ALF User Manual

Daniel A. Dalquen

August 11, 2015

1 Getting Started

1.1 Obtaining ALF

ALF is available as a web service as well as a stand-alone version. Both are available at `http://www.alfsim.org`. The web service can be used to run simulations directly or to generate parameter files for the stand-alone version. I recommend using the web service only for small simulations. If you want to be notified when new releases are available, you can register with your email address, when you download the stand-alone version.

1.2 Installing the stand-alone version

When you download and unpack the stand-alone version of ALF, the following files and directories are created:

`install.sh`
: install script (instructions below)

`bin/`
: - binaries of the darwin engine for Mac OS X and Linux
: - starter scripts for darwin and ALF
: - script to convert Fasta to Darwin files

`lib/`
: - the darwin library
: - the entire ALF source code is located in lib/simulator

`params/`
: sample parameter sets to reproduce some of the results from the publication

ALF can be started directly from this directory (see section 1.3), but it will probably be more convenient to install the software, because that will allow you to start ALF directly from any directory.

To install ALF on your system, run the script install.sh. By default, binary and script files will be copied to `/usr/local/bin` and the `lib` directory will be copied to `/usr/local/share/alfdarwin` (this will require super user privileges). If you wish to install ALF in a different location, pass the path to that location as parameter to `install.sh`. For example, if you want to install ALF in your home directory, use

```
./install.sh /path/to/home.
```

This will create the directories `bin/` and `share/alfdarwin/` in your home directory, if they don't exist already.

1.3 Running Simulations

If you did not install ALF, run it with the following command:

```
bin/alfsim [path to parameter file]
```

If you did install ALF and the script is on your PATH, then this reduces to

```
alfsim [path to parameter file]
```

When no paramater file is given, ALF will look for the file parameters.drw in the current working directory. Section 2 lists and explains all available parameters.

1.4 Citing ALF

If you use ALF in your research, please cite:

1.5 Contact information

If you have a question or want to report a bug, please send me an email to `d.dalquen@ucl.ac.uk`.

2 Parameters

Parameters for a simulation are passed in a parameter file that uses the Darwin syntax (see `http://www.biorecipes.com/DarwinHelp` for details). For example,

```
protStart := 200;
```

sets the parameter `protStart` to 200. Note that each assignment is terminated by a semicolon.

Some parameters expect a list (comma-separated values enclosed in square brackets, e.g. `[2.4, 133.8]`) or a set (comma-separated values in curly brackets, e.g. {`0, 1, 2.5`}).

Strings are enclosed in single quotes, e.g. `'a string'`.

Finally, comments start with the number sign (`#`).

2.1 Tree Parameters

ALF simulates sequences along a tree. This tree can be sampled randomly from a birth-death process or from a tree of life. Alternatively, you can also supply your own tree. The parameters below set up the species tree for the simulation.

- `treeType`
  Method for creating the species tree. Accepts the following values:

  `'BDTree'`
  Creates a birt-death tree with distance `mutRate` from origin to leaves, and $\lambda =$ `birthRate` and $\mu =$ `deathRate`. Uses the sampling process described by [?].

  `'ToLSample'`
  Creates a tree by sampling from the tree of life (currently consists of 1038 species derived from OMA)

  `'Custom'`
  Use a custom tree defined by `treeFile`.

- `mutRate`
  Distance from origin to species at the leaves (for random trees)

- `scaleTree`
  Set to `true`, if the branch lengths of the tree should be scaled. When `treeLength` is defined as well, the sum of all branches will be scaled

to match `treeLength`. Otherwise, the distance from root to deepest leaf is scaled to match `mutRate`. (default `false`).

- `treeLength`
  Sum of all branch lengths (only in conjunction with `scaleTree`).
- `birthRate`
  For `'BDTree'`: birth rate (λ)
- `deathRate`
  For `'BDTree'`: death rate (μ)
- `NSpecies`
  The number of species in the tree (for `'BDTree'` and `'ToLSample'`).
- `ultrametric`
  For `BDTree`: should resulting tree be ultrametric (default `true`).
- `treeFile`
  String with path to tree file with tree in Darwin or Newick format. You can also directly assign a darwin tree structure, which has the following format:

  ```
  Tree(Left,Height,Right,xtra)
  ```

  where `Left` and `Right` can be another tree structure or a Leaf (`Leaf(Label, Height, xtra)`), `Height` is the distance of the node from the root and `extra` is a field for additional information (for example used to annotate the tree with model switches, see below).
- `unitIsPam`
  Set to `false` if branch lengths are in substitutions per site. Set rate parameters for events accordingly. By default, ALF uses PAM distances.

2.2 Root Genome

The genome at the root of the tree can either consist of your own sequences or can be randomly generated. The following parameters set up the root genome.

- `realorganism`
  A string specifying the path to the Darwin DB with the sequence data. Use the script `fasta2darwin` in the `bin` directory to convert

Fasta files into the Darwin DB format. Note: Sequences containing ambiguous characters or special amino acids (B, J, O, U, X, Z for amino acid sequences; B, D, H, K, M, N, R, S, V, W, X, Y for nucleotide sequences) are removed.

Example (reads sequences from the file `se_ECOLI_core.db` in directory `realseed`):

```
realorganism := 'realseed/se_ECOLI_core.db';
```

- `protStart`
  The number of sequences that the first organism should have (only if the root genome is generated).

- `gammaLengthDist`
  List of parameters for the length distribution of the generated sequences ($\sim \Gamma(k, \theta)$).

  Example (lengths will be drawn from $\sim \Gamma(2.4, 133.8)$):

  ```
  gammaLengthDist := [2.4, 133.8];
  ```

  If lengths of all sequences should be identical, set $k = \theta = 1$ and specify the desired length with `minGeneLength`.

- `minGeneLength`
  Minimum length of a gene.

2.3 Sequence types

In ALF, you can configure different sequence types defined by a substitution model, an indel model and a model for rate variability among sites. The following sections describe how to set up the models, define sequence types and assign them to the sequences of the root genome. Switches between different sequence types can be performed during speciation or duplication.

2.3.1 Substitution Models

ALF supports a variety of nucleotide, codon and amino acid substitution models. Several models can be simulated in parallel for a subset of sequences. The parameters in this section allow you to set up substitution models.

- `substModels`
  List of substitution models, where each model definition has the following format:

```
SubstitutionModel(name:string, parameters:list,
                  frequencies:list, neutralDNA:boolean)
```

The number of arguments required depend on the model:

– models `CPAM`, `ECM`, `ECMu`, `GCB`, `JTT`, `WAG` and `LG` require just the name of the model (e.g. `SubstitutionModel('CPAM')`).
– When using a custom matrix (`CustomC`, `CustomP`), pass the path to the matrix file as parameter.
– For M-series models, pass also the codon frequencies (in the order AAA, AAC, AAG, AAT, ACA,. . . , TTT).
– Finally, for nucleotide models specify as fourth parameter whether non-sense mutations should be allowed.

Available models include:

nucleotide substitution
: `F84`, `GTR`, `HKY`, `TN93`

codon substitution
: `CPAM`, `ECM`, `ECMu`, `M0`, `M2`, `M3`, `M8`, `CustomC`

amino acid substitution
: `GCB`, `JTT`, `LG`, `WAG`, `CustomP`

Parameters are ordered as follows

custom empirical models
: `parameters[1]` should contain a path to a matrix in PAML format (lower triangular matrix of exchangabilitites, followed by a line with codon/aa frequencies).

M-series models
: `parameters[1]`: κ
`parameters[2]`: ω (single value or list)
`parameters[3]`: list of probabilities for ω-class[es]
`parameters[4]`: for M8, parameter p of beta distribution
`parameters[5]`: for M8, parameter q of beta distribution

nucleotide models
: `GTR`:
`parameters[1]`: a
$\vdots$

`parameters[6]`: f

`HKY`:
`parameters[1]`: α
`parameters[2]`: β

`F84`:
`parameters[1]`: κ
`parameters[2]`: β

`TN93`:
`parameters[1]`: α_1
`parameters[2]`: α_2
`parameters[3]`: β

Example (defines two substitution models, the CodonPam model for coding sequences and one a TN93 model with $\alpha_1 = 0.3, \alpha_2 = 0.4$ and $\beta = 0.7$ and equal base frequencies for non-coding sequences):

```
substModels := [SubstitutionModel('CPAM'),
                SubstitutionModel('TN93', [.3, .4, .7],
                                  [seq(0.25,4)], true)]:
```

- `blocksize`
  When no substitution model is given (pure gap simulation), select block size for gaps

2.3.2 GC Content Amelioration

GC content amelioration can be enabled by setting the variable `targetFreqs`. There are three possibilities:

- `targetFreqs := ['Random'];`
  Creates random target frequencies for all leaf species and all models. Overrides frequencies supplied in substitution models.

- Use specific frequencies per species and substitution model
  If you want to have specific target frequencies per species and substitution model, set `targetFreqs` to an array with the following structure:

```
targetFeqs := [freqsModel_1, freqsModel_2, ...]
```

where

```
freqsModel_i = [['speciesName_1', [freqList_1]],
                ['speciesName_2', [freqList_2]], ...]
```

Example (Simulation with 4 species using a single nucleotide model):

```
targetFreqs := [[['S1',[0.15, 0.35, 0.3, 0.2]],
                 ['S2',[0.2, 0.25, 0.3, 0.25]],
                 ['S3',[0.25, 0.2, 0.25, 0.3]],
                 ['S4',[0.35, 0.15, 0.2, 0.3]]]]:
```

Note: These target frequencies represent the stationary distribution of the underlying model. The actual frequencies will depend on the branch lengths.

- Use different models
  You can define different substitution models for each branch (see section 2.3.1).

2.3.3 Gap Models

You can define multiple models for insertions and deletions and assign each of them (or have them assigned) to a subset of sequences. Use the following parameters to define indel models.

- `indelModels`
  List of indel models, where each model definition has one of the following formats:

  `IndelModel(0)`
  for no indels

  `IndelModel(rate:nonnegative, model:string, parameters:list, maxLen:posint)`
  for one model for insertions and deletions with the same rate

  `IndelModel(gainRate:nonnegative, model:string, parameters:list, maxLen:posint, lossRate:nonnegative)`
  for one model for insertions and deletions with separate rates for insertions and deletions

  `IndelModel(gainRrate:nonnegative, gainModel:string, gainParameters:list, gainMaxLen:posint, lossRate:nonnegative, lossModel:string, lossParameters:list, lossMaxLen:posint)`
  for separate models for insertions and deletions

The parameters are as follows:

- `gainRate/lossRate`
  rate of insertions/deletions
- `model`
  can be one of the following:

  `'ZIPF'`
  Use a Zipfian distribution of the form $L^{-\texttt{Z\_c}}$, described in [?].

  `'NEGBIN'`
  Use a negative binomial distribution $\sim NB(\texttt{NB\_r}, \texttt{NB\_q})$

  `'QG'`
  Use the Qian-Goldstein distribution with parameters `QG_c` and `QG_t`, described in [?].

  `'GEOM'`
  Use a geometric distribution with parameter `E_p`. This corresponds to a negative binomial distribution $\sim NB(1, \texttt{E\_p})$

  `'CUSTOM'`
  Use a custom distribution with probabilities from `indelVector`.
- `gainParameters/lossParameters`
  the model parameters:

  `'ZIPF'`
  `parameters[1]`: c (Exponent of Zipfian distribution)

  `'GEOM'`
  `parameters[1]`: p (geometric distribution with mean $1/p$)

  `'QG'`
  `parameters[1]`: c
  `parameters[2]`: t (see [?])

  `'NEGBIN'`
  `parameters[1]`: r ($r \in \mathbb{N}$)
  `parameters[2]`: q ($0 < q < 1$)

  `'CUSTOM'`
  `parameters[1]`: p_1 ($\Pr(\textit{indel of length } 1)$)
  $\vdots$
  `parameters[N]`: p_N ($\Pr(\textit{indel of length } N)$)

  A list with probabilities defining a custom indel distribution. Should be `MaxLen` elements long.

  - `gainMaxLen`/`lossMaxLen`
    maximal length of indels

- `DawgPlacement`
  If `true` (default), each sequence is assumed to be part of a larger sequence and gaps can extend beyond either end of the sequence. This leads to a uniform distribution of gaps within the sequence. If `false`, gaps will be constrained to begin and end inside the sequence. This will lead to fewer gaps on both ends of the sequence.

2.3.4 Rate Variation Among Sites

The parameters in this section control rate variation among sites within sequences.

- `rateVarModels`
  List of models for rate variation among sites. Each model definition has one of the following formats:

  `RateVarModel()`
    for no rate variation

  `RateVarModel(model:string, areas:posint, motifFreq:nonnegative, alpha:nonnegative)`
    for defining a model

  Note: These parameters are ignored, if M-series models or custom rates are used (use `RateVarModel()` in these cases).

  The model definitions take the following parameters:

  - `model`
    possible values include:

    `'None'`
      No rate variation among sites.

    `'Gamma'`
      Use gamma rates. The number of bins is defined by `areas`, additionally, motifs occur with frequency `motifFreq`.

    `'Poisson'`
      Generates a random number of domains (at most `areas`) per gene with rates drawn from a Poisson distribution. `motifFreq` defines the fraction of domains with mutation rate 0 (motif).

  - `areas`
    For `'Poisson'`: maximum number of areas with different rate within a gene.
    For `'Gamma'`: number of rate classes.

  - `motifFreq`
    Proportion of invariable sites (motifs)
  - `alpha`
    Shape parameter of gamma distribution for gamma rates among sites.

- `areaPath`
  The path to a file, containing for each gene a list of areas with different substitution rates. The file should have the following format:

  ```
  [areas_gene_1, areas_genes_2, ...]
  ```

  where

  ```
  areas_gene_i = [area_1, area_2, ...]
  area_i = [start_pos, end_pos, rate]
  ```

  Note: If you use `areaPath` to provide custom rate variation, set `rateVarModels` to `[RateVarModel()]` and define your sequence types (see section 2.3.5) accordingly.

2.3.5 Assigning and Switching Between Sequence Types

The following parameters define different sequence types and probabilities for switching between different types.

- `seqTypes`
  A list of triplets `[i,j,k]`, where `i` defines the substitution model, `j` defines the indel model, and `k` defines the model for variation among sites.

- `seqTypeAssignments`
  Parameter concerning selection of sequence type for each sequence of the root genome. Either supply a list of frequencies of the types $1 \dots n$ defined in `seqTypes` for random assignment or a list of assignments for each sequence of the root genome.

  Example (assign T_1 with probability 0.75 and T_2 with probability 0.25):

```
seqTypes := [[1,1,1], [2,1,1]];
seqTypeAssignments := [0.75, 0.25];
```

- `modelSwitchS/modelSwitchD`
  A matrix with probabilities for a switch from sequence type T_i to type T_j after speciation or duplication, respectively.

  Example (switch from T_1 to T_2 with probability 0.2 during speciation in the new species):

  ```
  modelSwitchS := [[0.8,0.2],[0,1]];
  modelSwitchD := [[1,0],[0,1]]; % no switch
  ```

  Note: You can define model switches in more detail by using a custom tree and annotate it accordingly. Here is an example that creates the scenario depicted in figure 1 (branch lengths are 40 PAM for a and b, and 80 PAM for c - f):

  ```
  treeFile := Tree(Tree(Leaf('S1', 120), 40,
                        Leaf('S3', 120), [S,[[1,[3,4]]]]),
                   0, Tree(Leaf('S2', 120), 40,
                           Leaf('S4',120), [S,[[2,[5,6]]]]),
                   [S,[[1,[1,2]]]]);
  ```

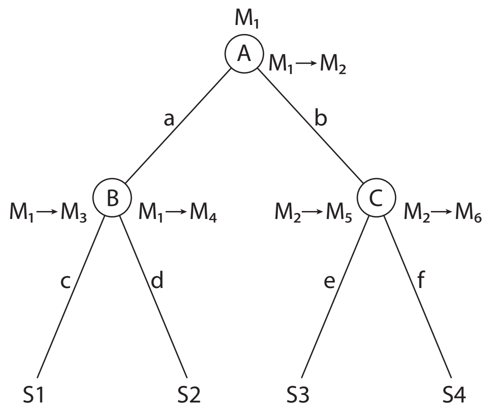

Figure 1: This tree represents the following scenario: All sequences are initially assigned T_1. At node A, a switch occurs from T_1 to T_2 for branch b. At node B a switch occurs from T_1 to T_3 for branch c and from T_1 to T_4 for branch d. At node C a switch occurs from T_2 to T_5 for branch e and from T_2 to T_6 for branch f.

2.4 Rate Variation Among Genes

The parameters in this section control rate variation among sequences.

- `amongGeneDistr`
  Distribution of rates among genes. Possible values include:

  `'None'`
  : No variation among genes.

  `'Gamma'`
  : Draw rate for each sequence from a gamma distribution with shape parameter `aGAlpha` and average 1.

  `'Custom'`
  : Use custom rates from file defined by `aGPath`.

- `aGAlpha`
  Shape parameter of among gene rate distribution.

- `aGPath`
  Path to file with custom rates (see example file, only for custom root sequences)

2.5 Gene Duplication and Loss

The following parameters control gene duplication and gene loss.

- `geneDuplRate`
  Rate of gene duplications (relative to substitutions).

- `transDupl`
  Probability of a tranlocation after duplication.

- `numberDupl`
  Maximum number of consecutive genes involved in one duplication event.

- `fissionDupl`
  Probability of a fission event after the duplication of a single gene.

- `fusionDupl`
  Probability of a fusion event after the duplication of two or more genes.

- `P_pseudogene`
  Probability of duplicate becoming a pseudogene (permanent rate change for duplicate).

- `ratefac_pseudogene`
  Factor by which the rate for the pseudogene is modified.

- `P_neofunc`
  Probability of duplicate undergoing neofunctionalization (temporary rate change for duplicate).

- `ratefac_neofunc`
  Factor by which the rate for the duplicate is modified.

- `life_neofunc`
  Life of increased rate (time to normalization of rate, in PAM units or substitutions per site)

- `P_subfunc`
  Probability of both copies undergoing subfunctionalization (temporary rate change for original and duplicate)

- `ratefac_subfunc`
  Factor by which the rate for the duplicate is modified.

- `life_subfunc`
  Life of rate change (time to normalization of rate, in PAM units or substitutions per site).

- `geneLossRate`
  Rate of gene losses (relative to substitutions)

- `numberLoss`
  Maximum number of consecutive genes involved in one loss event.

2.6 Lateral Gene Transfer

- `lgtRate`
  Rate of single lateral gene transfers (relative to substitutions).

- `orthRep`
  Proportion of lateral gene transfers that are orthologous replacements (i.e. the transferred gene replaces the orthologous gene in the recipient).

- `lgtGRate`
  Rate of lateral transfers of groups of genes.

- `lgtGSize`
  Maximum number of genes which can be transferred in one event.

2.7 Genome Rearrangement

- `invers`
  Rate of gene inversions (relative to substitutions).
- `invSize`
  Maximum number of genes which are inverted in one inversion event
- `transloc`
  Rate of gene translocations (relative to substitutions).
- `transSize`
  Maximum number of genes which are translocated in one go
- `invtrans`
  Rate of inverted translocations (relative to substitutions).

2.8 Gene Fusion and Fission

- `fissionRate`
  Rate of gene fissions without prior duplication (relative to substitutions).
- `fusionRate`
  Rate of gene fusions without prior duplication of fused genes (relative to substitutions).
- `numberFusion`
  Maximum number of genes fused in one event.

2.9 Output

The following parameters define what output is generated and where it is stored. ALF will always generate a species tree, that reflects the ancestry of the simulated species, and the set of genomes of the species at the leaves of that tree.

- `simOutput`
  A set of output files and formats. If this variable is not defined, all outputs are generated. Possible values include:

`'GeneTrees'`
: all gene trees

`'Ancestral'`
: output ancestral genomes

`'Dup'`
: output ancestral sequences at gene duplications. The output consists of one file per species. Each sequence is only stored in one file (i.e. the one of the species where the duplication occurred).

`'MSA'`
: MSAs of all related sequences

`'VP'`
: pairwise evolutionary relationships (ortho/para/xenologs)

`'DarwinTree'`
: output trees in Darwin format

`'Newick'`
: output trees in Newick format (default)

`'DarwinDB'`
: output genomes as Darwin databases

`'Fasta'`
: output genomes as Fasta files (default)

Example (creates species tree in Darwin format, MSAs of all gene families, and Darwin databases for all ancestral and leaf species):

```
simOutput := {'DarwinTree', 'MSA', 'DarwinDB', 'Ancestral'};
```

- `wdir`
  A string specifying the working directory, i.e. the directory where simulation results are stored. current working directory by default, can also be set as argument of alfsim

- `mname`
  A string specifying the name of the simulation. ALF will create a directory of that name in the working directory containing all result files.

2.10 Misc

- Initialize random number generator
  If you are interested in getting reproducible results, put `SetRand(`*seed*`);`

on the very first line of the parameter file, where *seed* is an arbitrary integer number.

3 The Evolutionary History of a Gene

The Darwin database files hold information about all evolutionary events that affected a gene. For each gene, the `<DE>` tag contains a string composed of the following elements:

- `a-(x)b` speciation at time x of species a into species b
- `-(gDx)a` gene duplication of gene g at time x in species a
- `-a(gLx)b` LGT (novel acquisition) of gene g at time x from donor species a into recipient species b
- `-a(gLox)b` LGT (orthologous replacement) of gene g at time x from donor species into recipient species b
- `-(gFix)a` gene fission of gene g at time x in species a
- `-(gFux)a` gene fusion of gene g with the current gene at time x in species a

4 Performance / Memory Usage

Simulation	**Execution Time (s)**	**Memory Usage (MB)**
root genome: 100 protein sequences, 500 aa each; 20 species; substitutions: WAG	7	300
root genome: 100 protein sequences, 500 aa each; 20 species; substitutions: WAG; indels: Zipfian; Γ rates	10	664
root genome: 100 codon sequences, 500 codons each; 20 species; substitutions: CPAM; indels: Zipfian; Γ rates	315	1506
root genome: 100 codon sequences, 500 codons each; 20 species; substitutions: CPAM; indels: Zipfian; Γ rates; duplications/losses with rate 0.001	277	1489
root genome: 1000 seqs, lengths $\sim \Gamma(3, 134)$; 20 species; substitutions: CPAM, TN93; indels: Zipfian; Γ-rates; duplications/losses/fusions/fissions with rate 0.001; LGTs with rate 0.0004; rearrangements with rate 0.005	1782	1801
root genome: 4352 E. coli genes; 20 species; substitutions: CPAM, TN93; indels: Zipfian; Γ-rates; duplications/losses with rate 0.003; LGTs with rate 0.0005	16577	3934

Table 1: Performance of a number of example simulations. Based on simulations run on a Indel Core 2 Quad with 2.33 GHz.