gemino: An efficient, flexible molecular evolution and sequencing simulator

Abstract

High-throughput sequencing (HTS) is central to the study of population genomics. Choices in sampling design for sequencing projects can include sequencing method (e.g., restriction-site associated DNA sequencing [RADseq] versus whole genome sequencing [WGS]), depth of coverage, number of individuals to sample, and selecting the appropriate restriction enzyme(s) (for RADseq). These choices are most often informed by previous work on highly diverged species, which ignores species- and population-specific genomic characteristics, demographies, and evolutionary histories. Simulating sequencing based on available genomic data better informs sampling strategies. However, current methods provide only rudimentary ways to simulate population structure and variation in coverage among sites. Here I present the R package `gemino` that efficiently (i) reads and simulates reference genomes; (ii) generates variants using summary statistics, phylogenies, Variant Call Format (VCF) files, and coalescent simulations—the latter of which can include selection, recombination, and demographic fluctuations; (iii) simulates sequencing error, mapping qualities, restriction-enzyme digestion, and variance in coverage among sites; and (iv) writes outputs to standard file formats. `gemino` can simulate single or paired-ended reads for WGS, RADseq, or genotyping-by-sequencing on the Illumina platform. Although defaults are provided for most functions, relatively few aspects of `gemino` are hard-coded, providing the user flexibility for their simulations. `gemino` can be extended to simulate different sequencing technologies, such as Pacific BioSciences, or Oxford Nanopore Technologies. Most functions are written in C++ to improve performance, and I employed OpenMP to allow for parallel processing. The stable version of `gemino` is available on CRAN (https://CRAN.R-project.org/package=gemino), and the development version is on GitHub (https://github.com/lucasnell/gemino).

Keywords: sequencing simulator, population genomics, high-throughput sequencing, Illumina, Pool-seq, RADseq

Introduction

High-throughput sequencing (HTS) is a cost-effective approach to generate vast amounts of genomic data and has revolutionized the study of genomes (Metzker, 2009). Large datasets combined with increased error rates—compared to Sanger sequencing—make bioinformatic pipelines an important aspect of research using HTS. Many bioinformatic tools exist, and new programs that are more accurate and computationally efficient are constantly being developed. To test these tools against known parameter values, in silico simulation of genomic data is needed.

Although there are many sequence simulators currently available (reviewed in Escalona, Rocha, & Posada, 2016), most have only rudimentary ways to generate population-level data. Events like population-size changes, selection, or population structure can drastically change null expectations for sequence data, but including these possibilities is impossible with most current methods.

In the present paper I introduce `gemino`, the first available HTS simulator in the R (R Core Team, 2018) environment. Designed for efficient memory use, flexibility, and speed, `gemino` combines the functionality of an HTS simulator with that of a molecular phylogenetics simulator. Genomes can be derived from FASTA files or simulated in silico. `gemino` can create variants from the reference genome based on basic population-genomic summary statistics, phylogenies, Variant Call Format (VCF) files, or coalescent simulations. These variants are simulated based on several popular molecular-evolution models. Another difference from other HTS simulators is that, in addition to simulating sequencing error and mapping qualities, `gemino` explicitly simulates variance in coverage among sites. This makes it possible to test bioinformatic pipelines that calculate allele frequencies. `gemino` can simulate single or paired-ended reads for WGS on the Illumina platform, and can be extended to simulate restriction-enzyme-associated sequencing methods and different sequencing platforms, including those

from Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (Nanopore).

After outlining the methods, I demonstrate the usefulness of `gemino` in informing study design through three common usage examples.

Features and methods

Most code is written in C++ and interfaces with R using the `Rcpp` package (Eddelbuettel & François, 2011). I used OpenMP to allow for parallel processing and the `PCG` family of pseudo-random number generators (O'Neill, 2014). An overview of the methods are show in Figure @ref(fig:gemino-overview).

Read and create genomes

Haploid reference genomes can be input from FASTA files using the function `read_fasta`, which also accepts a FASTA index file—created using `samtools faidx`—for faster processing. Both FASTA and index files can be either uncompressed or compressed using `gzip`. If a reference genome is not available, the `create_genome` function creates a reference genome of given equilibrium nucleotide distributions, and mean and standard deviation of the sequence-length distribution. I draw sequence lengths from a gamma distribution (X. Li et al., 2011).

Alter genomes

Using function `filter_sequences`, genomes can be filtered by minimum sequence size or by the smallest sequence that retains a given proportion of total reference sequence if sequences are sorted by descending size. Function `merge_sequences` shuffles reference sequences and merges them into one.

Add variants

Variants from the reference genome are generated as haploid genomes. For diploid organisms, two haploid variants can be randomly assigned as being from one individual, or haploid variants can be assigned to individuals by the user. Variants can be created in three different ways, all of which are encompassed in the `create_variants` function.

(1) Mutations are randomly distributed throughout the genome based on population-genomic statistics. Watterson's estimator (1975) informs the number of segregating sites and Nei and Li's θ (1979) informs the nucleotide diversity at segregating sites. I used "Algorithm D" by Vitter (1984) for sampling locations on genomes without replacement.

(2) Mutations are derived directly from a variant call format (VCF) file. Reading and writing VCF files uses package `vcfR` (Knaus & Grünwald, 2017).

(3) Variant sequences are simulated along a phylogenetic tree or based on coalescent simulation output. I used the TN93 model of nucleotide substitution (Tamura & Nei, 1993). Models JC69 (Jukes & Cantor, 1969), K80 (Kimura, 1980), F81 (Felsenstein, 1981), HKY85 (Hasegawa, Kishino, & Yano, 1985; Hasegawa, Yano, & Kishino, 1984), and most others in frequent use are special cases of the TN93 model (Yang, 2006). I incorporated indels (insertions and deletions) of any size into the TN93 model, resulting in the following substitution rate matrix $\mathbf{Q}$:

$$\mathbf{Q} = \begin{bmatrix} -(\alpha_1\pi_C + \beta\pi_R + \xi) & \alpha_1\pi_C & \beta\pi_A & \beta\pi_G \\ \alpha_1\pi_T & -(\alpha_1\pi_T + \beta\pi_R + \xi) & \beta\pi_A & \beta\pi_G \\ \beta\pi_T & \beta\pi_C & -(\alpha_2\pi_G + \beta\pi_Y + \xi) & \alpha_2\pi_G \\ \beta\pi_T & \beta\pi_C & \alpha_2\pi_A & -(\alpha_2\pi_A + \beta\pi_Y + \xi) \end{bmatrix}$$

where rows specify rates of change from T, C, A, and G, respectively; π_T, π_C, π_A, and π_G represent substitution rates and equilibrium frequencies; π_Y is the frequency of pyrimidines

(i.e., $\pi_T + \pi_C$); π_R is the frequency of purines (i.e., $\pi_A + \pi_G$); α_1 and α_2 are the rates of C↔T and A↔G transitions, respectively; β is the rate of transversions; and ξ is the summed rates of indels of all sizes.

The substitution-transition matrix ($\mathbf{M}$), giving the probabilities of each substitution given that a mutation occurs, is

$$\mathbf{M} = \begin{bmatrix} 0 & \frac{q_{TC}}{q_T} & \frac{q_{TA}}{q_T} & \frac{q_{TG}}{q_T} \\ \frac{q_{CT}}{q_C} & 0 & \frac{q_{CA}}{q_C} & \frac{q_{CG}}{q_C} \\ \frac{q_{AT}}{q_A} & \frac{q_{AC}}{q_A} & 0 & \frac{q_{AG}}{q_A} \\ \frac{q_{GT}}{q_G} & \frac{q_{GC}}{q_G} & \frac{q_{GA}}{q_G} & 0 \end{bmatrix}$$

where $q_{ij} = \mathbf{Q}_{ij}$ and $q_i = -q_{ii}$ (Yang, 2006). The sum of values in each row i in $\mathbf{M}$ equals $1 - \xi/q_i$.

The vector of indel rates by size ($\mathbf{\Xi}$) is

$$\mathbf{\Xi} = \begin{bmatrix} \xi_{(I)1} & \dots & \xi_{(I)n_I} & \xi_{(D)1} & \dots & \xi_{(D)n_D} \end{bmatrix}$$

where the first subscript indicates whether the rate pertains to an insertion (I) or deletion (D), the second subscript indicates size, and n_I and n_D represent the maximum sizes for insertions and deletions respectively. All indel rates are the same among nucleotides, and $\sum \mathbf{\Xi} = \xi$. Rates in $\mathbf{\Xi}$ can be independently set to any value. Default values for insertions and deletions are the same and are calculated, for size l from 1 to 10 (Albers et al., 2010), as such:

$$\xi_l = R\left(\frac{\alpha_1 + \alpha_2}{2} + \beta\right) \frac{\mathrm{e}^{-l}}{\sum_{k=1}^{10} \mathrm{e}^{-k}}$$

where R is the average rate of indels to substitutions in eukaryotes from Sung et al. (2016).

When choosing a substition or indel for a mutation at a position containing nucleotide i, I weight sampling based on a vector $\mathbf{W}_i$ consisting of row i in $\mathbf{M}$ truncated with $\mathbf{\Xi} \cdot {q_i}^{-1}$. For

example, if the mutation occurs at a position containing T, the sampling weight vector would be

$$\mathbf{W}_T = \begin{bmatrix} 0 & \frac{q_{TC}}{q_T} & \frac{q_{TA}}{q_T} & \frac{q_{TG}}{q_T} & \frac{\xi_{(I)1}}{q_T} & \cdots & \frac{\xi_{(I)n_I}}{q_T} & \frac{\xi_{(D)1}}{q_T} & \cdots & \frac{\xi_{(D)n_D}}{q_T} \end{bmatrix}$$

If the j^{th} item is selected, it is a substition to nucleotide j if $j \leq 4$, an insertion of length $j - 4$ if $4 < j \leq n_I + 4$, and a deletion of length $j - n_I - 4$ if $n_I + 4 < j$. To improve the efficiency of weighted sampling from a potentially high number of items within $\mathbf{W}_i$, I used alias sampling (Kronmal & Peterson, 1979; Walker, 1974).

Phylogenetic trees are accepted as `phylo` objects from the `ape` package (Paradis, Claude, & Strimmer, 2004) or from `ms`-style output from coalescent simulators such as `scrm` (Staab, Zhu, Metzler, & Lunter, 2015), `msms` (Ewing & Hermisson, 2010), `ms` (Hudson, 2002), or `msprime` (Kelleher, Etheridge, & McVean, 2016). `ms`-style output can include multiple trees per variant if recombination is included in the simulations; this is parsed properly in `gemino`. Reference sequences are simulated along tree branches by calculating exponential wait times for the Markov "jump" chain for each sequence, where the rate of the exponential distribution is the sum of mutation rates for each nucleotide in the sequence (Yang, 2006). The mutation rate for nucleotide i is q_i. At each jump, a position on the sequence is sampled with a probability proportional to the mutation rate for that nucleotide, using weighted reservoir sampling with exponential jumps (Efraimidis & Spirakis, 2006). A mutation type is then sampled with probabilities from $\mathbf{W}_i$. Jumps are performed until the sum length of all jumps is greater than the branch length.

`gemino` can use coalescent-simulation output that does not include phylogenetic tree(s) if they include segregating sites. In this case, each mutation is simply sampled from $\mathbf{W}_i$, where i is the nucleotide at the segregating-site position.

Digest genomes

An unlimited number of restriction-enzyme binding sites of varying lengths can be used to digest either a reference genome or a set of variants from the reference, using function `digest_genome`. Digesting variants explicitly simulates polymorphisms in restriction enzyme binding sites, an important source of potential bias for RADseq data (Andrews, Good, Miller, Luikart, & Hohenlohe, 2016).

Simulate sequencing data

Multiple sources cause variation in sequencing depth for WGS and RADseq (Andrews et al., 2016; Escalona et al., 2016). `gemino` includes explicit simulation of PCR, fragment size selection, and DNA shearing. These steps occur on the fly as FASTQ files are output, so these methods are all provided inside the `sequence` function.

PCR is modeled as a Galton–Watson discrete time branching process (Kebschull & Zador, 2015) for the number of fragments (N) after j rounds of PCR:

$$N_j = N_{j-1} + \text{Bin}(N_{j-1}, p)$$

where p is the probability of the fragment being duplicated during each round of PCR. The expected value and variance in N after j rounds of PCR is as follows (Athreya & Ney, 1972):

$$\begin{aligned} \text{E}(N) &= N_0(1+p)^j \\ \text{Var}(N) &= N_0(1-p)(1+p)^{j-1}\left[(1+p)^j - 1\right] \end{aligned}$$

where N_0 is the starting number of fragments. With even moderate numbers of starting fragments, the distribution of fragment numbers approaches normal (Figure @ref(fig:pcr-plot)). However, if starting with only 1 of each unique fragment (which most closely resembles WGS and, to a lesser extent, original RADseq), the distribution is neither normal nor unimodal.

For this reason, direct simulation of this branching process is provided for WGS and original RADseq. Arguments are provided to specify PCR bias due to fragment size and GC content.

Size selection, by default, simply filters fragments outside a range. Alternatively, the user can assign sizes a probability of being selected. Shearing is done by first sampling a sheared fragment size. Then locations in the genome are sampled, which can be done with or without weighting by GC content.

Sequencing error and quality scores for paired- or single-end Illumina reads are based on methods by Holtgrewe (2010). Default values are provided, but the user can specify error rates and quality-score distribution parameters (for a normal distribution) for each position in output reads. Although indels are rare in Illumina sequencing (Hu et al., 2012), they can be a large source of error in reads from PacBio or Nanopore (Kircher & Kelso, 2010; Ono, Asai, & Hamada, 2012; Robasky, Lewis, & Church, 2013). Indels are disabled by default in `gemino`, but indel rates can be set and affected by fragment size and GC content.

Writing output

Reference genome sequences can be written to FASTA files and sets of variants to VCF files using the `write` function. Sequencing reads are written to FASTQ files using the `sequence` function. Each of these functions can output to uncompressed or gzipped files.

Performance

Performance was tested on a 2013 MacBook Pro running macOS High Sierra with a 2.6GHz Intel Core i5 processor and 8 GB RAM. I compared the performance of `gemino` functions to those in other packages based on the time and maximum RAM required for each to perform its task. RAM usage is presented as the maximum usage during an R session where the only actions taken were to load the necessary package and run the function. Thus overhead associated with loading a package is included. For functions in packages outside `gemino`, I

use the R convention of displaying them as the package name, two colons, then the function name (e.g., `Package::Function`).

Reference-genome creating, reading, and digesting

For performance comparisons in creating, reading, and digesting reference genomes, I used the R packages `SimRAD` and `ShortRead` . `SimRAD` is designed to assist in research design for RADseq studies, and `ShortRead` performs data input in the Bioconductor system (Huber et al., 2015).

For the performance of creating a reference genome, I compared `create_genome` to `SimRAD::sim.DNAseq` in creating one 100 Mbp chromosome. Although `create_genome` can use multiple cores, I only use one for the comparison test because `SimRAD::sim.DNAseq` can only use one and because multithreading in `create_genome` is done across multiple sequences. I also report on the performance of `create_genome` in creating a 1 Gbp genome split among eight chromosomes.

To test reference-genome read and digestion performance, I used the threespine stickleback (*Gasterosteus aculeatus*) genome (438 Mbp; Peichel, Sullivan, Liachko, & White, 2017). For reading, I compared `read_fasta` (with and without a fasta index file) to `ShortRead::readFasta`, separately for uncompressed and gzipped files. I compared digestion between `digest_genome` and `SimRAD::insilico.digest` using the restriction enzyme *ApeKI* (1.3 million cut sites).

Table @ref(tab:SimRAD-comp-table) shows that `gemino` outperforms `SimRAD` across all functions for both elapsed time and RAM used. `gemino` outperformed `ShortRead` in reading FASTA files when an index file was used in the former, but `gemino` became slower without an index file. RAM usage for `ShortRead::readFasta` was higher than for `read_fasta` largely due to the overhead associated with loading required packages. Before running any functions, loading `SimRAD` or `ShortRead` increased RAM usage by ~325 MB, while `gemino` only increased it by ~25 MB.

Molecular evolution simulation

Here I will compare my method of simulating variants with those from package `phylosim`. However, I am not finished coding this portion of the package.

Example usage

Choose a restriction enzyme for RADseq

Many common restriction enzymes are already programmed into the `digest` command, so comparing different digestions is simple. The example below digests the stickleback genome using *ApeKI* and *AscI* and plots the resulting fragment sizes (Figure @ref(fig:choose-enzyme-run)).

```
genome <- read_fasta('stickleback_genome.fa')
dig_ApeKI <- digest(genome, 'ApeKI', n_cores = 4)
dig_AscI <- digest(genome, 'AscI', n_cores = 4)
plot_digest(list(dig_ApeKI, dig_AscI), genome)
```

Test variant-calling effectiveness in WGS versus RADseq

This portion is not yet coded.

Compute the power of various depths of coverage in WGS

This portion is not yet coded.

Conclusion

`gemino` outperforms current programs while providing a more flexible platform. This package should inform research design for projects employing HTS, particularly those in population genomics. Output from `gemino` will help develop more specific sequencing goals in funding applications and estimate the power of a given sequencing design. Furthermore, `gemino` can be used to test bioinformatic pipelines under assumptions of much more complex demographic histories than current HTS simulation platforms allow.

References

Albers, C. A., Lunter, G., MacArthur, D. G., McVean, G., Ouwehand, W. H., & Durbin, R. (2010). Dindel: Accurate indel calls from short-read data. *Genome Research*, *21*(6), 961–973. https://doi.org/10.1101/gr.112326.110

Andrews, K. R., Good, J. M., Miller, M. R., Luikart, G., & Hohenlohe, P. A. (2016). Harnessing the power of RADseq for ecological and evolutionary genomics. *Nature Reviews Genetics*, *17*(2), 81–92. https://doi.org/10.1038/nrg.2015.28

Athreya, K. B., & Ney, P. E. (1972). The Galton-Watson process. In J. L. Doob (Ed.), *Branching processes* (pp. 1–65). Berlin: Springer.

Eddelbuettel, D., & François, R. (2011). Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, *40*(8), 1–18. https://doi.org/10.18637/jss.v040.i08

Efraimidis, P. S., & Spirakis, P. G. (2006). Weighted random sampling with a reservoir. *Information Processing Letters*, *97*(5), 181–185. https://doi.org/10.1016/j.ipl.2005.11.003

Escalona, M., Rocha, S., & Posada, D. (2016). A comparison of tools for the simulation of genomic next-generation sequencing data. *Nature Reviews Genetics*, *17*(8), 459–469. https://doi.org/10.1038/nrg.2016.57

Ewing, G., & Hermisson, J. (2010). MSMS: A coalescent simulation program including recombination, demographic structure and selection at a single locus. *Bioinformatics*, *26*(16), 2064–2065. https://doi.org/10.1093/bioinformatics/btq322

Felsenstein, J. (1981). Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, *17*(6), 368–376. https://doi.org/10.1007/bf01734359

Hasegawa, M., Kishino, H., & Yano, T.-a. (1985). Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution*, *22*(2), 160–174. https://doi.org/10.1007/bf02101694

Hasegawa, M., Yano, T.-a., & Kishino, H. (1984). A new molecular clock of mitochondrial DNA and the evolution of hominoids. *Proceedings of the Japan Academy, Series B*, *60*(4), 95–98. https://doi.org/10.2183/pjab.60.95

Holtgrewe, M. (2010). *Mason–a read simulator for second generation sequencing data*. Institut für Mathematik und Informatik, Freie Universität Berlin.

Hu, X., Yuan, J., Shi, Y., Lu, J., Liu, B., Li, Z., ... Fan, W. (2012). pIRS: Profile-based illumina pair-end reads simulator. *Bioinformatics*, *28*(11), 1533–1535. https://doi.org/10.1093/bioinformatics/bts187

Huber, W., Carey, V. J., Gentleman, R., Anders, S., Carlson, M., Carvalho, B. S., ... Morgan, M. (2015). Orchestrating high-throughput genomic analysis with bioconductor. *Nature Methods*, *12*(2), 115–121. https://doi.org/10.1038/nmeth.3252

Hudson, R. R. (2002). Generating samples under a wright-fisher neutral model of genetic variation. *Bioinformatics*, *18*(2), 337–338. https://doi.org/10.1093/bioinformatics/18.2.337

Jukes, T. H., & Cantor, C. R. (1969). Evolution of protein molecules. In H. N. Munro (Ed.), *Mammalian protein metabolism* (Vol. 3, pp. 21–131). New York: Academic Press.

Kebschull, J. M., & Zador, A. M. (2015). Sources of PCR-induced distortions in high-throughput sequencing data sets. *Nucleic Acids Research*, gkv717. https://doi.org/10.1093/

nar/gkv717

Kelleher, J., Etheridge, A. M., & McVean, G. (2016). Efficient coalescent simulation and genealogical analysis for large sample sizes. *PLOS Computational Biology*, *12*(5), e1004842. https://doi.org/10.1371/journal.pcbi.1004842

Kimura, M. (1980). A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, *16*(2), 111–120. https://doi.org/10.1007/bf01731581

Kircher, M., & Kelso, J. (2010). High-throughput DNA sequencing - concepts and limitations. *BioEssays*, *32*(6), 524–536. https://doi.org/10.1002/bies.200900181

Knaus, B. J., & Grünwald, N. J. (2017). VCFR: A package to manipulate and visualize variant call format data in R. *Molecular Ecology Resources*, *17*(1), 44–53. Retrieved from http://dx.doi.org/10.1111/1755-0998.12549

Kronmal, R. A., & Peterson, A. V. (1979). On the alias method for generating random variables from a discrete distribution. *The American Statistician*, *33*(4), 214–218. https://doi.org/10.1080/00031305.1979.10482697

Li, X., Zhu, C., Lin, Z., Wu, Y., Zhang, D., Bai, G., ... Yu, J. (2011). Chromosome size in diploid eukaryotic species centers on the average length with a conserved boundary. *Molecular Biology and Evolution*, *28*(6), 1901–1911. https://doi.org/10.1093/molbev/msr011

Metzker, M. L. (2009). Sequencing technologies the next generation. *Nature Reviews Genetics*, *11*(1), 31–46. https://doi.org/10.1038/nrg2626

Nei, M., & Li, W. H. (1979). Mathematical model for studying genetic variation in terms of restriction endonucleases. *Proceedings of the National Academy of Sciences of the USA*, *76*(10), 5269–5273.

Ono, Y., Asai, K., & Hamada, M. (2012). PBSIM: PacBio reads simulatortoward accurate genome assembly. *Bioinformatics*, *29*(1), 119–121. https://doi.org/10.1093/bioinformatics/

bts649

O'Neill, M. E. (2014). *PCG: a family of simple fast space-efficient statistically good algorithms for random number generation.* Claremont, CA: Harvey Mudd College.

Paradis, E., Claude, J., & Strimmer, K. (2004). APE: analyses of phylogenetics and evolution in R language. *Bioinformatics*, *20*, 289–290.

Peichel, C. L., Sullivan, S. T., Liachko, I., & White, M. A. (2017). Improvement of the threespine stickleback genome using a hi-c-based proximity-guided assembly. *Journal of Heredity*, *108*(6), 693–700. https://doi.org/10.1093/jhered/esx058

R Core Team. (2018). *R: a language and environment for statistical computing.* Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Robasky, K., Lewis, N. E., & Church, G. M. (2013). The role of replicates for error mitigation in next-generation sequencing. *Nature Reviews Genetics*, *15*(1), 56–62. https://doi.org/10.1038/nrg3655

Staab, P. R., Zhu, S., Metzler, D., & Lunter, G. (2015). Scrm: Efficiently simulating long sequences using the approximated coalescent with recombination. *Bioinformatics*, *31*(10), 1680–1682. https://doi.org/10.1093/bioinformatics/btu861

Sung, W., Ackerman, M. S., Dillon, M. M., Platt, T. G., Fuqua, C., Cooper, V. S., & Lynch, M. (2016). Evolution of the insertion-deletion mutation rate across the tree of life. *G3: Genes|Genomes|Genetics*, *6*(8), 2583–2591. https://doi.org/10.1534/g3.116.030890

Tamura, K., & Nei, M. (1993). Estimation of the number of nucleotide substitutions in the control region of mitochondrial dna in humans and chimpanzees. *Molecular Biology and Evolution*, *10*(3), 512–526.

Vitter, J. S. (1984). Faster methods for random sampling. *Communications of the ACM*, *27*(7), 703–718. https://doi.org/10.1145/358105.893

Walker, A. (1974). New fast method for generating discrete random numbers with arbitrary

frequency distributions. *Electronics Letters*, *10*(8), 127. https://doi.org/10.1049/el:19740097

Watterson, G. A. (1975). On the number of segregating sites in genetical models without recombination. *Theoretical Population Biology*, *7*(2), 256–276.

Yang, Z. (2006). *Computational molecular evolution*. New York, NY, USA: Oxford University Press. https://doi.org/10.1093/acprof:oso/9780198567028.001.0001