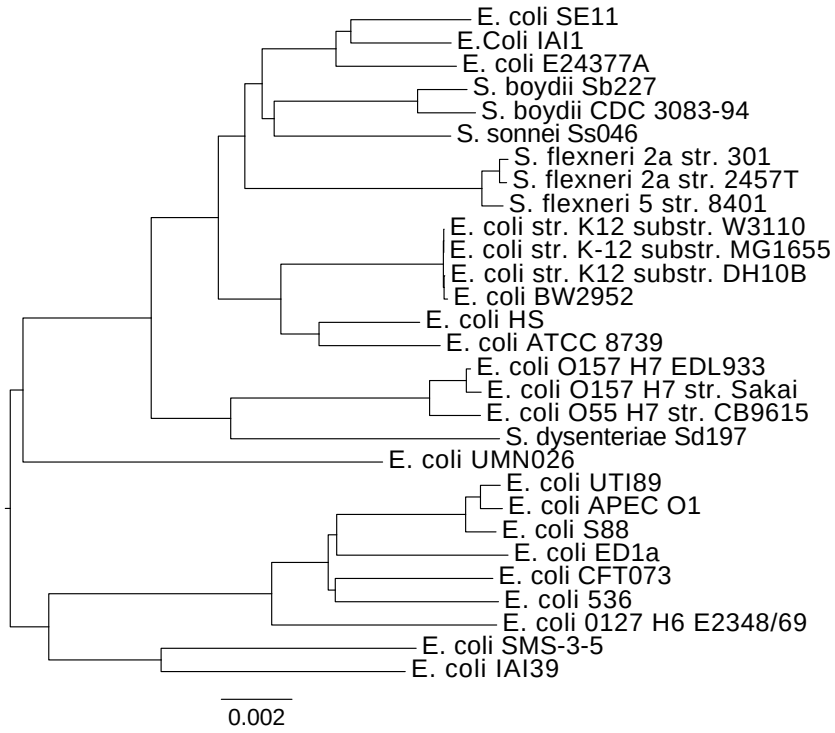
Documentation of ANDI

Rapid Estimation of Evolutionary Distances between Genomes

https://github.com/EvolBioInf/andi

Fabian Klötzl
kloetzl@evolbio.mpg.de

Version 0.9, 2015-04-30



Abstract

This is the documentation of the ANDI program for estimating the evolutionary distance between closely related genomes. It efficiently and accurately computes a distance for substitution rates up to 0.5. These distances are based on ungapped local alignments framed by exact matches, termed *anchors*. Anchors are efficiently found using an enhanced suffix array. As a result, ANDI scales well, even for data sets containing thousands of bacterial genomes.

License

Contents

1 Installation

The easiest way to obtain your own copy of ANDI, is to download the latest release from GitHub.[1] Please note, that ANDI requires LIBDIVSUFSORT[2] for optimal performance. If you wish to install ANDI without LIBDIVSUFSORT, consult Section 1.2.

ANDI runs best under a UNIX command line. All following examples are also intended for that environment.

1.1 Installing from Source Package

Once you have downloaded the package, unzip it and change into the newly created directory.

```
~ % tar -xzvf andi-0.9.tar.gz
~ % cd andi-0.9
```

Now build and install ANDI .

```
~/andi-0.9 % ./configure
~/andi-0.9 % make
~/andi-0.9 % sudo make install
```

This installs ANDI for all users on your system. If you do not have root privileges, you will find a working copy of ANDI in the `src` subdirectory. For the rest of this documentation, I will assume, that ANDI is in your $PATH.

Now ANDI should be ready for use. Try invoking the help.

```
~/andi-0.9 % andi -h
Usage: andi [-jrv] [-p FLOAT] FILES...
     FILES... can be any sequence of FASTA files. If no files are
        supplied, stdin is used instead.
Options:
 -j, --join   Treat all sequences from one file as a single genome
 -m, --low-memory Use less memory at the cost of speed
 -p <FLOAT>   Significance of an anchor pair; default: 0.05
 -r, --raw    Calculates raw distances; default: Jukes-Cantor
    corrected
 -v, --verbose Prints additional information
 -t <INT>     The number of threads to be used; default: 1
 -h, --help   Display this help and exit
    --version Output version information and acknowledgments
```

ANDI also comes with a man page, which can be accessed via **`man andi`**.

1.2 Installing without LIBDIVSUFSORT

In case you cannot or do not want to install LIBDIVSUFSORT, ANDI comes with its own implementation of a *suffix array construction algorithm*. It is called PSUFSORT and is also available, separately.[3] To activate it for ANDI, proceed as described in Section 1.1, but with the following twist:

[1] https://github.com/EvolBioInf/andi/releases
[2] https://github.com/y-256/libdivsufsort
[3] https://github.com/kloetzl/psufsort

```
~/andi % ./configure --without-libdivsufsort
```

Please note that this requires a C++11 compiler. Also, PSUFSORT may be slower than LIBDIVSUFSORT and thus lead to inferior runtimes.

1.3 Installing from Git Repository

To build ANDI from the GIT repo, execute the following steps.

```
~ % git clone git@github.com:EvolBioInf/andi.git
~ % cd andi
~/andi % autoreconf -i
```

For old versions of AUTOCONF you may instead have to use `autoreconf -i -Im4`.
Continue with the GNU trinity as described in Section 1.1.

2 Usage

The input sequences for ANDI should be in FASTA format. Any number of files can be passed. Each file may contain more than one sequence.

```
~ % andi S1.fasta S2.fasta
2
S1      0.0000 0.0979
S2      0.0979 0.0000
```

If no file argument is given, ANDI reads the input from STDIN. This makes it convenient to use in UNIX pipelines.

```
~ % cat S1.fasta S2.fasta | andi
2
S1      0.0000 0.0979
S2      0.0979 0.0000
```

The output of ANDI is a matrix in PHYLIP style: On the first line the number of compared sequences is given, 2 in our example. Then the matrix is printed, where each line is preceded by the name of the ith sequence. Note that the matrix is symmetric and the main diagonal contains only zeros. The numbers themselves are evolutionary distances, estimates as substitution rates.

2.1 Input

As mentioned before, ANDI reads in FASTA files. It recognizes only the four standard bases and is case insensitive (RegEx: `[acgtACGT]`). All other residue symbols are excluded from the analysis and ANDI prints a warning, when this happens.

If instead of distinct sequences, a FASTA file contains contigs belonging to a single taxon, ANDI will treat them as a unit when switched into JOIN mode. This can be achieved by using the `-j` or `--join` command line switch.

```
~ % andi --join E_coli.fasta Shigella.fasta
[Output]
```

When the JOIN mode is active, the file names are used to label the individual sequences. Thus, in JOIN mode, each genome has to be in its own file, and furthermore, at least one filename has to be given via the command line.

If ANDI seems to take unusually long, or requires huge amounts of memory, then you might have forgotten the JOIN switch. This makes ANDI compare each contig instead of each genome, resulting in many more comparisons! To make ANDI output the number of genome it about to compare, use the `--verbose` switch.

2.2 Output

The output of ANDI is written to `stdout`. This makes it easy to use on the command line and within shell scripts. As seen before, the matrix, computed by ANDI, is given in PHYLIP format.

```
~ % cat S1.fasta S2.fasta | andi
2
S1      0.0000 0.0979
S2      0.0979 0.0000
```

If the computation completed successfully, ANDI exits with the status code 0. Otherwise, the value of `errno` is used as the exit code. ANDI can also produce warnings and error messages for the user's convenience. These messages are printed to `stderr` and thus do not interfere with the normal output.

2.3 Options

ANDI takes a small number of commandline options, of which even fewer are of interest on a day-to-day basis. If **`andi`** `-h` displays a `-t` option, then ANDI was compiled with multi-threading support (implemented using OpenMP). By default ANDI uses all available processors. However, to restrict the number of threads, use `-t`.

```
~ % time andi ../test/1M.1.fasta -t 1
2
S1     0.0000 0.0995
S2     0.0995 0.0000
./andi ../test/1M.1.fasta 0,60s user 0,01s system 99% cpu 0,613 total
~ % time andi ../test/1M.1.fasta -t 2
2
S1     0.0000 0.0995
S2     0.0995 0.0000
./andi ../test/1M.1.fasta -t 2 0,67s user 0,03s system 195% cpu 0,362
    total
```

In the above examples the runtime dropped from 0.613 s, to 0.362 s using two threads. Giving ANDI more threads than input genomes leads to no further speed improvement. The other important option is `--join` (see Section 2.1).

By default, the distances computed by ANDI are *Jukes-Cantor* corrected. This means, the output is substitution rates, which is greater than the mismatch rate. To obtain the pure mismatch rate, use the `--raw` switch.

2.4 Example: ECO29

Here is a real-world example of how to use ANDI. As a data set we use ECO29; 29 genomes of *E. Coli* and *Shigella*. You can download the data here: `http://guanine.evolbio.mpg.de/andi/eco29.fasta.gz`. The genomes have an average length of 4.9 million nucleotides amounting to a total 138 MB.

ECO29 comes a single FASTA file, where each sequence is a genome. To calculate their pairwise distances, enter

```
~ % andi eco29.fasta > eco29.mat
andi: The input sequences contained characters other than acgtACGT.
   These were automatically stripped to ensure correct results.
```

The ECO29 data set includes non-canonical nucleotides, such as Y, N, and P, which get stripped from the input sequences. The resulting matrix is stored in `eco29.mat`; Here is a small excerpt:

```
~ % head -n 5 eco29.mat | cut -d ' ' -f 1-5
29
gi|563845 0.0000e+00 1.8388e-02 1.8439e-02 2.6398e-02
gi|342360 1.8388e-02 0.0000e+00 4.4029e-04 2.6166e-02
gi|300439 1.8439e-02 4.4029e-04 0.0000e+00 2.6123e-02
gi|261117 2.6398e-02 2.6166e-02 2.6123e-02 0.0000e+00
```

From this we compute a tree via neighbor-joining using a PHYLIP wrapper called EMBASSY.[1]

```
~ % fneighbor -datafile eco29.mat -outfile eco29.phylipdump
```

[1] `http://emboss.sourceforge.net/embassy/#PHYLIP`

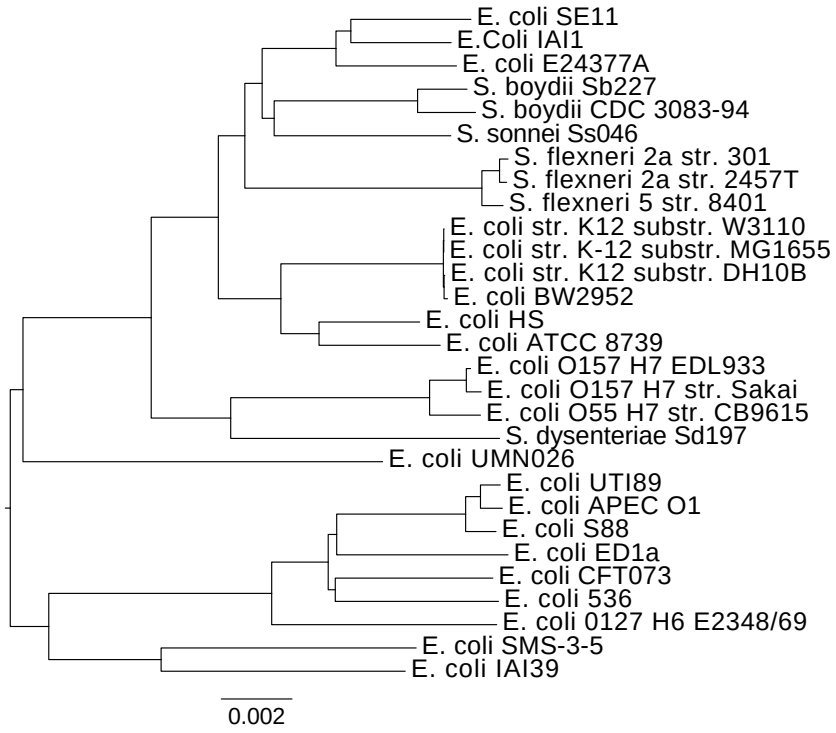
To make this tree easier to read, we can midpoint-root it.

```
~ % fretree -spp 29 -intreefile eco29.treefile -outtreefile eco29.
    tree <<EOF
M
X
Y
R
EOF
```

The file `eco29.tree` now contains the tree in Newick format. This can be plotted using

```
~ % figtree eco29.tree &
```

to yield



3 DevOps

ANDI is written in C/C++; mostly C99 with some parts in C++11. The sources are released on GITHUB as *free software* under the GNU GENERAL PUBLIC LICENSE VERSION 3 [?]. Prebundled packages using AUTOCONF are also available, with the latest release being v0.9 at the time of writing.

To prove correctness even under exceptional circumstances, the code was statically analyzed by the SCAN-BUILD utility from the LLVM framework [?].

3.1 Code Documentation

Every function in ANDI is documented using DOXYGEN style comments. To create the documentation run **`make`** `code-docs` in the main directory. You will then find the documentation under `./docs`.

3.2 Unit Tests

The unit tests are located in the ANDI repository under the **`./test`** directory. Because they require GLIB2, and a C++11 compiler, they are deactivated by default. To enable them, execute

```
~/andi % ./configure --enable-unit-tests
```

during the installation process. You can then verify the build via

```
~/andi % make check
```

The unit tests are also checked each time a commit is sent to the repository. This is done via TRAVISCI.[1] Thus, a warning is produced, when the builds fail, or the unit tests to not run successfully. Currently, the unit tests cover more than 80% of the code. This is computed via the TRAVIS builds and a service called COVERALLS.[2]

3.3 Known Issues

These minor issues are known. I intend to fix them, when I have time.

1. Stripping the filename from its path in join mode will not work properly under Windows because of the different path separator.
2. I currently do not check for integer multiplication overflows in mallocs. To fix this, I might be using `reallocarray` from BSD.

3.4 Creating a Release

A release should be a stable version of ANDI with significant improvements over the last version. For bug fixes, dotdot release may be used.

Once ANDI is matured, the new features implemented, and all tests were run, a new release can be created. First, increase the version number in `configure.ac`. Commit that change in git, and tag this commit with `vX.y`. Create another commit, where you set the version number to the next release (e. g., `vX.z-beta`).

[1] `https://travis-ci.org/EvolBioInf/andi`
[2] `https://coveralls.io/r/EvolBioInf/andi`

This assures that there is only one commit and build with that specific version. Now return to the previous commit **git** checkout vX.y.

Ensure that ANDI is ready for packaging with AUTOCONF.

```
~ % make distcheck
make dist-gzip am__post_remove_distdir='@:'
make[1]: Entering directory '/home/kloetzl/Projects/andi'
if test -d "andi-0.9.1-beta"; then find "andi-0.9.1-beta" -type d !
   -perm -200 -exec chmod u+w {} ';' && rm -rf "andi-0.9.1-beta" || {
    sleep 5 && rm -rf "andi-0.9.1-beta"; }; else :; fi
test -d "andi-0.9.1-beta" || mkdir "andi-0.9.1-beta"
 (cd src && make top_distdir=../andi-0.9.1-beta distdir=../andi
    -0.9.1-beta/src \
   am__remove_distdir=: am__skip_length_check=: am__skip_mode_fix=:
      distdir)

... Loads of output ...

================================================
andi-0.9.1-beta archives ready for distribution:
andi-0.9.1-beta.tar.gz
================================================
```

If the command does not build successfully, no tarballs will be created. This may necessitate further study of AUTOCONF and AUTOMAKE.