MEvoLib (**M**olecular **Evo**lution **Lib**rary) v1.01 Manual

by J. Álvarez-Jarreta
Universidad de Zaragoza, Spain

July 22, 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 databases, e.g. GenBank (http://www.ncbi.nlm.nih.gov/genbank/, 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. Furthermore, 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. The MEvoLib's home page (http://zaramit.org/software/mevolib) contains its latest stable version, the datasets used in the biological examples, this manual and other relevant information.

2 Download and Installation

The next section describes how to download and install MEvoLib.

2.1 Download

MEvoLib v1.01 is the last stable release of this library, available at the ZaraMit website (http://zaramit.org/download/software/MEvoLib_v1.01.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/download/software/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/download/vm/zaramit-vm.tar. For further instructions, please go to http://zaramit.org/software/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/download/software/consense-3.696.tar.gz and http://zaramit.org/download/software/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.

2.4 Test suite

In the version 1.01 of MEvoLib we have included a test suite to test all the modules available before the user installs the library. The recommended usage of this functionality is the following:

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

The *run_tests.py* script in the *Tests* folder provides more specific testing options. To get all the information execute the following command:

```
python Tests/run_tests.py --help
```

2.5 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*

The version 1.01 of MEvoLib includes an interface and a dictionary with common parameterizations of three software tools: Mafft, Clustal Omega and Muscle.

```
>>> from MEvoLib import Align
>>> Align.get_tools()
["mafft", "clustalo", "muscle"]
```

The 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* Biopython object.

```
>>> 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", args="--globalpair --thread 2 --quiet")
```

The fourth example saves the resultant Muscle alignment in a PHYLIP output file (*align.phy*) instead of saving it in a variable. The alignment is always returned even if it is also saved in an output file.

```
>>> from MEvoLib import Align
>>> Align.get_alignment("muscle", "inseqs.fasta", "fasta",
...     args="fastdna", outfile="align.phy",
...     outfile_format="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", args="-F", informats=["fasta"],
...     incmd="-d=")
```

3.2 MEvoLib.Cluster: *Group sequences or fragments*

The version 1.01 of MEvoLib includes four different clustering methods: naïve rows, naïve columns, genes and padded-Recursive-DCM3 decomposition (*PRD*).

```
>>> from MEvoLib import Cluster
>>> Cluster.get_tools()
["genes", "prd", "rows", "cols"]
```

The first example shows a naïve row clustering, where the input dataset is dividied in 200 sets with roughly the same number of sequences. The output will be a dictionary of unique keys, with one list of *Bio.SeqRecord.SeqRecord* Biopython objects 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 corresponds 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 included in the GENBANK format of each sequence. The *Genes* method retrieves the information in the feature fields of the metadata. The NCBI databases has a predefined list of feature keys that has been also included in the method.

```
>>> from MEvoLib.Cluster import Genes
>>> Genes.get_features()
["misc_feature", "STS", ..., "gap", "mRNA", ...,
"tmRNA", ..., "CDS", "tRNA"]
```

To cluster the genes of a dataset, the user can extract each available feature, or provide a list of features to keep, discarding the others. The *unprocessable* feature is created by the method to store all those information-less sequences (if any).

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

Furthermore, the *Genes* method can generate a log file with every pair of terms referring to the same gene that might be the result of an error in the metadata. For instance, if a sequence has a typo for two different

genes (e.g. ATP8 gene is referred as "ATPase 6" in only one sequence and, therefore, these two genes will be merged in the results), the statistical sampling threshold calculated by the *Genes* method will mark that pair of terms as suspicious and write them in the log file.

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

The *unprocessable* sequences can be handled by the method itself. To do so, the user must provide a reference sequence (either by a name of the MEvoLib.Data contents or by a file path) and an alignment tool.

```
>>> from MEvoLib import Cluster
>>> dec = Cluster.get_subsets("genes", "hmtDNA.gb", "gb",
...       feature_filter=["D-loop", "tRNA", "rRNA", "CDS"],
...       log_file="hmtDNA.log", ref_seq="rCRS",
...        alignment_bin="mafft")
>>> print(sorted(dec.keys()))
["CDS.atp6", ..., "D-loop.D-loop", "rRNA.rnr1", ...,
"tRNA.trnf", ..., "unprocessable"]
```

The last example clusters the sequences using the 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",
...       tree_file="intree.tree", file_format="newick",
...       subset_size=250, overlapping=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 v1.01 there is only one sequence available: the revised Cambridge Reference Sequence (*rCRS*).

```
>>> from MEvoLib import Data
>>> Data.get_refseqs()
["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*

This module allows the user to automatically fetch and merge sequences and trees from local files or NCBI databases. The first example shows how to load sequences from a local file and from a NCBI database. Currently, the NCBI databases available are: *nuccore* (GenBank), *nuccest*, *nucgss*, *popset* and *protein*.

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

The following example shows the available possibilities to access the data of the *BioSeqs* objects created in the previous 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"
[porgn] AND mitochondrion[Filter]
>>> hmtDNA.statistics()
(10, 14926.0, 4924.3348992528927, 153, 16571)
>>> hmtDNA.update()
>>> len(hmtDNA)
37728
```

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

```
>>> hmtDNA.write("hmtDNA.gb")
>>> len(seqdb)
83
```

```
>>> seqdb.join(hmtDNA)
>>> len(seqdb)
37811
```

Finally, we can load again a previously saved *BioSeqs* object and include new sequences from a sequence file without the need to create another *BioSeqs* object first.

```
>>> hmtDNA = BioSeqs.from_bioseqs("hmtDNA.gb")
>>> hmtDNA.include("inseqs.fasta", "fasta")
>>> len(hmtDNA)
37811
```

The next example shows the same functionalities as the previous four examples, but with *PhyTrees* (excluding the NCBI 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")
>>> treedb2.include("intrees1.newick", "newick")
>>> treedb3.from_phytrees("new_trees.newick")
```

3.5 MEvoLib.Inference: *Phylogenetic inference*

MEvoLib v1.01 includes two different phylogenetic inference software tools: FastTree and RAxML.

```
>>> from MEvoLib import Inference
>>> Inference.get_tools()
{"inference": ["raxml", "fasttree"],
 "bootstrap": []}
```

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

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

The next example estimates a phylogeny with FastTree using its *default* configuration (*GTR+CAT* evolution model), and returns the resultant tree in the *tree* variable, which will be a *Bio.Phylo.Newick.Tree* Biopython object. The method also returns the log-likelihood score of the phylogeny.

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

The third example infers a phylogeny with RAxML using the *default* configuration (*GTRCAT* evolution model), saving the resultant phylogeny in the NEWICK file *tree.newick*.

```
>>> from MEvoLib import Inference
>>> tree, score = Inference.get_phylogeny("raxml",
...     "inseqs.fasta", "fasta", args="default",
...     outfile="tree.newick", outfile_format="newick")
```

The last example estimates a phylogenetic tree using RAxML with a different parameterization and generating 100 bootstraps to add more statistical robustness to the inference process.

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

3.6 MEvoLib.PhyloAssemble: *Supertrees and consensus*

The current version of MEvoLib only includes the Consense method from PHYLIP in this module. At least one supertree method will be included in the next update of MEvoLib.

```
>>> from MEvoLib import PhyloAssemble
>>> PhyloAssemble.get_tools()
{"supertree": [], "consensus": ["consense"]}
```

The first example shows how to know the available keywords (and their corresponding arguments) for a specific tool.

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

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

```
>>> 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", args="default",
...     outfile="cons_tree.newick", outfile_format="newick")
```

4 Biological examples

In this section we provide a step-by-step example of the usage of MEvoLib in two real phylogenetic studies. These two tests require at least 12GB of available RAM to run.

4.1 Study of the *MT-ATP6* and *MT-ATP8* genes

We present here an example illustrating the potential of MEvoLib following the workflow presented in Figure 1. Our study is focused on the molecular evolution of two hmtDNA genes: *MT-ATP6* and *MT-ATP8*. We want to know if they have evolved similarly based on the following information: **i)** both genes encode proteins that belong to the same subunit of the ATP synthase enzyme; and, **ii)** these two genes are located in the same strand and they have an overlapping of 42bp.

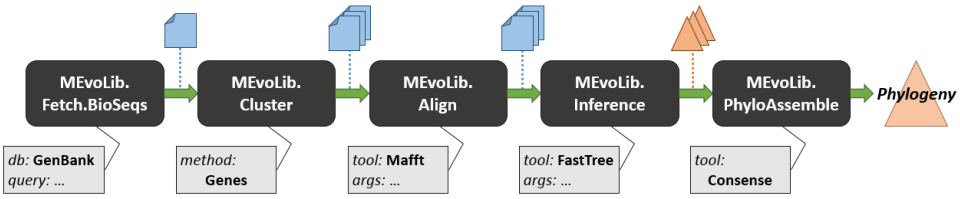

Figure 1: A common molecular evolution workflow using MEvoLib modules: data fetching, clustering, multialignment, phylogenetic inference and building of the consensus tree

First, we are going to download all the hmtDNA sequences from GenBank that have any information related with the *MT-ATP6* and *MT-ATP8* genes. The basic query to achieve this purpose would be: *"homo sapiens"[porgn] AND mitochondrion[Filter] NOT mRNA[Filter] AND (atp6 OR atp8)*. But these genes are referred with different nomenclatures in the GenBank metadata. Therefore, our fetching source code is:

```
from MEvoLib.Fetch.BioSeqs import BioSeqs

seq_db = BioSeqs.from_entrez(email="eg@test.com",
    entrez_db="nuccore",
    query='"homo sapiens"[porgn] AND mitochondrion[Filter] '  \
          'NOT mRNA[Filter] AND (atp6 OR atpase6 OR "atpase ' \
          '6" OR "atp synthase 6" OR "atpase subunit 6" OR '  \
          '"atp synthetase subunit 6" OR "atp synthase f0 '   \
          'subunit 6" OR "atp synthase fo subunit 6" OR atp8' \
          ' OR atpase8 OR "atpase 8" OR "atp synthase 8" OR ' \
          '"atpase subunit 8" OR "atp synthetase subunit 8" ' \
          'OR "atp synthase f0 subunit 8" OR "atp synthase '  \
          'fo subunit 8")')
seq_db.write('hmtDNA_ATP6.8_full.gb')
print(seq_db.statistics())
```

We would download 32548 sequences with the first query, whilst the one we have written in the source code fetches 32626 sequences (on 07/Jul/2016). We know two hmtDNA sequences (DQ862537.1 and FR695060.1) that have errors in their biological information, puzzling the genes' clustering of our target genes. Hence, we are going to remove them from the downloaded dataset:

```
for accession in ["DQ862537.1", "FR695060.1"] :
    del seq_db.data[accession]
seq_db.write("hmtDNA_ATP6.8_curated.gb")
print(seq_db.statistics())
```

Next, we need to extract the genes we are interested in from the filtered dataset. For this example, we are going to focus exclusively on the CDS feature. In a complete study, we would also require the *gene* feature to include those sequences that might have only tagged the genes under this feature. The following source code presents the gene extraction applying the *Genes* method:

```
from MEvoLib import Cluster

gene_dict = Cluster.get_subsets("genes",
    "hmtDNA_ATP6.8_curated.gb", "gb", feature_filter=["CDS"],
    log_file="hmtDNA_ATP6.8_curated.log")
results = {"atp6": [], "atp8": []}
for key, value in gene_dict.items() :
    if ( (len(value) > 0) and ("atp" in key) ) :
        if ( "6" in key ) :
            results["atp6"].extend(value)
        else : # "8" in key
            results["atp8"].extend(value)
```

We have saved the log file to be able to check all those pairs of terms that our method has merged that do not have enough statistical representation in the dataset. The *Genes* method only uses the metadata information provided to match all the terms of a single gene. Afterwards, we extract those elements in the dictionary that refer to the *MT-ATP6* and *MT-ATP8* genes and we save the corresponding sequence fragments in two separated FASTA files. We are going to build the phylogenetic trees with FastTree and the *GTR+CAT* evolution model. To do so, we need to align the gene sequences first. Thus, the following code implements the multialignment procedure using Mafft in its default configuration (*--auto*) and the phylogenetic inference of each gene:

```
from Bio import SeqIO
from MEvoLib import Align, Inference

for key, value in results.items() :
    SeqIO.write(value, "{}.fasta".format(key), "fasta")
    Align.get_alignment("mafft", "{}.fasta".format(key),
        "fasta", args = "default",
        outfile = "{}.aln".format(key),
        outfile_format = "fasta")
    tree, score = Inference.get_phylogeny("fasttree",
        "{}.aln".format(key), "fasta", args = "default",
        outfile = "{}.newick".format(key),
        outfile_format = "newick", bootstraps = 0)
    print("{}: {}".format(key, score))
```

As we can see, the resultant phylogeny is returned and saved in a NEWICK file, and its maximum likelihood score is also returned. The inference workflow for this research would be completed with these last results. For instance, to extend our study to include the last stage of the workflow shown in Figure 1, we could be interested in analyzing the consensus tree composed from the *MT-ATP6* phylogenies of different species, using the Consense method from PHYLIP in its default configuration (*majority rule consensus*):

```
from MEvoLib import PhyloAssembly

PhyloAssemble.get_consensus_tree("consense", "atps.newick",
    "newick", args="default", outfile="cons/atps.newick",
    outfile_format="newick")
```

4.2 Study of the bacterium *Borrelia burgdorferi s.l.*

The parasitology research group of Prof. Estrada-Peña was interested in obtaining a phylogenetic view of the strains of the bacterium *Borrelia burgdorferi* s.l. (*BB*). It is a human pathogen transmitted by several species of ticks of the genus *Ixodes* and reservoired by more than 400 species of hosts. The interest to produce a phylogeny of the literally thousands of strains recorded in the northern hemisphere is to associate the genetic profile with the given species of vectors or reservoirs. The multilocus sequence typing (*MLST*) is a method that uses the sequence of 8 inhouse genes of BB that produces the best known approach to the typing of the strains of BB (for further details: http://www.ncbi.nlm.nih.gov/pubmed/18574151). The complete dataset is available at PubMLST (http://pubmlst.org/borrelia/). The allel database contains 1535 sequences with 675 MLST profiles (on 15/Jul/2016).

We are going to use Biopython and MEvoLib to infer the phylogenetic tree. To do so, we need to compose the 675 MLST sequences from the 1535 alleles using the information provided by the MLST profile. The 8 allel files are available at our website in a *tar.gz* file (http://zaramit.unizar.es/download/datasets/borrelia_alleles.tar.gz) and we have also created a Python file with the MLST profiles saved in one list (http://zaramit.unizar.es/download/datasets/mlst_info.py). Therefore, the first step is to align each allel file with Mafft:

```
import os
from MEvoLib import Align

seq_files = [fname  for fname in os.listdir(".")
                        if ( fname.endswith(".fas") )]
```

```
for fname in seq_files :
    Align.get_alignment("mafft", fname, "fasta", args="default",
        outfile="{}.aln".format(fname[:-4]),
        outfile_format="fasta")
```

Analyzing the resultant files we can see that only the *clpA*, *clpX* and *pyrG* alleles were modified (the other alleles were already aligned). The next step is to create a *BioSeqs* object containing all the aligned sequences to retrieve them really fast by their identifier. We can also save the resultant database to avoid the alignment process in the future:

```
import os
from MEvoLib.Fetch.BioSeqs import BioSeqs

aln_files = [fname  for fname in os.listdir(".")
                        if ( fname.endswith(".aln") )]
seq_db = BioSeqs.from_seqfile(aln_files[0], "fasta")
for fname in aln_files[1:] :
    seq_db.include(fname, "fasta")
seq_db.write("bb_alleles.gb")
```

Then, we are ready to generate the 675 MLST sequences from the profile information (stored in *mlst_info.MLST*). Every MLST sequence is composed by 8 allels (one from each file):

```
from Bio import SeqIO
from Bio.SeqRecord import SeqRecord
from mlst_info import MLST

mlst_list = []
for entry in MLST :
    sequence = seq_db.data['clpA_{}'.format(entry[1])].seq + \
               seq_db.data['clpX_{}'.format(entry[2])].seq + \
               seq_db.data['nifS_{}'.format(entry[3])].seq + \
               seq_db.data['pepX_{}'.format(entry[4])].seq + \
               seq_db.data['pyrG_{}'.format(entry[5])].seq + \
               seq_db.data['recG_{}'.format(entry[6])].seq + \
               seq_db.data['rplB_{}'.format(entry[7])].seq + \
               seq_db.data['uvrA_{}'.format(entry[8])].seq
    record_id = 'mlst{:03d}'.format(entry[0])
    record = SeqRecord(sequence, id=record_id, name=record_id,
                       description='')
    mlst_list.append(record)
num_seqs = SeqIO.write(mlst_list, 'mlst.fasta', 'fasta')
print(num_seqs)
```

Finally, with all the MLST sequences saved in the *mlst.fasta* file, we can proceed to infer the phylogenetic tree for the BB. In this case, we are going

to apply the *GTR+CAT* evolution model through FastTree:

```
from MEvoLib import Inference

tree, score = Inference.get_phylogeny('fasttree', 'mlst.fasta',
    'fasta', args='default', outfile='mlst.newick',
     outfile_format='newick')
print(score)
```