SimTreeSDD: **Sim**ulating Phylogenetic **Tree**s Under **S**tate-**D**ependent **D**iversification

Emma E. Goldberg
`eeg@uic.edu`

September 3, 2010

Contents

Background

Preliminaries

Branching processes are often used as macroevolutionary null models or for fitting empirical data. In the process considered here, each lineage (species or other taxonomic level) at each time has a specified probability of giving rise to a new lineage (speciation/origination/birth), going extinct (extinction/death), or changing its state from one value of a character to another. This is "state-dependent diversification" when the value of the character affects the probabilities of speciation and extinction.

The purpose of this simulator is to produce trees under state-dependent diversification processes. Probabilities of birth, death, and character transition are assumed to be constant over time and across taxa in this implementation. The two processes available now are BiSSE (binary state speciation and extinction; Maddison, Midford & Otto (2007) SystBiol) and GeoSSE (geographic state speciation and extinction; Goldberg, Lancaster & Ree (in review) SystBiol).

As may be clear already, some terms are used interchangeably here:

lineage = species = taxon
birth = speciation = origination
death = extinction
trait value = (character) state

Methods

Two types of characters are possible here. First, for BiSSE, you can have a single binary character. For example, eyes may be present or absent, or breeding system may be self-compatible or self-incompatible. In this first case, there are (obviously) two possible states. Second, for GeoSSE, the character in question can be geographic location. In the two-region model implemented here, there are three possible states: a species may be present in both regions, it may be present only in the first region, or it may be present only in the second region. Transitions between these states occur through dispersal (a species present only in one region that disperses becomes present in both regions) or local extinction (a species present in both regions that went extinct in the first region becomes present only in the second region). Character state changes can also happen during speciation events if the parent lineage is in both regions.

Each tree is simulated beginning with a single lineage (GeoSSE) or a branching point (BiSSE) at an initial time and ending at a specified final time (or before then, if the tree goes extinct). Six rates must be specified for the branching process. For a binary character, these rates are

speciation in state 0
speciation in state 1
extinction in state 0
extinction in state 1
transition from state 0 to state 1
transition from state 1 to state 0

and for a geographic character, these rates are

speciation in region A
speciation in region B
extinction in region A
extinction in region B
dispersal from region A to region B
dispersal from region B to region A

A lineage present in both regions (AB) is subject to speciation in A (daughters are AB and A) or B (daughters are AB and B), or additionally, a between-region speciation rate may be given (daughters are A and B).

The simulations are carried out in continuous time, and the results can be written to a variety of output file types. The program is called from the command line, so it is easy to automate it to produce many trees organized however you like. It is quite fast: performance obviously depends on hardware and parameter values, but I have never waited more than a couple seconds for a batch of thousands of reasonably large trees.

How to use the program

`SimTreeSDD` is written in C, and the program is command line-based, so the instructions below assume you are working from a terminal (e.g. `xterm` in Linux or `Terminal` in Mac OS X). There is no reason why it shouldn't work under Windows, but I haven't tried—if you have, please send me a howto and I will incorporate it here.

Commands you should issue in the terminal are displayed like this:

```
type this command
then type this command
```

and the contents of input or output files are displayed like this:

File contents: `example.txt`

```
this is the first line of the file
this is the next line
```

Installation

`SimTreeSDD` is available as C source code, so first download `SimTreeSDD_xxx.tar.gz` from my website (currently `http://www.uic.edu/~eeg`; `xxx` is the date). Put this archive wherever you like and navigate there in the terminal. Then, unpack the files and compile the code:

```
tar zxvf SimTreeSDD_xxx.tar.gz
cd SimTreeSDD_xxx/src/
make
```

This will create an executable called `SimTreeSDD`. You may want to copy this to somewhere in your path, e.g.

```
cp SimTreeSDD /usr/local/bin/
```

If you don't do this, then in the examples below, replace `SimTreeSDD` with the relative or full path, which might be something like ~`yourname/SimTreeSDD_xxx/src/SimTreeSDD`.

Input options

`SimTreeSDD` can be told what to do (i.e., what parameter values to use in the simulation and what output to write) in two ways. First, you can use an input file, and second, you can pass options on the command line.

Input file

An input file to `SimTreeSDD` consists of lines of the form `option = value`. The comment character is #, so all text that follows a # on a line is ignored. A minimum input file might look like this:

File contents: `simple_sim_input.dat`

```
trait_type = character # for a binary character
```

```
        birth0 = 0.5
        birth1 = 0.3
        death0 = 0.2
        death1 = 0.2
        alpha  = 0.4
        beta   = 0.1
        end_t  = 5
```

To simulate a tree with these values, type

```
        SimTreeSDD simple_sim_input.dat
```

(giving the path to `SimTreeSDD` if necessary).

A sample input file with more options and comments is included as `misc/params_SimTreeSDD.dat`. Here is a complete list of options and their possible values:

`trait_type` the type of character to model
- `= character` a single binary character
- `= region` geographic location in a two-region system

`birth0` the speciation rate for trait 0 or region A
- `=` any real non-negative number

`birth1` the speciation rate for trait 1 or region B
- `=` any real non-negative number

`birthAB` the between-region speciation rate (only for `trait_type=region`)
- `=` any real non-negative number [default is 0]

`death0` the extinction rate for trait 0 or region A
- `=` any real non-negative number

`death1` the extinction rate for trait 1 or region B
- `=` any real non-negative number

`alpha` the transition rate from trait 0 to 1, or the dispersal rate from region A to B
- `=` any real non-negative number

`beta` the transition rate from trait 1 to 0, or the dispersal rate from region B to A
- `=` any real non-negative number

`root_state` the state of the initial node or lineage
- `= -1` for `trait_type=character`, draw the root state from the stationary distribution [default]
- `= 0` for `trait_type=character`, root state is 0;
  for `trait_type=region`, root state is present in both regions [default]
- `= 1` for `trait_type=character`, root state is 1;
  for `trait_type=region`, root state is present only in region A
- `= 2` for `trait_type=region`, root state is present only in region B

`min_tips` the minimum number of tips a tree must have to be kept as a successful run; you can set this to whatever seems convenient, but keep in mind that a large value here may introduce bias in some analyses
- `=` any non-negative integer [default is `0`]

`min_two_states` whether a tree must have more than one tip state represented to be kept as a successful run
- `= 0` no, keep trees with whatever tip states [default]
- `= 1` yes, must have at least two states among the tips

`num_trees` the number of (successful) trees to simulate
- `=` any non-negative integer [default is `1`]

`file_prefix` a prefix to give output files
- `=` any text string [default is `run`]; this can include a path, e.g., `sim-runs/run1`

`num_start` the number with which to start labeling output files (if `num_trees` is greater than 1); for example, you could produce trees named run-1.tre, run-2.tre, run-3.tre
  `=` any non-negative integer [default is `1`]

`verbosity` how much information to print to the terminal as the program is running
  `= 0` print nothing
  `= 1` print some stuff
  `= 2` print lots [default]

`write_newick` create an output file containing the simulated tree in Newick format
  `= 0` don't create a `.tre` file for each tree [default]
  `= 1` do create a `.tre` file for each tree

`write_nexus` create an output file containing the simulated tree and character states in NEXUS format
  `= 0` don't create a `.nexus` file for each tree
  `= 1` do create a `.nexus` file for each tree [default]

`write_bmstrait` create an output file containing the tip character states in a format useable with BayesMultiState
  `= 0` don't create a `.bmstrait` file for each tree [default]
  `= 1` do create a `.bmstrait` file for each tree

`write_ttn` create an output file containing the simulated tree and character states in TTN format (see below)
  `= 0` don't create a `.ttn` file for each tree [default]
  `= 1` do create a `.ttn` file for each tree

Command-line options

Any option that can be given in the input file can also be given on the command line. This is sometimes handy when calling many runs of the program from a script. If an option is specified in both places, the value on the command line takes precedence. Options are separated by spaces. Spaces can *not* be used around the = sign on the command line. An input file *must* be given as the *first* argument. Here is an example in which the values in `simple_sim_input.dat` (above) are used but `end_t` is set to 2.5 instead and 10 trees are created:

```
SimTreeSDD simple_sim_input.dat end_t=2.5 num_trees=10
```

Output options

Depending on what you want to do with your simulated trees, you may want the output written in various forms. For viewing trees with, for example, TreeViewX, FigTree, or Dendroscope, set `write_newick=1`. The tree string written contains tip and node labels showing the character states. Here is an example file for a small, 4-tip tree (but note that this is a single line in the real file):

File contents: `small_tree.tre`

```
(((0_tip0:0.051603,0_tip1:0.051603)0_n4:0.002604,0_tip2:0.054207)0_n5:
0.045793,1_tip3:0.100000)1_n6:0.000000;
```

For use with Mesquite, set `write_nexus=1`. For use with HyPhy, set `write_newick=1` and `write_nexus=1`. For use with BayesMultiState, set `write_nexus=1` and `write_bmstrait=1`.

For use with R, you can pick any of the output formats. I like to use my own TTN format (`write_ttn=1`), which consists of the tree as a Newick string on the first line, then a list of the tips and their trait values, and then (optionally) a downpass-ordered list of the nodes and their trait values. So TTN stands for Tree, Tips, Nodes. I've included R functions for reading, writing, and displaying TTN files in `misc/ttn.R`. I might eventually create an interface for calling `SimTreeSDD` through R, to facilitate use with the BiSSE and GeoSSE parameter estimation code which is currently in the `diversitree` package. Here is an example TTN file for the same small tree as above:

File contents: `small_tree.ttn`

```
(((0:0.051603,1:0.051603)4:0.002604,2:0.054207)5:0.045793,3:0.100000)6:0.000000;
0 0
1 0
2 0
3 1
4 0
5 0
6 1
```

Large tree warning

It is rather easy to choose parameter values that lead to extremely large trees. This can cause trouble in (at least) three ways. First, output files may be unwieldy for other programs to handle. Second, `SimTreeSDD` itself may require a lot of memory, bogging down your computer. To avoid this, `SimTreeSDD` will abort itself if a tree reaches 50,000 nodes. This is an arbitrary cutoff, and if you need it to take a different value you can change the value of `TOO_BIG` in `src/build.h`. Third, the tree-building algorithm uses heavy recursion, so it is possible to get so deep in recursive layers as to exceed your stack size. `SimTreeSDD` can not check for this problem and so just dies with a segmentation fault. Sorry about that, but it should not be an issue for reasonable values of the `birth*`, `death*`, and `end_t` parameters. If you have this problem and really need to use whatever extreme parameter values you have set, you can increase the stack size with `ulimit` or `limit`—consult the documentation for your shell.

Procedural stuff

Bugs and help

This code has been tested and used by me and others since 2006 (some of it even dates back to John Huelsenbeck's 2003 Phylogenetic class at UC San Diego). The input interface (`src/keyvalue.*`) was written by Walter Brisken. As far as I know everything works as it should, but of course, any software may have bugs, so if you encounter an unexplainable crash or a result that seems incorrect, please let me know. I would also be happy to hear if you have usability suggestions (e.g., making the documentation clearer or error-checking during use of the program), ideas for additional features that would be useful, or if you are making substantial changes to take the code in a new direction.

License

This software is issued under the Gnu Public License (http://www.gnu.org/licenses/gpl.html). The full text of the license is included with the code, but the gist of it is (1) you can do whatever you want with this code and modify it however you like, and (2) if you redistribute this code or your modified version, it must also be under the GPL.