# Introduction

MetaSMC is a simulator of shotgun sequencing reads from evolving microbial or tumour cell populations. MetaSMC simulates reads by first generating start and end positions of reads, then simulating sequence of local genealogies of reads, and finally impose mutations on genealogies to generate reads. The model for simulating local genealogies is Sequentially Markov Coalescent, and the models for simulating mutations include infinite-site model and various DNA substitution models, including Jukes-Cantor, K80, F81, HKY85, and Generalised time-reversible(GTR). MetaSMC supports all demographic models that supported by ms, as well as heterogeneous mutation model for different populations. MetaSMC provides a command-line program and library interface which can be used in more complex applications, such as metagenomics simulator.

# Installation

MetaSMC has to be installed on Unix-like environments such as GNU/Linux and FreeBSD. The following packages are required:

* Linear Algebra PACKage (LAPACK) for computing eigendecomposition of mutation matrix
* LAPACKE: C interface of LAPACK
* GNU make
* GNU C Complier (GCC) 4.8.3 or later

After downloading and extracting the package, compile the program by typing:

$ cd metasmc/  
$ ./configure  
$ make  
$ make install

If you wish to change installation path, replace second command by

$ ./configure --prefix=<installation\_path>

To remove installed programs, type

$ make uninstall

By default, MetaSMC use mt19937-64 to generate random numbers. To change random number generator, modify src/rand.c and modify the source file of mt19937-64 in src/Makefile.am. Seed of random number can be specified using **-s** switch (See below). If **-s** switch is not used, seed is automatically assigned using **time(NULL)**.

# File Formats

## Format of read profile

A read profile begin by three lines of header. The first line is the path to FASTA file containing reference genome. The second line consists of two integer separated by a ','. The first number is the length of reference genome and the second one is number of fragments n. The third line consists of a sequence of numbers. The first is an integer indicating number of subpopulations. If p > 1$, is followed by a sequence of real number seperated by ',', which indicates size of each subpopulation relative to subpopulation (Note that size of first subpopulation is always ).

The header is followed by lines describing fragments. Each line consists of fields separated by TAB characters. The first field is fragment id,the second is subpopulation number from which the fragment is sampled. The following two fields are start position and end position of this fragment. The next field is the number of reads contained by this fragment. This is followed by pairs of fields describing start and end position of each read.

The following is a profile of 15 reads. All fragments contains only one read with position the same as that of the fragment. All reads are sampled from the same subpopulation. The length of reference genome is .

reference.fasta  
25,15  
1  
1 0 8 16 1 8 16  
2 0 0 8 1 0 8  
3 0 8 16 1 8 16  
4 0 3 11 1 3 11  
5 0 0 8 1 0 8  
6 0 3 11 1 3 11  
7 0 8 16 1 8 16  
8 0 3 11 1 3 11  
9 0 0 8 1 0 8  
10 0 12 20 1 12 20  
11 0 12 20 1 12 20  
12 0 12 20 1 12 20  
13 0 17 25 1 17 25  
14 0 17 25 1 17 25  
15 0 17 25 1 17 25

The following is a profile of another 15 fragments. These reads are sampled from two subpopulations. The size of second subpopulation is half of first subpopulation. Fragment 7-11 are sampled from second subpopulation (subpopulation 1) and other reads are sampled from first subpopulation (subpopulation 0). The first 8 fragments are sequenced by two reads. For example, the first fragment contains two reads, one ranges from 8 to 14 and another ranges from 10 to 16.

reference.fasta  
25,15  
2,0.5  
1 0 8 16 2 8 14 10 16  
2 0 0 8 2 0 6 2 8  
3 0 8 16 2 8 14 10 16  
4 0 3 11 2 3 9 5 11   
5 0 0 8 2 0 8 2 8  
6 0 3 11 2 3 9 5 11  
7 1 8 16 2 8 14 10 16  
8 1 3 11 2 3 9 5 11  
9 1 0 8 1 0 8  
10 1 12 20 1 12 20  
11 1 12 20 1 12 20  
12 1 12 20 1 12 20  
13 0 17 25 1 17 25  
14 0 17 25 1 17 25  
15 0 17 25 1 17 25

## Output format

Output is a set of reads in FASTA format, as follows

>fragment\_number:<i/i> subpopulation, fragment\_start-fragment\_end, read\_start-read\_end  
TCATTGCGACAATGGG ...

# Usage of Command-Line Programs

## Attached Read Sampler

MetaSMC take as input a read profile describing start positions and end positions of reads. Our package contains a default read sampler which generates reads of fixed lengths. Basic command line of read sampler is

$ sampler nfrags [Options]

where **nfrags** is the number of fragments to be simulated.

### Options of Read Sampler

The following two switches are mandatory:

**-g fasta\_file**: Set path to FASTA file containing reference genome.

**-f fragment\_size**: Set fragment size.

The following switches are optional

**-n npop**: Number of populations. Default value is 1.

**-s size\_1 size\_2 ...**: Relative size of population 1, 2...

**-p**: If this switch is used, sampler will generate paired-end reads.

**-l read\_length**: If -p switch is used, user should specify read size using this switch. Otherwise, read length will equal to fragment length

## Coalescent Simulator

The basic command of coalescent simulator is

$ metasmc [Options]

### Options of Coalescent Simulator

**-i profile\_path**: Assign path to read profile. If this switch is not used, the input is read from stdandard input.

**-s seed**: Assign seed of pseudorandom number generator. If this switch is not used, default seed is generated using time(NULL).

**-F M**: Set the maximum number of fragments per read group.

**-t theta1,theta2,...**: per-locus mutation rate of populations that exists at time 0. The number of thetas are equal to either 1 or the number of populations that exists at time 0. If only 1 theta is assigned, the value is assigned to all populations that exist at time 0.

**-r rho**: Per-locus recombination rate.

**-T treefile**: Print Newick-formatted coalescent trees to **treefile**.

**-m model1,model2,...**: Assign DNA evolution models of population that exists at time 0, **model1** for population 1, **model2** for population 2, and so on. Available models are **JC69**, **K80**, **F81**, **HKY85**, **T92**, **TN93** and **GTR**. The description of these mutation models can be found in http://en.wikipedia.org/wiki/Models\_of\_DNA\_evolution. Infinite-site model is assumed if user do not specify this switch. Under infinite-site model, mutant allele is chosen uniformly at random from three possible nucleotides.

**-f freq1\_A,freq1\_C,freq1\_G,freq\_T;freq2\_A,freq2\_C,freq2\_G,freq2\_T;...**: Base frequencies for mutation model. Prefix **freq1** stands for base frequencies of population 1, **freq2** stand for base frequencies of population 2, and so on.

**-R par11,par12,...;par21,par22,...;...**: Set parameters of mutation model. The length of parameter list depends on mutation model. parij indicate j-th parameter of population i. The length of parameter list is 0 for JC69, 1 for K80, 0 for F81, 1 for HKY85, 1 for T92, 2 for TN93 and 6 for GTR. The physical meanings of parameters can be seen in http://en.wikipedia.org/wiki/Models\_of\_DNA\_evolution .

The following are used to specify demographic model. Some switches have effect on specific population which is designated by 0-based index. For example, the first population specified in read profile has index 0, the second population has index 1, and so on.

**-G alpha**: Set exponential growth rate of all populations to **alpha** at time 0

**-g i alpha**: Set exponential growth rate of population **i** that exist at time 0.

**-M m\_12,m\_13,...,m\_21,m\_23,...**: Migration matrix at time 0.

The following switches are used to set demographic events. *t* indicates the time when the event occurs.

**-es t i proportion**: Split population **i**. This generated a new population numbered according to the order appear in command line. For example, assume 2 population is specified in read profile and there are two another **-es** switch before current one, the newly generated population is numbered 4. **proportion** is probability that each lineage stays in population i.

**-ej t i j**: Join population i and population j without changing sizes and growth rates. Note that all lineages in population **j** are moved to population **i** after this event.

**-eG t alpha**: Set growth rate of all populations to **alpha**.

**-eg t i alpha\_i**: Modify growth rate of population **i**.

**-eM t mig\_rate**: Modify the mig matrix so all elements are **mig\_rate/(npop-1)**, where **npop** is the number of population at time .

**-em t i j m\_ij**: Set migration rate between populations **i** and **j** to **m\_ij**

**-eN t size\_all**: Modify all population sizes to **size\_all\*N0**, where **N0** is the size of population 0 at time 0.

**-en t i size\_i**: Set size of population **i** to **size\_i\*N0**.

## Example

To simulate 100 single-end reads from single population with per-base mutation rate 0.01, type

$ sampler 100 -g reference.fasta -f 500 >profile.txt  
$ metasmc -i profile.txt -t 0.01 >read.fasta

The desired reads can be found in read.fasta and all reads have length 500.

To simulate two populations with migration rate 0.1 from population 1 to 2 and 0.2 from 2 to 1, type

$ sampler 100 -g reference.fasta -f 500 -n 2 > profile.txt  
$ metasmc -i profile.txt -t 0.01 -M 0.1,0.2 >reads.fasta

If user wish to simulate two populations with different mutation models, type

$ sampler 100 -g reference.fasta -f 500 -n 2 > profile.txt  
$ metasmc -i profile.txt -t 0.01 -M 0.1,0.2 -t 0.01 -m JC69,GTR -R :0.1,1,0.2,1,2,4 >reads.fasta

In this example, both populations have mutation rate 0.01, but first population mutates according to Jukes-Cantor model whereas second population mutates according to Generalised time-reversible model. **-R** is used to assign model parameters and parameter lists for different populations are separated by ':'.

To simulate completely isolation of two populations at time 0.1, type

$ sampler 100 -g reference.fasta -f 500 -n 2 > profile.txt  
$ metasmc -i profile.txt -t 0.01 -M 0,0 -ej 0.1 0,1 >reads.fasta

An interesting application of MetaSMC is to simulate change of mutation rate. This can be achieved by combining heterogeneous mutation rate and population splitting:

$ sampler 100 -g reference.fasta -f 500 > profile.txt  
$ metasmc -i profile.txt -t 0.01,0.02 -m JC69 -es 0.1 0 0 >reads.fasta

Initially, there is only 1 population (population 0) with mutation rate 0.01. At time 0.01, a splitting event causes all lineages moving to population 1 (the population index is automatically assigned) with mutation rate 0.02. Since migration between two populations is disallowed, population 0 is essentially replaced by population 1.

# Library Interface

## Overview

This subsection gives an overview of the usage of the library interface. For detailed usage of each function, see API Reference. The library interface is defined in global.h, smc.h and mutation.h, which can be found in **<installation\_path>/include/**. To use library interface, programmer need to add **-lsmc** and **-L<installation\_path>/lib** in gcc command.

To simulate reads from a species, the first step is to load read profile. This can be done by calling **load\_profile**, which returns a object of type **struct profile**. Next, we need to set up configuration object by

struct config \*create\_config(struct profile \*, int)

*create\_config* returns configuration object that contains basic configuration without demographic events, migration matrix and growth rates. After calling **create\_config**, demographic events can be added using a set of functions. There are 8 types of demographic events described as follows:

* ggro: change growth rate of all populations
* grow: change growth rate of a specific population
* gmig: change migration rate between all pairs of populations
* rmig: change migration rate between a specific pair of populations
* gsiz: change size of all populations
* join: join two populations
* splt: split a population
* size: change size of a population

Events can be added by calling function **add\_event\_<event type>**. Migration matrix and growth rates can be added by

int set\_growth\_rates(struct config \*, double \*)

and

int set\_migration\_matrix(struct config \*, double \*)

If these functions are not used, all entries of migration matrix and growth rates are set to 0.

Next, we need to allocate genealogy object, which can be done by calling

struct genealogy \*alloc\_genealogy(struct config \*, struct profile \*)

Genealogy objects holds informations required during simulation process. After genealogy object is created, reads are simulated by calling

int simulate(struct genealogy \*, struct profile \*);.

Reads can be founded in profile object. See API Reference of **load\_profile** to see how to extract reads from profile object. If user wish to reuse genealogy object, call

void clear\_genealogy(struct genealogy \*)

to reinitalize genealogy object.

Profile objects, configuration objects and genealogy objects can be destroyed by calling

void destroy\_genealogy(struct genealogy \*);  
void destroy\_config(struct config \*);  
void unload\_profile(struct profile \*);

## API Reference

### NAME

**#include <global.h>**

**load\_profile**, **unload\_profile** - loading and unloading read profile

### SYNOPSIS

struct profile \*load\_profile(char \*filename)  
void unload\_profile(struct profile \*prof)

### DESCRIPTION

**load\_profile** loads read profile in path **filename** and returns profile objects. Profile objects can be destroyed by **unload\_profile**.

Profile object also holds reads after simulation. Reads can be found in **prof->fgset**, which is of type **struct frag** defined as follows:

struct frag{  
 int id;  
 double start;  
 double end;  
 int pop;  
 int nread;  
 struct read \*rd;  
};

This corresponds to fragment in read profile (See File Formats). **struct read** is defined as follows

struct read{  
 int start;  
 int end;  
 char \*seq;  
 char \*qual;  
};

This corresponds to actual read in read profile (See File Formats).

### RETURN VALUE

**load\_profile** returns a pointer to profile object of type **struct profile** and returns **NULL** on error.

### NAME

**#include <global.h>**

**create\_config**, **destroy\_config** - Handling Configuration Object

### SYNOPSIS

struct config \*create\_config(struct profile \*prof, int print\_tree)  
void destroy\_config(struct config \*cfg)

### DESCRIPTION

**prof**: Pointer to profile object

**print\_tree**: A flag which is set to 1 if the programmer wants to output tree.

**cfg**: Pointer to configuration object.

### RETURN VALUE

**create\_config** returns pointer to configuration object of type **struct config** and returns **NULL** on error.

### NAME

**#include <global.h>**

**#include <mutation.h>**

**add\_event\_ggro**, **add\_event\_grow**, **add\_event\_gmig**, **add\_event\_rmig**, **add\_event\_gsiz**, **add\_event\_join**, **add\_event\_splt**, **add\_event\_size** - adding events to configuration object

### SYNOPSIS

int add\_event\_ggro(struct config \*cfg, double t, double alpha);  
int add\_event\_grow(struct config \*cfg, double t, int pop, double alpha);  
int add\_event\_gmig(struct config \*cfg, double t, double rmig);  
int add\_event\_rmig(struct config \*cfg, double t, int popi, int popj);  
int add\_event\_gsiz(struct config \*cfg, double t, double size);  
int add\_event\_join(struct config \*cfg, double t, int popi, int popj);  
int add\_event\_splt(struct config \*cfg, double t, int pop, double prop);  
int add\_event\_size(struct config \*cfg, double t, int pop, double size);

### DESCRIPTION

**cfg**: Pointer to configuration object.

**t**: Event time

**pop**: When the event involves only one population, **pop** is the index of the involved population.

**alpha**: Growth rate.

**popi**, **popj**: In the case of rmig event, **popi** is the source population and *popj* is destination population. In the case of join event, **popi** and **popj** are populations to be joined.

**rmig**: Migration rate between all pairs of populations.

**size**: Population size.

**prop**: Proportion of lineages that move to new population upon splt event.

### RETURN VALUE

All functions returns 0 on success and returns -1 on error.

### NAME

**#include <global.h>**

**set\_growth\_rates** - set growth rates at time 0

### SYNOPSIS

int set\_growth\_rates(struct config \*cfg, double \*grate)

### DESCRIPTION

**cfg**: Pointer to configuration object.

**grate**: If there is populations at time 0, **grate** is a vector of length that contains exponential growth rates of the populations.

### RETURN VALUE

**set\_growth\_rates** returns 0 on success and returns -1 on error.

### NAME

**#include <global.h>**

**set\_migration\_matrix** - set migration matrix at time 0

### SYNOPSIS

int set\_migration\_matrix(struct config \*cfg, double \*mmig)

### DESCRIPTION

**cfg**: Pointer to configuration object.

**mmig**: If there is populations at time 0, **mmig** is a matrix that contains migration rates between populations.

### RETURN VALUE

**set\_migration\_matrix** returns 0 on success and returns -1 on error.

### NAME

**#include <smc.h>**

**alloc\_genealogy**, **destroy\_genealogy**, **clear\_genealogy** - handling genealogy object

### SYNOPSIS

struct genealogy \*alloc\_genealogy(struct config \*cfg, struct profile \*prof)  
void destroy\_genealogy(struct genealogy \*G)  
void clear\_genealogy(struct genealogy \*G)

### DESCRIPTION

**cfg**: Pointer to configuration object.

**prof**: Pointer to profile object.

**G**: Pointer to genealogy object.

### RETURN VALUE

**alloc\_genealogy** returns pointer to object of type **struct genealogy** and returns **NULL** on error.

### NAME

**#include <global.h>**

**#include <mutation.h>**

**register\_mutation\_model** - Register mutation model to configuration object.

### SYNOPSIS

int register\_mutation\_model(struct config \*cfg, int pop, struct mutation \*mmut)

### DESCRIPTION

This function assigns mutation model **mmut** to population **pop**. Mutation model object is defined as follows:

struct mutation {  
 int model;  
 double theta;  
 double mpar[SQNUCS - NUM\_NUCS];  
 double pi[NUM\_NUCS];  
 double D[NUM\_NUCS];  
 double Cijk[NUM\_NUCS][NUM\_NUCS][NUM\_NUCS];  
};

Programmer need to specify **model**, **theta**, **mpar** and **pi**. **D** and **Cijk** can be computed by calling **init\_mutation\_model**. The value of **model** can be **MODEL\_JC69**, **MODEL\_K80**, **MODEL\_F81**, **MODEL\_HKY85**, **MODEL\_T92**, **MODEL\_TN93** and **MODEL\_GTR**. **theta** is new scaled mutation rate, **mpar** are parameter list whose length depends on mutation model, and **pi** is equilibrium allele frequency.

### RETURN VALUE

### NAME

**#include <global.h>**

**#include <mutation.h>**

**init\_mutation\_model** - initialize mutation model.

### SYNOPSIS

void init\_mutation\_model(struct mutation \*mmut)

### DESCRIPTION

This function computes **D** and **Cijk** in **struct mutation**, through eigenvalue decomposition. After calling this function, **Cijk** holds coefficient of where is rate matrix of mutation model, and holds eigenvalues of .

### RETURN VALUE