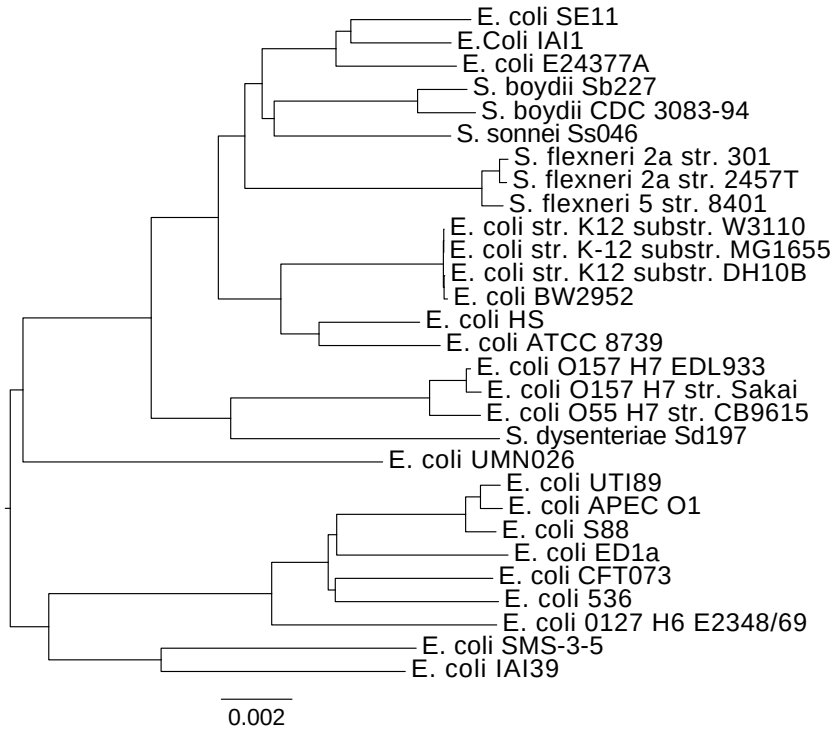
Documentation of ANDI

Rapid Estimation of Evolutionary Distances between Genomes

https://github.com/EvolBioInf/andi

Fabian Klötzl
kloetzl@evolbio.mpg.de

Version 0.13, 2020-02-11



Abstract

This is the documentation of the ANDI program for estimating the evolutionary distance between closely related genomes. These distances can be used to rapidly infer phylogenies for big sets of genomes. Because ANDI does not compute full alignments, it is so efficient that it scales well up to thousands of bacterial genomes.

This is scientific software. Please cite our paper [**?**] if you use ANDI in your publication. Also refer to the paper for the internals of ANDI. Additionally, there is a Master's thesis with even more in depth analysis of ANDI [**?**].

License

Contents

1 Installation

1.1 Package Manager

The easiest way to install ANDI is via a package manager. This also handles all dependencies for you. Debian and Ubuntu:

```
~ % sudo apt-get install andi
```

macOS with homebrew:

```
~ % brew tap brewsci/bio
~ % brew install andi
```

ArchLinux AUR package with aura:

```
~ % aura -A andi
```

ANDI is intended to be run in a UNIX commandline such as `bash` or `zsh`. All examples in this document are also intended for that environment. You can verify that ANDI was installed correctly by executing **`andi`** `-h`. This should give you a list of all available options (see Section 2.3).

1.2 Source Package

To build ANDI from source, download the latest release from GitHub. Please note, that ANDI requires the GNU SCIENTIFIC LIBRARY and LIBDIVSUFSORT[1] for optimal performance [?].

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

```
~ % tar -xzvf andi-0.12.tar.gz
~ % cd andi-0.12
```

Now build and install ANDI.

```
~/andi-0.12 % ./configure
~/andi-0.12 % make
~/andi-0.12 % 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, it is assumed, that ANDI is in your $PATH.

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

```
~/andi-0.12 % Usage: andi [OPTIONS...] FILES...
 FILES... can be any sequence of FASTA files.
 Use '-' as file name to read from stdin.
Options:
 -b, --bootstrap=INT Print additional bootstrap matrices
    --file-of-filenames=FILE Read additional filenames from FILE; one
       per line
 -j, --join     Treat all sequences from one file as a single genome
 -l, --low-memory Use less memory at the cost of speed
 -m, --model=MODEL Pick an evolutionary model of 'Raw', 'JC', 'Kimura'
     ; default: JC
```

[1] https://github.com/y-256/libdivsufsort

```
-p FLOAT      Significance of an anchor; default: 0.025
   --progress=WHEN Print a progress bar 'always', 'never', or 'auto';
       default: auto
-t, --threads=INT Set the number of threads; by default, all
    processors are used
   --truncate-names Truncate names to ten characters
-v, --verbose  Prints additional information
-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**.

1.3 Installing from Git Repository

To build ANDI from the GIT repo, you will also need the AUTOTOOLS. Refer to your OS documentation for installation instructions. Once done, execute the following steps.

```
~ % git clone git@github.com:EvolBioInf/andi.git
~ % cd andi
~/andi % autoreconf -fi -Im4
```

Continue with the GNU trinity as described in Section 1.2.

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, estimated from 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 not enough file names are provided, ANDI will try to read sequences from the standard input stream. This behaviour can be explicitly triggered by passing a single dash (`-`) as a file name, which is useful in pipelines.

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! Since version 0.12 ANDI produces a progressmeter on the standard error stream. ANDI tries to be smart about when to show or hide the progress bar. You can manually change this behaviour using the `--progress` option.

Starting with version 0.11 ANDI supports an extra way of input. Instead of passing file names directly to ANDI via the commandline arguments, the file names may also be read from a file itself. Using this new `--file-of-filenames` argument can work around limitations imposed be the shell.

The following three snippets have the same functionality.

```
~ % andi --join *.fasta
[Output]
```

```
~ % ls *.fasta > filenames.txt
~ % andi --join --file-of-filenames filenames.txt
[Output]
```

```
~ % ls *.fasta | andi --join --file-of-filenames -
[Output]
```

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 $0.613\,\mathrm{s}$, to $0.362\,\mathrm{s}$ 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 [?]. Other evolutionary models are also implemented (Kimura, raw). The `--model` parameter can be used to switch between them.

Since version 0.9.4 ANDI includes a bootstrapping method. It can be activated via the `--bootstrap` or `-b` switch. This option takes a numeric argument representing the number of matrices to create. The output can then be piped into PHYLIP. For more information on computing support values from distance matrices see [?].

```
~ % andi -b 2 ../test/1M.1.fasta
2
S1      0.0000 0.1067
```

```
S2      0.1067 0.0000
2
S1      0.0000 0.1071
S2      0.1071 0.0000
```

The original PHYLIP only supports distance matrices with names no longer than ten characters. However, this sometimes leads to problems with long accession numbers. Starting with version 0.11 ANDI prints the full name of a sequence, even if it is longer than ten characters. If your downstream tools have trouble with this, use `--truncate-names` to reimpose the limit.

Also new in version 0.11 is the `--file-of-filenames` option. See Section 2.1 for details.

2.4 Example: ECO29

Here follows a real-world example of how to use ANDI. It makes heavy use of the commandline and tools like PHYLIP. If you prefer R, check out this excellent blog post by Kathryn Holt.[1]

As a data set we use ECO29; 29 genomes of *E. Coli* and *Shigella*. You can download the data from 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.[2]

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

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

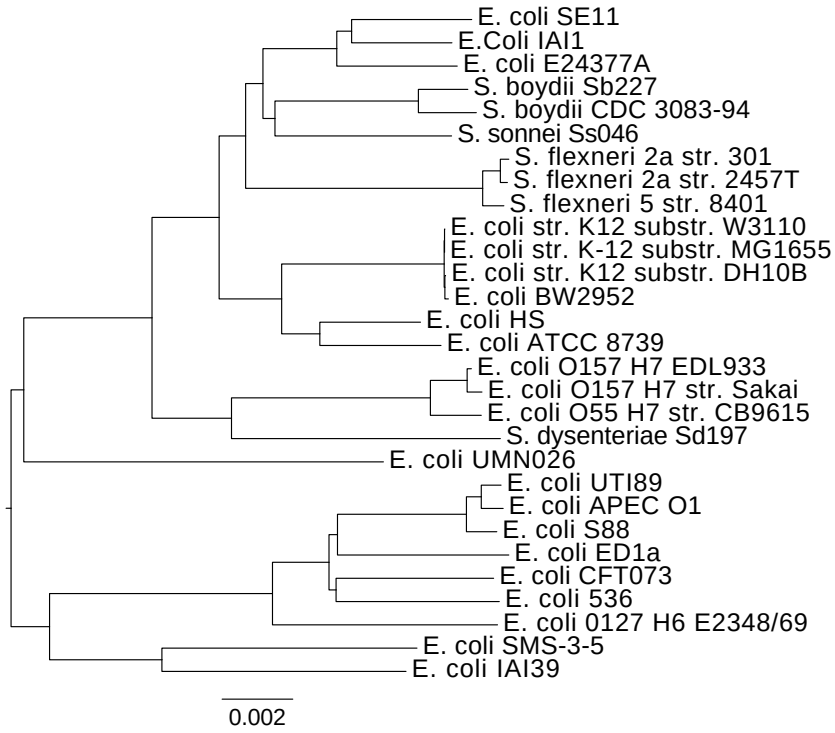
```
~ % fretree -spp 29 -intreefile eco29.treefile -outtreefile eco29.tree
    <<EOF
M
X
Y
R
EOF
```

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

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

to yield

[1] `http://holtlab.net/2015/05/08/r-code-to-infer-tree-from-andi-output/`
[2] `http://emboss.sourceforge.net/embassy/#PHYLIP`


E. coli SE11
E.Coli IAI1
E. coli E24377A
S. boydii Sb227
S. boydii CDC 3083-94
S. sonnei Ss046
S. flexneri 2a str. 301
S. flexneri 2a str. 2457T
S. flexneri 5 str. 8401
E. coli str. K12 substr. W3110
E. coli str. K-12 substr. MG1655
E. coli str. K12 substr. DH10B
E. coli BW2952
E. coli HS
E. coli ATCC 8739
E. coli O157 H7 EDL933
E. coli O157 H7 str. Sakai
E. coli O55 H7 str. CB9615
S. dysenteriae Sd197
E. coli UMN026
E. coli UTI89
E. coli APEC O1
E. coli S88
E. coli ED1a
E. coli CFT073
E. coli 536
E. coli 0127 H6 E2348/69
E. coli SMS-3-5
E. coli IAI39
0.002

3 Warnings and Errors

Here be an explanation of all possible errors. Other errors may occur and are due to the failure of underlying functions (e. g. **read**(3)). All warning messages are printed to `stderr`. Most errors are non-recoverable and will result in ANDI exiting with a non-zero state.

3.1 Sequence Related Messages

Unexpected Character

ANDI is pretty pedantic about the formatting of FASTA files. If you violate the syntax, ANDI will print the file name, the line and the problematic character. These errors are non-recovering, meaning no further sequences are read from the invalid file. The checks are implemented by the PFASTA library.

Non acgtACGT Nucleotides Stripped

Our models of genome evolution (JC, Kimura) only work on the four canonical types of nucleotides. All others are stripped from the sequences. This can be ignored in most cases.

Too Short Sequence

ANDI was designed for big data sets of whole genomes. On short sequences the distance estimates are inaccurate. Use an multiple sequence alignment instead.

Too Long Sequence

LIBDIVSUFSORT limits the length of a sequence to 31 bits. That count includes the reverse complement. So the technical limit for a sequence analysis is $2^{30} = 1.073.741.824$. Unfortunately, that excludes (full) human and mice genomes. Per-chromosome analysis works just fine.

Empty Sequence

One of the given sequences contained either no nucleotides at all, or only non-canonical ones.

Less than two sequences given

As ANDI tries to compare sequences, at least two need to be supplied. Note that ANDI may have regarded some of your given sequences as unusable.

Maximum Number of Sequences

The maximum number of sequences ANDI can possible compare is huge (roughly $457.845.052$). I doubt anyone will ever reach that limit. Please send me a mail, if you do.

3.2 Technical Messages

Out of Memory

If ANDI runs out of memory, it gives up. Either free memory, run ANDI on a bigger machine, try the `--low-memory` mode or reduce the number of threads.

RNG allocation

Some technical thing failed. If it keeps failing repeatedly, file a bug.

Bootstrapping failed

This should not happen.

Failed index creation

This should not happen, either.

Skipped and ignored Arguments

Some command line parameters of ANDI require arguments. If these are not of the expected type, a warning is given. See Section 2.3 for their correct usage.

3.3 Output-related Warnings

As the input sequences get more evolutionary divergent, ANDI finds less homologous anchors. With less anchors, less nucleotides are considered homologous between two sequences. If no anchors are found, comparison fails and `nan` is printed instead. See our paper and especially Figure 2 for details.

NaN

No homologous sections were found. Your sequences are very divergent ($d > 0.5$) or sprout a lot of indels that make comparison difficult.

Little Homology

Very few anchors were found and thus only a tiny part of the sequences is considered homologous. Expect that the given distance is erroneous.

Too long name

If you added the `--truncate-names` switch and an input name is longer than ten characters, you will receive this warning.

4 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 0.13 at the time of writing.

If you are interested in the internals of ANDI, consult the paper [?] or my Master's thesis [?]. Both explain the used approach in detail. The latter emphasizes the used algorithms, data structures and their efficient implementation.

4.1 Dependencies

Here is a complete list of dependencies required for developing ANDI.

- A C and a C++11 compiler,
- the AUTOTOOLS,
- the GNU SCIENTIFIC LIBRARY,
- PDFLATEX with various packages for the manual,
- GIT,
- GLIB2 for the unit tests,
- DOXYGEN,
- and LIBDIVSUFSORT.

4.2 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`.

4.3 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 did not run successfully. Currently, the unit tests cover more than 75% of the code. This is computed via the TRAVIS builds and a service called COVERALLS.[2]

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

4.4 Known Issues

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

1. This code will not work under Windows. At two places Unix-only code is used: filepath-separators are assumed to be / and file-descriptors are used for I/O.
2. Unit tests for the bootstrapped matrices are missing.
3. Cached intervals are sometimes not "as deep as they could be". If that got fixed `get_match_cache` could bail out on `ij.lcp < CACHE_LENGTH`. However the `esa_init_cache` code is the most fragile part and should be handled with care.

4.5 Creating a Release

A release should be a stable version of ANDI with significant improvements over the last version. dotdot releases should be avoided.

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`. Tags should be annotated and signed, if possible. This manual then needs manual rebuilding.

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.

Also verify that the recent changes did not create a performance regression. This includes testing both ends of the scale: ECO29 and PNEU3085. Both should be reasonable close to previous releases.

Create another commit, where you set the version number to the next release (e. g., `vX.z-beta`). This assures that there is only one commit and build with that specific version.