DIFFpop

Jeremy Ferlic, Jiantao Shi, Thomas O. McDonald, and Franziska Michor

DIFFpop

DIFFpop is an R-wrapped C++ package to simulate cell labeling experiments performed on differentiation hierarchies.

Dependencies

- GNU Scientific Library
  - (OSX) brew install gsl with Homebrew or from (http://ftpmirror.gnu.org/gsl/).
  - (Windows) download and extract the file local###.zip and create an environmental variable LIB_GSL to add the directory (see notes about Windows installation below for more details).
  - (Linux) install libgsl0-dev and gsl-bin.
- Rtools (Windows only)
- devtools

Important notes about Windows installation

Rtools contains the necessary resources to compile C++ files when installing packages in R. GSL is also required which can be downloaded from (http://www.stats.ox.ac.uk/pub/Rtools/goodies/multilib/local323.zip). After downloading, unzip to your R directory. Depending on if your computer is 32 or 64-bit, move the library files from local###/lib/i386 (32-bit) or local###/lib/x64 (64-bit) to local###/lib.

To set the environmental variable LIB_GSL on a Windows 7/10 computer, go to "Advanced system settings" in Control Panel > System and Security > System and click Environmental Variables. Create a new system variable with

- Variable Name: LIB_GSL
- Variable Value: "C:/path/to/local323" (include quotes)

Recommended R packages

- igraph
- ggplot2
- R.utils

Installation

To install in R, type:

```
install.packages("devtools")
devtools::install_git("https://github.com/ferlicjl/diffpop.git")
```

Uses

DIFFpop is an R package that uses C++ to simulate the maturation process of cells through user-defined differentiation hierarchies according to user-defined event rates. The software is capable of simulating hierarchies in two manners: using a branching process with growing populations and using a multitype Moran process with fixed population sizes.

For simulating populations with changing sizes, the system will grow or decline as a branching process according to the exact rates specified by the user. If at any point any population becomes extinct, the simulation halts. A branching process is a stochastic process used to model the growth and composition of reproducing populations. Assumptions made in branching processes are that individuals live for a random amount of time before some event. Here, we only consider events that are analogous to cellular differentiation events, i.e. mitosis, differentiation, and apoptosis. Individuals of the same type are independent and identically distributed. Simulation of the branching process is an application of the direct Gillespie Stochastic Simulation Algorithm.

Simulations with fixed population sizes are modeled using a multitype Moran Process. A traditional Moran Process consists of a population of cells, each belonging to a certain class and having an individual fitness. In each time step, a random cell is selected for division and a random cell is selected for apotosis, ensuring that the population size remains constant. To introduce selection, cells with higher fitness values are more likely to be selected for division. To modify the traditional Moran Process and place it in the setting of differentiation, we included additional cellular events corresponding to differentiation. By coupling events together, such as enacting a differentiation event following a mitosis event, the population size is maintained strictly constant. Simulation of this multitype Moran Process is carried out using tau-leaping, where the total number of expected events for one time unit is calculated by multiplying the sum of the events rates by the total number of cells and then that number of events is enacted.

Applications

Binary Cell Labeling

DIFFpop is capable of tracking the uptake and progression of a binary label throughout a differentiation hierarchy. In experimental settings, these are often fluorescent labels that allow the researchers to sort samples into cell types and then quantify proportion of cells that express the label.

Simulating a binary labeling scheme in DIFFpop can be achieved in two ways. In the first, the user manually enters the number of initial cells that belong to the unlabeled and labeled populations. In the second, the user specifies with what probability a cell will gain the label upon simulation initiation. The proportion of labeled cells can be tracked over time in the label and census output file.

As an example of applying DIFFpop to this type of data, we have included a vignette replicating the results from introducing a yellow fluorescent protein (YFP) reporter into the hematopoietic cells of the bone marrow (Busch et al., Nature 2015). After validating that our stochastic simulations closely match the results from the in-vivo experiments, we could further investigate the system using DIFFpop, including inferring hematopoietic clonal dynamics from the system if the investigators introduced unique barcode labeling.

Confetti-style Labeling

A confetti-style labeling scheme is one in which one particular label from amongst a series of possible labels is expressed through random segregration and reintegration into the host cells genome. As an example system, 4 possible colored reports, labeled green, blue, yellow, and red, are added side-by-side in the host cell's genome. Upon induction by CRE recombinase, these label sections are random spliced out of the genome

and reintegrated, with ultimately only one label color being expressed. This same label is expressed in all daughter cells and can be traced as cells replicate and differentiate.

Simulating a confetti-style labeling scheme can be achieved in DIFFpop by simply specifying the initial number of cells to express each particular label. The census files can then be analyzed upon simulation completion to track the changes in label expression throughout the system over time.

As an example of this type of experimental procedure, we point the reader to a confetti-style labeling scheme implemented in the hematopoietic system of mice (Ganuza et al., Nat Cell Biol. 2017). Such a labeling experiment could be easily simulated using DIFFpop assuming the proper population sizes and transition rates were known.

Unique Cell Barcoding

In addition to simulating fluorescent cell labels, DIFFpop can also be used in combination with unique cell barcoding experiments. Unique cell labeling can be achieved by introducing a mobile transposon into the genome. Upon induction of labeling with tamoxifen, this transposon is spliced from the genome, and randomly reintegrated at some point in the host genome. Assuming the probability that the transposon randomly integrating into the same location in two cells is negligible, each cell now contains the transposon in a unique genomic position. The transposon in this location will then be passed to all offspring cells and be maintained through replication and differentiation events. At the end of an experiment, cell populations can be sorted and then sequenced for the presence or absence of these barcodes. Alternatively, a sample of cells can be sent off for single cell sequencing, allowing for not only the presence or absence of a particular barcode, but also an estimate of the size of a particular barcode-defined clone.

Simulating unique cell barcoding in DIFFpop can easily be achieved by simply specifying the proportion of cells to be successfully labeling upon system initialization. The census files can then be analyzed upon simulation completion to track the barcode frequencies in the system over time. One can even simulate a single cell barcoding experiment by randomly sampling from the barcode population.

As an example of a unique barcoding population, we point the reader to an experimental procedure in which cells of the hematopoietic system are labeled in-vivo and analyzed in native hematopoiesis, not requiring the use of cell transplantation (Rodriguez-Fraticelli et al., Nature 2018).

Using DIFFpop in R

The first step to utilize DIFFpop for simulation of a differentiaton hierarchy is to specify the populations of the hierarchy. Populations of cells are created using functions that correspond to a specific software class, i.e. GrowingPop, FixedPop, or DiffTriangle. Users must give each population a unique name as well as an initial population size. Optionally, users may specify an initial cell barcoding frequency, which represents the proportion of initial cells that receive a unique barcode. If this parameter is not set, no unique barcodes will be created for the population.

The next step is to specify the transitions between populations. For that purpose, the addEdge function is used, along with the correct parameters: the initiating population, the receiving population, event type as a string (either “alpha”, “beta”, “gamma1”, “gamma2”, “delta”, “zeta”, or “mu”), and event rate. For events involving only one population (“alpha”, “delta”, or “mu”), users set that population as both the initiating and receiving population.

The last specification step is to specify which population is the root of the differentiation hierarchy, that is, which population is the furthest upstream, using the setRoot function in R.

The simulateTree function is then used to initiate the simulation.

Simulation parameters

Table 4 describes the parameters of the simulation.

Table 1: Description of simulation parameters

Parameter	Variable Type	Description
tree	DiffTree	specifies which hierarchy to simulate
fixed	boolean	TRUE if simulating using FixedPops and DiffTriangles, FALSE if simulating using GrowingPops
time	integer	number of time units to simulate
census	integer	how often to output full census of populations
indir	string	directory location of input files
outdir	string	directory location for output files
seed	numeric	optional seed for the random number generator

Observations are made and output files updated at every integer time unit through *nObs*. In addition, full outputs of the cell states in each population are made every *census* time unit(s). The *indir* directory informs the C++ backend where the input files for the differentiation hierarchy are located and *outdir* specifies a particular directory in which to place all output files. Optionally, the user can specify a numeric *seed* for the GSL random number generator used throughout the simulation.

Birth-Death Example

Towards learning how to utilize DIFFpop to simulate cellular differentiation, we present the following birth-death process. Using this simple example as a starting-off point, we will then show how by the addition of relatively few lines, this model can be modified and expanded.

The following script will create the example.

```
library(diffpop)

# Create an empty DiffTree object
tree1 = DiffTree()

# Create a population "pop1" with 100 unlabeled cells
GrowingPop(tree = tree1, name = "pop1", size = 100, label = 0.0)

# Add cell birth event to pop1
addEdge(tree = tree1, parent = "pop1", child = "pop1", type = "alpha", rate = 0.5)
# Add cell death event to pop1
addEdge(tree = tree1, parent = "pop1", child = "pop1", type = "delta", rate = 0.3)

# Set the root of tree1 to "pop1"
setRoot(tree = tree1, popName = "pop1")

# Simulate tree1 for 10 time units writing results to ./output/
simulateTree(tree = tree1, fixed = FALSE, time = 20,
             indir = "./input/", outdir = "./output/")
```

After installing DIFFpop using the steps outlined above, the library must first be loaded into the current R session. In order to begin creating a differentiation hierarchy, we must create an empty DiffTree object we wil call “tree1”. We then begin by creating a single population named “pop1”, and initializing it to contain

100 unlabeled cells, and add it to the tree. Because we want to model a population whose size will fluctuate over time, we will use a GrowingPop to model this population.

We will then add transitions to this population. We will start by adding a self-renewal event, which will occur at a rate of 0.5 events per cell per time unit. We will also add the death event, which will occur at a rate of 0.3 events per cell per time unit.

The last remaining steps are to set "pop1" as the root of the differentiation tree and start our simulation. We will simulate this tree for 20 units of time, writing all of the inputs to the input folder and all output files to the output folder. We will also set the simulation parameter fixed to FALSE, notifying DIFFpop we are modeling the system using GrowingPops.

Before expanding our basic process, let us take a look at some of the output files from this simulation. Each simulation is given a unique file prefix. For example, the output files from the last run were all prefixed with "out_11-11-2018-202313_54045", letting us know the simulation was initiated at 8:23 PM on November 11, 2018. Because we had completely unlabelled cells and did not allow for mutation, the only system statistics of interest are the population sizes, which are output to the "out_11-11-2018-202313_54045_pop.csv" file each time unit. Let's plot the population size of "pop1" over the course of the simulation.

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.5.1
```

```
popfiles = list.files("./output/", pattern="^out.*_pop.csv$", full.names=T)

# Read in population sizes file
pop = read.csv(popfiles[length(popfiles)])

# Plot the population size of "pop1" vs. simulation
plot(pop$time, pop$pop1, xlab = "Time", ylab = "Population Size",
      main = "Simple Birth-Death Process")
```

Extending the Birth-Death Example

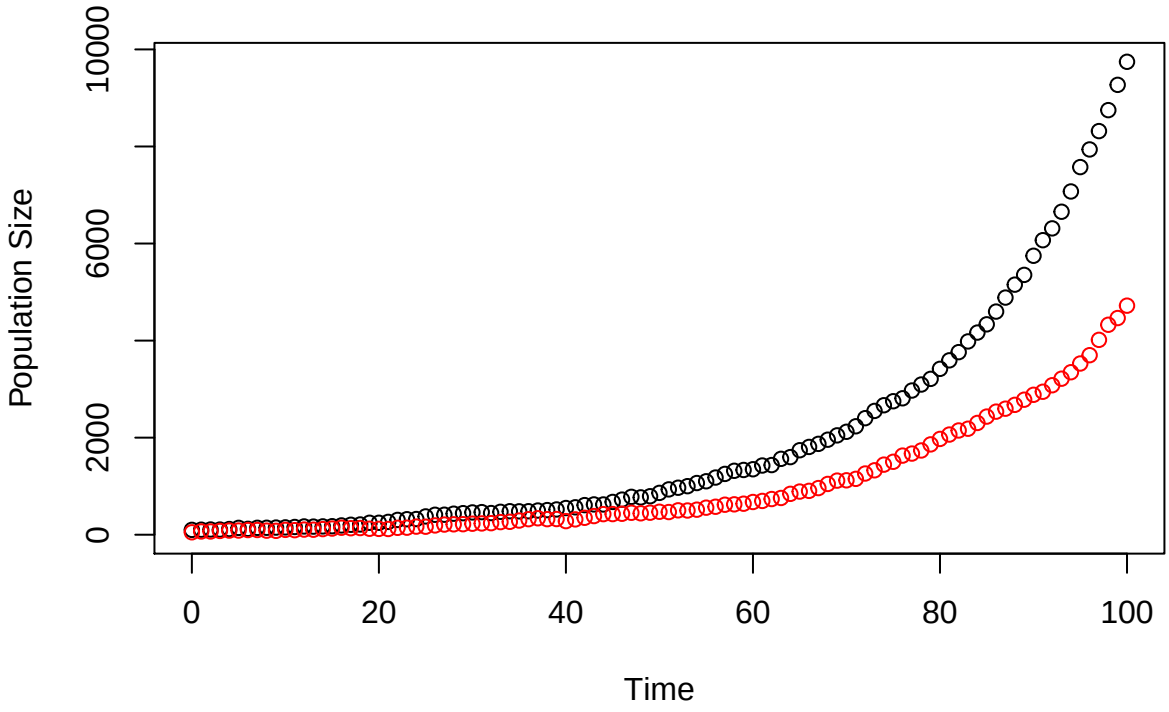
In our next example, let us look at exploring some other features of DIFFpop beyond a simple birth-death process by labeling 50% of cells in "pop1", adding an additional population "pop2", adding differentiation from "pop1" to "pop2", and allowing for mutations to occur in "pop1". We can store this updated tree as "tree2". Whenever a mutation occurs, we will introduce a new fitness change to be drawn from a standard normal distribution. Below is the entire script needed to simulate this model.

```
library(diffpop)

# Create an empty DiffTree object
tree2 = DiffTree()

# Create two populations, labeling on average 50% of cells in pop1
GrowingPop(tree = tree2, name = "pop1", size = 100, label = 0.50)
GrowingPop(tree = tree2, name = "pop2", size = 50, label = 0.0)

# Add cell birth/death events to pop1
addEdge(tree = tree2, parent = "pop1", child = "pop1", type = "alpha", rate = 0.4)
addEdge(tree = tree2, parent = "pop1", child = "pop1", type = "delta", rate = 0.3)

# Add cell birth/death event to pop2
addEdge(tree = tree2, parent = "pop2", child = "pop2", type = "alpha", rate = 0.35)
addEdge(tree = tree2, parent = "pop2", child = "pop2", type = "delta", rate = 0.4)
```

```
# Add differentiation from pop1 to pop2
addEdge(tree = tree2, parent = "pop1", child = "pop2", type = "gamma1", rate = 0.05)

# Add mutation in pop1, occurs during each mitosis event with probability 1e-4
addEdge(tree = tree2, parent = "pop1", child = "pop1", type = "mu", rate = 1e-4)

# Set fitness change distribution when a new mutant arises
setFitnessDistribution(tree = tree2,
                       distribution = "normal",
                       alpha_fitness = 0,
                       beta_fitness = 1,
                       pass_prob = 0,
                       upper_fitness = NA,
                       lower_fitness = 0)

# Set the root of tree1 to "pop1"
setRoot(tree = tree2, popName = "pop1")

# Simulate tree1 for 50 time units writing results to ./output2/
simulateTree(tree = tree2, fixed = FALSE, time = 100,
             indir = "./input2/",  outdir = "./output2/")
```

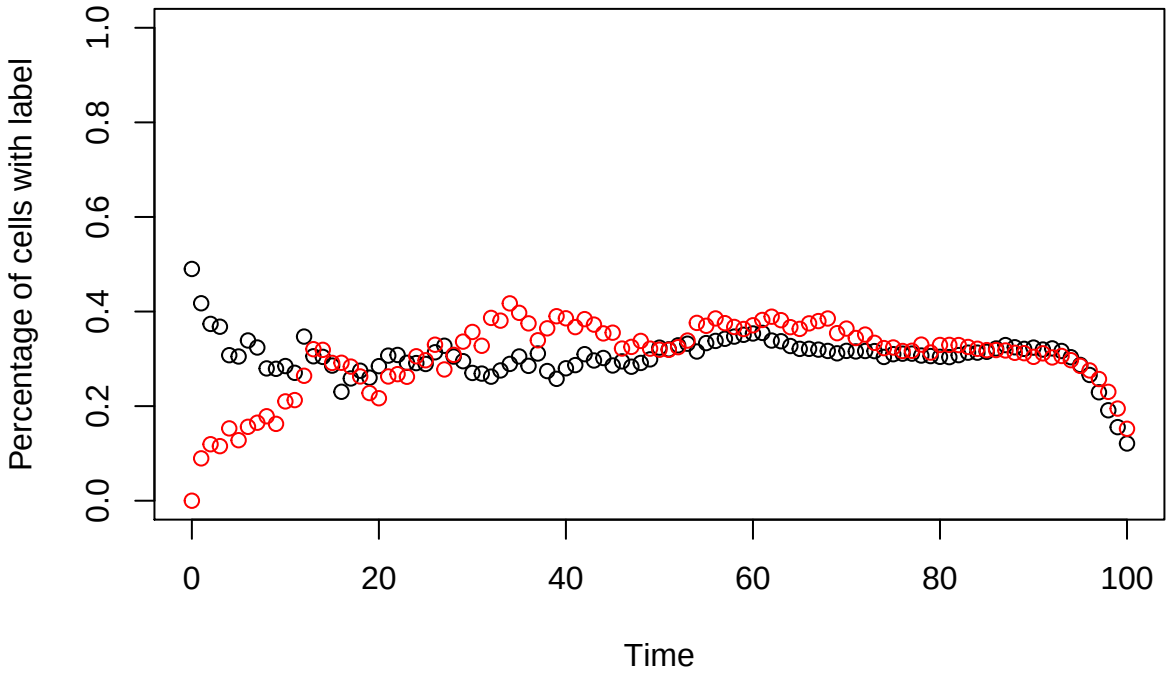
Let us explore some of the output files from this simulation run. First, let us once again start with a plot of the population sizes over time.

```
popfiles = list.files("./output2/", pattern="^out.*_pop.csv$", full.names=T)

# Read in population sizes file
pop = read.csv(popfiles[length(popfiles)])

# Plot the population size of "pop1" vs. simulation time
plot(pop$time, pop$pop1, col = "black",
     xlab = "Time", ylab = "Population Size", ylim = c(0, max(pop)))
points(pop$time, pop$pop2, col = "red")
```



We can also look at how quickly the label, which was only initialized in pop1, was taken up in pop2. To do this, we will plot from the label output file, which shows the percentage of cells in the population that contain the label.

```
# Read in label file
lblfiles = list.files("./output2/", pattern="^out.*_label.csv$", full.names=T)

# Read in label file
label = read.csv(lblfiles[length(lblfiles)])

# Plot the label percentage  vs. simulation time
plot(label$time, label$pop1, col = "black",
     xlab = "Time", ylab = "Percentage of cells with label", ylim = c(0, 1))
points(label$time, label$pop2, col = "red")
```



We can view the mutation output file to see which mutations occurred, in which populations they occurred, and the resulting fitnesses of those clone.

```
# Read in label file
mutfiles = list.files("./output2/", pattern="^out.*_mut.csv$", full.names=T)

# Read in population sizes file
mut = read.csv(mutfiles[length(mutfiles)])

head(mut)
```

```
##   mutant    time population  fitness
## 1      1 29.3029       pop1 0.996159
## 2      2 75.0166       pop1 0.736024
## 3      3 85.5805       pop1 1.724180
## 4      4 86.3273       pop1 1.025380
## 5      5 90.2656       pop1 3.590480
## 6      6 91.8170       pop1 2.597500
```

For more detailed examples, please set the vignettes.

Software design and class structures

The software classes implemented in DIFFpop are described below and summarized in Table 1.

GrowingPop

A GrowingPop is the class used to designate the various cell types throughout a differentiation tree. A GrowingPop contains a list of cell states, functions to enact cellular events on those cell states, and event rates at which to perform those functions. The hierarchical structure is maintained by pointers to upstream and downstream CellPopulations.

FixedPop

A FixedPop is a class derived from a GrowingPop. In order to maintain a constant population size, cellular events are coupled together; i.e. a mitosis event generating an additional cell is immediately followed by a differentiation or death event. Similarly, if the number of cells in the FixedPop population increases by one from upstream differentiation, a differentiation or death event of its own is immediately enacted to maintain the population level.

DiffTriangle

A DiffTriangle cell type is used to represent the downstream fully differentiated cells. Cells are arranged in a triangle formation. Cells enter the population on the highest level of the triangle, experiencing further differentiation and division to progress down the triangle. When a new cell enters the DiffTriangle population, it causes an already existing cell on the highest level to divide and further differentiate to the next level of the triangle. Those two cells each displace a existing cell, causing them to divide and differentiate, thus generating four newly displaced cells, which in turn displace cells that further displace cells until reaching the lowest level of the triangle. A displaced cell from the last row of the triangle can either be passed on to an offspring cell type if there are further cell types in the hierarchy or die out. Importantly, DiffTriangle structures will not initiate any cellular events of their own, as differentiation waves throughout a triangle are only initiated when receiving a new cell from an upstream population.

The population size of a DiffTriangle is specified by two parameters; the first, z, is the number of cell divisions until full maturation or the number of levels in the triangle. The second is the $mfactor$, which is the number of cells that exist in the first stage of maturation, i.e. the number of cells in the first level of the DiffTriangle. If $mfactor$ is greater than 1, then a cell entering the DiffTriangle population simply chooses at random which specific triangle to enter.

Table 2: Software classes implemented in DIFFpop

Class Type	Population Size	Use
GrowingPop	Dynamic	dynamically sized population with exponential event waiting times
FixedPop	Constant	constant size homogenous population
DiffTriangle	Constant	constant size population with z levels of maturity

NodeList

The cells of each population are maintained by a NodeList. A NodeList is a doubly-linked list of nodes. Each node keeps track of a particular cell state, defined by the combination of a barcode, mutation status, and fitness value, as well as a count of how many cells belong to that particular state. In addition, DiffTriangle nodes also contain data to record in which triangle and at which level a cell resides. Mutation status is a string listing which mutations have occurred in that state. Information about particular mutations can be found in the mutation information file. The NodeList keeps track of the total number of cells in the list as well as the sum of the fitness values for the cells. This sum of fitness values is used to weight a uniform[0,1)

random variable to select cells for mitosis by fitness. Individual cells may be inserted or removed from the list, maintaining a left-balanced list by cell count – that is, cell states with higher cell counts are found to the left of the list. Figure 1 shows an example of this left-balancing after the addition of cells to a particular state. This approach allows for more efficient indexing by cell count, particularly as diversity in the compartment decreases as dominant clones arise.

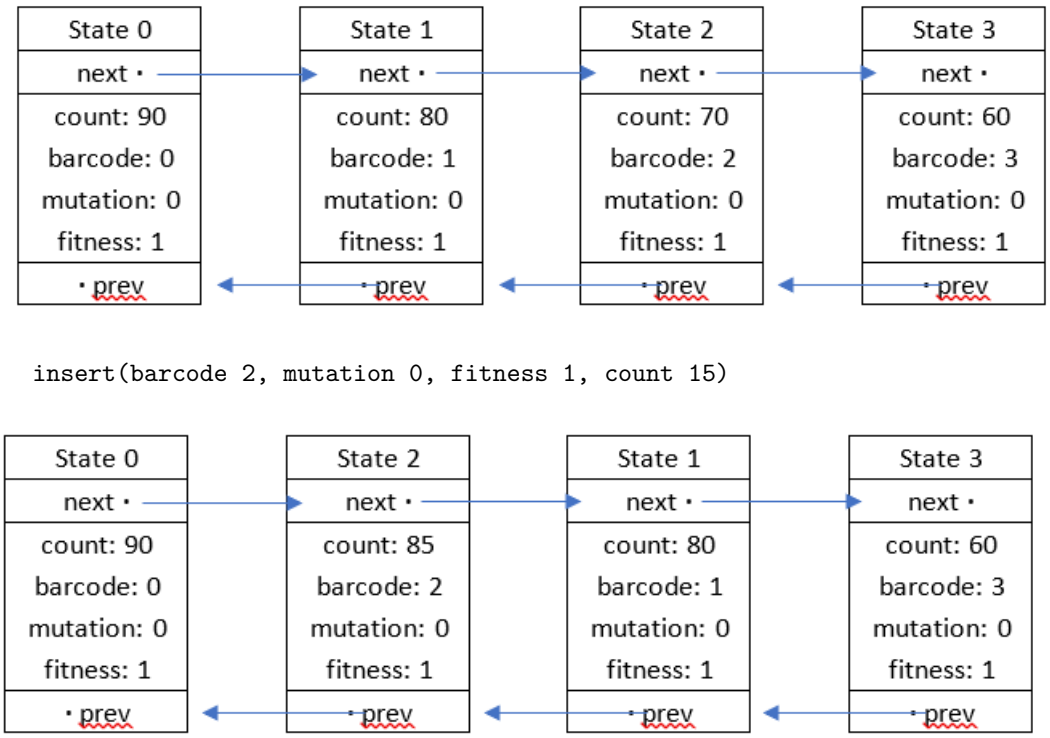


Figure 1: A NodeList maintains the order of states by count, allowing for faster indexing as dominant clones arise. This order is updated after every insertion or deletion from the list. Each node state consists of a unique combination of barcode, mutation history, and fitness. The number of cells in each state are kept as a count. Nodes are linked together to form a list using the pointers *next* and *prev*.

Event types and parameters

Cellular events in DIFFpop are enacted according to their accompanying parameter rates given in units of number of events per cell per time unit. Table 2 shows the events implemented in DIFFPop.

Table 3: Cellular events included in DIFFpop. The rate at which the specific event is enacted, in units of number of events per cell per time unit, is designated using lowercase Greek letters.

Rate parameter	Variable type	Description
α (alpha)	double	mitotic self-renewal rate
β (beta)	double	asymmetric differentiation rate to downstream cell type
γ_1 (gamma1)	double	mitosis-independent (one-to-one) differentiation rate to downstream cell type
γ_2 (gamma2)	double	mitosis-dependent (one-to-two) differentiation rate to downstream cell type
ζ (zeta)	double	de-differentiation rate to upstream cell type
δ (delta)	double	apoptosis rate
μ (mu)	double	probability of mutation per mitotic event

Note that population-specific parameters are specified using subscripts, i.e. $\alpha_{(LT-HSC)}$ is the mitotic self

renewal rate for the LT-HSC population and $\gamma_{1(LT-HSC,ST-HSC)}$ is the mitosis-independent differentiation rate from the LT-HSC population to the ST-HSC population.

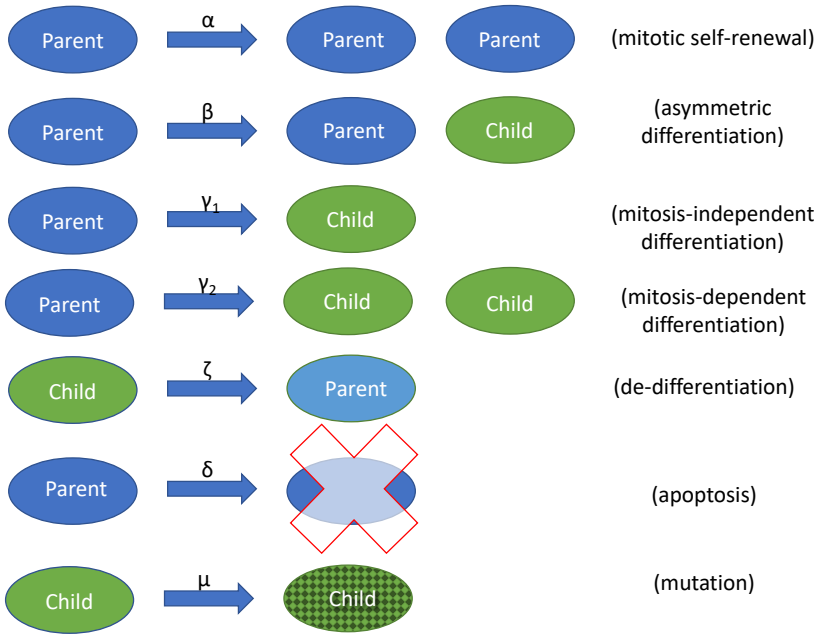


Figure 2: Events implemented in DIFFpop include apoptosis, asymmetric differentiation, mitosis-independent differentiation, mitosis-dependent differentiation, de-differentiation, apoptosis, and mutation. The lowercase Greek letter above each event describes the rate at which the event occurs in units of number of events per cell per time unit.

Events can be classified into three categories based on how they affect the population size of the compartment: "i-1" events resulting in a one-cell deficit include differentiation (γ_1/γ_2), de-differentiation (ζ), and apoptosis (δ); "i" events that maintain the population size include asymmetric differentiation (β); "i+1" events that result in a one-cell surplus include mitosis (α).

Mutations in DIFFpop occur only during mitosis events. Each mitosis event results in a new mutation with probability μ. Therefore in a general population A, the rate of mitosis events resulting in a new mutation is $\alpha_{(A)}\mu_{(A)}$ and the rate of mitosis events resulting in no mutation is $\alpha_{(A)}(1-\mu_{(A)})$. Mutations accumulate according to the infinite allele assumption, that is, every new mutation leads to a new allele that has yet to be seen in the population.

In the FixedPop setting, all de-differentiation events result in an apoptosis event in the upstream population. This consideration is necessary to avoid circular equations when calculating adjustments to net proliferation in order to maintain a constant population size.

Simulation overview

After the tree structure has been specified in R, the task of simulating is handed off to a C++ backend. Before events can be enacted, the tree structure and other user-specified parameters must be read from the input files, which are automatically generated by the R functions.

Initializing a simulation

1. Read tree structure from input files
2. Initialize populations
3. Write parameter information
4. Output tree structure to user

Simulation for branching process via the Gillespie Algorithm

1. Initiate simulation at time 0
2. Generate the time to the next event
    a. Iterate through population hierarchy, collecting total number of expected events from each population by multiplying the sum of all population event rates by the current population size
    b. Generate a time from exponential distribution with parameter equal to sum of event rates from all populations
3. Update output files with statistics on current system state
4. Update state
    a. Choose population to enact event by drawing from discrete distribution with probability equal to the total number of expected events calculated in 2a.
    b. Enact single event on that population
5. If simulation time remains, repeat from step 2

Simulation for multi-type Moran process via tau leaping

1. For each time unit:
    a. Iterate through tree in a breadth-first manner, i.e. starting with upstream populations
    b. For each population:
        i. Calculate expected number of i-1 and i events [γ_1, γ_2, δ, ζ, and β] = j for one time unit by multiplying the sum of those event rates by the current population size
        ii. Enact j events choosing events with probability proportional to their event rates. If event resulted in a one-cell deficit, enact a mitosis [α] event

Input files

The following input files are automatically generated in R after specifying the structure of the differentiation hierarchy and simulating. These files may then be edited manually or sent to collaborators for the purpose of simulating the same tree structure.

1. Population information (node.csv)
    - Population name, software class of population, root, initial barcode percentage, DiffTriangle height, DiffTriangle mfactor
2. Event information (edges.csv)
    - initiating population, receiving population, event type, event rate
3. Fitness change distribution (fitnessdist.txt)
    - Parameters of user-defined fitness distribution. A blank file means the default parameters will be used, meaning changes in fitness will be drawn from a $N(0, 1)$ distribution with 0 as lower limit on fitness for all clones.
4. Population order (bfs.txt)
    - Populations in the tree listed in breadth-first order, meaning that upstream populations are listed before downstream populations. Used to order populations for output files

Output files

Each run of the simulation is given a unique file prefix, consisting of the date and time when the simulation initiated, followed by a random integer. The following output files are updated throughout the course of a simulation.

1. Population size (*prefix*_pop.csv)
   - Size of each population at each observation time
2. Diversity indices (*prefix*_diversity.csv)
   - Shannon diversity and Simpson diversity are calculated at each observation time for each population along with the total number of barcode species present in each population at each obervation time
   - Shannon diversity $= -\sum_j p_j \ln p_j$, Simpson diversity $= \left(\sum_j p_j^2\right)^{-1}$, where p_j is the proportion of cells belonging to the jth clone.
3. Fraction of labelled cells (*prefix*_label.csv)
   - Fraction of cells that contain a unique barcode/label at each observation time for each population
4. Event rates (*prefix*_events.csv)
   - Number of events that occurred during the last unit of simulation time
5. Mutation summary (*prefix*_mut.csv)
   - Time of mutation, population in which the clone arose, and new fitness value for the clone
6. Parameter summary (*prefix*_params.csv)
   - Summary of parameters given by user, as well as recalculated event rates if using FixedPops
7. Census files (*prefix_population*_census.csv)
   - Complete census of cell states for each census time, including barcode, mutation history, fitness, and cell count
8. Done file (*prefix*.done)
   - File is created when simulation is complete. Alerts user that they can begin moving or manipulating other simulation result files

Maintaining a constant population size

In order to maintain a constant population size, a relationship must exist between the event rates of the compartment. Specifically, those event rates that result in an excess of cells in the compartment [i+1 rates] must be balanced with those event rates that result in a deficit of cells [i-1 rates]. Let $\alpha_{(x)}$ denote the mitotic self-renewal (α) rate of population x. Let $\gamma_{1(x,y)}$ denote the one-to-one differentiation (γ_1) rate from population x to population y. Let $\gamma_{2(x,y)}$ denote the one-to-two differentiation (γ_2) rate from population x to population y. Let $\beta_{(x,y)}$ denote the asymmetric differentiation (β) rate from population x to population y. Let $\zeta_{(x,y)}$ denote the one-to-one de-differentiation (ζ) rate from population x to population y. Let $\delta_{(x)}$ denote the cell death (δ) rate of population x. Let $n_{(x)}$ be the size of population x.

Then, for any compartment A, we have

$$n_{(A)}\alpha_{(A)} + \sum_{\text{pop } i \neq A} n_{(i)} \left(\gamma_{1(i,A)} + 2\gamma_{2(i,A)} + \beta_{(i,A)} + \zeta_{(i,A)}\right) = n_{(A)} \left[\delta_{(A)} + \sum_{\text{pop } i \neq A} \left(\gamma_{1(A,i)} + \gamma_{2(A,i)} + \zeta_{(A,i)}\right)\right] \tag{1}$$

and therefore,

$$\delta_{(A)} = \frac{n_{(A)}\left[\alpha_{(A)} - \sum_{\text{pop } i \neq A}\left(\gamma_{1(A,i)} + \gamma_{2(A,i)} + \zeta_{(A,i)}\right)\right] + \sum_{\text{pop } i \neq A} n_{(i)}\left(\gamma_{1(i,A)} + 2\gamma_{2(i,A)} + \beta_{(i,A)} + \zeta_{(i,A)}\right)}{n_{(A)}} \tag{2}$$

For the population size to remain constant, (1) must hold. That is, events that increase the population size (self-renewal and influx from other populations) must be balanced by events that decrease the population (cell death and differentiation). In the modified Moran Process, we force this to hold by automatically calculating delta for each population. If this calculated delta value is positive, we simply set the effective death rate equal to this value. If this calculated delta value is negative, we increase the alpha rate of the population by this value. Although DIFFpop uses (2) to adjust net proliferation, a user could use (1) to calculate the rate for any event given that the other event rates are known or can be estimated.

Fitness distribution

Throughout the differentiation hierarchy, whenever a new clone arises due to mutation, a change in fitness can be drawn from a random distribution. The parameters of that distribution can be specified by the user in R. Table 3 shows the parameters used to specify the fitness distribution.

Table 4: Description of parameters used to specify distribution from which changes in fitness are drawn

Parameter Name	Type	Description
fitness_distribution	string	"doubleexp", "normal", "uniform"
alpha_fitness	double	alpha parameter for fitness distribution
beta_fitness	double	beta parameter for fitness distribution
pass_prob	double	probability that mutation does not result in a fitness change
upper_fitness	double	upper bound on fitness
lower_fitness	double	lower bound on fitness

If the distribution function selected is normal, fitness additions are drawn from a $N(alpha\_fitness, beta\_fitness)$ distribution. If the distribution function selected is uniform, fitness additions are drawn from a $U(alpha\_fitness, beta\_fitness)$ distribution. If the distribution function selected is double exponential, alpha_fitness refers to the rate parameter of an exponential distribution for the positive range and beta_fitness refers to the rate parameter of an exponential distribution for the negative range.