User Guide of ESPRIT

Yijun Sun[§], Yunpeng Cai[¶], Li Liu[§], Michael L. Farrell[♯]
William McKendree[♯], William Farmerie[§]

[§]Interdisciplinary Center for Biotechnology Research
[¶]Department of Electrical and Computer Engineering
University of Florida, Gainesville, FL 32610-3622
[♯]Materials Technology Directorate
Air Force Technical Applications Center
1030 S. Highway A1A, Patrick AFB, FL 32925-3002

This is an open-source project. If you use the algorithm, please cite the following paper: Y Sun, Y Cai, L Liu, F Yu, ML Farrell, W McKendree and W Farmerie. ESPRIT: Estimating Species Richness Using Large Collections of 16S rRNA Pyroequences. *Nucleic Acids Research*, vol. 37, no. 10 e76, 2009.

1 Introduction

ESPRIT is a computational algorithm developed for estimating microbial diversities using 16S rRNA pyrosequencing data. It consists of four modules: (1) removes low-quality reads using various criteria, (2) computes pairwise distances of reads, (3) groups reads into OTUs at different dissimilarity levels, and (4) performs statistical inference to estimate species richness. We developed two versions of ESPRIT, one for personal computers and one for computer clusters. The personal-computer version is used for small and medium-scale project and can process several tens of thousands sequences within a few minutes, and the computer-cluster version is for large-scale problems and is able to analyze several hundreds of thousands of reads within one day. The source code is freely available at `http://plaza.ufl.edu/sunyijun/ESPRIT.htm`. If you have any questions and comments, please feel free to contact Dr. Yijun Sun at `sunyijun@biotech.ufl.edu`.

Throughout the manual, parameters in angle brackets <> are mandatory, while those in square brackets [ ] are optional.

2 Installation

The executable code for various platforms (Windows, 32-bit and 64-bit Linux) is available in the released package. Copy the code to your destination directory, and add it to the system execution path. If you need to modify the program or use ESPRIT in other platforms, you can compile the source code. Download the source code into your designated directory and modify `Makefile` as follows:

Windows Users: use the definitions `RM=del /F /S` and `CFLAGS = ... -DWIN`. Comment out other `RM` and `CFLAG` definitions by adding `#` at the beginning of the lines.

Linux/Unix/Cygwin Users: use the definitions `RM=rm -rf` and `CFLAGS` with no `-DWIN` option. Comment out other `RM` and `CFLAG` definitions.

Mac Users: use the Linux setting described above, and remove `-static` from the `LDLIBS` definition.

After choosing the proper setting, type

```
>> make
```

to compile. You can also use one of the following commands to generate the PC and CC versions:

```
>> make esprit_pc
```

```
>> make esprit_cc
```

If you need to compile the source code multiple times, it is recommended to use

```
>> make clean
```

to remove the previously generated files before compiling.

ESPRIT can process up to 1 million reads. To process a larger dataset, you can modify the `Max_SEQS` and `Max_Buf` definitions in `util.h`, and recompile the code.

3 Personal Computer Version

The following command initiates the program using the default parameters. The inputs `sequence.fas` and `primer.fas` contain 16S rRNA and primer sequences, respectively. They have to be in the FASTA format. `[PATH]` provides the path information of the input files. If `[PATH]` is not given, the path information should be included in the file names. The input `primer.fas` is optional. If missing, no trimming is performed against the primer sequences.

```
>> esprit_pc [-r PATH] -i sequence.fas [-p primer.fas]
```

ESPRIT generates five output files, including `sequence.Chao1`, `sequence.ACE`, `sequence.OTU`, `sequence.Rarefaction`, and `sequence.Cluster`. They are saved in the same directory as the input files. The file `sequence.Cluster` provides the detailed information on how the sequences are clustered at different distance levels. In order to reduce the size of the file, each sequence is represented by a number, instead of the original sequence ID. Since the low-quality sequences are removed from the original sequence file, it should be noted that

Table 1: Input and output files of the PC version of ESPRIT

Input	
`<input>.fas`	16S rRNA sequences in FASTA format
`primer.fas`	Primer sequences in FASTA format
Output	
`<input>.Chao1`	Chao1 estimates and 95% confidence intervals at different distance levels
`<input>.ACE`	ACE estimates at different distance levels
`<input>.Rarefaction`	Results of rarefaction analysis
`<input>.OTU`	The number of OTUs estimated at different distance levels
`<input>.Cluster`	Clustering results

each sequence is numbered based on the sequence file `sequence_clean.fas`. The cluster information allows users to compute other ecological metrics, to derive a consensus sequence of each cluster, and to align the sequences of rarely occurred OTUs against a database, which may lead to the identification of new microorganisms. Please refer to Table 1 for a detailed description of each output file.

ESPRIT uses *k*mer statistics to remove unnecessary sequence comparisons. If the *k*mer distance between two sequences are larger than a certain threshold, pairwise alignment will not be performed. The default setting is 0.5. However, the users can use a different threshold, say 0.2, using the following command:

```
>> esprit_pc -u 0.2 -i sequence.fas -p primer.fas
```

A smaller threshold can significantly reduce the computational complexity. It is useful if users are only interested in microbial diversity at a small distance level (e.g., 0.03). We provide an auxiliary program `DetermineThreshold` in the software package that allows users to determine a threshold by plotting *k*mer distances against genetic distances by using a small subset of their samples (see Section 7 for details).

ESPRIT allows users to bypass the trimming process and to use the data filtered by a customized trimming procedure. To do so, simply type:

```
>> esprit_pc -f -i sequence.fas
```

Table 2: Command line reference of ESPRIT

Flag	Possible Value	Default	Description
`-i`	File name	/	16S rRNA sequences in FASTA format.
`-p`	File name	/	primer sequences in FASTA format.
			If missing, no trimming is performed against primer sequences.
`-a`	File name	/	an array of distance levels where statistical analysis is performed.
			If missing, statistical analysis is performed at all levels.
`-r`	File Path	/	path of input files.
			If missing, each file name should include its path.
`-u`	$x \in (0, 1)$	0.5	threshold of $k$mer distances.
`-f`	/	None	If present, trimming is not performed.
`-t`	/	None	Process protein sequences.
`-k`	$x \in \{2, 3, 4, 5, 6, 7\}$	6	length of $k$mer.
`-g`	$x \in [1.0, 100.0]$	10	gap open penalty of the Needleman-Wunsch algorithm.
`-e`	$x \in [0.0, 10.0]$	0.5	gap extension penalty of the Needleman-Wunsch algorithm.

3.1 One Example

We provide a test data set, namely `FS396.fas`, in the software package for demonstration purposes. The data set contains about 17,000 reads downloaded from [1]. It has undergone a systematic trimming process. Simply type:

```
>> esprit_pc -f -i FS396.fas
```

It takes about 7 minutes to finish the entire analysis.

4 Computer Cluster Version

Since different cluster systems may use different commands to submit a job, we are unable to write the code into one program. Instead, we present the pseudo-code of the CC version of ESPRIT. The users can easily modify it into a Unix script to submit jobs to computer clusters. The users who have no access to a computer cluster to analyze their data may contact the corresponding author. All of the flags used in the PC version are also applicable. The only difference is that the users have to specify the number of computer nodes, on

which the computation is carried out. In the following pseudo-code, $k$mer counting and the Needleman-Wunsch algorithm are performed in 55 and 100 computer nodes, respectively.

```
 1 preproc [-p primer.fas] [-w] [-v 1.0] sequence.fas;
 2 for ((i=1; i<=10; i++)) do
 3     for ((j=i; j<=10; j++)) do
 4         kmerdist_par sequence_Clean.fas 10 $i $j;
 5     done;
 6 done;
 7 cat *.dist >> kmer.dist;
 8 splitdist -s 100 kmer.dist;
 9 for ((i=0; i<=99; i++)) do
10     needledist sequence_Clean.fas kmer_dist.$i needle.dist_$i;
11 done;
12 cat needle.dist_* >> sequence.ndist;
13 hcluster sequence.ndist sequence_Clean.frq
14 do_stat sequence.Cluster_List
```

Algorithm 1: The pseudo-code of the CC version of ESPRIT

The CC version of ESPRIT consists of five main programs (i.e., `preproc, kmerdist_par, needledist, hcluster and do_stat`), and one auxiliary program `splitdist`.

Line 1: The function `preproc` removes low-quality reads using various criteria. All of the files `*.fas` are in the FASTA format. The input `primer.fas` is optional. `preproc` removes the reads containing ambiguous nucleotides (N), and those with more than one mismatch with the PCR primers at the beginning of a read. Also, we eliminate the sequences with atypical lengths. If two sequences are identical or one sequence is a substring of the other, only the longer sequence is retained, and the number of occurrence of the retained sequence is recorded in `sequence_Clean.frq`.

Lines 2-6: The function `kmerdist_par` computes the $k$mer distances of sequences. In this example, the sequences in `sequence_Clean.fas` are equally divided into 10 blocks, and `kmerdist_par sequence_Clean.fas 10 $i $j` computes the $k$mer distances between two sequences from the i-th and j-th blocks, respectively. Hence, the computation is carried out in 55 computer nodes. The results are saved in a sparse distance matrix `sequence_clean_$i_$j.dist`, $1 \leqslant i \leqslant 10, i \leqslant j \leqslant 10$. Only the $k$mer distances larger than the threshold are kept. **CAUTION**: Each `kmerdist_par` thread is submitted as an individual CC job. Do not write a for-loop in your submission script. Ask your CC administrator about how to submit batched jobs.

Line 8: The function `splitdist` divides `kmer.dist` into 100 segments. The results are saved in `kmer.dist_$i`, $0 \leqslant i \leqslant 99$.

Lines 9-11: The function `needledist` computes the genetic distance of each sequence pair. In this example, the computation is carried out in 100 computer nodes. The results are saved in `needle.dist_$i`, $0 \leqslant i \leqslant 99$.

Line 13: The function `hcluster` assigns reads into OTUs at different distance levels. The outputs include `sequence.OTU`, `sequence.Cluster` and `sequence.Cluster_List`.

Line 14: The function `do_stat` performs statistical inference of species richness. The outputs include `sequence.Chao1, sequence.ACE` and `sequence.Rarefaction`, which are identical to those generated by `esprit_pc`.

5 Map Clustering Result back to Original Data

In the above procedure, the clustering result generated by `hcluster` is correspondent to the cleaned and trimmed FASTA file. Number 0 in the `*.Cluster` file refers to the first sequence in the `*_Clean.fas` file, and number 1 the second sequence. In some cases, users may want to map the clustering result back to the original FASTA file. We provide a perl script to carry out the task. Users should have `perl` installed in their system and available in the execution path.

After `preproc` or `esprit_pc` is executed, a `*_Clean.map` file is generated. Users can type the following command to convert the output file:

```
>> perl invmap.pl sequence.Cluster sequence_Clean.map sequence.org.Cluster
```

The cluster result file `sequence.org.Cluster` is then generated. The numbers in the result file are the zero-based indices of the sequences in the original FASTA file `sequence.fas`.

The following command can be used to generate FASTA fiels of the sequences in each OTU defined at the 0.03, 0.04 and 0.05 levels:

```
>> parsecluster sequence.fas sequence.org.Cluster 0.03 0.05
```

Clustering results generated at the 0.03, 0.04 and 0.05 distance levels are placed in the directories `groups_0.03`, `groups_0.04` and `groups_0.05`, respectively.

The following command generates a FASTA file containing the representative (consensus) sequences of OTUs at the 0.05 level:

```
>> consensus sequence_Clean.fas sequence_Clean.frq sequence.Cluster 0.05
```

The representative sequences are listed in `sequence_Clean.cons0.05.fas` and the corresponding cluster sizes are given in `sequence_Clean.cons0.05.frq`. For each OTU, the most abundant sequence is selected as the representative sequence.

6 Average Linkage based Clustering

`hcluster` supports only complete and single linkage based hierarchical clustering. We provide a function, called `aveclust`, for average linkage based clustering. After a distance file is generated, type the following command to perform average-linkage clustering:

```
>> aveclust -n num_seqs [-f sequence_Clean.frq] sequence.ndist
```

where `num_seqs` is the number of sequences after preprocessing, which can be obtained by typing:

```
>> wc -l sequence_Clean.frq
```

The clustering result contains three files: `*.A-OTU` reports the number of OTUs at each distance level, `*.A-Cl` reports the detailed clustering result, and `*.A-CList` contains OTU information which can be used as input of `do_stat` for performing statistical inference.

We currently only provide the executable code of `aveclust` for Windows and Linux. Since `aveclust` does not employ online-learning techniques, it can process fewer sequences than `hcluster`. Table 3 lists the number of sequences that can be handled with different memory limitations.

Table 3: The number of sequences that `aveclust` can process with different memory limiations

	32Bit Systems [a]				64Bit Systems			
memory	1GB	2GB	3GB	4GB	8GB	16GB	32GB	64GB
# of reads	30000	40000	50000	60000	90000	120000	180000	250000

[a]For Windows systems, due to the memory limit set by OS, users might only be able to use 2GB/3GB memory.

7 Detailed Syntax of Each Function

The default parameters work well for most data sets, as shown in the paper. However, if the users want to have more options, we provide a detailed description of each function.

preproc

Remove low-quality reads and merge duplicated reads.

```
preproc [-e] [-t] [-p primer_file] [-m mis_allowed] [-w] [-v var_allowed]
[-l minlen] [-u maxlen] <input.fas> [output] [freq_output]
```

`-t`:	Process protein sequences.
`-e`:	Merge identical sequences only. By default, `preproc` merges reads of zero distance (ignoring end gaps).
`primer_file`:	FASTA file containing PCR primers.
`mis_allowed`:	maximum allowed mismatch between a sequence and primer sequences, default 1.
`-w`:	Remove sequences containing ambiguous bases. By default, only the ambiguous bases are removed, not the sequences.
`-v var_allowed`:	Remove sequences with lengths that differ from average length larger than `var_allowed` times one standard deviation.
`-l minlen`:	Remove sequences shorter than `minlen`.
`-u maxlen`:	Remove sequences longer than `maxlen`.
`input`:	FASTA file containing original sequences.
`output`:	FASTA file containing processed sequences, default <input>_Clean.fas.
`freq_output`:	sequence frequency output, default <input>_Clean.frq.

kmerdist

Compute pairwise $k$mer distances.

```
kmerdist [-u threshold] [-m] [-d] [-t] [-k klen] <input[.fas]>
```

[<output>]

klen: $k$mer length, default 6 (nucleotide) or 3 (amino acid).

threshold: $k$mer threshold, default 0.5.

-d: Print the distances. By default, only the indices of sequence pairs with $k$mer distance smaller than the threshold are printed.

-m: Print the entire distance matrix.

-t: Process protein sequences.

kmerdist_par [-u threshold] [-k klen] <input[.fas]> num_segs $i $j

klen: $k$mer length, default 6 (nucleotide) or 3 (amino acid).

threshold: $k$mer threshold, default 0.5.

num_segs: number of total segments.

-t: Process protein sequences.

$i, $j: indices of the i-th and j-th segments for computing $k$mer distances. The result is stored in <input>_$i_$j.dist.

needledist

Compute pairwise genetic distances.

needledist [-n] [-v] [-t] [-x] [-d align_output] [-g gap_open] [-e gap_extend] <seq_file> <dist_file> <output>

needledist [-a] [-n] [-v] [-t] [-x] [-d align_output] [-g gap_open] [-e gap_extend] <seq_file><output>

-x: Penalize end gaps;

-d: Print the detailed alignment results to align_output;

-t: Align protein sequences.

-v: Show alignment progress.

-n: Do not perform alignments; compute pairwise distances for pre-aligned sequences.

gap_open: gap open penalty of the NW algorithm, default 10.

gap_extend: gap extension penalty of the NW algorithm, default 0.5.

seq_file: FASTA file containing processed sequence.

dist_file: distance file generated by kmerdist or kmerdist_par.

output: genetic distances computed based on globally aligned sequences.

hcluster

Perform hierarchical clustering.

`hcluster [-u] [-s stepsize] [-c seqcount] [-p method] [-e end_level] [-t table_size] [-b buffer_size] [-o tree_file] <input.dist> [freq_file]`

`-u`: The input distances have been sorted in an ascending order. If not specified, the program sorts the distances and creates `input.dist_sort`.

`stepsize`: step size between two consecutive distance levels, default 0.01.

`table_size`: A `table_size`×`table_size` matrix is created to store temporal link information between clusters. If `table_size` is too small, a "Link Table Full" error will be reported. Default 10,000.

`buffer_size`: size of buffer used for internal sorting. Default 10,000,000. A bigger buffer will reduce the time used in sorting the distances.

`seqcount`: number of sequences in the data set. Only valid when using with `-u`. If not specified, the program automatically goes through `<input.dist>` to find out the maximum sequence index.

`end_level`: Stop clustering after the maximum distance level `end_level` is reached. By default, `hcluster` processes all distance records in the input.

`method`: 0 complete linkage (default), 1 single linkage.

`-o`: Generate detailed hierarchical clustering tree. `<tree_file>` contains 4 columns:[new cluster id] [cluster 1 id] [cluster 2 id] [distance]

`input.dist`: genetic distances calculated by `needledist`.

`freq_file`: frequency information of sequences generated by `prerpoc`. If not specified, it is assumed that each sequence appears only once.

do_stat

Perform ACE, CHAO1 and rarefaction statistical analysis.

`do_stat [-a markfile] <input.Cluster_List>`

`input.Cluster_List`: `input.Cluster_List` generated by `hcluster`;

`markfile`: a file containing the distance levels that require statistical analysis. For example, "0.03 0.05 0.1" means that the analysis is performed only at the three specified distance levels. By default, the analysis is performed at all distance levels.

splitdist

Split a distance file for parallel computing.

`splitdist -s segnum [-n total] <input>`

`segnum`:	number of segments.
`input`:	distance file.
`total`:	total number of records in the input file. If not specified, the program will compute it.

fastasplit

Split a FASTA file.

`fastasplit <-s segnum | -u num_seqs> [-n total_seqs] <input> <input>`

`-s segnum`:	Split a file into a given number of segments.
`-u num_seqs`:	Split a file into segments each containing a given number of sequences.
`total_seqs`:	total number of sequences in the input file. Need to specify it if there are more than 10^7 sequences.

DetermineThreshold

Compare $k$mer distances with genetic distances to determine a $k$mer threshold.

`DetermineThreshold [-k klen] [-g gap_open] [-e gap_extend] <input> <output>`

`klen`:	$k$mer length.
`gap_open`:	gap open penalty, default 10.
`gap_extend`:	gap extension penalty, default 0.5.
`input`:	sequence file in FASTA format.
`output`:	The output file contains two columns. The first one contains the $k$mer distances and the second one contains the corresponding genetic distances.

parsecluster

Generate a set of FASTA files each containing the sequences in an OTU defined at the given distance levels.

`parsecluster <fastafile> <clusterfile> <low> <high>`

`fastafile`:	original or trimmed sequence data in FASTA format.
`clusterfile`:	corresponding clustering result (*.Cluster).
`[low, high]`:	the minimal and maximum distance levels.

consensus

Generate FASTA files containing representative sequences at given distance level. The most abundant sequence in each OTU is selected as the representative sequence.

`consensus <fastafile> <freq_file> <clusterfile> <level>`

`fastafile`:	pre-processed FASTA sequences.
`freq_file`:	frequency information of sequences generated by `preproc`.
`clusterfile`:	corresponding clustering result (*.Cluster).
`level`:	distance level.

aveclust

Average Linkage Based Hierarchical Clustering

`aveclust <-n numsamp |-p > [<-f freq_file>] [-u level] [-s stepsize] [-o <tree_file>] <input.dist>`

`numsamp`:	number of non-redundant sequences.
`-p`:	Accept input in PHYLIP distance matrix format. `aveclust` accepts full or lower-triangle matrix but not upper-triangle matrix.
`freq_file`	frequency information of sequences generated by `preproc`. If not specified, it is assumed that each sequence appears only once.
`-o`:	Generate detailed hierarchical clustering tree. <`tree_file`> contains 4 columns: [new cluster id] [cluster 1 id] [cluster 2 id] [distance]
`-u`:	Stop clustering when the maximum distance level is reached.
`stepsize`:	step size between two consecutive distance levels, default 0.01.

References

[1] Sogin ML, Morrison HG, Huber JA, Welch DM, Huse SM, Neal PR, Arrieta JM, Herndl GJ. (2006) Microbial diversity in the deep sea and the underexplored "rare biosphere". *Proc Natl Acad Sci.* **103**:12115-12120.