BEAR User Manual

Downloading BEAR

Download the `bear.zip` file at `http://www.cs.usask.ca/research/research_groups/birl/bear.index`. Inside the `.zip` file you will find a directory called `BEAR_release` and two subdirectories (`scripts` and `doc`). The `scripts` directory contains the scripts that comprise BEAR, while the `doc` directory contains this user manual, in addition to the version history and known bugs with the software. It is recommended that the `scripts` directory be added to your `PATH` variable if you wish to use the commands from this manual verbatim.

Prerequisites

To use BEAR, you must have `Perl` and `Python` installed. BEAR is known to work with `Perl` v5.14.2 and `Python` v2.7.3. You will need the `Perl Getopt::Long`, `Bio::SeqIO`, and `List::Util` modules, the `Python Bio` and `NumPy` packages, and the `sys, csv, StringIO, random, decimal` and `argparse` modules.

BEAR also requires the use of `DRISEE`, which is part of the `MG-RAST` project. Details of `DRISEE`-specific prerequisites and installation procedures can be found at `https://github.com/MG-RAST/DRISEE`. This version of BEAR uses `DRISEE` v1.2.

Running BEAR

BEAR can perform the following functions:

1. **Homology-based abundance profile generation**: This function first performs taxonomic classification on the sequences in a given `FASTQ` file based on homology to known protein sequences. The sequences that are classified to at least the species level are then used to create an abundance profile based on the frequency of each species. This technique for abundance profile generation will work best for large files of WGS reads that are expected to have many coding sequences with significant similarity to existing protein sequences. If this is not the case, it is advised to use parametric abundance profile generation instead.

2. **Parametric abundance profile generation**: This function generates abundance values derived from power functions that can correspond to metagenomic communities with low, medium, or high species complexity.

3. **Error model generation (using `DRISEE`)**: This function uses `DRISEE` to generate error models derived from duplicate reads present in a given `FASTQ` file.

4. **Quality score model generation for erroneous nucleotides**: The `DRISEE` output is processed to determine the quality scores that are assigned to erroneous nucleotides.

5. **Simulated read generation using empirically-derived error rates and quality scores**: Given a `multi-FASTA` file of genome sequences, `FASTQ` file of metagenomic data, and the output from the previous steps, reads are sampled from previously sequenced genomes according to an abundance profile. The reads are then modified based on error rates and quality scores are generated.

The LINUX/UNIX shell commands for executing each function listed below. The portions of the shell commands in matching angle brackets (e.g., <search_results>) should be replaced with a value conforming to their descriptions.

1. (a) **Homology-based abundance profile generation** first requires the user to have a file of sequence information in genpept format from the refseq_protein database. As of November 13, 2013, the following command can be used to download all files in genpept format from NCBI's refseq_protein database:
wget -A .protein.gpff.gz -r ftp://ftp.ncbi.nlm.nih.gov/refseq/release/complete/

   i. All of the files downloaded using this command can then be decompressed using the gunzip command and concatenated into one large file in genpept format using the cat command. This will be the database for homology-based searching, referred to as <genpept_format_file> in step (iii).
   ii. With the large genpept file as the database, use a search tool (e.g., RAPSearch2) to generate a file of search results using a file of WGS reads as the set of queries.
      - If RAPSearch2 was used, the first five lines must be removed from the search results file and the first, second, and 12th columns must be extracted; e.g.,
        tail -n +6 <search_results> | cut -f 1,2,12 > <search_results_trimmed>
      - If a different search tool was used, three columns from the resulting search file must be extracted such that a trimmed, tab-delimited search file with the following format is generated:
        <query_id> <subject_id> <bit_score>
   iii. Run the following command to map protein accession numbers to taxonomic lineages:
        make_accession_to_taxonomy_map.pl <genpept_format_file> > <mapping_file>
        The output of this command, stored in <mapping_file>. is a tab-delimited file with the following format:
        <refseq_id> <taxonomic_lineage>
   iv. Run the following command to generate a homology-based abundance profile using the FASTQ and <search_results_trimmed> files from step (ii):
        homology_abundance.pl <search_results_trimmed> <mapping_file> <WGS_read_file>
        This will result in three files as output:
      - <search_results_trim>.lca_map: a file in tab-delimited text format where the first column is the identifier of a read in the RAPSearch2 results file, and the second column is the lineage assigned to that query.
      - <search_results_trim>.lca_summary: a file in tab-delimited text format where the second column is a lineage, and the first column is the number of query IDs assigned to that lineage.
      - <search_results_trim>.lca_summary.normalized: a file in tab-delimited text format where the second column is a classification, and the first column is the relative frequency of that classification (between 0 and 1).
   v. With a multi-FASTA file consisting of genome sequences that will make up the simulated community, run the following command to generate a homology-based abundance file using only the sequences that have been classified to the species level:
        normalize_abundances.pl <search_results_trim>.lca_summary <genomes_file> > <abundance_file>

   The <abundance_file> contains the following two tab-delimited columns:
   <genome_id> <abundance_value>
   The values for each <abundance_value> will be a relative frequency between 0 and 1, such that the sum of all abundance values equals 1.

(b) **Parametric abundance profile generation** requires only that the user have a `multi-FASTA` file of genomes that will later form the simulated community. BEAR currently supports `low`, `medium`, and `high` complexity communities derived from power functions. Only one command is necessary to generate abundance profiles using this method:
`parametric_abundance.pl <genomes_file> <low|med|high> > <abundance_file>`
The abundance files generated by this function use the same format as described at the end of step 1a.

2. To perform **error model generation**, the user first must ensure that `DRISEE` is installed and running correctly.

   (a) Copy the `drisee.py` and `run_find_steiner.py` scripts from the `BEAR/scripts/` directory to `DRISEE-master/` directory. Add the `DRISEE-master/` directory to the `PATH` variable.

   (b) Generate the `DRISEE`-derived error rates by using the following command:
   `drisee.py --percent -n <read_min> -b <num_max> -x <read_max> -t fastq -s <nr> <WGS_read_file> <error_rate_file>`
   where `<nr>` is the total number of reads in the dataset, `<WGS_read_file>` is a `multi-FASTQ` file of WGS reads, and `<error_rate_file>` is the desired prefix of the output files. The other required parameters are as follows:

Parameter	**Description**	**Suggested parameter value**
`-n <read_min>`	Minimum bin size of duplicate reads for it to be considered in the error calculations.	`-n 3`
`-b <num_max>`	Maximum number of prefix bins to process.	`-b 10000`, but any large number should be sufficient
`-x <read_max>`	Maximum number of reads to process from each bin of prefix identical reads.	`-x 10000`, but any large number should be sufficient

   `DRISEE` will generate two output files, `<error_rate_file>.per` and `<error_rate_file>.uc`. The `<error_rate_file>.per` file will be in `DRISEE` format, described in detail at `https://github.com/MG-RAST/DRISEE`. The `<error_rate_file>.uc` file will be in `uclust` format, described in detail at `http://www.drive5.com/uclust/uclust_userguide_1_1_579.html`.

3. Using the (presumably large) uclust file generated by `DRISEE`, **generate the quality score models for erroneous nucleotides** by using the following command:
   `error_quality.pl <WGS_read_file> <error_rate_file>.uc`
   This command will generate two files:

   (a) `<quality_model_file>`, a file in `DRISEE` format with quality scores in place of error rates (`.err.qual` file extension).

   (b) `<matrix_file>`, a file with a containing two 4x4 matrices; one that determines the transition/transversion rates of each nucleotide and another that determines the rate of homopolymer insertion errors compared to regular insertion errors (`.err.matr` file extension).

4. (a) Now that the error models and abundance files have been generated, we can start to simulate the reads. First, the `generate_reads.py` script is used to generate uniform-length simulated reads from a `multi-FASTA` file of genome sequences. The general form for running this script is:
   `generate_reads.py <parameters>`

   The following parameters are **required**:

Parameter	Description
`-r <genomes_file>`	`Multi-FASTA` file containing genomic sequences from which reads will be sampled.
`-a <abundance_file>`	Tab-delimited abundance file with an abundance value for each corresponding genome sequence in `<reference_fasta>`. Can be generated by following step 1a or 1b.
`-o <output_file>`	Name for output file containing simulated uniform-length reads.
`-t <total_reads>`	The total number of reads to sample from all genomes.
`-l <longest_read>`	The length, in bp, of the longest possible read to simulate.

The following parameters are **optional**:

Parameter	Description
`-i <insert_mean_length>`	Average length of insert for paired-end reads. Default value: 0
`-s <insert_stddev>`	Standard deviation of insert length for paired-end reads. Default value: 0

(b) The final step trims the reads from the previous step, introduces sequencing errors, and generates quality scores. The general form for running this script is:
`trim_reads.py <parameters>`

The following parameters are **required**:

Parameter	Description
`-i <WGS_read_file>`	`FASTQ` file from which the error and quality models were derived. In this script, it is used to generate quality scores for non-erroneous nucleotides.
`-f <fasta_file>`	Read file to be trimmed and subjected to errors and quality score generation. This should be the `FASTA` file containing the uniform-length reads generated in Step 4.
`-o <output_file>`	Name for output FASTQ file containing the simulated data.
`-r <error_rate_file>`	Generates erroneous nucleotides. The error rate file (with `.per` extension) was generated by `DRISEE` in Step 2. If Step 2 was not performed, use `-r 0` for a simple error model.
`-q <quality_model_file>`	Generates quality scores for erroneous nucleotides. The error quality value file was generated by `error_quality.py` in Step 3. If Step 3 was not performed, use `-q 0` for a simple quality score model.
`-m <matrix_file>`	Determines transition/transversion rates for substitution errors and nucletotide-specific insertion errors. This should be the `<matrix_file>` generated by `error_quality.py` in Step 3. If Step 3 was not performed, use `-m 0` to assume that rates are equal among all nucleotides.