The Aevol User Manual

for version 5.0 or newer

Contents

Introduction

1 What is Aevol?

Aevol is a digital genetics model: populations of digital organisms are subjected to a process of selection and variation, which creates a Darwinian dynamics.

By modifying the characteristics of selection (e.g. population size, type of environment, environmental variations) or variation (e.g. mutation rates, chromosomal rearrangement rates, types of rearrangements, horizontal transfer), one can study experimentally the impact of these parameters on the structure of the evolved organisms. In particular, since Aevol integrates a precise and realistic model of the genome, it allows for the study of structural variations of the genome (e.g. number of genes, synteny, proportion of coding sequences).

The simulation platform comes along with a set of tools to help analyse phylogenies and to measure many characteristics of the organisms and populations along evolution.

2 License

3 The AEVOL Community

AEVOL's primary ressource is its website `http://www.aevol.fr/` where you shall find information about the project and its contributors.

To subscribe to the (low trafic) users' mailing lists, please visit
`http://lists.gforge.liris.cnrs.fr/mailman/listinfo/aevol-users`.

You may also want to report bugs and ask for new features to be implemented.
To do so, simply write to
`aevol-bugs@lists.gforge.liris.cnrs.fr` or
`aevol-feat-request@lists.gforge.liris.cnrs.fr`

Chapter I

Installation

AEVOL can run on Linux and on OS X.

1 Linux users

1.1 Pre-built packages

AEVOL 4.4 is available as a deb package in all the repositories. So, you can `apt install aevol` provided the 4.4 version fits your needs. AEVOL 5 should also hit the repositories soon.

1.2 Installation from Source

Installation commands are given for Debian-like systems (`apt`) as well as Fedora-like systems (`dnf`).

Required Dependencies

- **Build Tools**.
  `apt install build-essential` or `dnf install gcc-c++`.
- **Compression library**. AEVOL compresses most of the data it generates.
  `apt install zlib1g-dev` or `dnf install zlib-devel`.
- **Boost library**. AEVOL also relies on the Boost Filesystem library.
  `apt install libboost-filesystem-dev` or `dnf install boost-devel`.

Optional Dependencies

- **X libraries**. AEVOL uses the X11 library for the graphical outputs. `apt install libx11-dev` or `dnf install libX11-devel`.

  Note, however, that AEVOL can be compiled without graphical outputs, and hence no need for X libraries, by typing `./configure --without-x` instead of `./configure` (see installation instructions below for more information). This option is useful if you want to run AEVOL on a computer cluster, for example.

Installation Instructions

Download the latest release of AEVOL at `http://aevol.fr/download/`and save it to a directory of your choice. Open a terminal and use the `cd` command to navigate to this directory. Then follow the steps below to extract the files and build the executables:

```
tar zxf aevol-VERSION.tar.gz
cd aevol-VERSION
./configure
make
```

If you have administration privileges, you can finally make the AEVOL programs available to all users on the computer by typing:

```
sudo make install
```

If you don't have administration privileges, you may still install AEVOL "locally" by doing the following:

```
./configure --prefix=/install/path
make
make install
```

where `/install/path` is a directory where you have write permission. Don't forget to add `/install/path` to your `PATH` environment variable.

2 Mac users

2.1 Pre-built packages

This option is not available yet for Mac users.

2.2 Installation from Source

Required Dependencies

- **C++ command-line compiler**. Mac users need a command-line C++ compiler like `g++` or `clang` installed.

  There are three options:

  App Store If your computer runs OS X El Capitan (10.11), you can first install XCode (freely downloadable from the App Store), then start XCode and install the Command Line Tools package from the menu XCode / Preferences / Downloads / tab " Components".

  Standalone Command Line Tools Alternatively, but still provided your computer runs OS X El Capitan (10.11), you can install the Command Line Tools package for Xcode (without installing Xcode itself) by downloading it from Apple's developer site. A free registration required there, then just search for "Command Line Tools".

  Homebrew For all versions of OS X, another way is to install Homebrew to get the appropriate packages. To install Homebrew, you have to open a terminal and type (or copy/paste, more likely) the following command:

  ```
  /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/
      Homebrew/install/master/install)"
  ```

  Then you can install the appropriate dependencies with:

  ```
  brew install boost clang-omp
  ```

- **Compression library**. Aevol compresses most of the data it uses using the zlib1g library. This library is already included as part of OS X so there is no need to install it.

Optional Dependencies

- **X libraries**. For the graphical outputs, Mac users should also have X11 installed. X11 is not included with OS X, but X11 server and client libraries for OS X are

available from the XQuartz project. You will need to log out and log in after the installation to have X11 properly setup. Note, however, that AEVOL can be compiled without graphical outputs, and hence no need for X libraries, by typing `./configure --without-x` instead of `./configure` (see below). This option is useful if you want to run AEVOL on a computer cluster, for example.

You can either download XQuartz manually from (`http://xquartz.macosforge.org`) or, if you previously chose the Homebrew way, you may just `brew install Caskroom/cask/x`
In either case, don't forget to re-log for proper install to be completed.

Installation Instructions

Download the latest release of AEVOL at `http://aevol.fr/download/`and save it to a directory of your choice. Open a terminal and use the `cd` command to navigate to this directory. Then follow the steps below to extract the files and build the executables:

```
tar zxf aevol-VERSION.tar.gz
cd aevol-VERSION
./configure
make
```

If you have administration privileges, you can finally make the AEVOL programs available to all users on the computer by typing:

```
sudo make install
```

If you don't have administration privileges, you may still install AEVOL "locally" by doing the following:

```
./configure --prefix=/install/path
make
make install
```

where `/install/path` is a directory where you have write permission. Don't forget to add `/install/path` to your `PATH` environment variable.

Chapter II

Tutorial: Using AEVOL

1 Introduction

AEVOL is made up of 4 main tools (`aevol_create`, `aevol_run`, `aevol_propagate` and `aevol_modify` – man pages provided in appendix 1) and a set of post-treatment tools (prefixed by `aevol_misc_`).

Everything in AEVOL relies on an ad-hoc file organization where all the data for an experiment is stored: organisms in the `populations` directory, the task they are selected for in `environment`, the experimental setup in `exp_setup` and so on. It is not recommended to manually modify these files since this may cause some inconsistency leading to undefined behaviour. Besides, most of these files are compressed.

Once created, an experiment can either be run, propagated or modified.

Running an experiment simply means simulate evolution for a given number of generations.

Propagating an experiment means creating a fresh copy of it (setting the current generation number to 0).

Modifying an experiment actually means modifying some of its parameters. The `aevol_modify` tool virtually allows for the modification of any parameter of the experiment, including manipulations of the whole population or of individual organisms (*e.g.* "I want the population to be filled with clones of the organism having the longest genome" or "I want a random subset of organisms to be switched to super mutators"). To date, only the most common experiment modifications have been implemented but feel free to ask for more (`aevol-feat-request@lists.gforge.liris.cnrs.fr`).

AEVOL comes along with a set of simple but representative examples. Following these

examples is probably the best way to get going with Aevol and have a quick overview of the possibilities it offers. In any case, keep in mind that you can always get help by typing `man aevol_cmd` (only available for the 4 main commands) or `aevol_cmd -h` (available for all the commands).

Most examples are showcases for different features of the model such as spatially structured populations, plasmids and horizontal transfer. They can all be run with the same very simple commands. Simply follow the instructions from section 2. The `workflow` example proposes a typical *"experiments on a previously generated wild-type"* workflow. It will lead you through the whole experimental process, including a sample of possible post-treatments you can use to analyse the outcome of your different simulations.

2 Basic examples

To run all but the `workflow` examples, simply follow the following steps:

1. Install Aevol, preferentially with graphics enabled (see chapter I)
2. cd into the directory of the example (*e.g.* `examples/basic`)
3. run `aevol_create`
4. run `aevol_run`
5. Have a look at the graphical outputs (Ctrl+Q to quit)

Optional Explore the different statistics created in the `stats` subdirectory.

3 The *workflow* example

The workflow example provides an example of one of the many different workflows that can be used for experiments with Aevol. The main idea underlying this workflow is to parallel wet lab experiments, which are conducted on evolved organisms. To use already evolved organisms for Aevol experiments, one can either use an evolved genome provided by the community or evolve one's own. This example describes the latter (more complete) case.

3.1 *Wild-Type* generation

Generating a Wild-Type in Aevol is very easy, all you need is a parameter file describing the conditions in which it (the Wild-Type) should be created (population size, mutation

rates, task to perform, ...). However, have in mind that founding effects can influence the course of evolution, especially in the case of overconstrained evolution. It is recommended to use mild mutation and rearrangement rates and to let the environment vary over time to avoid overconstrained or overspecialized genomes. A sample parameter file is provided in `examples/workflow/wild_type`. Once your parameter file is ready, simply run the following commands (it is recommended you do that in a dedicated directory, called `wild_type` for example):

```
cd wild_type
aevol_create -f your_param_file
aevol_run -n number_of_generations
```

3.2 Experimental setup

This is where the setup of the campaign of experiments is done. As it would be done in a wet lab experiment, different populations will be allowed to evolve in different conditions to compare the different outcomes. In this example, we will start from an evolved population called the "wild type", created as above. We will use this wild type to start 10 evolutionary lines that will have to adapt to a new environment. Five of them will evolve under the same rates of chromosomal rearrangements as the wild type, whereas the other five will be "mutators" evolving under higher rates of chromosomal rearrangements. Both groups will evolve during 10,000 generations.

First, the wild type population should have been created with `aevol_create` and `aevol_run -n 5000` (for example). Then, the `aevol_propagate` tool allows for an exact copy of the whole data structure required by AEVOL with a reset of the current generation number to 0. Followed by a call to `aevol_modify`, it allows us to set up our example in the 2 following steps:

Propagate the experiment

The `aevol_propagate` tool allows for the creation of fresh copies of an experiment (as it was at a given time). The `-i` option sets the input directory and the `-o` option, the output directory. You must provide a distinct output directory for each of the experiments you wish to run. If the output directory does not exist, it will be created. If, as we do here, you use `aevol_propagate` repeatedly to initialize several simulations, you should specify a different seed for each simulation, otherwise all simulations will yield exactly the same results. You can use the option -S to do so. In this case, the random drawings will be different for all random processes enabled in your simulations (mutations, stochastic gene expression, selection, migration, environmental variation, environmental noise). Alternatively, to change the random drawings for specific random processes only, do not use -S

but the options -m, -s, -t, -e, -n (see `aevol_propagate -h` for more information on those options).

```
cd ..
aevol_propagate -g 5000 -i wild_type -o line01 -S 97558
aevol_propagate -g 5000 -i wild_type -o line02 -S 535241
aevol_propagate -g 5000 -i wild_type -o line03 -S 1499
aevol_propagate -g 5000 -i wild_type -o line04 -S 916189
aevol_propagate -g 5000 -i wild_type -o line05 -S 677
aevol_propagate -g 5000 -i wild_type -o line06 -S 43743
aevol_propagate -g 5000 -i wild_type -o line07 -S 7265
aevol_propagate -g 5000 -i wild_type -o line08 -S 11942
aevol_propagate -g 5000 -i wild_type -o line09 -S 29734
aevol_propagate -g 5000 -i wild_type -o line10 -S 43155
```

Modify parameters to meet the experiment requirements

For each of the propagated experiments, create a plain text file (*e.g.* "newparam.in") containing the parameters to be modified. Parameters that do not appear in this file will remain unchanged. The syntax is the same as for the parameter file used for `aevol_create`. For example, for the lines 1 to 5, we will create a text file called "newparam-groupA.in" will consist in the following lines:

```
# New environment
 ENV_GAUSSIAN  0.5   0.2   0.05
 ENV_GAUSSIAN  0.5   0.4   0.05
 ENV_GAUSSIAN  0.5   0.8   0.05
 ENV_VARIATION none
```

For the lines 6 to 10, we also want to modify the rearrangement rates, hence the file "newparam-groupB.in" will consist in the following lines:

```
# New environment
 ENV_GAUSSIAN  0.5   0.2   0.05
 ENV_GAUSSIAN  0.5   0.4   0.05
 ENV_GAUSSIAN  0.5   0.8   0.05
 ENV_VARIATION none
# New rearrangement rates
 DUPLICATION_RATE        1e-5
 DELETION_RATE           1e-5
 TRANSLOCATION_RATE      1e-5
 INVERSION_RATE          1e-5
```

Then we will run the following commands:

```
cd line01; aevol_modify --gener 0 --file ../newparam-groupA.in; cd ..
cd line02; aevol_modify --gener 0 --file ../newparam-groupA.in; cd ..
cd line03; aevol_modify --gener 0 --file ../newparam-groupA.in; cd ..
cd line04; aevol_modify --gener 0 --file ../newparam-groupA.in; cd ..
cd line05; aevol_modify --gener 0 --file ../newparam-groupA.in; cd ..

cd line06; aevol_modify --gener 0 --file ../newparam-groupB.in; cd ..
cd line07; aevol_modify --gener 0 --file ../newparam-groupB.in; cd ..
cd line08; aevol_modify --gener 0 --file ../newparam-groupB.in; cd ..
cd line09; aevol_modify --gener 0 --file ../newparam-groupB.in; cd ..
cd line10; aevol_modify --gener 0 --file ../newparam-groupB.in; cd ..
```

3.3 Run the simulations

Each of the propagated experiments can be run thus:

```
aevol_run -n <number_of_generations>
```

Of course, all the runs being completely independent, you can submit these tasks to a cluster of your choice to save time.

3.4 Analyse the outcome

In addition to the set a statistics files that are recorded in the `stats` directory, Aevol includes a set of post-treatment tools to further analyse the outcome of your experiments, please refer to section 4.

4 Post-treatment Tools

In addition to the set a statistics files that are recorded in the `stats` directory, Aevol includes a set of post-treatment tools to further analyse the outcome of your experiments.

Please note that these tools have only been tested on simple experimental setups and can fail with exotic ones. For example, the tools listed below are fully functional under a single-chromosome setup, but are still under development for most complicated settings with both a chromosome and exchangeable plasmids. However, in most cases,

the problems can easily be remedied. Please do not hesitate to send us your request (aevol-feat-request@lists.gforge.liris.cnrs.fr).

4.1 aevol_misc_view_generation

The `view` tool is probably the easiest and most straightforward tool provided with Aevol. It allows one to visualize a generation using the exact same graphical outputs used in `aevol_run`. However, since it relies on graphics, it is only available when Aevol is compiled with X enabled (which is the default).

Usage: `aevol_misc_view -g generation_number`

There must have been a backup of the population at this generation. For example, if the program is called with the option `-g 4000`, there must be a file called `pop_004000.ae` in the `populations` directory.

4.2 aevol_misc_create_eps

The `create_eps` tool takes a generation number as an input, and produces several EPS files describing an individual of this population (the best one by default) at this generation:

- `best_genome_with_CDS.eps`, where the chromosome is represented by a circle, and coding sequences on the leading (resp. lagging) strand are drawn as arcs outside (resp. inside) the circle.
- `best_genome_with_mRNAs.eps`, where the chromosome is represented by a circle, and transcribed sequences on the leading (resp. lagging) strand are drawn as arcs outside (resp. inside) the circle. Gray arcs correspond to non-coding RNAs and black arcs correspond to coding RNAs.
- `best_phenotype.eps`, where the phenotype resulting from the interaction of all genes is superimposed to the environmental target.
- `best_triangles.eps`, where all triangles resulting from the translation of a coding sequence are superimposed.

Usage: `aevol_misc_create_eps [-i INDEX | -r RANK] -g GENER`

There must have been a backup of the population at this generation. For example, if the program is called with the option `-g 4000`, there must be a file called `pop_004000.ae` in the `populations` directory. The program will then create a subdirectory called `analysis-generation004000` and write the EPS files therein. If neither index nor rank are specified, the program creates the EPS files of the best individual.

4.3 aevol_misc_mutagenesis

This `mutagenesis` tool creates and evaluates single mutants of an individual saved in a backup, by default the best of its generation. Use option `-g` to specify the generation number contanining the individual of interest. There must have been a backup of the population at this generation. For example, if the program is called with the option `-g 4000`, there must be a file called `pop_004000.ae` in the `populations` directory.

Use either the `-r` or the `-i` option to select another individual than the best one: with `-i`, you have to provide the ID of the individual, and with `-r` the rank (1 for the individual with the lowest fitness, N for the fittest one).

The type of mutations to perform must be specified with the `-m` option. Choose 0 to create mutants with a point mutation, 1 for a small insertion, 2 for a small deletion, 3 for a duplication, 4 for a large deletion, 5 for a translocation or 6 for an inversion.

For the point mutations, all single mutants will be created and evaluated. For the other mutation types, an exhaustive mutagenesis would take too much time, hence only a sample of mutants (1000 by default) will be generated. Use option `-n` to specify another sample size.

The output file will be placed in a subdirectory called `analysis-generationGENER`.

Usage:

```
aevol_misc_mutagenesis -g GENER [-i INDEX | -r RANK]
                       [-m MUTATIONTYPE] [-n NBMUTANTS]
```

4.4 aevol_misc_robustness

The `robustness` tool computes the replication statistics of all the individuals of a given generation, like the proportion of neutral, beneficial, deleterious offsprings. This is done by simulating $NBCHILDREN$ replications for each individual (1000 replications by default), with its mutation, rearrangement and transfer rates. Depending on those rates and genome size, there can be several mutations per replication. Those global statistics are written in `analysis-generationGENER/robustness-allindivs-gGENER.out`, with one line per individual in the specified generation.

The program also outputs detailed statistics for one of the individuals (the best one by default). The detailed statistics for this individual are written in `analysis-generationGENER/robustness-singleindiv-details-gGENER-iINDEX-rRANK.out`, with one line per simulated child of this particular individual.

Usage: `aevol_misc_robustness -g GENER [-n NBCHILDREN] [-r RANK | -i INDEX]`

If neither index nor rank are specified, the program computes the detailed statistics for the best individual of generation `GENER`.

4.5 aevol_misc_lineage

The `lineage` tool allows for the reconstruction of the lineage of a given individual. It requires the phylogenetic tree to be recorded during the evolutionnary run (see the `TREE_MODE` parameter). Using this phylogenetic tree, it will produce a binary file containing the whole evolutionary history of any given individual, *i.e.* for each of its ancestors, which organism in the previous generation it is an offspring of, and the list of mutations that occured during replication. This file will be named *e.g.*
`lineage-b000000-e050000-i999-r1000.ae` which means we retraced the evolutionary history of the organism with rank 1,000 (that had the index 999) at generation 50,000 and that its history was retraced all the way down to generation 0. This file is not readable in a text editor, it is meant to be used by other programs like `ancstats`, `fixed_mutations` or `gene_families` (see below).

Usage: `aevol_misc_lineage [-i index | -r rank] [-b gener1] -e gener2`

If neither index nor rank are specified, the program creates the EPS files of the best individual of generation `gener2`.

4.6 aevol_misc_ancstats

The `ancstats` tool issues the "statistics" for the line of descent of a given individual (providing its lineage file, see section 4.5). It will produce a set of files similar to those created in the `stats` directory during the simulation but regarding the successive ancestors on the provided lineage, instead of the best organism of each generation. These files are placed in the `stats/ancstats` directory. The program works by loading the initial genome at the beginning of the lineage, and then by replaying each mutation recorded in the lineage file. Environmental variations are also replayed exactly as they occured during the main run.

Usage: `ae_misc_ancstats [-c | -n] [-t tolerance] -f lineage_file`

With the option `-c` or `--fullcheck` enabled, the program will check that the rebuilt genome sequence and the replayed environment are correct every `<BACKUP_STEP>` generations, by comparing them to the data stored in the backups in the `populations` and `environment` directories. The default behaviour is faster as it only performs these checks at the final generation only. The option `-n` or `--nocheck` diasbales genome sequence

checking completely. Although it makes the program faster, it is not recommended. The option `-t tolerance` is useful when `ancstats` in run on computer different from the one that performed the main evolutionary run: In this case, differences in compilators can lead to small variations in the computation of floating-point numbers. The tolerance specified with this option is used to decide whether the replayed environment is sufficienlty close to the one recorded during the main run in the `environment` directory.

4.7 aevol_misc_fixed_mutations

The `fixed_mutations` tool issues the detailed list of mutations that occurred in the lineage of a given individual (providing its lineage file, see section 4.5). This text file is placed in the `stats` directory. The program works by loading the initial genome at the beginning of the lineage, and then by replaying each mutation recorded in the lineage file. Environmental variations are also replayed exactly as they occured during the main run. The output file indicates, for each mutation, at which generation it occurred, which type of event it was (point mutation, small insertion, inversion...), where it occurred on the chromosome and how many genes (actually how many coding RNAs) where affected. More details are given in the first lines of the file itself.

Usage: `ae_misc_fixed_mutations [-c | -n] [-t tolerance] -f lineage_file`

With the option `-c` or `--fullcheck` enabled, the program will check that the rebuilt genome sequence and the replayed environment are correct every `<BACKUP_STEP>` generations, by comparing them to the data stored in the backups in the `populations` and `environment` directories. The default behaviour is faster as it only performs these checks at the final generation. The option `-n` or `--nocheck` disables genome sequence checking altogether. Although it makes the program faster, it is not recommended. The option `-t tolerance` is useful when `fixed_mutations` is run on computer different from the one that performed the main evolutionary run: In this case, differences in compilators can lead to small variations in the computation of floating-point numbers. The tolerance specified with this option is used to decide whether the replayed environment is sufficienlty close to the one recorded during the main run in the `environment` directory.

4.8 aevol_misc_gene_families

The `gene_families` tool issues the detailed history of each gene family on the lineage of a given individual (providing its lineage file, see section 4.5). A gene family is defined here as a set of coding sequences that arised by duplications of a single original gene. The original gene, called the root of the family, can either be one of the genes in the initial ancestor, or a new gene created from scratch (for example by a local mutation that transformed a non-coding RNA into a coding RNA). The history of gene duplications, gene losses and gene mutations in each gene family is represented by a binary tree. The

program starts by loading the initial genome at the beginning of the lineage and by tagging each gene in this initial genome. Each of these initial genes is marked as the root of a gene family. Then, each mutation recorded in the lineage file is replayed and the fate of all tagged genes is followed and recorded in their respective families. When a gene is duplicated, the corresponding node in one of the gene trees becomes an internal node, and two children nodes are added to it, representing the two gene copies. When a gene sequence is modified, the mutation is recorded in its corresponding node in one of the gene trees. When a gene is lost, the corresponding node in one of the gene trees is labelled as lost. When a new gene appears from scratch, i.e. not by gene duplication, it becomes the root of a new gene tree. Environmental variations are also replayed exactly as they occured during the main run.

When all mutations have been replayed, several output files are written in a directory called `gene_trees`. Two general text files are produced. The file called `gene_tree_statistics.txt` contains general data on each gene family, like its creation date, its extinction date, or how many nodes it contained. The file called `nodeattr_tabular.txt` contains information about each node of each gene tree, like when it was duplicated or lost or how many mutations occurred on its branch. In addition, for each gene tree, two text files are generated: a file called `genetree******-topology.tre` contains the topology of the gene tree in the Newick format, and a file called `genetree******-nodeattr.txt` that contains the list of events that happened to each node in the tree file, before it was either duplicated or lost.

Usage: `ae_misc_gene_families [-c | -n] [-t tolerance] -f lineage_file`

With the option `-c` or `--fullcheck` enabled, the program will check that the rebuilt genome sequence and the replayed environment are correct every `<BACKUP_STEP>` generations, by comparing them to the data stored in the backups in the `populations` and `environment` directories. The default behaviour is faster as it only performs these checks at the final generation only. The option `-n` or `--nocheck` diasbales genome sequence checking completely. Although it makes the program faster, it is not recommended. The option `-t tolerance` is useful when `gene_families` is run on computer different from the one that performed the main evolutionary run: In this case, differences in compilators can lead to small variations in the computation of floating-point numbers. The tolerance specified with this option is used to decide whether the replayed environment is sufficienlty close to the one recorded during the main run in the `environment` directory.

Appendix : AEVOL Parameters (param.in)

5 Initialization Parameters

5.1 INIT_POP_SIZE

Meaning

Initial Population Size (constant in many setups)

Default Value

$1,000$

5.2 INIT_METHOD

Meaning

Initialisation (bootstrapping) method.

It is strongly recommended to use the default method which is explained hereafter.

Default Value

ONE_GOOD_GENE CLONE

A random sequence of size INITIAL_GENOME_LENGTH is generated and evaluated with regard to the defined task. This process is repeated until the generated genome perform any subset of the task (*i.e.* has a better fitness than an organism with no genes). The population is then filled with clones of the generated organism.

5.3 INITIAL_GENOME_LENGTH

Meaning

Size of the initial, randomly generated genome(s).

Default Value

5,000

6 Artificial Chemistry Parameters

6.1 MAX_TRIANGLE_WIDTH

Meaning

Maximum degree of protein pleiotropy.

This value must be strictly greater than 0 (which would mean that a protein cannot do anything) and lower than 1 (which means that a protein can contribute to every possible metabolic process).

Default Value

0.033333333

7 Selection Parameters

7.1 SELECTION_SCHEME

Meaning

Selection scheme to use (`fitness_proportionate`, `linear_ranking` or `exponential_ranking`)

In the `fitness_proportionate` scheme, the probability of reproduction of each organism is proportional to its fitness. The probability of reproduction is proportional to $exp(-k \times g)$, where k determines the intensity of selection (it can be set using the SELECTION_PRESSURE keyword) and g is the "metabolic error" (see the model description).

The other two selection schemes are based on the rank of the organisms in the population, which allows one to maintain a constant selective pressure throughout the entire evolutionary process. Organisms are thus first sorted by increasing fitness (the worst individual in the population having rank 1). Then, their probability of reproduction can be computed depending on their rank r and according to whether the linear or exponential scheme is used.

For the `linear_ranking` scheme, the probability of reproduction of an individual is given by $p_{reprod} = \frac{1}{N} \times (\eta^- + (\eta^+ - \eta^-) \times \frac{r-1}{N-1})$, where $\frac{\eta^+}{N}$ and $\frac{\eta^-}{N}$ represent the probability of reproduction of the best and worst individual respectively. For the population size to remain constant, the sum over N of this expression must be equal to 1 and so η^- must be equal to $2 - \eta^+$. As for η^+, it must be chosen in the interval $[1, 2]$ so that the probability increases with the rank and remains in $[0, 1]$. To date, variable population size is not supported with the `linear_ranking` scheme, thus only η^+ is required and can be specified using the SELECTION_PRESSURE parameter

For the `exponential_ranking` scheme, the probability of reproduction is given by $p_{reprod} = \frac{c-1}{c^N-1} \times c^{N-r}$, where $c \in]0, 1[$ determines the intensity of selection (it can be set using the SELECTION_PRESSURE keyword). The closer it is to 1, the weaker the selection.

Default Value

exponential_ranking

7.2 SELECTION_PRESSURE

Meaning

Intensity of selection.

This value is interpreted differently according to the selection scheme being used (see the SELECTION_SCHEME parameter).

Default Value

0.998 (fit for the `exponential_ranking` scheme)

8 Local Mutations' Parameters

8.1 POINT_MUTATION_RATE, SMALL_INSERTION_RATE, SMALL_DELETION_RATE

Meaning

These parameters set the spontaneous per replication, per base rate of point mutations, small insertions and small deletions (indels) respectively.

Default Value

1×10^{-5}

8.2 MAX_INDEL_SIZE

Meaning

Sets the maximum size of indels (small insertions and small deletions) whose actual size will be uniformaly drawn in $[1; MAX\_INDEL\_SIZE]$

Default Value

6

9 Chromosomal Rearrangements' Parameters

There are two distinct ways to perform chromosomal rearrangements, either taking sequence homology into account (which is time consuming) or not (the breakpoints are then chosen at random).

Only the simple case where sequence homology is ignored will be covered here, please see for homology driven rearrangements.

9.1 DUPLICATION_RATE, DELETION_RATE, TRANSLOCATION_RATE, INVERSION_RATE

Meaning

These parameters are used when sequence homology is ignored. They set the spontaneous per replication, per base rate of each kind of chromosomal rearrangements. The breakpoints defining the sequence that will be either duplicated, deleted, translocated or inverted are drawn at random (uniform law on the genome size).

Default Value

1×10^{-5}

10 To be continued...