

EvolSimulator v2.1.0

Robert G. Beiko and Robert L. Charlebois

August 8, 2006

Contents

1 Overview	1
1.1 Citing EvolSimulator	2
2 Building and running EvolSimulator	2
2.1 Download and build	2
2.2 Program execution	3
3 Program output	5
4 List of parameters in EvolSimulator	5

1 Overview

EvolSimulator is a program that can simulate the evolution of whole genomes, with changes in nucleotide and protein sequence, gene content, and population structure controlled by parameters within the program. One goal of EvolSimulator is to test hypotheses of genome evolution in light of lateral genetic transfer. To carry out a meaningful simulation of such processes, many potentially confounding evolutionary phenomena need to be taken into consideration, including different rates of gene and protein evolution, genome-specific mutation pressures, and cryptic paralogy arising through gene duplication and loss. Different conditions can be set on the propensity of genes to be transferred and the propensity of organisms to share them: proposed transfers can be evaluated in light of gene content compatibility or time of divergence, for instance.

A large number of user-specified parameters are needed to direct the behaviour of different evolutionary phenomena within EvolSimulator. While the complete set of parameters presents an extremely complex picture, it is important to know that most of the phenomena can be disabled to permit targeted analyses which consider only a subset of all implemented effects. For instance, an analysis of the effects of biased mutation regimes on phylogenetic reconstruction would not require the modelling of gene gains, losses, or transfers, so the rate of these events would be set to zero.

1.1 Citing EvolSimulator

The current citation for EvolSimulator is:

Beiko, R.G. and R.L. Charlebois. A simulation test bed for hypotheses of genome evolution (submitted to *Bioinformatics*).

This citation will be updated as the status of the manuscript changes.

2 Building and running EvolSimulator

2.1 Download and build

The EvolSimulator package, including C++ source and some binaries, is available from <http://bioinformatics.org.au/evolsim>. To build EvolSimulator, you will need to obtain the (free) Boost library from www.boost.org. There is no need to “install” the library; just untar/unzip the downloaded file, then adjust the `-I` option within EvolSimulator’s makefile to point to the folder containing “boost”. See the supplied makefile for an example. If you have any trouble compiling the source code, or if you would prefer a ready-made executable for your system beyond what we already provide, please contact us. Once the Boost library is in place, and you have edited the makefile to point to it, just type

```
make
```

on any UNIX/Linux system with the GNU Compiler Collection (`gcc/g++`) installed.

2.2 Program execution

The command line for EvolSimulator is as follows (optional arguments in square brackets):

```
EvolSimulator -f paramFileName  
[-p savePath] [-e]  
[-s randomSeed_gene] [-z randomSeed_genome]  
[-retryLGT numAttempts] [-forceSp] [-d eventFileName]  
[-ngibws] [-n seriesName]
```

- **-f paramFileName**

Specifies the name of the input parameter file used by EvolSimulator. A sample file is provided with the EvolSimulator release package. This file must define values for every variable, even those that are irrelevant to a given run.

- **-p savePath**

Defines the output directory into which results will be saved. If unspecified, defaults to the program's directory.

- **-e**

Causes the program to output genome files at every speciation and extinction event, so that most-recent common ancestors and extinct relatives can be included in subsequent analyses if desired.

- **-s randomSeed_gene**

The randomSeed_gene option allows one to specify a seed to the random number generator that directs stochastic events during the course of the simulation; if unspecified the seed is selected at random. This argument is useful for reproducing a run exactly, where file-based parameters are all kept the same except perhaps for the length of the run or the frequency of program output, and where command-line arguments are also kept the same except perhaps for enabling event writes (-e) or specifying output format (-ngibws).

- **-z randomSeed_genome**

A number provided with this option specifies the random number seed that directs genome evolution. Any pair of runs initiated with the same ‘-z’ number will have the same history of genome-level events: speciation/extinction, and the drift of each genome’s target size, receptivity to LGT, mutation rate and mutational biases (G+C, purine/pyrimidine, transition/transversion).

- **-retryLGT numAttempts**

LGT scenarios attempt to transfer a gene from a random donor to a random recipient, but an attempt may fail (probabilistically; see the section on LGT scenarios) depending on the relationship between the members of the pair. This command-line argument (default value 1) allows an unsuccessful attempt at LGT to be retried, by reselecting another random donor and recipient, up to the number of times specified.

- **-forceSp**

For a speciation event to succeed, the freshly spawned genome must be able to find a niche in which to inhabit, but all niches in which it can inhabit might already be full. Given that the habitat system is necessarily controlled by the ‘-s’ random number generator, a run would not be reproducible at the genome level if such speciation events were frustrated. One usually therefore specifies -forceSp when one wishes to reproduce a run’s genome-level events by specifying the same -z seed as before. This forces speciation to occur by overoccupying the parent’s niche, when necessary.

- **-d eventFileName**

Specifies the file name of a deterministic series of events, in the proper format. It is used when one specifies a speciationModel of 2 in the parameter input file. This feature of EvolSimulator is still under development and should not yet be used.

- **-ngibws**

Rather than generating a flat set of .gbk and .faa files, specifying this flag causes EvolSimulator to produce a nested hierarchy of folders and files required for creating an NGIBWS database (Charlebois, R.L., G.D.P. Clarke, R.G. Beiko and A. St. Jean. 2003. *FEMS Microbiol. Lett.* 225, 213-220) for subsequent comparative genomic analysis.

- `-n seriesName`

Provides a string tag to append to genome names, useful to keep track of things when one wishes to produce a variety of different runs exploring parameter space.

3 Program output

At regular intervals (if desired, controlled by the parameter `outInterval`) and at the end of the simulation, a set of GenBank-format (`.gbk`) and FASTA (`.faa`) files, or files compatible with NGIBWS, are produced for each extant genome, time-stamped with the current evolutionary iteration. The `'-e'` command-line option causes the program to output genome files at every significant event (speciation or extinction) as well, to allow the tracking of biases and genetic complements at internal nodes of the tree. Tree files are also produced, in two versions: one including only extant genomes and one also including extinct genomes (indicated by an asterisk). Evolutionary histories are built into the gene and “product” names for each gene. Gene names are unique for each ortholog, and include times at which prior duplicative events occurred.

4 List of parameters in EvolSimulator

The following is a brief overview of the parameters used by EvolSimulator to guide the evolution of genes and genomes. The parameters listed below are organized by the conceptual level at which they act. The legal range of each parameter is indicated as well; the conceptual upper bound of many parameters is infinity, though one must understand that practical bounds exist. A square bracket indicates that a parameter may have the boundary value indicated; a parenthesis indicates that it cannot. Where integers are specified, integer parameter values are expected; where floating-point boundaries are indicated, floating-point parameter values are expected.

Granularity of simulation

numIterations

Number of evolutionary time intervals (generations)

Range : $[1, \infty)$

The granularity of evolution is a function of the number of time intervals, in combination with the multitude of per-iteration events and adjustments specified below. One can choose to perform longer runs with finer adjustments, or shorter runs with coarser adjustments, depending on the need.

outInterval

Interval (in generations) between recording of information

Range : $[1, \infty)$

Setting *outInterval* to a value greater than *numIterations* will produce one set of output files at the end of the run; setting it to a smaller value will generate intermediate output allowing one to examine ancestral states.

Population-level parameters

preferredMinNumLineages

Loose lower bound on population size

Range : $[1, \infty)$

When *speciationModel* 1 is selected (stochastic growth and shrinkage of the population), the base extinction probability is halved when the population size is below *preferredMinNumLineages*. This therefore tends to grow an impoverished population to keep within the range *preferredMinNumLineages* to *maxNumLineages*, without precluding the occasional bottleneck. This parameter has no meaning for *speciationModel* 0. If requested by the user community, we will be happy to convert the “halved”, mentioned above, into its own parameter.

maxNumLineages

Strict upper bound on population size

$$\text{Range} : [\text{preferredMinNumLineages}, \infty)$$

For *speciationModel* 0 (fixed population size), *maxNumLineages* is the size of the population. For *speciationModel* 1, it represents the upper bound on population size. This parameter also affects niche and habitat distribution, and LGT probabilities (discussed below).

speciationAndExtinctionProb

Base probability of speciation or extinction per iteration

$$\text{Range} : [0.0, 1.0)$$

Under *speciationModel* 0, *speciationAndExtinctionProb* is the actual probability of a pair of speciation + extinction events occurring in an iteration. Under *speciationModel* 1, the probability of speciation in a given iteration is

$$p = \text{speciationExtinctionProb} * \ln\left(1.0 + \frac{\text{maxNumLineages}}{\text{numLineages}}\right)$$

if *numLineages* (the current number) is less than *maxNumLineages*; $p = 0$ otherwise. Again under *speciationModel* 1, the probability of extinction in a given iteration is

$$p = \text{speciationExtinctionProb} * \frac{\text{numLineages}}{\text{maxNumLineages}}$$

If *numLineages* is greater than 1; otherwise $p = 0$. As mentioned above, the extinction probability under *speciationModel* 1 is furthermore halved when *numLineages* falls below *preferredMinNumLineages*.

speciationModel

Fixed or stochastic control of population size

$$\text{Valid settings} : \{0, 1, 2*\}$$

With *speciationModel* 0, the population is fixed at a size of *maxNumLineages*, cloning the cenancestor. With *speciationModel* 1, a single cenancestral genome is created, then population growth follows the speciation and extinction regime regulated by the next three parameters. *SpeciationModel* 2 should not presently be used, as it is still under development. It will allow deterministic evolution specified from an event file in the proper format (which we won't tell you about just yet, given that "*SpeciationModel* 2 should not presently be used"! We will update the code and this documentation once that model's functionality has been validated).

meanNumLGTperIteration

Mean number of LGT events attempted in each time interval

Range : [0.0, ∞)

The actual number of attempted LGT events per iteration is drawn from a Poisson distribution with a mean of

$$\text{meanNumLGTperIteration} * \frac{\text{numLineages}}{\text{maxNumLineages}}$$

The latter factor normalizes the frequency of LGT to the size of the population, so that smaller populations are not overburdened by LGT nor larger populations deprived of LGT. Rather, this one parameter effectively controls the number of LGT per *maxNumLineages* genomes per iteration. Note that LGT attempts are not always successful, depending on the scenario(s) specified (see Genome-level parameters, below) and the value of the `-retryLGT` command-line option.

Ecological parameters

selectionModel

Regulating which genomes are more likely to speciate or extinguish

Valid settings : {0, 1, 2, 3, 4}

There are five selection models currently implemented:

- 0: Neutral selection, where every genome has an equal chance of speciating or going extinct.

- 1: Generalist selection, where the sum of fitnesses (see *paralogUsefulnessFactor*, below) over all inhabitable niches biases the choice of which genome should speciate or go extinct.
- 2: Specialist selection, where the best fitness among inhabitable niches biases the choice of which genome should speciate or go extinct.
- 3: Selection for large genomes, where larger genomes are better candidates for speciation and poorer candidates for extinction.
- 4: Selection for small genomes, where smaller genomes are better candidates for speciation and poorer candidates for extinction.

numberOfHabitats

The potential number of cross-breeding populations

$$Range : [1, \infty)$$

The cenancestor is originally totipotent, but depending on *meanPropHabReq* (below) and chance, populations may become isolated from one another. Isolation affects the selection of partners for habitat-restricted LGT, but also plays a role in the competitive survival of lineages, given the limited number of available niches. This arrangement can be disabled by setting *numberOfHabitats* to 1.

numberOfHabitatNiches

Niche spaces are distributed among habitats

$$Range : [numberOfHabitats, \infty)$$

Each habitat includes one or more niches, the *numberOfHabitatNiches* of them distributed by a Poisson process among the *numberOfHabitats*. Again if one wishes to disable the space-competitive dimension of a simulation, one may set *numberOfHabitatNiches* to 1, along with *numberOfHabitats*. Otherwise, distinct gene-content specializations will develop, adapting genomes to specific niches. Note that LGT may occur among genomes in different niches, perhaps rescuing them from overspecialization. Under the scenario of habitat-restricted LGT, LGT may occur between genomes in different niches only if those niches are in the same habitat.

numberOfHabitatSpaces

Spaces are distributed among niches within habitats

Range : $[max(maxNumLineages, numberOfHabitatNiches), \infty)$

Each niche is guaranteed one space, with the remaining spaces distributed by a Poisson process to the collection of niches. If the parameter has a high value, competition for space is light and genomes are free to move around. If spaces are limiting, competition will be much more important, and certain genomes may have an advantage over others in their ability to expand (speciate) into the scarce collection of unoccupied niches.

probMigrationPerIter

Probability per iteration that a genome's niche will be reassigned

Range : $[0.0, 1.0]$

Niche reassignment is determined by the cumulative fitness probability of niches inhabitable by a genome. It must possess all of the habitat-required genes and the niche-required genes of a target niche in order to be eligible to move there, and the chance of moving to a particular niche further depends upon the genome's fitness in that niche (see *paralogUsefulnessFactor*, below). Reassignment can be to its current home, even if other niches are suitable.

paralogUsefulnessFactor

For computing a gene's usefulness in a given niche or habitat

Range : $[0.0, \infty)$

The fitness of a genome in a particular niche is a function of the usefulness of its genes in that niche. There are three levels of usefulness: global, habitat, and niche, representing three levels of selection. A gene's usefulness, contributing to a genome's fitness, is then the maximum of its usefulness globally, in the particular habitat, and in the particular niche in the particular habitat. We model global usefulness to be less likely than habitat usefulness, and habitat usefulness to be less likely than niche usefulness, as follows: where r is an independently sampled uniform random deviate for each ecological unit:

- global usefulness for a new gene is $e^{-3r*paralogUsefulnessFactor}$
- habitat usefulness is $e^{-2r*paralogUsefulnessFactor}$
- niche usefulness is $e^{-r*paralogUsefulnessFactor}$

Besides contributing to a genome's fitness and thereby directing its migration, speciation and extinction probabilities, a gene's usefulness in the currently inhabited niche affects the probability that it will be retained and not accidentally lost (see the section on gene duplication and loss).

exactPropGlobalReq

The proportion of indispensable (universal core) gene families

$$Range : [0.0, 1.0]$$

Regardless of a gene's usefulness, loss of the last member of a randomly preselected number of gene families from a genome is precluded. As long as one member of the family remains in a genome, others may be lost (as a function of their usefulness), but at least one member of the family must remain. These are indispensable core genes. It can be useful to specify this, in that it assures that every lineage at the end of a run will have at least one member of this gene family, if one is interested for example in making phylogenies where a gene has a representative from every taxon.

meanPropHabReq

The mean proportion of gene families indispensable in any given habitat

$$Range : [0.0, 1.0]$$

As for global genes, habitat-required gene families cannot be entirely lost from a genome, while the genome is in that habitat. Should the genome move to a different habitat, it may lose genes required in its original home, blocking the reverse migration. A given habitat will have a number of required gene families determined by a Poisson process with mean *meanPropHabReq*.

meanPropNicheReq

The mean proportion of gene families indispensable in any given niche

Range : [0.0, 1.0]

This parameter operates just like *meanPropHabReq*, described above, but for gene families indispensable at the niche level.

Genome-level parameters

rateAndBiasAdjustFrequency

Coarseness of adjustments of rates and biases, expressed in generations

Range : [1, ∞)

In addition to the control of granularity provided by the global parameter *numIterations*, genome-level rates and biases need not be updated at each generation, if this parameter is set above 1. Drift rates and such (below) are still specified per-iteration, but *rateAndBiasAdjustFrequency* allows one to lump updates. This is primarily useful for cutting down on the number of genome-level events that must be recorded to file, if that option is enabled. One would normally keep this parameter set at 1.

minGenomeSize

Minimum number of genes in each genome

Range : [1, ∞)

maxGenomeSize

Maximum number of genes in each genome

Range : [*minGenomeSize*, ∞)

initialTargetSize

Initial value for cenancestor's target genome size

Range : [*minGenomeSize*, *maxGenomeSize*]

Target genome size represents the number of genes towards which a genome will tend via the mechanisms of gene gain (duplication of existing genes or acquisition of new genes via LGT) and gene loss.

sizeAdjFactor

Base change in genome size per generation

Range : [0.0, ∞)

A genome has a target size, as explained above, which may drift (see *targetSizeDriftFactor*, below). The parameter *sizeAdjFactor* regulates the speed at which the genome's actual size moves towards its target. If *sizeAdjFactor* is set to 0.0, target size is moot, since no losses or gains can occur. If *sizeAdjFactor* is set too high, many duplications and losses will occur and gene family diversity is likely to suffer from the erratic shifts in genome size. Actual adjustments are done as follows: First, the ratio p of current size to target size is computed. The number of net gains for the iteration is then set as $p1 - p2$, where $p1$ is drawn from a Poisson distribution with mean ($sizeAdjFactor/p$), and $p2$ is drawn from a Poisson distribution with mean ($sizeAdjFactor*p$). If net gains is positive, that number of gene duplications is attempted, each attempt involving a random gene and testing its duplicability against a uniform random deviate. If instead, net gains is negative, then up to that number of losses occurs, each attempt involving a random gene and testing to see if it is dispensable in the current niche and testing its usefulness in that niche against a uniform random deviate. Gains and losses are furthermore constrained to keep the genome within the range of *minGenomeSize*, *maxGenomeSize*.

When a paralog is created, its parent's duplicability is repartitioned between the parent and its new copy, in order to preempt runaway duplication. The paralog's inertia (general resistance to mutation) is set randomly (the parent's inertia is unchanged). The paralog's selective pressure is set to a random fraction of *maxInitialSelectivePressure* (selective pressure constrains

the types of amino acid residue substitutions; see Protein residue-level parameters below), and the parent gene's selective pressure is reduced by a random fraction of that in recognition of the redundancy. Finally, the new paralog gets its own independent measure of ecological usefulness, and its amino acid residues each get a new auxiliary substitution matrix (again, see Protein residue-level parameters, below) assigned to them.

Note that if LGT is prescribed and *sizeAdjFactor* is set to zero, nothing but orthologous displacement can occur, since new paralogs would never form, and incoming genes would always find a never-lost ortholog to replace. As *sizeAdjFactor* increases, incoming paralogs can swell genome size, driving loss. Large genomes (genomes with large target sizes) will be paralog factories, and small genomes (genomes with small target sizes) will be ever trying to shed excess baggage, rarely generating paralogs of their own. Clearly there is a good deal of complexity, and interesting dynamics, to explore by playing with *sizeAdjFactor*, *targetSizeDriftFactor* (below), and the relative rates of LGT.

targetSizeDriftFactor

Maximum change in genome target size per generation

Range : [0.0, ∞)

A genome's target size may drift upwards or downwards by an amount up to *targetSizeDriftFactor* per iteration, but bounded by *minGenomeSize* and *maxGenomeSize*.

mutationRateDriftFactor

Maximum drift per generation of mutation probability

Range : [0.0, 1.0]

A genome's mutation rate may drift upwards or downwards by an amount up to *mutationDriftFactor* per iteration, but bounded by *minMutationRate* and *maxMutationRate*.

minMutationRate

Minimum mutation probability

Range : (0.0, 1.0)

maxMutationRate

Maximum mutation probability

Range : [*minMutationRate*, 1.0)

mutationRate

Cenacestral (starting) mutation rate

Range : (0.0, 1.0)

A genome's mutation rate affects the probability that a nucleotide will mutate in a given iteration, when the gene's inertia permits. In other words, for each gene at each iteration, if the gene's inertia is inferior to a uniform random deviate and the genome's *mutationRate* is also inferior to a (separate) uniform random deviate, then the nucleotide may mutate. Mutation is biased (see below) towards G+C or A+T, towards transitions or transversions, and towards purines or pyrimidines, and may actually result in the nucleotide remaining unchanged, if the "polymerase" makes a lucky guess. The mutation is also subject to rejection at the amino acid residue level (see Protein residue-level parameters, below).

transitionProbDriftFactor

Maximum drift per generation of the proportion of proposed mutations that are transitions

Range : [0.0, 1.0]

A genome's transition probability may drift upwards or downwards by an amount up to *transitionProbDriftFactor* per iteration, but bounded by *minTransitionProb* and *maxTransitionProb*.

minTransitionProb

Minimum transition probability

Range : (0.0, 1.0)

maxTransitionProb

Maximum transition probability

Range : [*minTransitionProb*, 1.0)

transitionProb

Cenancestral (starting) transition probability

Range : (0.0, 1.0)

This is the initial probability that an attempted mutation (see the discussion at *mutationRate*, above) is a transition rather than a transversion.

gcBiasDriftFactor

Maximum drift per generation of G+C bias in proposed mutations

Range : [0.0, 1.0]

A genome's G+C bias may drift upwards or downwards by an amount up to *gcBiasDriftFactor* per iteration, but bounded by *minGCbias* and *maxGCbias*.

minGCbias

Minimum G+C bias

Range : (0.0, 1.0)

maxGCbias

Maximum G+C bias

Range : [*minGCbias*, 1.0)

gcBias

Cenancestral (starting) G+C bias

Range : (0.0, 1.0)

This is the initial probability that an attempted mutation (see the discussion at *mutationRate*, above) generates a G or C rather than an A or T.

purineBiasDriftFactor

Maximum drift per generation of purine bias in proposed mutations

Range : [0.0, 1.0]

A genome's purine bias may drift upwards or downwards by an amount up to *purineBiasDriftFactor* per iteration, but bounded by *minPurineBias* and *maxPurineBias*.

minPurineBias

Minimum purine bias

Range : (0.0, 1.0)

maxPurineBias

Maximum purine bias

Range : [*minPurineBias*, 1.0)

purineBias

Cenacestral (starting) purine bias

Range : (0.0, 1.0)

This is the initial probability that an attempted mutation (see the discussion at *mutationRate*, above) generates an A or G rather than a C or T.

randomLGTprob

Proportion of LGT events that are automatically successful

Range : [0.0, 1.0]

Note that for all LGT scenarios, success also depends on the recipient genome's receptivity to LGT; and if the recipient already has that particular ortholog, on the parameter *orthologReplacementProbability* (described below) as well. Also note that the sum of *randomLGTprob*, *divergenceLGTprob*, *geneCompLGTprob*, *gcLGTprob*, and *habitatLGTprob* should be between 0.0 and 1.0.

divergenceLGTprob

Proportion of LGT events decided with the divergence criterion

$$Range : [0.0, 1.0]$$

See *divergenceFactor*, below.

geneCompLGTprob

Proportion of LGT events decided with the gene complement criterion

$$Range : [0.0, 1.0]$$

See *geneCompSimilarityFactor*, below.

gcLGTprob

Proportion of LGT events decided with the G+C similarity criterion

$$Range : [0.0, 1.0]$$

See *gcSimilarityFactor*, below.

habitatLGTprob

Proportion of LGT events decided with the habitat criterion

$$Range : [0.0, 1.0]$$

Habitat-restricted LGT succeeds only if the donor and recipient are currently residing in the same habitat.

divergenceFactor

Divergence denominator for relations-biased LGT

$$Range : (0.0, \infty)$$

When relations-biased LGT is called for: Where d is the divergence time between a donor and potential recipient and p is the probability of exchange,

$$p = \max\left(0.0, 1.0 - \sqrt{\frac{d}{divergenceFactor}}\right)$$

Exchange between genomes will therefore be precluded when d is greater than *divergenceFactor*.

geneCompSimilarityFactor

Proportion denominator for gene content-biased LGT

$$Range : [0.0, 1.0)$$

When gene complement-biased LGT is called for: Where g is the number of orthologous genes in common between a donor and potential recipient, s is the size of the smaller of the two genomes, and p is the probability of exchange,

$$p = \max\left(0.0, 1.0 - \frac{1.0 - g/s}{1.0 - \text{geneCompSimilarityFactor}}\right)$$

Exchange between genomes will therefore be precluded when the proportion of shared orthologs is less than *geneCompSimilarityFactor*.

gcSimilarityFactor

G+C denominator for G+C-biased LGT

$$Range : (0.0, 1.0]$$

When G+C-biased LGT is called for: Where c is the G+C-compositional difference between the donor and potential recipient and p is the probability of exchange,

$$p = \max\left(0.0, 1.0 - \frac{c}{\text{gcSimilarityFactor}}\right)$$

Exchange between genomes will therefore be precluded when c is greater than *gcSimilarityFactor*.

orthologReplacementProbability

Probability of an orthologous replacement

$$Range : [0.0, 1.0]$$

If an ortholog of an incoming laterally transferred gene exists in the genome, it can be replaced at *orthologReplacementProbability*. Otherwise, the LGT event is considered unsuccessful, and may be retried with different partners and different genes.

lgtReceptivityDriftFactor

Maximum drift per generation of LGT receptivity

Range : [0.0, 1.0]

A genome's LGT receptivity may drift upwards or downwards by an amount up to *lgtReceptivityDriftFactor* per iteration, but bounded by *lgtMinReceptivity* and *lgtMaxReceptivity*.

lgtMinReceptivity

Minimum LGT receptivity

Range : [0.0, 1.0]

lgtMaxReceptivity

Maximum LGT receptivity

Range : [*lgtMinReceptivity*, 1.0]

lgtReceptivity

Cenacestral (starting) LGT receptivity

Range : [0.0, 1.0]

A genome's LGT receptivity is the probability of acceptance of a gene that is being offered by a donor via LGT. If a genome's LGT receptivity is low, gene acquisition by LGT will often fail. If the parameter is high, then acquisition by LGT is welcome. This regulator, in combination with selection schemes (outlined above) may help to understand the cost or value of LGT under different circumstances.

Gene-level parameters

duplicabilityDriftFactor

Duplicability is the propensity of a gene to duplicate

Range : [0.0, 1.0]

A gene's duplicability may drift between 0.0 and 1.0, the amount of change up or down being as much as *duplicabilityDriftFactor*, per iteration.

inertiaDriftFactor

Inertia is the general resistance of a gene to mutation

Range : [0.0, 1.0]

A gene's inertia may drift between 0.0 and 1.0, the amount of change up or down being as much as *inertiaDriftFactor*, per iteration.

selectiveDriftFactor

Selection is the general resistance of a gene to amino acid substitution

Range : [0.0, 1.0]

A gene's selective pressure may drift between 0.0 and 1.0, the amount of positive change being as much as $((2.0 - \textit{selectiveDriftBias}) * \textit{selectiveDriftFactor})$, and the amount of negative change being as much as $(-\textit{selectiveDriftBias} * \textit{selectiveDriftFactor})$, per iteration.

selectiveDriftBias

Amount of pressure toward increasing selective pressure on a gene

Range : [0.0, 1.0]

Set near 0.0 for rapid drift towards high selective perfection of proteins, and near 1.0 for random drift up and down of selective perfection. Normally, one would set this parameter slightly below 1.0, to simulate gradual perfection. Selective pressure, as explained in the section on Protein residue-level parameters below, constrains the range of substitutions allowed. Conceptually, one can imagine a new paralog with much freedom to evolve, soon developing a novel function and focusing its sequence towards optimality.

maxInitialSelectivePressure

Maximum selective pressure on a new gene or paralog

Range : (0.0, 1.0]

Selective pressure limits the range of allowable substitutions available to an amino acid. See the section Protein residue-level parameters below for an explanation of how this limitation is effected. This parameter should normally be set to a low value, with selective perfection of a new gene thereby developing gradually over time, via *selectiveDriftFactor* and *selectiveDriftBias*.

Protein residue-level parameters

minGeneSize

Minimum gene length (in amino acids)

$$\textit{Range} : (1, \infty)$$

maxGeneSize

Maximum gene length (in amino acids)

$$\textit{Range} : [\textit{minGeneSize}, \infty)$$

resistanceAdjFactor

Resistance to change of a residue, relative to the previous residue

$$\textit{Range} : [0.0, 1.0]$$

Simulating domains in proteins that are more or less conserved, a chain of resistance-to-substitution is created by first setting the first residue's evolutionary resistance R_0 to a random value between 0.0 and 1.0, then iteratively adjusting each subsequent residue R_i 's evolutionary resistance in proportion to *resistanceAdjFactor*:

$$R_{i+1} = R_i * (1 - \textit{resistanceAdjFactor}) + \textit{rnd}(\textit{resistanceAdjFactor})$$

where $\textit{rnd}(\textit{resistanceAdjFactor})$ is a random number between 0.0 and *resistanceAdjFactor*.

substitutionMatrix

Global substitution matrix that applies to all residues (400 individual values)

Range : [0.0, 2.0]

A suitable matrix such as EX provides the general guidelines for amino acid substitution, further specialized by the auxiliary matrices, described below.

numAuxiliaryMatrices

Number of auxiliary matrices

Range : [1, ∞)

Each residue of each new protein is assigned one of these auxiliary matrices, providing idiosyncrasy in its substitution patterns.

auxMatrixScaleFactor

Scale of entries in auxiliary matrix

Range : [1.0, ∞)

An *auxMatrixScaleFactor* of 1.0 makes the auxiliary matrices moot (all values = 1.0), whereas a value greater than one produces a range of auxiliary multipliers ranging between $(1.0/\text{auxMatrixScaleFactor})$ and *auxMatrixScaleFactor*.

We use the following formula in order to decide on the propensity ψ of a given site toward substitution, based on the selective pressure s and the site-specific rate r :

$$\psi = 1 - s - q(0.5 - r)^{0.5}$$

with q as a normalizing constant equal to s if $r < 0.5$, and $1 - s$ if $r > 0.5$. The value of q is irrelevant when $r = 0.5$. Thus, when a gene's selective pressure is high, and when a residue's evolutionary resistance is high, the propensity to substitution at that site will be close to 0.

The acceptance of a proposed substitution from residue i to residue j is dependent on this site-specific propensity, as well as a user-supplied substitution matrix entry $M_{i,j}$ for the conversion from i to j , and the auxiliary matrix entry $A_{i,j}$ specific to that site. The following formula determines the probability R that a specific residue change at a specific site will be accepted:

$$R = 1 - \psi(M_{i,j})^{A_{i,j}}$$

The proposed substitution is successful if a random number sampled from the uniform distribution $[0.0,1.0]$ is less than R . Small values of $M_{i,j}$ will consequently produce small values of R , unless the corresponding auxiliary matrix entry is very large.

goodMutationProb

Probability of redefining the target residue as the current residue

$$Range : [0.0, 1.0]$$

A residue's reference amino acid is considered archetypal, and exploration of sequence space is constrained to residues within reach of this reference. However, a "good mutation" can sometimes occur, which is superior to the prior type. This parameter permits references to be redefined. For example, if set to 0.0, then proteins remain equivalent to their ancestors, but if set to 1.0 in contrast, there is no memory and proteins freely redefine themselves.

numProteinTypes

Number of protein classes with a given starting amino acid frequency in the cenancestor

$$Range : [1, \infty)$$

In nature, several major compositional classes of protein exist. Naively, for example, there are hydrophobic membrane proteins and there are soluble proteins; in truth several more subtle variations exist. This parameter allows one to define a number of compositional classes, currently just at the level of amino acid composition.

numOfProteinType $[i]$

Count of cenancestral proteins of type i

Range : $[1, \maxGenomeSize; \minGenomeSize \leq \sum_i \leq \maxGenomeSize]$

For each of the compositional classes of proteins desired, how many are of each type?

freqOf_ACDEFGHIKLMNPQRSTVWY $[i]$

Frequency vector of amino acids for proteins of type i

Range : $[0.0, 1.0; \sum = 1.0]$

A description of the relative frequency of each amino acid is provided, for each compositional class. Cenancestral proteins will sample from this distribution in constructing their sequences.