Supplementary Online Material for "An individual-based model simulator for transposable elements dynamics and evolution"

Felipe Figueiredo[*] Claudio Struchiner

May 28, 2015

Contents

[*]philsf79@gmail.com

1 Overview of the simulator

The `TRepid` simulator is an Object-Oriented system, with individual hosts and TEs each represented by objects. It's main interface with the user is provided by a plain-text configuration file that should be present in directory of the simulation. The simulation sub-directory tree will be created, and data and log files written by the simulator as required.

1.1 The main players

The main class defines simulation objects that contain all the information and data related to a given simulation. Associated with each simulation object is a configuration object to access or modify any parameter of the simulation, including equations parameters, file names, etc. It collects all configuration parameters and variables from configuration files, or use defaults if values are not provided by the user.

1.2 Basic usage: Input

The principal way of setting up a new simulation is through a configuration file called `trepid.conf`, inside which the user defines parameters for all the models.

The default configuration file that comes with the simulator has all parameters filled with default values, and comments explain what they mean. This file is divided in sections to organize semantics for the user. These categories are not interpreted though, since only variable names are used and users are free to remove the section names and define the config file in the order they prefer.

After preparing the configuration file, the user can run the `trepid` program inside the directory where the config file is located. Upon starting, the simulator looks for the `trepid.conf` file, and if found, create a directory structure for the population (even if the population is not saved at the end of the simulation), a directory for logs and results, and finally a directory for the Templates for the hosts with TEs.

The Templates dir is where one or more genome templates are located, and used to create the GM (Genetically Modified, i.e. with TEs) sub-population. If the user wants to use a GM population, at least one GM template must exist. If the templates directory is previously non-existent, a GM population won't be created, and in this case the simulator aborts.

If the configuration file is not found or is empty, default values will be used for all parameters. The same happens to any parameter not defined in the config file, if it is incomplete. The user can check what parameters were set for each simulation in the log header.

1.3 The default config file

In this section we describe all of the variables definable in the config file, showing the default values as illustration. These are the parameters used for a very simple simulation of a Master Gene model in a small host population that does not have generation overlap.

1.4 General

Simulation time and initial sub-populations

```
#####################################
[General]
# Number of generations (default: 10)
it_max = 10
```

FIXME: include other general variables, save_*, sample_gm, interactive, often_max.

1.5 Initial

FIXME: include other initial variable: init_gender .

```
#####################################
[Initial]
# Initial Susceptibles (default: 900)
S_0 = 0
# Initial Infectious (default: 100)
I_0 = 10000
# Initial Removed (default: 0)
R_0 = 0
```

The parameter `it_max` determines how many iterations of the model, i.e., how many generations will be simulated.

The F_0 population is determined by three initial sub-populations. In these parameters, the notations inspired by the SIR epidemic model of the early prototypes were maintained. `S_0` symbolizes the initial wild sub-population, `I_0` the initial GM sub-population (i.e. individuals that carry TEs) and `R_0` the Deceased or past individuals.

If a population was previously saved to disk, the files will be automatically read and the F_0 population will be constructed using those individuals. If the initial parameters for either of `S_0`, `I_0` and `R_0` are greater than the pre-existing individuals, additional ones will be created to match the simulation criteria.

1.6 Reproduction

Age structure

```
#####################################
[Reproduction]
# Life expectancy (default: 4)
age_max = 4
# Reproductive age (default: 3)
age_reprod = 3
# Offspring count (default: 100)
offspring_count = 100
```

The two parameters age_max and age_reprod determine the age structure in the population. They are measured in generations. Setting the age_reprod to one unit lower than age_max configures a population without generation overlap. Setting age_max to 2 and age_reprod to 1 disables the age structure in the sense that all individuals last only one generation after being created.

```
# Recombination model (default: 1step)
recombination_model = "none"
```

The above parameters set the total offspring per couple, and turn off the recombination for the simulation.

```
# Reproduction model (default: Logistic)
reproduction_model="Logistic"

# Logistic parameters
# reproductive (default: 0.1)
r = 20

# carrying capacity (default: 5000)
K = 20000
```

The above options select the ecological model (in this case a logistic population) and define the model parameters (in this case r and K).

```
# ploidy parameters (default = 2)
ploidy = 1
```

The ploidy can be artificially set to diploid or haploid. Haploid populations are created in the following manner: upon creation of a new individual, the first chromosome comes from the father and the second from the mother, but since only one chromosome will be considered throughout the simulation. The contents of both chromosomes are merged, possibly overwriting a specific TE, in such cases where there is a TE in the same site in both parents' chromosomes. This is done by design in order to guarantee that there exists only one active TE in the haploid case, as opposed to what would be expected in the diploid case in which one active element would be inherited from the father, and another element from the mother. In the next generation, two would be inherited from the father and two from the mother, in geometric progression.

1.7 Transposition

```
######################################
[Transposition]
# Species' categories sites (default: k=0, S=30, N=70)
killer_sites = 0
severe_sites = 0
neutral_sites = 200
```

Sizes of each chromosome section. *Killer sites* are those that kill the individual immediately upon birth, if occupied by an element. *Severe sites* add fitness costs, and *Neutral sites* are ignored.

```
# Fitness cumulative impact of transposition (default .05)
severe_impact = .05
```

This value is incremented for each TE that occupies a *severe site*. The total impact is rounded up to the next integer. This total impact is the quantity of offspring the individual will not be able to produce, due to TE activity. In this sense, the total TE impact will be subtracted from the amount of offspring the individual will contribute to the population, and it is calculated for both the father and the mother.

```
# Transposition model: constant, exponential or SKR (default: SKR)
transposition_model = "constant"
```

The *constant* model produces the same amount of new elements per transposition event, i.e., in the gametogenesis. At least one active element is required for the creation of new copies.

```
# Transposition rate for constant and exponential model (default: .5)
transposition_rate=1
```

The *transposition rate* parameter has different meanings, depending on which transposition model was selected. If the *constant model* is being used, this parameter is the amount of new copies being created at each gametogenesis. In the case of the *exponential model* or the *SKR model* it is the probability that new copies will be created, i.e., the probability of occurrence of a transposition event in that gametogenesis.

```
# Excision rate for constant and exponential models
excision_rate = 0
```

The *excision rate* is analogous to the *transposition rate*, but instead of contributing to the creation of new copies, if subtracts TEs from chromosomes. Both the possible values and meaning for each transposition model are analogous to the interpretation in the *transposition model*, as stated above.

```
# Deleterious threshold (default: severe + neutral)
# could be set to any arbitrary value
#delete = 10
```

FIXME: explain.

```
#Inactivation model: mastergene, random and progressive (default: mastergene)
inact_model = "mastergene"
```

The activity *status* is represented by a number between 0 and 1. Any positive number implies the element is active, while 0 means the element is inactive.

There are three *inactivation models* available. The *Master Gene* postulates that every new copy is automatically inactive (thus, has a *status* of 0). The *random* model samples a random number between 0 and the *status* of the template element, thus reducing the mean *status* over successive generations. The *progressive* model decreases the *status* of each new copy by a fixed amount, also decreasing mean *status* over time. Small *status* values are truncated to 0.

```
# Invasion threshold for GM sub population (default: 0.9). Set to 1 to disable.
invasion_threshold = 0.9
```

```
# Loss threshold for GM sub population (default: 0.01). Set to 0 to disable.
loss_threshold = 0
```

FIXME: explain.

1.8 Evolution

```
######################################
[Evolution]
# Probability that a mutation (substitution) will occur (default: .1)
mutate_prob = 1
```

The *mutate probability* is the probability that a evolutionary event will occur in the selected transposition event. This implies that at some amount of point substitutions is bound to occur, although some might be silent substitutions.

```
# Number of point mutations to be introduced in each evolutionary event (default: 2)
mutation_count = 2
```

FIXME: explain.

1.9 Output

Over the course of a simulation some output files are created: a log file, two CSV files (one for the demography and one with TE data), and if an individual can be sampled from the population in the last generation, it will be saved to disk in two files (in the formats FASTA and NEXUS). All of these files are prefixed with the simulation date, and a serial number, so files from successive simulations can be observed in chronological order.

1.9.1 Log file

FIXME: explain

1.9.2 CSV tables

FIXME: explain.

1.9.3 Sampled individual

FIXME: explain.

1.10 Advanced usage: API for Perl programmers

Although the configuration file can provide a practical input for several simulation scenarios, some users might have more complex needs. Besides the configuration file there is an API which can be directly accessed by software, scripts or an interactive shell in Perl language which facilitates step-by-step procedures, analysis, reanalysis and more detailed data acquisition.

All data generated in each component can be accessed from the API, and accessor methods are provided for each data type for all object classes. Additionally, the whole generation algorithm can be bypassed and simulations can be manually run, step by step, and scrutinized in any way desired. One can change config parameters after a few generations for any of the models, or even the models themselves, insert or remove individuals in the population, etc.

The API documentation along with the complete list of functions and class and object methods are provided inline with the code and can be accessed by the usual `perldoc` command. PDF and HTML documents are also provided in the software package.

Each simulation run is identified by a unique ID string, and all objects related to that simulation are tagged with this ID, so several simulations can be run by the same instance or script without risk of collision of variable values, either in batch or in parallel. We provide an example script to run a batch of simulations with different parameters.

2 Algorithms and models

The simulator is flexible in construction and able to produce a realistic scenario in which transposition events occur in finite populations that agree with the Wright-Fisher conditions [Hartl and Clark(1998)Hartl and Clark]. Transposable elements (TEs) replicate according to an arbitrary transposition model and TE sequences' evolve according to Kimura's infinite-sites model [Tajima(1996)Tajima], and an arbitrary model of molecular evolution [Yang(2006)Yang].

2.1 Population

As described above, the population has two independent structures: gender and age. We have drawn inspiration from the SIR epidemiological model to structure the population in discrete compartments. The age structure is also discrete, as described below.

The population is divided into **Wild** and **GM** (genetically modified) compartments of hosts similarly to the SIR compartmental model classes *Susceptible* and *Infected*. There's also a **Deceased** compartment, in analogy to the *Recovered* compartment, where individuals from earlier generations are kept for future reference or analysis. A given individual cannot change from the **Wild** to the **GM** compartment, though, and vice versa. Instead, these compartments' dynamics evolve through generations of sexual reproduction of the host population.

Time is measured in simulation steps representing discrete generations. The age structure is defined by two classes, regarding wither or not a host is mature enough to reproduce. Each new individual remains non-reproductive for a maturation period, after which it becomes mature and enter the reproductive pool in the population.

Age of sexual maturity and maximum lifespan are passed to the simulator as configuration parameters by the user. Default parameters for maturation, lifespan and offspring count reflect behavior expected for strongly r-selected species with no generation overlap, suitable for modeling some insect populations such as mosquitoes or fruit flies.

Default age parameters	
Initial age	**1**
Reproductive age	**3**
Maximum age	**4**

The growth rate of the population is determined at the beginning of each generation from an arbitrary ecological model (see below), and from this rate the net income of new individuals for the next generation is obtained. The necessary amount of reproductive encounters to reach the appropriate number of new individuals for the next generation is then determined by dividing the total income by the expected number of offspring per couple as indicated in equation (1). Note that assuming a non-monogamous species, this is equal to the number of females expected to mate in this generation, as there should always be enough males to impregnate any available fertile females.

$$\text{reproductive encounters} = \frac{\text{total income of new hosts}}{\text{expected number of offspring}} \tag{1}$$

Couples are then pooled from the available mature population to satisfy the necessary number of reproductive encounters. We are modeling host species that are not monogamous, so males are chosen from the population with replacement while females are chosen without replacement. This approximates the random mating behavior (i.e., no spatial structure), while retaining the realistic behavior that a given female can only copulate at most once per generation.

The host fitness is defined as the total offspring count that host has generated. The mean fitness over the population for a given generation is the arithmetic mean of the fitnesses of all live individuals at the end of the generation.

The offspring amount for each reproductive encounter could be drawn from a distribution whose mean is determined for the population. For simplicity, however, we consider in this version of the simulator a fixed initial amount of offspring for each couple. From this number it will be subtracted a quantity proportional to the impact TEs might have on the host fecundity. We postpone the definition of fitness impact and how it is calculated to the transposition model section.

Migration between meta-populations can be introduced in the simulator by the use of several geographic patches. Each of these patches is a micro-environment of its own, with the same compartment structure. For simplicity, in this version of the simulator we consider only one geographic patch, so migration events are ignored. This feature is planned for a future release.

2.1.1 Ecological Model

At each generation, the total number of new offspring must be determined before mating begins. To this end, an ecological model is consulted to determine the dynamics of the population in the absence of external influence (in our case, impact from TEs, which is described below).

The simulator is modular in the sense that the ecological and transposition models are implemented and considered independently from the rest of the system framework, so they function as exchangeable parts in the mechanism. This way, many other models can be included in the simulator in order to suit particular demands for each of these sections.

Indeed, any population model can be used in this system provided it can be represented in the form:

$$p_{n+1} = F(p_n) \tag{2}$$

where F is a function that depends on the current population size p_n. It is not required that F be deterministic, i.e., depends only on p_n. It can optionally depend on a random variable ξ_n to introduce intrinsic stochasticity in the growth rate:

$$p_{n+1} = F(p_n, \xi_n) \tag{3}$$

As a result, it is also not required that the model should be written explicitly as a differential or difference equation. The only requisite is that the input is the current population size, and the output is the expected quantity for the next generation.

We currently implement five population models, two models of unrestricted growth and and two models for saturated growth: constant population, constant growth (linear), exponential growth (or decay), the Logistic and the Hassel [Hassell(1975)Hassell] equations. All of these are discretized from Ordinary Differential Equations, as described in section 4.1.

2.1.2 Age structure

The population has an age structure in which it is divided in discrete classes. The behavior of such models has been thoroughly studied in the ecological literature [Nisbet and Gurney(1982)Nisbet and Gurney].

One can configure the parameters to set any arbitrary number of age classes for a simulation, but there are two age classes of obvious interest: before and after reaching the reproductive age. The durations for the maturation stage and the adult stage last are selected as configuration parameters to the simulator, and can be chosen to reproduce several scenarios, according to the organism being modeled. The default values for maturation and longevity are appropriate to species that don't have generation overlap, such as most insects.

(FIXME: move to another section) The default value total offspring for each reproductive encounter is also selected as to represent r-selected species with high fecundity and high mortality, such as insects.

2.2 Molecular evolution

2.2.1 The genome

The genome in each individual is composed by two chromosomes, and each chromosome is represented discretely as a list of insertion sites that act as *loci* for TEs. Each insertion site in each chromosome can therefore be either empty or occupied by a TE. To reflect the fact that some insertions can cause disadvantageous, deleterious or even fatal mutations on the host, we categorize the insertion sites into three classes: fatal (*killer sites*), meaning they disrupt essential genes in a way they render the cell useless; severe (*severe sites*), which may disrupt non-essential genes, or metabolic pathways, and neutral (*neutral sites*). Under the hypothesis that within a species genomes from different hosts should bear more similarities than differences, the number of insertion sites in each of the above classes should not vary much within a species, and could be considered species specific if such numbers can be estimated based on genome size.

The genome size in this simulator is thus defined as a result of config parameters choices. Each chromosome has $\eta = k + s + n$ *loci*, each of which can contain one or zero TEs. These sites can be reordered, without loss of generality, in such a way that the first k sites are *killer sites*, the next s are *severe sites*, and the remaining n are of *neutral sites.*

2.2.2 Recombination and ploidy

A model of crossing-over recombination is optionally used to promote additional variability of chromosome content. It is implemented in three variants called *1-step*, *2-step* and *all-step.* The resulting gamete retains the original size in insertion sites, and each site is filled consulting one of the parent's corresponding site from one of the parental gamete.

In the *1-step* model a site is chosen at random and this site is the cutting point between the parts coming from each chromosome. The resulting gamete will be filled up to this site identical to the first chromosome and then the remaining sites are merged from the second chromosome.

The *2-step* model is similar to the above, but there are two cutting points instead of one. This means the first and third sections of the gamete are merged from the first chromosome and the second is merged from the other chromosome.

There's also an *all-step* model that abstracts from the two models above in the sense that every site is a cutting point. This way every site is taken from a random chromosome preserving only the order from which it came. This will provide maximum variability considering recombination events.

If none of the above recombination models is selected, the resulting gamete will be equal to one of the parental chromosomes, chosen at random.

Additionally, although we are mainly interested in simulating sexual populations, a workaround is provided to represent haploid simulations, for the benefit of simplicity. If this option is chosen, the second chromosome of the offspring being created is always ignored, and its contents copied to the first one respecting the original chromosome site, possibly overwriting an existing TE that previously existed there. This "haploid sexual population" is a practical way of creating simple simulations and hypotheses.

2.2.3 The evolutionary model

Over time mutations occur and are accumulated in the genome of simulated hosts. Whenever a mutation occurs a substitution drawn from an evolutionary model.

We implemented a simple model that follow the assumptions of the Jukes-Cantor substitution model [Jukes and Cantor(1969)Jukes and Cantor]. All mutations are substitutions and equally likely to occur. As a result no special assumption is made over sites in the sequence which are more likely to change, nor transition/transversion bias. *All sites are considered equally likely to change as well as all nucleotides changes are considered equally likely to occur.*

As happens with the ecological section, almost any evolutionary model can be implemented in the simulator. The requirement here is that the input is the original sequence, and the output is the mutated sequence.

Which changes are allowed or disallowed are completely up to the model in question. The evolutionary model is any model or function that, given an input nucleotide sequence, outputs a similar homologous sequence.

2.3 Transposition model

The existence of TEs in a host genome can cause several kinds of impacts in the host, ranging from fitness reduction, longevity reduction and even host inviability. There are several ways to model both how the TEs replicate in the genome, and the impact that such replication causes in the host that carry them.

The three key elements to be considered in the modeling of these phenomena are the way the total amount of TE copies vary, the ability of TEs to be transposed and the impact that each TE might cause in the host.

2.3.1 Activity cycle

The initial phase of the TE invasion, should be regarded as the period during which the TE is most active. Otherwise, genetic drift may lead the TE to extinction [Le Rouzic and Capy(2005)Le Rouzic and Capy]. After this initial phase, if the TE succeeds in fixating in the host population, it must decrease its activity so as not to disrupt too much the host's fertility.

Transposable elements usually undergo some degeneration process, that inactivates TE copies. We take this phenomenon into account in the model in the form of an *status score*, that is a fixed number between zero and one for each TE. When a status of zero is selected, it renders the TE inactive. This score is also used to determine whether of not the TE is actively transposing.

Every time a new TE is created from an existing TE, the new copy's *status score* is chosen to be less than the original score. How fast this score drops to zero in successive transpositions can be implemented in several ways, and could be used to reproduce several interesting scenarios:

- if the status is constant and zero, then every new copy is inactive, and the only active copy is the original one. This is known as the *master gene model*;
- if the status is constant and non-zero, then every new copy is active, which represents the *transposon model*;
- the status can decrease in a constant rate, which will render new copies inactive after a few
- the new status can be chosen randomly, between zero and the original value

2.3.2 Forms of impact on hosts

New TEs can cause an impact on the host, depending on where in the genome it lands when created. This is implemented in the three categories of insertion sites where TEs might appear. If a TE is created in a *killer site*, it will be considered a deleterious transposition. Hosts can be born dead due to deleterious transposition, and otherwise the impact will be considered a fitness toll, thus the host will have a lower offspring count when compared.

All transposition models can be used with or without individual fitness impact on hosts, as an additive fitness impact can be optionally defined as a config parameter, which is a real number. All elements that have such an effect on the host are accounted for and their relative impact is summed and rounded up. This total impact is the total amount of offspring this particular host will be unable to produce due to disadvantageous transpositions in its genome.

Lifespan can also be decreased if the impact is too great, albeit not fatal (as proposed in [Le Rouzic and Deceliere(2005)Le Rouzic and Deceliere]). We include this characteristic in the model as follows. First we take the ratio between the total fitness impact as calculated above and the maximum age defined in the config for the simulation, then this ratio is rounded down. This integer is the total age classes the host will be unable to achieve due to deleterious transpositions.

2.3.3 TE activity dynamics

Several transposition models can be implemented in the system. We implemented both neutral models in respect to natural selection (constant and exponential growth), and a transposition model that takes into account intrinsic deleterious effects by insertion of new elements [Struchiner *et al.*(2005)Struchiner, Kidwell, and Ribeiro].

Transposition models implemented in our framework have two main components: one for the the acquisition of new copies and one for the excision of existing copies.

The acquisition model (usually called by a `transpose()` function) determines the total amount of new copies that should be created for the gamete. The excision model (usually called by a `excise()` function) is the exact opposite. Parameter choices in the simulation configuration should consider cases where the creation of new copies never falls behind the deletion of old copies, otherwise the

2.4 The population genealogy

We are interested in analyzing sequence data from an individual to reconstruct the phylogenetic relations between its TEs. Therefore it becomes necessary to compare the reconstructed history with the real parental history we simulated.

Each host is identified by a name defined as a serial number, and carries the names of both parents. A simple recursive breadth-first algorithm is used to recover the names of parents, grandparents and so on. As such the genealogy can be fully reconstructed up to any valid time interval. The population stored in the **Deceased** compartment can be consulted to create genealogies of different depths after the execution of the simulation, or inspect each individual for it's genome, in order to compare the elements present in the sampled individual, and its ancestors.

2.5 Sources of stochasticity

Although most of models described so far involve deterministic models the overall behavior is stochastic. This happens due to both sampling effects and the fact that the system is an individual based model, or agent based model [Bonabeau(2002)Bonabeau]. The following are sources of such uncertainty:

1. sampling of individuals in finite population
2. recombination
3. replication and excision of TEs
4. mutations (substitutions) within TE sequences

The item 1 refers to Wright-Fisher models of populations [Hartl and Clark(1998)Hartl and Clark]. Since we're simulating small populations (typically $< 10^5$, but there's no restriction to population size) there is a sampling effect to be considered when an individual is chosen for reproduction. Should we simulate very large simulations the importance of sampling effects could be diminished, and we could approximate the behavior of an infinite population (as in a Hardy-Weinberg population).

Since we're simulating sexual populations, we incorporate recombination by crossing over of games. This can be implemented in several ways, and the item 2 is related to the recombination model used to promote additional variability to the chromosome pool.

FIXME: complete this section

The item 3 is related to the availability of new TEs in the chromosomes (as defined by different *loci* and sequence).

The item 4 is related to the evolutionary model used to promote variability across generations.

3 Availability

The simulator has been developed and tested in GNU/Linux systems running Ubuntu Linux. As it is a portable language, it should run on any system for which its dependencies are available. Those include Mac OS X and Windows, besides UNIX/Linux in general. Packaging has been prepared following guidelines typical of Perl modules and applications, which should ease the installation in any platform Perl is available with the help of the CPAN framework[1], as well as Debian/Ubuntu DEB packages.

The software is released in the open-source license GPL and is available from https://launchpad.net/trepid .

4 Implementation details

4.1 Population models

4.1.1 Implementation of the logistic model

We will use the logistic equation as an example to describe the implementation because of its simplicity; the implementation of the Hassel model is analogous.

The differential equation is discretized as a single iteration of Euler's method, with a step size $h = 1$. This results in formula (4):

$$p_{n+1} - p_n = \Delta p = Ceiling\left(\frac{r \times p \times \left(1 - \frac{p}{K}\right)}{\text{offspring count}}\right) \tag{4}$$

where $Ceiling(x)$ ($Floor(x)$) is the smallest (greatest) integer greater (less) than or equal to x. Note that this implementation already takes into account the normalization by the total offspring count as per equation (1), so the result is the number of reproducing couples for that generation, instead of the total income of new individuals.

This model is deterministic so the caching system described in section 4.3 is used for it.

4.1.2 Constant model

The simplest population model available is arguably the constant population, which assumes equilibrium and zero net migration.

The parameter considered for the constant growth is the r config parameter (reproductive rate), which is used in this case as the total amount of new individuals to be created for each generation, rounded up if it's not already an integer.

$$\Delta p_n = r \tag{5}$$

4.2 Transposition models

4.2.1 Exponential model

The exponential model is a generalization of the constant model, but instead of only depending on a fixed quantity, it also depends on the relative number of copies existing at each time. This model also belongs to a wider class of neutral models that don't take into account any intrinsic regulation of copy number by fitness disadvantage to the host [Le Rouzic and Deceliere(2005)Le Rouzic and Deceliere], where the transposition rate u_n is a bounded function of c_n and equilibrium is attained when it saturates at the same value of the excision rate e.

[1]http://www.cpan.org

$$\Delta c_n = (u_n - e)c_n \tag{6}$$

Considering the transposition rate u_n constant (therefore u), it can be rewritten as:

$$c_{n+1} - c_n = \Delta c = (u - e)c \tag{7}$$

that only depends (deterministically) on the current quantity of copies c in the reproducing host. As such, the same discretization scheme used in the population models can be used to provide a cache-able model that continually increases or decreases copy number over generations, depending on the value of $u-e$. If $u - e > 0$, the total copy number will always increase, whereas if $u - e < 0$ the simulator will tend to remove more copies that are added.

The model is consulted each time a host produces a gamete both for the amount of new TE copies to be created, and the amount of existing copies to be excised. If there must be new TE copies created, for each new copy a random active TE is sampled (with replacement) from the host genome, and replicated. Afterwards, if there must be excision of existing copies, for each excised copy the genome is sampled for a random TE (without replacement) and this copy is removed from the genome. Note that only active TEs get transposed, but any TE can be excised.

4.2.2 Constant model

The constant model of transposition is analogous to the respective population model counterpart. At each time the model is consulted for the amount of new TE copies to be created, the transposition rate config parameter *transposition_rate* is used as the total amount of new copies to be generated (rounded up). There is no significant dynamics in regard to the copy number growth, and it's not bounded per individual, except for the total number of insertion sites in the genome, defined in the config file (*killer_sites*, *severe_sites* and *neutral_sites*).

The excision of copies is also constant, defined by the config parameter e as the total amount of previously existing TE copies to be removed from the gamete.

$$\Delta c_n = u - e \tag{8}$$

4.2.3 SKR model

This model assumes an intrinsic self-regulation of TE copy number, and can assume various shapes and growth forms. The original article proposes that the form of the $U(c_n)$ function can be chosen as any functions that share the qualitative behavior described, and suggests (and tests) three forms indexed as U_1, U_2 and U_3 that we reproduced in this framework.

$$c_{n+1} = Ceiling(c_n + c_n \times T_0 \times U(c_n)) \tag{9}$$

$$U_1(c) = 2^{(-c/C_{0.5})} \tag{10}$$

$$U_2(c) = 1 - \frac{c^5}{(C_{0.5}^5 + c^5)} \tag{11}$$

$$U_3(c) = 1 + (c - \frac{0.5}{C_{0.5}}) \tag{12}$$

Transposition with this model is provided by the preceding equations, while excision is provided by the exact same function of the exponential model.

4.3 Caching of deterministic models

In order to save unnecessary calculations, caching of model data is implemented for both population and transposition models. All values calculated for deterministic models are cached to avoid the performance toll of unnecessary repeated calculations. This cache is implemented as a hash[2], and is consulted before each calculation to see if it's necessary.

The cached data are stored independently per simulation and per data type (population or transposition), so in case a user wants to instantiate several different simulations in the same session [3], data from different simulations are guaranteed not to mix.

5 Data structures

The model is implemented in Perl5, using its object-oriented paradigm. The main classes `TRepid::Population`, `TRepid::Host` and `TRepid::TE` define, respectively, objects for the population, the hosts that constitute the population and the TEs that populate each host genome.

Two classes inherit methods from Bioperl classes [Stajich *et al.*(2002)Stajich, Block, Boulez, Brenner, Chervitz, Dagdigian, Fuellen, Gilbert, Korf, Lapp, Lehvaslaiho, Matsalla, Mungall, Osborne, Pocock, Schattner, Senger, Stein, Stupka, Wilkinson, and Birney]. The TE class inherits the `Bio::Seq` class, and the Host class inherits the `Bio::SeqIO` class. This guarantees the simulator can import and export sequences using a great range of sequence formats. It also uses the abstract model of representing DNA sequences provided by the Bioperl project in a way that unifies the usage of any sequence format (Fasta, genbank, embl, swissprot, etc) so that any of these formats can be used at the discretion of the user. We provide a wrapper for including header information in the Fasta format, which we describe in detail in a later section.

The documentation of the API is bundled within the code with the POD (Plain Old Documentation) format, which can be accessed by the ordinary means of any Perl distribution. Besides being available inline in the code, all this material is also converted and provided in more convenient formats like HTML and PDF.

5.1 Host data structure

The structure inherited from `Bio::SeqIO` provide methods to collect and save sequences from a file, which inspired an implementation for a file-based storage. Besides these general-purposed methods we defined some object attributes to store meta-data related to each individual like its name, age, gender, fitness, the two chromosomes the file name to which the `Host` object is associated.

Each of the two chromosomes is an array, and each of its positions stores a reference for a TE object if one is present. Methods for consulting the number or position of TEs and de-reference them are provided in the class API.

5.2 Population data structure

The object attributes for the `Population` class store information related to where the files are stored in disk, and some statistics that are frequently consulted.

5.3 TE data structure

Besides the structure inherited from `Bio::Seq`, which includes methods for setting and retrieving several meta-data as the sequence string, header information, etc.

[2]Sometimes referred to as an associative array.

[3]This functionally requires usage of the library via the API. A sample script is provided as an example for programmers.

5.4 Meta-data format for Fasta sequences

We default to using sequences in Fasta format. To this end, we propose a protocol for inserting meta-data into the Fasta header and a set of corresponding parser (writer) methods in order to retrieve (save) the corresponding information.

```
>string:int:int:real
```

The first field (string) denotes the **TE description**. It can vary in size and can contain spaces. Usually it comprises the whole Fasta header if a real sequence is used, as acquired from any genomic database, should one such sequence be used.

The next two fields that contain integer numbers. The first is the **site** position in the chromosome, and the second defines in which **chromosome** this particular TE is located. The total number of sites available can be chosen by the user, and the default is 100 positions (from 0 to 99). There are always two chromosomes (denoted by the numbers 0 and 1) since we are modeling sexual diploid populations.

The last field is a real number, in the interval $[0, 1]$. It describes the **status score** of activity for that TE, where any non-zero value means the TE is active. Each new copy generated from this copy should have a lower score, effectively reproducing a deactivation function for the TE family.

References

[Bonabeau(2002)Bonabeau] Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences of the United States of America*, **99**(Suppl 3), 7280–7287.

[Hartl and Clark(1998)Hartl and Clark] Hartl, D. L. and Clark, A. G. (1998). *Principles of population genetics*. Sinauer Associates, 3 edition.

[Hassell(1975)Hassell] Hassell, M. P. (1975). Density-dependence in single-species populations. *Journal of Animal Ecology*, **44**(1), pp. 283–295.

[Jukes and Cantor(1969)Jukes and Cantor] Jukes, T. H. and Cantor, C. R. (1969). Evolution of protein molecules. In H. N. Munro, editor, *Mammalian protein metabolism*, pages 21–123. Academic Press, New York.

[Le Rouzic and Capy(2005)Le Rouzic and Capy] Le Rouzic, A. and Capy, P. (2005). The first steps of transposable elements invasion: parasitic strategy vs. genetic drift. *Genetics*, **169**(2), 1033–43.

[Le Rouzic and Deceliere(2005)Le Rouzic and Deceliere] Le Rouzic, A. and Deceliere, G. (2005). Models of the population genetics of transposable elements. *Genet Res*, **85**(3), 171–81.

[Nisbet and Gurney(1982)Nisbet and Gurney] Nisbet, R. M. and Gurney, W. S. C. (1982). *Modelling fluctuating populations*. John Wiley, Chichester. US.

[Stajich *et al.*(2002)Stajich, Block, Boulez, Brenner, Chervitz, Dagdigian, Fuellen, Gilbert, Korf, Lapp, Lehvaslaiho, Matsall
Stajich, J. E., Block, D., Boulez, K., Brenner, S. E., Chervitz, S. A., Dagdigian, C., Fuellen, G., Gilbert, J. G., Korf, I., Lapp, H., Lehvaslaiho, H., Matsalla, C., Mungall, C. J., Osborne, B. I., Pocock, M. R., Schattner, P., Senger, M., Stein, L. D., Stupka, E., Wilkinson, M. D., and Birney, E. (2002). The Bioperl toolkit: Perl modules for the life sciences. *Genome Res*, **12**(10), 1611–8.

[Struchiner *et al.*(2005)Struchiner, Kidwell, and Ribeiro] Struchiner, C. J., Kidwell, M. G., and Ribeiro, J. M. C. (2005). Population Dynamics of Transposable Elements: Copy Number Regulation and Species Invasion Requirements. *Journal of Biological Systems*, **13**(4), 455–475.

[Tajima(1996)Tajima] Tajima, F. (1996). Infinite-allele model and infinite-site model in population genetics. *Journal of Genetics*, **75**(1), 27–31.

[Yang(2006)Yang] Yang, Z. (2006). *Computational Molecular Evolution*. Oxford Series in Ecology and Evolution. Oxford University Press.