GemSIM:

General, Error Model based SIMulator of next-gen sequencing data

Introduction

GemSIM is a software package for generating realistic simulated next-generation sequencing reads. It can simulate reads from any sequencing technology with output in the formats FASTQ (raw reads) and SAM (aligned reads), including, for example, Illumina and Roche/454 data. GemSIM uses empirical error models and distributions calculated from real data, and can simulate both single- and paired-end reads from a single genome, collection of genomes, or set of related haplotypes. It can be used for deep sequencing, metagenomic, and resequencing projects. Its uses include assessing:

- Most appropriate technology and coverage for a sequencing project
- Aligner software and parameters
- SNP calling software, haplotype reconstruction, and evolutionary inference software
- Quality control

Installation

GemSIM is implemented in Python. It requires the freely available Python package NumPy. Download and installation instructions for NumPy are available at http://numpy.scipy.org/.

To generate run specific error models, users will also require an alignment in SAM format and the corresponding reference genome.

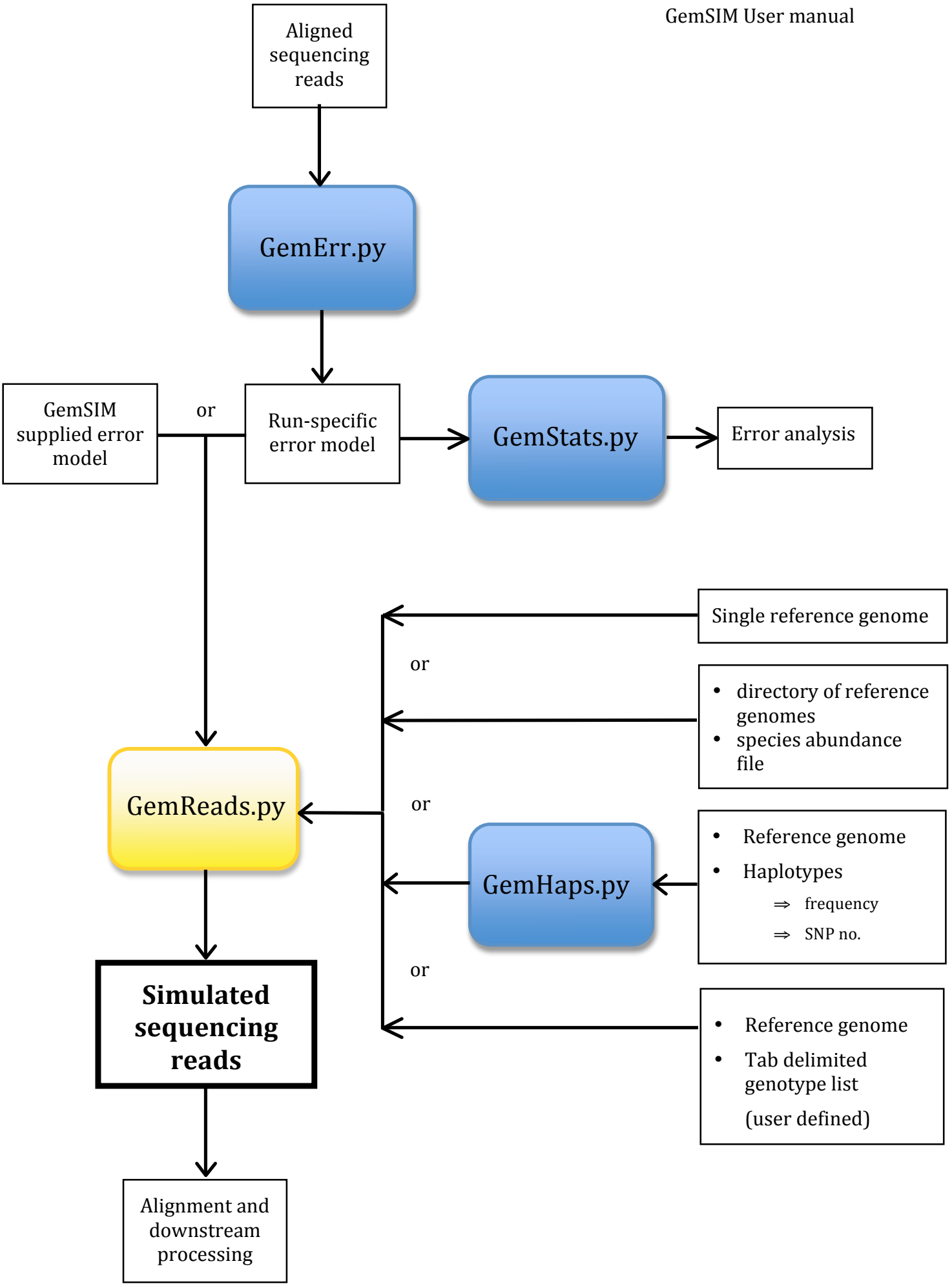
To install GemSIM simply download the package and run the programs from the command line interface (eg, Terminal in Mac OS X). Typing –h after a program name will give you basic usage instructions (more detailed instructions can be found later in this manual). For instance, typing:

```
./GemErr.py –h
```

will give you instructions on how to make a user-generated error model for input into GemReads.py.

Usage

The GemSIM package contains four programs: GemReads.py, GemErr.py, GemStats.py and GemHaps.py. The workflow is described by the following diagram:


Aligned sequencing reads
GemErr.py
GemSIM supplied error model
or
Run-specific error model
GemStats.py
Error analysis
Single reference genome
or
directory of reference genomes
species abundance file
GemReads.py
or
Reference genome
Haplotypes
⇒ frequency
⇒ SNP no.
GemHaps.py
or
Simulated sequencing reads
Reference genome
Tab delimited genotype list
(user defined)
Alignment and downstream processing

GemReads.py forms the main simulation program. It generates reads from a reference genome and an error model. For instance, to simulate one million single-end 100bp long Illumina reads, from a single circular reference genome, using a supplied error model, type:

./GemReads.py –r ref.fasta –n 1000000 –l 100 -m ill100v5_s.gzip –q 64 –o my_out -c

Alternatively, users may generate their own error models from real data using GemErr.py. GemHaps.py may also be used to generate a set of haplotypes derived from a single reference genome, or users may define their own set of related haplotypes.

Parameters for each program are as follows:

GemErr.py

-r	read length. Set to the LONGEST read in a dataset (it doesn't matter if you set it a few bases longer than the longest read, if unsure).
-f	reference genome in fasta format.
-s	input file in SAM format. This file will be parsed to create the error model.
-n	prefix for output filenames.
-c	Use this flag when you wish to calculate error models based on circular genomes.
-i	(optional) use only every i^{th} read for error model. MUST BE ODD. Improves speed of error model calculation.
-m	(optional) maximum indel size tracked by the model. Default=4.
-p	Set ONLY if you have paired end data. This option does not require an argument.
-e	(optional) comma-separated list of genomic sites to exclude (e.g. known SNPs). For example, -e 1301, 1500.
-k	minimum number of times a k-mer must be present in the control reference, for it to be included in the error model (default=0).

When aligning your data to create the SAM input file for GemErr.py, please consider the alignment parameters carefully. Remember, GemErr.py will create a specific error model for an ALIGNMENT, not raw reads. In particular, any reads which do not pass alignment quality filtering will be excluded from the error model. So variations in alignment parameters can have an impact on the error model. It may be advisable to create several error models and sets of simulated data, for several different alignment settings. This has the added benefit of allowing one to test how alignment parameters affect your data analysis. Some general considerations include:

- Pre-filtering data prior to alignment. This may be advisable, so that really bad quality reads to not artificially distort the average error rate of 'normal' reads. Be aware, however, that you should simulate only the number of *aligned* reads you anticipate, rather than the number of raw reads provided directly by your sequencing centre.
- Masking repetitive sequences. Depending on the organism, this may improve results. However, GemSIM does have some inbuilt resilience to small errors in alignment, as error rates are taken as the average of all genomic positions containing the relevant k-mer
- Trimming reads – for instance, it may be advisable to trim the trailing 'B' sections of Illumina reads prior to calculating error models. When simulating reads from such a model, you would then use the –l d option.

GemHaps.py

-r reference genome in fasta format

-g haplotype list (see below)

The halpotype list must be enclosed within inverted commas. List the frequency of the haplotype, followed by a comma, followed by the number of SNPS in the haplotype. Different haplotypes must be separated by a space. For instance, to have one haplotype making up 80% of the population and identical to the reference, and one haplotype making up 20% of the population with 15 SNPS compared to the reference, type:

./GemHaps.py –r [reference.fasta] –g '0.80,0 0.20,15'

Haplotype frequencies **must** sum to one. GemHaps simply places the requested number of random SNPS across the length of the genome, for each haplotype. Alternatively, users can create their own tab-delimited haplotype file for input to GemReads.py, or simply use a reference file. A user defined haplotype file may be preferable when one wants to simulate reads from a particular evolutionary model. A user defined haplotype file should have the following format:

```
0.6
0.05    193326      C       1002694     A
0.1     254506      T       515835      T       708279      T
0.25    93802       C       552798      G       586680      G
```

The start of each line contains the haplotype frequency. The positions of each SNP, followed by the SNP nucleotide, are then listed after each other. Haplotype frequencies, SNP positions, and SNP nucleotides are all separated by tabs.

GemHaps automatically names files according to the reference file given: for instance, if you supply ecoli.fa, the associated output file will be called ecoli.txt. If supplying your own haplotype file, please follow this naming convention.

GemHaps may also be used to generate haplotypes for individual references in metagenomics mode. In this case, please place all the haplotype files in their own directory, which can then by supplied to GemReads.

GemStats.py

-m	error model file
-n	prefix for output files.
-p	set if the error model file is from paired end data. This option does not require an argument.

GemStats.py will create a text file (or two text files, for paired-end data) giving the following information about the error model:

- average error rate
- error rate for each nucleotide
- error rate by position with read
- any nucleotide sequences which have an error rate more than two standard deviations greater than the average
- insertion rate
- deletion rate

This information can be used for quality control and to gain insights into the error profiles of individual sequencing runs.

GemReads.py

-r	reference genome.
-R	directory of reference genomes (metagenomics mode only!). If this option is used, do not also use –r.
-a	Species-abundance file (metagenomics mode only!). Must be provided if –d is given. See below for format.
-n	number of reads (or pairs of reads) to simulate
-g	haplotype file (optional). Generated by GemHaps.py or user defined (see above)
-G	directory of haplotype files (metagenomics mode only!).
-l	simulated read length. Integer value. Users may optionally specify "–l d" to use an empirical read length distribution (output as part of the model created by GemErr.py)

-m	error model file. *_single.gzip or *_paired.gzip, where * is your chosen model prefix.
-c	Use this flag when you wish to simulate reads from a circular genome.
-q	quality score offset. 33 for Sanger format, 64 for Solexa/Illumina 1.3+. (refer also to: http://en.wikipedia.org/wiki/FASTQ_format)
-o	output filename prefix
-p	Set to create paired-end reads (optional). This option does not require an argument.
-u	Specifies mean fragment for paired end reads. Use –u d for empirical fragment lengths. If a length is set, please use –s to specify the fragement length standard deviation.
-s	Use to specify the standard deviation of paired end fragments.

Reference genomes must be in fasta format. If –g is not specified, all reads will be generated from the reference genome, with no introduced SNPs. In metagenomics mode, each reference genome must be a separate fasta file. All reference genomes must be within the specified directory. Any haplotype files corresponding to references in the metagenomics file should also be in a separate directory.

The species abundance file must be a text file listing the filename of each reference genome, and its abundance, separated by a tab. The abundance can be in absolute numbers or relative. For instance, say you want to simulate reads drawn from a population that has 160 cells of bacteria 1, and 40 cells of bacteria two. The reference files are found at /myref/bac1.fna and /myref/bac2.fna. The species abundance file can then look like either:

bac1.fna 160
bac2.fna 40

or:

bac1.fna .8
bac2.fna .2

Usage examples

1. Generate 20000000 Illumina single reads (100bp) from the *Escherichia coli* genome using a supplied error model:

```
./GemReads.py –r ecoli.fasta –n 20000000 –l 100 -m models/ill100v5_s.gzip –c –q 64 –o ecoli_sim
```

This will create an output file ecoli_sim_single.fastq.

2. Generate 20000000 (40000000 reads in total) Illumina paired end reads using supplied error model for chemistry v4 and an empirical read length and fragment length distribution:

```
./GemReads.py –r ecoli.fasta –n 20000000 –l d –u d –m models/ill100v4_p.gzip –c  –q 64
        –o ecoli_paired_sim –p
```

This will create two output files, ecoli_paired_sim_fir.fastq and ecoli_paired_sim_sec.fastq.

2. Generate 20000000 (40000000 reads in total) Illumina paired end reads using supplied error model for chemistry v4, an empirical read length distribution, and fragment length distribution with mean 350 and standard deviation 20:

```
./GemReads.py –r ecoli.fasta –n 20000000 –l d –u 350 –s 20 –m models/ill100v4_p.gzip –c  –q 64
        –o ecoli_paired_sim –p
```

3. Generate 1000000 Roche/454 Titanium reads from the HIV genome, using the supplied error model, and output using Sanger quality score encoding:

```
./GemReads.py –r HIV.fasta –n 1000000 –l d –m models/r454ti_s.gzip –q 33 –o hiv_sim
```

This will create an output file hiv_sim_single.fastq

4. Generate an error model from every 5th read of an alignment of control lane PhiX Illumina 150bp data, excluding the known polymorphic site 1301, and use it to simulate trimmed 120bp reads from the *P. aeruginosa* genome:

```
./GemErr.py  -r 150 –f PhiX.fasta –s PhiX.sam –n ill150 –c  –i 5 –e 1301
```

This will generate an output error model file, ill150_s.gzip for input into GemReads:

```
./GemReads.py –r p_aeruginosa.fasta –n 25000000 –l 120 –m ill150_s.gzip –c –q 64 –o p_aeruginosa_sim_120
```

5. Make some *P. aeruginosa* haplotypes, and simulate some full-length Illumina reads using the model created in step 4:

```
./GemHaps.py –r p_aeruginosa.fasta –g '.50,10 .25, 50 .15, 100, .095,500 .005,1000' –o p_aerug
```

This will create a tab delimited haplotype file p_aerug.haps. Now simulate the reads:

```
./GemReads.py –r p_aeruginosa.fasta –n 20000000 –g p_aerug.haps –l d -m ill150_s.gzip –c
        –q 64 –o p_aerug_hap_sim
```

This will produce the output file p_aerug_hap_sim_single.fastq

6. Simulate a Roche/454 Titanium metagenomic project, using a supplied error model:

```
./GemReads.py –R /home/references –a abund.txt –n 1000000 –l d -m models/r454ti_s.gzip –c
        –q 33 –o meta
```

7. Simulate a Roche/454 Titanium metagenomic project, using a supplied error model, and some haplotype files found in the directory haps:

```
./GemReads.py –R /home/references –a abund.txt  -G /home/haps –n 1000000 –l d -m models/r454ti_s.gzip –c
        –q 33 –o meta
```

Error models

Error models currently included in the GemSIM package:

ill100v4_s: Illumina GA IIx with Illumina Sequencing Kit v4 chemistry, single reads

ill100v4_p: Illumina GA IIx with Illumina Sequencing Kit v4 chemistry, paired reads

ill100v5_s: Illumina GA IIx with TrueSeq SBS Kit v5-GA, single reads

ill100v5_p: Illumina GA IIx with TrueSeq SBS Kit v5-GA, paired reads

r454ti_s: Roche/454 Titanium, single reads