**A computational method to simulate evolutionary branching trees for analysis of tumor clonal evolution**

Thomas O. McDonald and Franziska Michor

Department of Biostatistics and Computational Biology, Dana-Farber Cancer Institute, and Department of Biostatistics, Harvard T. H. Chan School of Public Health, Boston, MA 02215, USA.

**Abstract**

KATE (K-type branching process Analysis software for simulating Tumor Evolution) is an R package that uses C++ to simulate homogeneous and inhomogeneous stochastic branching processes under a very flexible set of assumptions. The software simulates clonal evolution with the emergence of driver and passenger mutations under the infinite-allele assumption. The software is an application of the Gillespie Stochastic Simulation Algorithm expanded to a large number of types and scenarios with the intention of allowing users to easily modify existing models and create their own model. Visualization options are included in the code as well.

**Availability**

Available as R package and C++ executables on Github.

**Introduction**

Branching processes are a class of stochastic processes used to model the growth and composition of reproducing populations. These processes are ideal for modeling clonal evolution of cancer cells, where new mutations may appear and give rise to subclones with different fitness than their parents (citation), leading to a potentially large number of subclones in an exponentially growing population. Simulation of these complex processes is possible with the Gillespie Stochastic Simulation Algorithm (Gillespie, 1977) which is typically used to model chemical reaction systems, but is also useful in birth-death processes where individuals have exponentially distributed lifetimes before undergoing some event.

KATE adapts the SSA direct method to simulate an infinite-allele branching process where mutant cells are of a unique type and have random variables for their birth and death rates. The process is formally described in Foo et al (2015) and Gao et al (2016). The process begins with an individual ancestor with some birth rate, , and death rate, . Upon giving birth, the new daughter may mutate into a new type with probability . If an individual gives birth to a new mutant, the mutant has the birth rate , where is chosen from a probability distribution . Each new mutant has a birth rate equal to the sum of its mother and a random variable with distribution . We expand upon this process by including the ability for cells to have time-dependent rates, and , and mutation probabilities to change in new clones as well.

This class of branching processes can be useful in modeling the effect of driver mutations while still accounting for genetic drift from passenger mutations since the fitness of new mutants is chosen from a distribution, removing the need for distinguishing between mutation types in the formulation of the branching process. The implementation of a fitness landscape also allows different mutations to have different fitness effects instead of requiring the same change with each new mutation (citation). Gao et al. (2016) use a model with random fitness effects to show the accumulation of copy-number alterations in triple-negative breast cancer can not be explained by gradual evolution and instead support a model of punctuated equilibrium. The models also consider the effect of epistasis and varying mutations rates which are both implemented in KATE.

**Description**

KATE uses the direct Stochastic Simulation Algorithm to advance the simulation by first determining the time until the next event (birth, death, mutation, etc.) and then choosing the clone that undergoes the event and which type of event that takes place. When a birth takes place, the new individual can be a mutant with probability where is the new type of individual. Under this condition, a new clone is formed which is unique to the process, and the parameters for this type can be chosen from their respective distributions. The code was written in a modular manner to simplify customizing the process and distributions, so the user is able to easily change the underlying distribution in the code and recompile to achieve different results.

KATE simulates large birth-death-mutation processes in C++ with flexibility in model selection, including multiple (even infinite) types of cells. Throughout the process the number of cells within each type is stored and the final count along with information about each type is output at the end of the simulation. Clones are treated as nodes in a double linked list along with a link to the parent clone and the birth rate, death rate, mutation rate, parent node, and time of appearance is stored. Each clone is given a labelled value equal to the order of its appearance in the process, and the identity of the clone contains the labels of all ancestors of the clone. This approach allows the user to trace the lineage of any clone in the process. Ancestors can be input into the model with their own labels as well.

We currently allow the new types to have birth and death rates from a double exponential distribution as in (Foo, et al. 2015) with an atom at 0 with mass equal to the probability that the mutation is a passenger and has no effect on fitness. The mutation probability of a daughter can also be determined from a random distribution, where .

The model inputs determine the type of process run, with the most general process allowing for mutations with random fitness contributions. Setting the mutation probability to zero yields a branching process where no new clones appear, so the branching process is single type or has a type-space based on the ancestor list provided as a separate file. Allowing for a mutation probability, but setting the passenger probability to 1 or removing information about the fitness distribution simulates the infinite-allele branching process from (Pakes 1989).

**Ancestor Information and Input**

A separate input file may contain information about ancestors, allowing the user to begin a process with as many ancestors as desired, or restart a process that previously ended. The ancestor information is a tab-delimited file where each row describes a clone and must contain at least the number of ancestor cells associated with a clone, but may also contain the birth rate, death rate, mutation probability, and clone identifier. The output of a process can serve as the ancestor input, so the time of appearance and information about a clone’s descendants may also be included but are only for record-keeping. Given a list of cell counts without any other information, the simulator will use values coming from the input file that contains global parameters.

The input file allows the user to designate parameters and options for simulating and outputting the distribution. The various parameters are described in the appendix. Default values are given so that the process may be run without any input and a single-type birth-death process will be simulated.

**Time-Dependent Processes**

A separate program is included that extends the birth-death-mutation process simulation time-inhomogeneous processes where the birth and death rates vary as a function of time. We implement the SSA using adaptive thinning to choose the next time and the state is updated at each time step by integrating over the propensity function (Lewis and Shedler, 1979). Since adaptive thinning includes simulating a random variable from an exponential distribution with a constant propensity function that bounds the propensity function at that time, we distinguish between the birth rate function which should be bounded on [0, 1] and a multiplier parameter. This way we just set the bounding function as the sum of all rates of individuals at a particular time. We provide functions to allow polynomial rate functions and a general logistic function as examples, but the user can create their own functions and recompile the program. Inputs into the functions are structures containing the parameters. These parameters are included in the ancestor list of input file.

**Sampling**

Motivated by single-cell sequencing studies, an option to sample from the final population is included where the user specifies the number of samples and the sample size (Gao et al 2016, Wang et al 2014). The output is a list of identifiers for the cells sampled which represent the mutations present in each cell. Sampling proceeds without replacement within each sample. Selecting this option allows the user to observe differences in heterogeneity and phylogenies between the true population and samples. Single cell sequencing is currently limited to a small number of cells, so the cells sampled would most likely be from dominant clones. Sampling can give a glimpse of the expected evolutionary structure of these dominant clones, but should rarely include subclones with smaller counts.

**Application to Analysis of Tumor Punctuated Equilibrium**

KATE includes two preprogrammed scenarios to illustrate how a user can customize a process by creating a function class that advances the state of the process (selecting the next event). The functions that advance time and state are individual function classes that are members of abstract base classes. This approach provides some flexibility in the types of processes that can be studied.

As an example, we use a scenario where punctuated equilibrium is simulated by allowing a burst of multiple mutations to occur that will typically increase the death rate in the resulting subclones, but may increase the birth rate instead with a given probability (Gao, et al. 2016). To implement this process, we created a function class that advances the state of the process by selecting a clone to give birth to a new individual or die. If it gives birth, an offspring can arise as a new clone with some probability. Given this condition, a burst may appear in which multiple mutations arise with an additional probability, and the number of mutations comes from a Poisson distribution. The burst in mutations leads to a contribution to the birth or death rate by some multiplicative factor. The punctuated model is different from the base models by adding in steps when a new mutation arises. We treat this as a completely separate function class from the base class to give flexibility to a user that may want to implement different models, and also allow us to determine the scenario at the beginning of the simulation instead of proceeding through unnecessary conditional statements in each iteration of the SSA.

**Supplement**

Since we implement the simulation using the direct SSA, the process can be potentially slower based on large mutation probabilities or ancestors. To improve performance slightly, after each birth or death, the clone that experienced the event can either switch places in the linked list with the previous or next clone based on the total propensity. Over time, these individual switches will begin to create an ordering where preference is given to clones with larger counts or higher rates. This approach will eventually lead to a semi-sorted linked list that requires less steps on average per time step to search the linked list for the next event that occurs. As new clones enter the process, they are added to the end of the linked list, so they are automatically sorted.

**Instructions for Use**

**Installation**

Requirements**:**

* GNU Scientific Library (<https://www.gnu.org/software/gsl/)>
  + Install using homebrew in Terminal with  
    *brew install gsl*

**OR**

* + Download the latest GSL library from <http://ftpmirror.gnu.org/gsl/>
  + Unpack and follow the instructions in the INSTALL file
* C++ compiler (gcc or clang)

Compilation:

* Open terminal, point to directory with Makefile and type  
  make
* Two executables should be created, reflecting time-homogeneous and inhomogeneous simulations

Running:

To run in Terminal, type ./bdm or ./bdm-td and a simple version of the process should run. The following options are also given to specify input files, ancestor files, and output location.

* -in input file containing parameters shared by all clones (not required)
* -outoutput location (defaults to current directory)
* -anc ancestor file containing information about ancestors (not required)

A full command to run the program should look like:

mkdir results

./main –in ./inputfile.txt –anc ./ancestors.txt –out ./results/

Input File

The input file is a 2-column tab-delimited file containing the following arguments. A hashbang indicates a comment (line is ignored). The variable name is in the left column and the value is in the right. Defaults are provided to create a minimal model, so no inputs are required.

|  |  |  |
| --- | --- | --- |
| **tot\_life** | double | total lifetime of branching process |
| **max\_pop** | double | maximum population before ending process |
| **start\_time** | double | starting time (typically 0) |
| **ancestors** | int | number of individuals per ancestor clone (if ancestor file not provided) |
| **ancestor\_clones** | int | number of initial types |
| **num\_sims** | int | total number of simulations of the same process |
| **allow\_extinction** | boolean (1/0) | 1 if allow a simulation to go extinct |
| **detection\_threshold** | double | minimum proportion of the total population such that a clone is output. Otherwise its individuals are absorbed into the parent |
| **num\_samples (not implemented yet)** | int | number of samples to take |
| **sample\_size** | int | size of sample to take |
| **birth\_rate** | double > 0 | default birth rate |
| **death\_rate** | double > 0 | default death rate |
| **mutation\_prob** | double [0, 1] | default mutation probability for new clone |
| **trace\_ancestry** | bool (1/0) | should we track info on parent of each clone |
| **count\_alleles** | bool (1/0) | adds/subtracts and individual to allele\_count of individual and all ancestors |
| **FITNESS DISTRIBUTION PARAMETERS** | | |
| **alpha\_fitness** | double > 0 | exponential distribution parameter for positive side of fitness distribution |
| **beta\_fitness** | double > 0 | exponential distribution parameter for negative side of fitness distribution |
| **pass\_prob** | double [0, 1] | probability that additional fitness of new mutant is 0 |
| **upper\_fitness** | double | upper bound to fitness distribution |
| **lower\_fitness** | double ≤ upper\_fitness | lower bound to fitness distribution |
| **MUTATION DISTRIBUTION PARAMETERS** | | |
| **alpha\_mutation** | double > 0 | alpha parameter for Beta distribution for additional mutation probability in new mutant clone |
| **beta\_mutation** | double > 0 | beta parameter for Beta distribution for additional mutation probability in new mutant clone |
| **PUNCTUATED EQUILIBRIUM PARAMETERS** | | |
| **punctuated\_prob** | double [0, 1] | probability of mutation burst |
| **poisson\_param** | double > 0 | rate parameter for zero-truncated Poisson distribution number of mutations in burst |
| **punctuated\_fitness\_multiplier** | double | amount to multiply additional fitness by |
| **punctuated\_advantageous\_prob** | double [0, 1] | probability that burst affects birth rate instead of death rate |
| **EPISTATIC PARAMETERS** | | |
| **epistatic\_mutation\_threshold** | int > 0 | number of mutation required before burst in fitness due to epistasis |
| **epistatic\_multiplier** | double | amount to multiply fitness contribution in new clone by due to epistasis occurring |
| **birth\_function** | 1,2,3 | see below |
| **death\_function** | 1,2,3 | see below |
| **td\_birth\_params** | vector of doubles (space-delimited) | see below |
| **td\_death\_params** | vector of doubles (space-delimited) | see below |
| **B\_max** | 1 | max value of birth function |
| **D\_max** | 1 | max value of death function |

See example files for their use. To implement, in the terminal run

./main –in input.txt

Ancestor File

The ancestor file is a tab-delimited file with the same structure as the output. The first line contains variable names and each line contains information for a single clone to serve as an ancestor population. The only requirement for this file is a column containing the number of cells. If not provided, the program will look at the arguments **ancestors** and **ancestor\_clones** described above to run with nonunique clones. If those are not provided a default of a single ancestor individual is used. The following table describes the possible variables for the ancestor file. Some parameters below are included since they are present in the output and store information when continuing a previous simulation.

|  |  |  |
| --- | --- | --- |
| **unique\_id** |  | an id for the ancestor |
| **numcells** | int | the number of cells for the ancestor |
| **birthrate** | double > 0 | the birth rate |
| **deathrate** | double > 0 | the death rate |
| **mutprob** | double [0, 1] | the probability of initiating a new clone given a birth occurs |
| **initialtime** | double > 0 | the time the clone appeared (not used) |
| **subclone\_count** | int > 0 | number of descendant subclones(not used) |
| **num\_mut** | int > 0 | number of mutations from all ancestors (not used) |
| **driver\_count** | int > 0 | number of driver mutations (not used) |
| **is\_driver** | bool | 1 if the mutation that initiated this subclone is a driver |
| **TIME DEPENDENT PARAMETERS** | | |
| **birth\_function** | 0 = constant, 1 = linear,  2 = logistic | Function type |
| **death\_function** |
| **bf\_params** | list of parameters | Parameters for associated birth(death) function |
| **df\_params** |

**On the parameter sets for the birth or death functions:** The functions all take a clone’s birthrate as a scaling factor for the curve. The parameters should be listed the same regardless of which function is used in the form of “x1,x2,x3,x4,x5…” in the tab-delimeted file (see example). The curves are parameterized as follows:

Linear:

Logistic:

https://en.wikipedia.org/wiki/Generalised\_logistic\_function

**Examples**

Examples of input and ancestor files for both the time-homogeneous (bdm) and the time-inhomogeneous (bdm-td) are provided. The following is a complete step-by-step workthrough and results for a time-inhomogeneous process with 5 ancestors having different rates. The files are found in /examples/inhomogeneous/workthrough:

1. In Terminal, navigate to folder with the exectuables, and run the following command:

./main\_td –out ./examples/inhomogeneous/workthrough –in ./examples/inhomogeneous/workthrough/inputfile.txt –anc ./examples/inhomogeneous/workthrough/ancestors.txt

1. Three new files should appear with named clonedata.txt, sim\_stats.txt, timedata.txt. These contain information about all clones at the end of the simulations (clonedata.txt), information about the simulation (sim\_stats.txt), and time-course data about clone counts (timedata.txt). Example runs are provided from a previous run through with the prefix “ex\_”.
2. Open R and change code to point to source directory. View population information by importing data and running the provided function on your own.