy_reads1.fq -o "Escherichia coli" -n 25 -e sample_ecoli -d /data/
↪results/
```

1.3.3 Kraken Method

Kraken is a kmer-based phylogeny tool that can be used to idenify the strains present in a metagenomic dataset; the installation and usage instructions can be found here

- Download and install Kraken, along with the MiniKraken database from their website.
- Run Kraken on your isolate's reads, and generate the Kraken report.

```
kraken --db MiniKraken reads1.fq reads2.fastq  > sequences.kraken
kraken-translate --db MiniKraken sequences.kraken > sequences.labels
kraken-report --db MiniKraken sequence.kraken
```

Because the MiniKraken database was built from all the complete genomes from RefSeq, it should be easy to identify which strain in the database has the closest match to your sequenced isolate.

PS: This is a great time to check if you have any contamination in your sample; thanks, Kraken!

1.3.4 reads2type and cgFind Method

reads2type is also a kmer-based phylogeny tool, but it relies on a lightweight, prebuilt database that allows the analysis to be performed in your web browser, and it doesn't require you to upload your whole read file to a webserver. It works by taking one read at a time from your file, generating 55-mers, and comparing to its databse. If there is not enough taxanomic information to indentify the isolate off of that read alone, additional reads will be processed until a single taxonomy is achieved. This method works best on trimmed reads. Instructions and the webserver can be found here

Once you have a genus and species, you can use `cgfind`, a tool we developed to provide easy access to downloadable genomes based on the complete prokaryotic genomes found in NCBI. it can be found here Just enter your genus and species name, and select one of the available strains to download.

1.4 Usage

1.4.1 Minimal example: `ribo run`

The pseudogenome was constructed from the 7 rDNAs separated by several kb of flanking DNA. If can be found under *./riboSeed/integration_test/concatenated_seq.fasta*. If you have installed using setuptools, the *integration_test* folder will be installed in the site-packages dir, such as */venv-riboSeed/lib/python3.5/site-packages/riboSeed/integration_data/*.

Two read files can be found in the same directory.

To run the whole riboSeed pipline, use the following command:

```
ribo run ./riboSeed/integration_data/concatenated_seq.fasta \
          -F ./riboSeed/integration_data/test_reads1.fq \
          -R ./riboSeed/integration_data/test_reads2.fq \
          -o ./test1/ -v 1
```

1.4.1.1 Whats going on:

`ribo run` is used to run the pipeline with the most commonly used settings. It first creates a config file, tracking down your system executables for the required tools, and setting the default parameters for things not specified as args to run_riboSeed.

Then, `ribo scan` is run to re-annotate your reference, `ribo select` calls the rDNA operons, and `ribo seed` runs the *de fere novo* assembly.

If you want to change the behaviour of the programs under the hood, all of the command line options not set by `ribo run` are defined in the config file in the output directory. After editing the parameters in the config file, you can submit it to `ribo run` using the -c flag.

1.4.2 Running individual scripts

All of the elements of the package can be run individually: Perhaps you want to modify barrnap's behaviour in `scan`, or you want to experiment with different feature selectors in `select`. Go for it!

```
$ ribo

Description: A suite of tools to perform de fere novo assembly to bridge
gaps caused by rDNA repeats

Usage:   ribo <command> [options]

Available commands:
-run        execute pipeline (scan, select, seed, sketch, and score)
-scan       reannotate rRNAs in a FASTA file
-select     group rRNA annotations into rDNA operons
-seed       perform de fere novo assembly
-snag       extract rDNA regions and plot entropy
-sim        perform simulations used in manuscript
-sketch     plot results from a de fere novo assembly
-stack      compare coverage depth in rDNA regions to rest of genome
-score      score batches of assemblies with BLASTn
-swap       swap contigs from assemblies
```

(continues on next page)

(continued from previous page)

```
-spec       use assembly graph to speculate number of rDNAs
-structure  view the rRNA operon structure across several genomes
-config     write out a blank config file to be used with `run`
```

1.5 riboSeed Pipeline

1.5.1 Usage

The pipeline consists of 3 main stages: preprocessing, de fere novo assembly, and visualization/assessment. As of version 0.4.21, the pipeline is run by invoking `ribo` and then one of the following commands:

[preprocessing]

- `scan`
- `select`

[de fere novo assembly]

- `seed`

[visualizations/assessment]

- `snag`
- `stack`
- `sketch`
- `swap`
- `score`
- `spec`

1.5.2 Before We Start

Please back up any and all data used, and work within a virtualenv.

Genome assembly gobbles RAM. If you, like me, are working on a 4gb RAM lappy, don't run seed in parallel and instead run in series by using the `--serialize` option. That should prevent you from running out of RAM during the final SPAdes calls.

1.5.3 0: Preprocessing

1.5.3.1 `scan`

`scan` preprocesses sequences straight from a multifasta or one or more fasta. The issue with many legacy annotations, assemblies, and scaffold collections is rDNAs are often poorly annotated at best, and unannotated at worst. This is shortcut to happiness without using the full Prokka annotation scheme. It requires `` `barrnap `` <http://www.vicbioinformatics.com/software.barrnap.shtml>'__ and seqret (from `` `emboss `` <http://www.ebi.ac.uk/Tools/emboss/>'__) to be available in your path. #### Usage

`scan` can either use a directory of fastas or one (multi)fasta file. If using a directory of fastas, provide the appropriate extension using the `-e` option. If using a (multi)fasta as input, it write out each entry to its own fasta in the `contigs` subdirectory that it makes in the output. For each of the fastas, the script renames complex headers (sketchy), scans with barrnap and captures the output gff. It then edits the gff to add fake incrementing locus_tags, and uses the original sequence file through seqret to make a GenBank file that contains just annotated rRNA features. The last step is a concatenation which, whether or not there are multiple files, makes a single (possibly multi-entry) genbank file perfect for seed-ing.

NOTE: If using a reference with long names or containing special characters, use the –name argument to rename the contigs to something a bit more convenient and less prone to errors when piping results.

```
usage: ribo scan -o OUTPUT [-e EXT] [-k {bac,euk,arc,mito}] [-t ID_THRESH]
                   [-b BARRNAP_EXE] [-n NAME] [-c {1,2,4,8,16}]
                   [-s SEQRET_EXE] [-v {1,2,3,4,5}] [-h]
                   contigs

Given a directory of one or more chromosomes as fasta files, this facilitates
reannotation of rDNA regions with Barrnap and outputs all sequences as a
single, annotated genbank file

positional arguments:
  contigs_dir           directory containing one or more chromosomal sequences
                        in fasta format

required named arguments:
  -o OUTPUT, --output OUTPUT
                        output directory; default:
                        /home/nicholas/GitHub/seed


optional arguments:
  -k {bac,euk,arc,mito}, --kingdom {bac,euk,arc,mito}
                        whether to look for eukaryotic, archaeal, or bacterial
                        rDNA; default: bac
  -e extension          extension of the chromosomal sequences, usually
                        '.fa', '.fasta' or similar; default: .fa
  -t ID_THRESH, --id_thresh ID_THRESH
                        partial rRNA hits below this threshold will be
                        ignored. default: 0.5
  -b BARRNAP_EXE, --barrnap_exe BARRNAP_EXE
                        path to barrnap executable; default: barrnap
  -n NAME, --name NAME  name to give the contig files; default: infered from
                        file
  -s SEQRET_EXE, --seqret_exe SEQRET_EXE
                        path to seqret executable, usually installed with
                        emboss; default: seqret
  -v {1,2,3,4,5}, --verbosity {1,2,3,4,5}
                        Logger writes debug to file in output dir; this sets
                        verbosity level sent to stderr. 1 = debug(), 2 =
                        info(), 3 = warning(), 4 = error() and 5 = critical();
                        default: 2
  -h, --help            Displays this help message
```

NOTE: If using a reference with long names or containing special characters, use the –name argument to rename the contigs to something a bit more convenient and less prone to errors when piping results.

1.5.3.2 `select`

`select` searches the genome for rRNA annotations, clusters them into likely ribosomal groups, and outputs a colon-separated list of clustered rRNA locus tags by record id.

You will probably want to preview your file to figure out the syntax used. (ie, 16s vs 16S, rRNA vs RRNA, etc…)

If not using `scan` or if not working with a prokaryotic genome, you will need to change `--specific_features` appropriately to reflect the annotations in your reference (ie, for a fungal genome, use `--specific_features 5_8S:18S:28S`).

NOTE: the format of the output text file is very simple, and due to the relatively small number of such coding sequences in bacterial genomes, this can be constructed by hand if the clusters do not look appropriate. The format is `genome_sequence_id locus_tag1:locus_tag2`, where each line represents a cluster. See example below, where 14 rRNAs are clustered into 6 groups:

NOTE 2: In order to streamline things, as of version 0.0.3 there will be a commented header line with the feature type in the format "#$ FEATURE ", such as `#$ FEATURE rRNA`.

```
#$ FEATURE rRNA
CM000577.1 FGSG_20052:FGSG_20051:FGSG_20053
CM000577.1 FGSG_20048:FGSG_20047
CM000577.1 FGSG_20049:FGSG_20050
CM000577.1 FGSG_20054:FGSG_20056:FGSG_20055
CM000577.1 FGSG_20058:FGSG_20057
CM000577.1 FGSG_20075:FGSG_20074
```

Usage

```
usage: ribo select [-h] [-o OUTPUT] [-f FEATURE] [-s SPECIFIC_FEATURES]
                    [--keep_temps] [--clobber] [-c CLUSTERS] [-v VERBOSITY]
                    [--debug]
                    genbank_genome

This is used to identify and cluster rRNA regions from a gb file, returns a
text file with the clusters

positional arguments:
  genbank_genome        Genbank file (WITH SEQUENCE)

optional arguments:
  -h, --help            show this help message and exit

required named arguments:
  -o OUTPUT, --output OUTPUT
                        output directory;default:
                        /home/nicholas/GitHub/seed

optional arguments:
  -f FEATURE, --feature FEATURE
                        Feature, rRNA or RRNA; default: rRNA
  -s SPECIFIC_FEATURES, --specific_features SPECIFIC_FEATURES
                        colon:separated -- specific features; default:
                        16S:23S:5S
  --keep_temps          view intermediate clustering filesdefault: False
  --clobber             overwrite previous output files: default: False
  -c CLUSTERS, --clusters CLUSTERS
                        number of rDNA clusters;if submitting multiple
                        records, must be a colon:separated list whose length
                        matches number of genbank records. Default is inferred
                        from specific feature with fewest hits
```

(continues on next page)

(continued from previous page)

```
  -v VERBOSITY, --verbosity VERBOSITY
                        1 = debug(), 2 = info(), 3 = warning(), 4 = error()
                        and 5 = critical(); default: 2
  --debug               Enable debug messages
```

1.5.4 2: *De fere novo* Assembly

1.5.4.1 `seed`

`seed` maps reads to a genome and (1) extracts reads mapping to rDNA regions, (2) perfoms subassemblies on each pool of extracted reads to recover the rDNA complete with flanking regions (resulting in a pseudocontig) (3) concatenates a;; pseudocontigs into them into a pseudogenome with 5kb spacers of N's in between, (5) map remaining reads to the pseudogenome, and (6) repeat steps 1-5 for a given number of iterations (default 3 iterations). Finally, seed runs SPAdes assemblied with and without the pseudocontigs and the resulting assemblies are assessed with QUAST.

Output

The results directory will contain a 'final_long_reads' directory with all the pseudocontigs, the mapped fastq files, and `final_de_novo_assembly` and `final_de_fere_novo_assembly` folders, containing the SPAdes results.

NOTE:

If using a consumer-grade computer, it will be advantagous to run with `-z/--serialize` enabled to run asseblies in serial rather than parallel.

Usage:

minimal usage: `ribo seed clustered_accession\list.txt -F FASTQ1 -R FASTQ2 -r REFERENCE_GENOME -o OUTPUT`

```
usage: ribo seed -r REFERENCE_GENBANK -o OUTPUT [-F FASTQ1] [-R FASTQ2]
                   [-S1 FASTQS1] [-n EXP_NAME] [-l FLANKING] [-m {smalt,bwa}]
                   [-c CORES] [-k KMERS] [-p PRE_KMERS] [-s SCORE_MIN]
                   [-a MIN_ASSEMBLY_LEN] [--include_shorts] [--linear]
                   [--ref_as_contig {None,trusted,untrusted}] [--keep_temps]
                   [--skip_control] [-i ITERATIONS] [-v {1,2,3,4,5}]
                   [--target_len TARGET_LEN] [-t {1,2,4}] [-z]
                   [--smalt_scoring SMALT_SCORING] [--mapper_args MAPPER_ARGS]
                   [-h] [--spades_exe SPADES_EXE]
                   [--samtools_exe SAMTOOLS_EXE] [--smalt_exe SMALT_EXE]
                   [--bwa_exe BWA_EXE] [--quast_exe QUAST_EXE]
                   [--python2_7_exe PYTHON2_7_EXE]
                   clustered_loci_txt

Given cluster file of rDNA regions from select and either paired-end or
single-end reads, assembles the mapped reads into pseduocontig 'seeds', and
uses those with the reads to runde fere novo and de novo assembly with SPAdes

positional arguments:
  clustered_loci_txt    output from select
```

(continues on next page)

(continued from previous page)

```
required named arguments:
  -r REFERENCE_GENBANK, --reference_genbank REFERENCE_GENBANK
                        genbank reference, used to estimate insert sizes, and
                        compare with QUAST
  -o OUTPUT, --output OUTPUT
                        output directory; default:
                        /home/nicholas/GitHub/seed

optional arguments:
  -F FASTQ1, --fastq1 FASTQ1
                        forward fastq reads, can be compressed
  -R FASTQ2, --fastq2 FASTQ2
                        reverse fastq reads, can be compressed
  -S1 FASTQS1, --fastq_single1 FASTQS1
                        single fastq reads
  -n EXP_NAME, --experiment_name EXP_NAME
                        prefix for results files; default: seed
  -l FLANKING, --flanking_length FLANKING
                        length of flanking regions, in bp; default: 1000
  -m {smalt,bwa}, --method_for_map {smalt,bwa}
                        available mappers: smalt and bwa; default: bwa
  -c CORES, --cores CORES
                        cores for multiprocessing; default: None
  -k KMERS, --kmers KMERS
                        kmers used for final assembly, separated by commas;
                        default: 21,33,55,77,99,127
  -p PRE_KMERS, --pre_kmers PRE_KMERS
                        kmers used during seeding assemblies, separated bt
                        commas; default: 21,33,55,77,99
 -s SCORE_MIN, --score_min SCORE_MIN
                        If using smalt, this sets the '-m' param; default with
                        smalt is inferred from read length. If using BWA,
                        reads mapping with ASscore lower than this will be
                        rejected; default with SWA is half of read length
   -a MIN_ASSEMBLY_LEN, --min_assembly_len MIN_ASSEMBLY_LEN
                        if initial SPAdes assembly largest contig is not at
                        least as long as --min_assembly_len, exit. Set this to
                        the length of the seed sequence; if it is not
                        achieved, seeding across regions will likely fail;
                        default: 6000
  --include_shorts      if assembled contig is smaller than
                        --min_assembly_len, contig will still be included in
                        assembly; default: inferred
  --linear              if genome is known to not be circular and a region of
                        interest (including flanking bits) extends past
                        chromosome end, this extends the seqence past
                        chromosome origin forward by --padding; default: False
  --ref_as_contig {None,trusted,untrusted}
                        if 'trusted', SPAdes will use the seed sequences as a
                        --trusted-contig; if 'untrusted', SPAdes will treat as
                        --untrusted-contig. if '', seeds will not be used
                        during assembly. See SPAdes docs; default: untrusted
  --keep_temps          if not --keep_temps, mapping files will be removed
                        once they are no no longer needed during the
                        iterations; default: False
  --skip_control        if --skip_control, no de novo assembly will be done;
```

(continues on next page)

(continued from previous page)

```
                        default: False
  -i ITERATIONS, --iterations ITERATIONS
                        if iterations>1, multiple seedings will occur after
                        subassembly of seed regions; if setting --target_len,
                        seedings will continue until --iterations are
                        completed or --target_len is matched or exceeded;
                        default: 3
  -v {1,2,3,4,5}, --verbosity {1,2,3,4,5}
                        Logger writes debug to file in output dir; this sets
                        verbosity level sent to stderr. 1 = debug(), 2 =
                        info(), 3 = warning(), 4 = error() and 5 = critical();
                        default: 2
  --target_len TARGET_LEN
                        if set, iterations will continue until contigs reach
                        this length, or max iterations (set by --iterations)
                        have been completed. Set as fraction of original seed
                        length by giving a decimal between 0 and 5, or set as
                        an absolute number of base pairs by giving an integer
                        greater than 50. Not used by default
  -t {1,2,4}, --threads {1,2,4}
                        if your cores are hyperthreaded, set number threads to
                        the number of threads per processer.If unsure, see
                        'cat /proc/cpuinfo' under 'cpu cores', or 'lscpu'
                        under 'Thread(s) per core'.: 1
  -z, --serialize       if --serialize, runs seeding and assembly without
                        multiprocessing. This is recommended for machines with
                        less than 8GB RAM: False
  --smalt_scoring SMALT_SCORING
                        if mapping with SMALT, submit custom smalt scoring via
                        smalt -S scorespec option; default:
                        match=1,subst=-4,gapopen=-4,gapext=-3
  --mapper_args MAPPER_ARGS
                        submit custom parameters to mapper. And by mapper, I
                        mean bwa, cause we dont support this option for SMALT,
                        sorry. This requires knowledge of your chosen mapper's
                        optional arguments. Proceed with caution! default: -L
                        0,0 -U 0
  -h, --help            Displays this help message
  --spades_exe SPADES_EXE
                        Path to SPAdes executable; default: spades.py
  --samtools_exe SAMTOOLS_EXE
                        Path to samtools executable; default: samtools
  --smalt_exe SMALT_EXE
                        Path to smalt executable; default: smalt
  --bwa_exe BWA_EXE     Path to BWA executable; default: bwa
  --quast_exe QUAST_EXE
                        Path to quast executable; default: quast.py
  --python2_7_exe PYTHON2_7_EXE
                        Path to python2.7 executable, cause QUAST won't run on
                        python3. default: python2.7
```

1.5.5 Key Parameters

Results can be tuned by changing several of the default parameters.

- `--score_min`: This can be used to set the minimum mapping score. If using BWA, the default is not to supply a minimum and to rely on the BWA default. If submitting a `--score_min` to BWA, double check that it is appropriate. It appears to be extremely sensitive to read length, and having a too-low threshold for minimum mapping can seriously ruin ones day. Check out IGB or similar to view your mappings if greater than, say, 5% or the reads are mapping in subsequent iterations.

- `-l, --flanking_length`: Default is 2000. That seems to be a good compromise between gaining unique sequence and not relying too much on the reference.
- `--kmers` and `--pre_kmers`: Adjust these as you otherwise would for a *de novo* assembly.
- `--min_assembly_len`: For bacteria, this is about 7000bp, as the rDNA regions for a typical operon of 16S 23S and 5S coding sequences combined usually are about that long. If you are using non-standard rDNA regions, this should be adjusted to prevent spurious assemblies.
- `--ref_as_contig`: This can be used to guide how SPAdes treats the long read sequences during the assembly (`trusted` or `untrusted`). By default, this is infered from mapping percentage (`trusted` if over 85% of reads map to the reference)
- `--iterations`: Each iteration typically increases the length of the long read by approximately 5%.

1.5.6 3: Visualization/Assessment

1.5.6.1 `snag`

`snag` takes the list of clustered locus tags and extracts their sequences with flanking regions, optionally turning the coding sequences to N's to minimize bias towards reference. Is used to pull out regions of interest from a Genbank file. Outputs a directory with a fasta file for each clustered region (and a log file).

Additionally, it does a lot of plotting to visualize the Shannon entropy, coverage, occurrences, and other useful metrics.

Usage:

```
usage: ribo snag [-o OUTPUT] [-n NAME] [-l FLANKING] [--msa_kmers] [-c]
                   [-p PADDING] [-v VERBOSITY] [--clobber] [--no_revcomp]
                   [--skip_check] [--msa_tool {mafft,prank}]
                   [--prank_exe PRANK_EXE] [--mafft_exe MAFFT_EXE]
                   [--barrnap_exe BARRNAP_EXE]
                   [--makeblastdb_exe MAKEBLASTDB_EXE]
                   [--kingdom {mito,euk,arc,bac}] [-h]
                   genbank_genome clustered_loci

Use to extract regions of interest based on supplied Locus tags and evaluate
the extracted regions

positional arguments:
  genbank_genome        Genbank file (WITH SEQUENCE)
  clustered_loci        output from select

required named arguments:
  -o OUTPUT, --output OUTPUT
                        output directory; default:
                        /home/nicholas/GitHub/seed

optional arguments:
  -n NAME, --name NAME  rename the contigs with this prefixdefault: date
                        (YYYYMMDD)
```

(continues on next page)

(continued from previous page)

```
  -l FLANKING, --flanking_length FLANKING
                        length of flanking regions, in bp; default: 1000
  --msa_kmers           calculate kmer similarity based on aligned sequences
                        instead of raw sequences;default: False
  -c, --circular        if the genome is known to be circular, and an region
                        of interest (including flanking bits) extends past
                        chromosome end, this extends the seqence past
                        chromosome origin forward by 5kb; default: False
  -p PADDING, --padding PADDING
                        if treating as circular, this controls the length of
                        sequence added to the 5' and 3' ends to allow for
                        selecting regions that cross the chromosom's origin;
                        default: 5000
  -v VERBOSITY, --verbosity VERBOSITY
                        1 = debug(), 2 = info(), 3 = warning(), 4 = error()
                        and 5 = critical(); default: 2
  --clobber             overwrite previous output filesdefault: False
  --no_revcomp          default returns reverse complimented seq if majority
                        of regions on reverse strand. if --no_revcomp, this is
                        overwriddendefault: False
  --skip_check          Dont bother calculating Shannon Entropy; default:
                        False
  --msa_tool {mafft,prank}
                        Path to PRANK executable; default: mafft
  --prank_exe PRANK_EXE
                        Path to PRANK executable; default: prank
  --mafft_exe MAFFT_EXE
                        Path to MAFFT executable; default: mafft
  --barrnap_exe BARRNAP_EXE
                        Path to barrnap executable; default: barrnap
  --makeblastdb_exe MAKEBLASTDB_EXE
                        Path to makeblastdb executable; default: makeblastdb
  --kingdom {mito,euk,arc,bac}
                        kingdom for barrnap; default: bac
  -h, --help            Displays this help message
```

1.5.6.2 `stack`

Decause assembly using short reads often collases rDNA repeats, it is not uncommon to find a reference genome that has less than the actual number of rDNAs. stack uses `bedtools` and `samtools` to determine the coverage across rDNA regiosn, adn compares that coverage depth to 10 sets of randomly selected non-rDNA regions. If the number of rDNAs in the reference matches the number of rDNAs in your sequecned isolate, the coverage should be pretty similar. However, if the coverage in your rDNA regions is significantly higher, than there are likely more rDNAs in your sequenced isoalte that there are in the reference, which is something to be aware of.

It requires a mapping BAM file and the scan output directory as input.

1.5.6.3 `swap`

Infrequently, `seed` has joined together contigs that appear incorrect according to your reference. If you are at all unhappy with a bridging, `swap` allows swapping of a “bad” contig for one or more syntenic contigs from the *de novo* assembly. #### USAGE

```
usage: ribo swap -o OUTPUT [-v {1,2,3,4,5}] [-h]
                      de_novo_file de_fere_novo_file bad_contig good_contigs

Given de novo and de fere novo contigs files, a misjoined de fere novo contig
name, and a colon:separated list of de novo contig names, replace the
offending contig with the de novo contig(s)

positional arguments:
  de_novo_file          multifasta containing de novo contigs
  de_fere_novo_file     multifasta containing de fere novo contigs
  bad_contig            name of the bad contig
  good_contigs          colon separated good contigs for replacement

required named arguments:
  -o OUTPUT, --output OUTPUT
                        output directory; default:
                        /home/nicholas/GitHub/seed

optional arguments:
  -v {1,2,3,4,5}, --verbosity {1,2,3,4,5}
                        Logger writes debug to file in output dir; this sets
                        verbosity level sent to stderr. 1 = debug(), 2 =
                        info(), 3 = warning(), 4 = error() and 5 = critical();
                        default: 2
  -h, --help            Displays this help message
```

1.5.6.4 `spec`

One limitation in resolving the rDNA repeats is the lack of confidence in the reference genomes that were assembled from short reads along. `ribo spec` parses the SPAdes assembly graph in fastg format to take a guess at how many rDNAs are in the genome based on the nodes and edges represinting the region in the graph. This can help alert the user that the number of rDNAs in the reference may disagree with the actual number in the genome.

1.6 Accessory Scripts

1.6.1 Assessment

`riboScore.py` ~~~ Suppose you have a whole bunch of assemblies to assess. The most rigorous way of checking the assemblies would be to use Mauve (or a similar whole genome alignment visualizer tool) for the job, and manually check the quality of each assembly, listening to the ends of the contigs, seeking one-ness with the data. Thats all well and good if you are (a) independantly wealthy and enjoy doing this sort of thing, (b) seeking a meditative state through mindless clicking, or (c) an undergrad assistant, but for the rest of us, we are willing to sacrifice a bit of accuracy for throughput. This is, after all, why we aren't sequencing on gels anymore.

`ribo score` outputs two types of score repors as text files: one which is easy for humans to read, and the other that can be easily combined with hundreds like it to make various types of graphs etc.

```
usage: ribo score [-h] [-o OUTPUT] [-l FLANKING] [-p MIN_PERCENT]
                  [-f ASSEMBLY_EXT] [-g REF_EXT] [-F] [-v {1,2,3,4,5}]
                  indir

This does some simple blasting to detect correctness of riboSeed results

```

(continues on next page)

(continued from previous page)

```
positional arguments:
  indir                 dir containing a genbank file and other file

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        directory in which to place the output files
  -l FLANKING, --flanking_length FLANKING
                     length of flanking regions, in bp; default: 1000
  -p MIN_PERCENT, --min_percent MIN_PERCENT
                     minimum percent identity
  -f ASSEMBLY_EXT, --assembly_ext ASSEMBLY_EXT
                     extenssion of reference, usually fasta
  -g REF_EXT, --ref_ext REF_EXT
                     extension of reference, usually .gb
  -F, --blast_Full      if true, blast full sequences along with just the
                     flanking. Interpretation is not implemented currently
                     as false positives cant be detected this way
  -v {1,2,3,4,5}, --verbosity {1,2,3,4,5}
                     Logger writes debug to file in output dir; this sets
                     verbosity level sent to stderr. 1 = debug(), 2 =
                     info(), 3 = warning(), 4 = error() and 5 = critical();
                     default: 2
```

1.6.2 Visualization

1.6.2.1 riboSnag.py

riboSnag.py takes the list of clustered locus tags and extracts their sequences with flanking regions, optionally turning the coding sequences to N's to minimize bias towards reference. Is used to pull out regions of interest from a Genbank file. Outputs a directory with a fasta file for each clustered region (and a log file).

Additionally, it does a lot of plotting to visualize the Shannon entropy, coverage, occurrences, and other useful metrics.

Usage:

```
usage: riboSnag.py [-o OUTPUT] [-n NAME] [-l FLANKING] [--msa_kmers] [-c]
                   [-p PADDING] [-v VERBOSITY] [--clobber] [--no_revcomp]
                   [--skip_check] [--msa_tool {mafft,prank}]
                   [--prank_exe PRANK_EXE] [--mafft_exe MAFFT_EXE]
                   [--barrnap_exe BARRNAP_EXE]
                   [--makeblastdb_exe MAKEBLASTDB_EXE]
                   [--kingdom {mito,euk,arc,bac}] [-h]
                   genbank_genome clustered_loci

Use to extract regions of interest based on supplied Locus tags and evaluate
the extracted regions

positional arguments:
  genbank_genome        Genbank file (WITH SEQUENCE)
  clustered_loci        output from riboSelect

required named arguments:
```

(continues on next page)

(continued from previous page)

```
  -o OUTPUT, --output OUTPUT
                        output directory; default:
                        /home/nicholas/GitHub/riboSeed

optional arguments:
  -n NAME, --name NAME  rename the contigs with this prefixdefault: date
                        (YYYYMMDD)
  -l FLANKING, --flanking_length FLANKING
                        length of flanking regions, in bp; default: 1000
  --msa_kmers           calculate kmer similarity based on aligned sequences
                        instead of raw sequences;default: False
  -c, --circular        if the genome is known to be circular, and an region
                        of interest (including flanking bits) extends past
                        chromosome end, this extends the seqence past
                        chromosome origin forward by 5kb; default: False
  -p PADDING, --padding PADDING
                        if treating as circular, this controls the length of
                        sequence added to the 5' and 3' ends to allow for
                        selecting regions that cross the chromosom's origin;
                        default: 5000
  -v VERBOSITY, --verbosity VERBOSITY
                        1 = debug(), 2 = info(), 3 = warning(), 4 = error()
                        and 5 = critical(); default: 2
  --clobber             overwrite previous output filesdefault: False
  --no_revcomp          default returns reverse complimented seq if majority
                        of regions on reverse strand. if --no_revcomp, this is
                        overwriddendefault: False
  --skip_check          Dont bother calculating Shannon Entropy; default:
                        False
  --msa_tool {mafft,prank}
                        Path to PRANK executable; default: mafft
  --prank_exe PRANK_EXE
                        Path to PRANK executable; default: prank
  --mafft_exe MAFFT_EXE
                        Path to MAFFT executable; default: mafft
  --barrnap_exe BARRNAP_EXE
                        Path to barrnap executable; default: barrnap
  --makeblastdb_exe MAKEBLASTDB_EXE
                        Path to makeblastdb executable; default: makeblastdb
  --kingdom {mito,euk,arc,bac}
                        kingdom for barrnap; default: bac
  -h, --help            Displays this help message
```

1.6.2.2 `ribo stack`

Because assembly using short reads often collases rDNA repeats, it is not uncommon to find a reference genome that has less than the actual number of rDNAs. riboStack uses `bedtools` and `samtools` to determine the coverage across rDNA regiosn, adn compares that coverage depth to 10 sets of randomly selected non-rDNA regions. If the number of rDNAs in the reference matches the number of rDNAs in your sequecned isolate, the coverage should be pretty similar. However, if the coverage in your rDNA regions is significantly higher, than there are likely more rDNAs in your sequenced isoalte that there are in the reference, which is something to be aware of.

It requires a mapping BAM file and the riboScan output directory as input.

1.6.3 Utilities

1.6.3.1 riboSwap.py

Infrequently, riboSeed has joined together contigs that appear incorrect according to your reference. If you are at all unhappy with a bridging, `ribo swap` allows swapping of a "bad" contig for one or more syntenic contigs from the *de novo* assembly. #### USAGE

```
usage: ribo swap -o OUTPUT [-v {1,2,3,4,5}] [-h]
                   de_novo_file de_fere_novo_file bad_contig good_contigs

Given de novo and de fere novo contigs files, a misjoined de fere novo contig
name, and a colon:separated list of de novo contig names, replace the
offending contig with the de novo contig(s)

positional arguments:
  de_novo_file          multifasta containing de novo contigs
  de_fere_novo_file     multifasta containing de fere novo contigs
  bad_contig            name of the bad contig
  good_contigs          colon separated good contigs for replacement

required named arguments:
  -o OUTPUT, --output OUTPUT
                        output directory; default:
                        /home/nicholas/GitHub/riboSeed

optional arguments:
  -v {1,2,3,4,5}, --verbosity {1,2,3,4,5}
                        Logger writes debug to file in output dir; this sets
                        verbosity level sent to stderr. 1 = debug(), 2 =
                        info(), 3 = warning(), 4 = error() and 5 = critical();
                        default: 2
  -h, --help            Displays this help message
```

1.6.3.2 seedRand.py

There is no convenient unix command to generate seeded random numbers from the command line. This standalone script uses numpy (if availible) or the built-in random module to generate *n* random numbers given a seed.

Note: numpy *should* give you the same random numbers given the same seed across platforms: this is *not* the case with python's build-in random module.

```
usage: seedRand.py [-h] seed n

Given a seed, return a pseudrando integer between 1 and 9999, separated by
newlines, to stdout. usage : `seedRand.py 27 10` would return 10 random
numbers seeded with 27

positional arguments:
  seed        seed
  n           number of random numbers to return, must be > 0

optional arguments:
  -h, --help  show this help message and exit
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search