MEvoLib (**M**olecular **Evo**lution **Lib**rary) v1.0 Manual

by J. Álvarez-Jarreta
Universidad de Zaragoza, Spain

February 12, 2016

Contents

1 Introduction

MEvoLib is a Molecular Evolution library, designed for Python 2.7 and Python 3.3 or newer, that gathers a wide range of software applications and methods usually applied in phylogenetics and bioinformatics. It includes methods to interact with NCBI's databases (GenBank (http://www.ncbi.nlm.nih.gov/genbank/ among others), alignment tools, gene and other

data clustering methods, phylogenetic inference tools, consensus trees and supertree methods. It is intended to facilitate the usage of methods and tools to expert users, including automatic processing and parallelization of applications. Futhermore, it is also intended to help new users with an easy interface and by providing default configurations of some of the most valuable tools.

The ZaraMit website (http://zaramit.org) hosts this library, among other software our group has developed and published at MEvoLib's home page (http://zaramit.org/mevolib).

2 Download and Installation

The next section describes how to download and install MEvoLib.

2.1 Download

MEvoLib v1.0 is the first stable release of this library, available at the ZaraMit website (http://zaramit.org/downloads/MEvoLib_v1.0.tar.gz). The user can also access to the latest source code of MEvoLib at GitHub (http://github.com/JAlvarezJarreta/MEvoLib). MEvoLib has been released under the GNU General Public License (see *LICENSE* file for further details).

We have included a shell script to ease the download and installation of several bioinformatics tools (e.g. Mafft, FastTree, RAxML,...), available at http://zaramit.org/downloads/tool_installer.tar.gz.

Finally, for an easy and fast test of the library, we have created a Debian Virtual Machine (.ovf) where MEvoLib, all its dependencies and all the software tools have been installed. It can be imported to VirtualBox and VMWare (and other tools that support .ovx virtual machines). It can be downloaded from http://zaramit.org/downloads/zaramit-vm.tar. For further instructions, please go to http://zaramit.org/mevolib.

2.2 Dependencies

MEvoLib requires Python 2.7 or Python 3.3 (or newer) to work. In addition, three Python libraries must be installed:

- **Biopython** v1.65 or newer (http://biopython.org/).
- **NumPy** v1.10.1 or newer (www.numpy.org).

- **Dendropy** v4.0.3 or newer (http://www.dendropy.org/).

All the following software tools must be installed to allow the related MEvoLib's functions work properly: Mafft, Clustal Omega, Muscle, FastTree, RAxML, Consense (from PHYLIP) and DCM (from DACTAL). Consense has been extracted and modified to improve its functionality. DCM has been extracted from DACTAL package to facilitate its configuration and installation. They can be downloaded from http://zaramit.org/downloads/consense-3.696.tar.gz and http://zaramit.org/downloads/dcm.tar.gz, respectively.

2.3 Installation

MEvoLib includes a *README* file with extended information. The Unix installation process can be summarized in the execution of these two commands in the command line:

```
python setup.py build
python setup.py install
```

For the latter command, the user might need superuser privileges. Remember to replace *python* for the corresponding version of Python you want to install MEvoLib into.

2.4 Support

If you have any problem or doubt do not hesitate to contact us at zaramit@googlegroups.com.

3 Modules

In this section we show how to use the different modules of MEvoLib with examples of common cases. For further details, go to the documentation included in each source file.

3.1 MEvoLib.Align: *Multiple Sequence Alignment*

This first example shows how to know the available keywords (and their corresponding arguments) for a specific alignment tool, like Mafft.

```
>>> from MEvoLib import Align
>>> Align.get_keywords('mafft')
{'default': '--auto --quiet --thread 2',
```

```
  'linsi': '--localpair --maxiterate 1000 --quiet --thread 2',
  'parttree': '--parttree --retree 2 --partsize 1000 --quiet
              --thread 2'}
```

The second example gets an alignment with Mafft using its *default* configuration (*--auto*), and saves it in the *align* variable, which will be a *Bio.Align.MultipleSeqAlignment* object (from Biopython).

```
>>> from MEvoLib import Align
>>> align = Align.get_alignment('mafft', 'inseqs.fasta',
...       'fasta')
>>> type(align)
<class 'Bio.Align.MultipleSeqAlignment'>
```

The next example shows how to include a different argument list to generate a new alignment with Mafft.

```
>>> from MEvoLib import Align
>>> align = Align.get_alignment('mafft', 'inseqs.fasta',
...       'fasta', '--globalpair --thread 2 --quiet')
```

The fourth example saves the resultant alignment with Muscle in an output file (*align.phy*), in PHYLIP format, instead of saving it in a variable.

```
>>> from MEvoLib import Align
>>> Align.get_alignment('muscle', 'inseqs.fasta', 'fasta',
...       'fastdna', 'align.phylip', 'phylip')
```

The last example uses a tool not included in MEvoLib, PRANK, to generate the alignment. Hence, MEvoLib requires some extra parameters in order to know how to communicate with the new tool.

```
>>> from MEvoLib import Align
>>> align = Align.get_alignment('prank', 'inseqs.fasta',
...       'fasta', '-F', informats=['fasta'], incmd='-d=')
```

3.2 MEvoLib.Cluster: *Group sequences or fragments*

The first example shows a naïve row clustering, where the input dataset is dividied in 200 sets with roughly equal number of sequences. The output will be a dictionary of unique keys, with one list of *Bio.SeqRecord.SeqRecord* objects (from Biopython) per key. Each object corresponds to a complete input sequence.

```
>>> from MEvoLib import Cluster
>>> dec = Cluster.get_subsets('rows', 'inseqs.fasta', 'fasta',
...       200)
```

```
>>> print(sorted(dec.keys()))
['rset001', 'rset002', 'rset003', ..., 'rset200']
>>> type(dec['rset01'])
<class 'list'>
>>> type(dec['rset01'][0])
<class 'Bio.SeqRecord.SeqRecord'>
```

The second example generates a naïve column clustering, where the resultant sets are formed by a fragment of all the input sequences. Each set correspond to a different fragment of roughly the same length, based on the largest sequence of the input dataset.

```
>>> from MEvoLib import Cluster
>>> dec = Cluster.get_subsets('cols', 'inseqs.fasta', 'fasta',
...     10)
>>> print(sorted(dec.keys()))
 ['cset01', 'cset02', 'cset03', ..., 'cset10']
```

The third example divides the input dataset into gene sets, using the information in the GENBANK format of each sequence. The user can provide a list of features to filter from all the possible features stored at NCBI's databases.

```
>>> from MEvoLib import Cluster
>>> dec = Cluster.get_subsets('genes', 'hmtDNA.gb', 'gb',
...     ['D-loop', 'tRNA', 'rRNA', 'CDS'])
>>> print(sorted(dec.keys()))
['CDS.atp6', ..., 'D-loop.D-loop', 'rRNA.rnr1', ...,
'tRNA.trnf', ..., 'unprocessable']
```

The last example clusters the sequences using the *padded-Recursive-DCM3* decomposition (PRD) method from DACTAL. This method requires an input tree, the maximum number of sequences of each set (250) and the number of overlapping sequences between sets (50).

```
>>> from MEvoLib import Cluster
>>> dec = Cluster.get_subsets('prd', 'inseqs.fasta', 'fasta',
...     'intree.tree', 'newick', 250, 50)
>>> print(sorted(dec.keys()))
['prdset01', 'prdset02', 'prdset03', ..., 'prdset12']
```

3.3 MEvoLib.Data: *Handy biological information*

This module stores biological data required for some methods and tools. In MEvoLib v1.0 there is only one sequence available: the *revised Cambridge Reference Sequence* (rCRS).

```
>>> from MEvoLib.Data import rCRS
>>> rCRS.RECORD
SeqRecord(seq=Seq('GATCACAGGTCTATCACCCTATTAACCACTCA...ATG',
IUPACAmbiguousDNA()), id='NC_012920.1', name='NC_012920',
description='Homo sapiens mitochondrion, complete genome.',
dbxrefs=['BioProject:PRJNA30353'])
>>> rCRS.FILEPATH
'/usr/local/lib/python2.7/dist-packages/MEvoLib/Data/rCRS.gb'
```

3.4 MEvoLib.Fetch: *Fetching sequences and trees*

The first example shows how to load sequences from a local file or from a NCBI's database (GenBank in this example).

```
>>> from MEvoLib.Fetch.BioSeqs import BioSeqs
>>> seqdb = BioSeqs.from_seqfile('inseqs.fasta', 'fasta')
>>> hmtDNA = BioSeqs.from_entrez('nuccore', 'test@show.com',
...       '"Homo sapiens"[Organism] AND mitochondrion',
...       max_fetch=10)
```

The second example shows the possible data access available with the *BioSeqs* objects created in the first example.

```
>>> len(seqdb)
83
>>> print(hmtDNA)
DB: {KR902533.1, KT002469.1, KR026958.1, KP300793.1,
KR902534.1,...}
Num. sequences: 10
History:
2016/02/12 19:34:47    entrez    nuccore    "Homo sapiens"
[Organism] AND mitochondrion[All Fields]
>>> hmtDNA.statistics()
(10, 14926.0, 4924.3348992528927, 16571, 153)
>>> hmtDNA.update()
>>> len(hmtDNA)
37728
```

The third example shows the possible interactions within *BioSeqs* objects and how to write them into a new GENBANK file.

```
>>> seqdb.join(hmtDNA)
>>> len(seqdb)
37811
>>> seqdb.write('new_seqs.gb')
```

The next example shows the same possibilities as the previous three examples, but with *PhyTrees* (excluding the NCBI's database fetching).

```
>>> from MEvoLib.Fetch.PhyTrees import PhyTrees
>>> treedb1 = PhyTrees.from_treefile('intrees1.newick',
...     'newick')
>>> print(treedb1)
Num. trees: 5
History:
2016/02/12 10:24:05    /home/user/intrees1.newick    newick
>>> treedb2 = PhyTrees.from_treefile('intrees2.newick',
...     'newick')
>>> len(treedb2)
2
>>> treedb2.statistics()
(2, 15.0, 20, 10)
>>> treedb1.join(treedb2)
>>> treedb1.write('new_trees.newick')
```

3.5 MEvoLib.Inference: *Phylogenetic inference*

This first example shows how to know the available keywords (and their corresponding arguments) for a specific phylogenetic inference tool, like FastTree.

```
>>> from MEvoLib import Inference
>>> Inference.get_keywords('fasttree')
{'default': '-gtr -nt -nopr -quiet',
 'GTR+G': '-gtr -nt -nocat -gamma -nopr -quiet',
 'JTT+CAT': '-nopr -quiet',
 'WAG+CAT': '-wag -nopr -quiet',
 'GTR+CAT': '-gtr -nt -nopr -quiet',
 'JC+G': '-nt -nocat -gamma -nopr -quiet',
 'JC+CAT': '-nt -nopr -quiet'}
```

The second example gets a phylogeny with FastTree using its *default* configuration, and saves it in the *tree* variable, which will be a *Bio.Phylo.Newick.Tree* object (from Biopython). The method also returns the log-likelihood score of the phylogeny.

```
>>> from MEvoLib import Inference
>>> tree, score = Inference.get_phylogeny('fasttree',
...     'inseqs.fasta', 'fasta')
>>> type(tree)
<class 'Bio.Phylo.Newick.Tree'>
>>> score
-29643.345383
```

The third example infer a phylogeny with RAxML using the *default* configuration, saving the resultant phylogeny in the file *tree.newick*, in NEWICK

format.

```
>>> from MEvoLib import Inference
>>> tree, score = Inference.get_phylogeny('raxml',
...     'inseqs.fasta', 'fasta', 'default', 'tree.newick',
...     'newick')
```

The last example infer a phylogeny using RAxML with different arguments and generating 100 bootstraps to add more statistical robustness to the method.

```
>>> from MEvoLib import Inference
>>> tree, score = Inference.get_phylogeny('raxml',
...     'inseqs.fasta', 'fasta', '--JC69', bootstraps=100)
```

3.6 MEvoLib.PhyloAssemble: *Supertrees and consensus*

This first example shows how to know the available keywords (and their corresponding arguments) for a specific supertree or consensus tree tool, like Consense (from PHYLIP).

```
>>> from MEvoLib import PhyloAssemble
>>> PhyloAssemble.get_keywords('consense')
{'default': 'R 2 Y'}
```

The next example constructs a consensus tree with Consense (from PHYLIP) in its *default* configuration. The method returns the consensus tree in a *Bio.Phylo.Newick.Tree* object (from Biopython)

```
>>> from MEvoLib import PhyloAssemble
>>> cons_tree = PhyloAssemble.get_consensus_tree('consense',
...     'intrees.newick', 'newick')
>>> type(cons_tree)
<class 'Bio.Phylo.Newick.Tree'>
```

The last example performs the same operation as the example before, but it saves the consensus tree in the *cons_tree.newick* file, in NEWICK format.

```
>>> from MEvoLib import PhyloAssemble
>>> PhyloAssemble.get_consensus_tree('consense',
...     'intrees.newick', 'newick', 'default',
...     'cons_tree.newick', 'newick')
```