jackelope: A swift, versatile 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 platform, depth of coverage, and number of individuals to sample. 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, most current methods provide only rudimentary ways to simulate population structure and variation in coverage among sites. Here I present the R package `jackelope` 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, multiplexing, and optical/PCR duplicates; and (iv) writes outputs to standard file formats. `jackelope` can simulate single, paired-end, or mate-pair Illumina reads, as well as reads from Pacific BioSciences. Most functions are written in C++ to improve performance, and I employed OpenMP to allow for parallel processing. `jackelope` is available on GitHub (https://github.com/lucasnell/jackelope).

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

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 `jackelope`, the first available HTS simulator in the R (R Core Team, 2019) environment. Designed for efficient memory use, flexibility, and speed, `jackelope` 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. `jackelope` 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 can be simulated based on one of several popular molecular-evolution models. `jackelope` simulates single, paired-ended, or mate-pair reads on the Illumina platform, and it generates Pacific Biosciences (PacBio) reads. Both types of sequencing output to FASTQ files that can be optionally compressed.

After outlining the methods, I demonstrate the usefulness of `jackelope` 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 thread-safe, pseudo-random number generators (O'Neill, 2014). Package `RcppProgress` provides the thread-safe progress bar (Forner, 2018). An overview of the methods are show in Figure @ref(fig:jackelope-overview).

The `ref_genome` class

Haploid reference genomes are represented by the class `ref_genome`, an R6 (Chang, 2019) class that acts as a wrapper around a pointer to an underlying C++ object that stores all the sequence information. They can be generated from FASTA files using the function `read_fasta`. This function also accepts FASTA index files—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).

The access provided by the R class `ref_genome` is designed to both maximize flexibility and minimize copying and the chances of printing extremely large strings to the console. Methods in `ref_genome` allow the user to view the number of sequences, sequence sizes, sequence names, and individual sequence strings. Users can also edit sequence names and remove one or more sequences by name. Method `filter_sequences` filters genomes by the minimum sequence size or by the smallest sequence that retains a given proportion of total reference sequence if sequences are sorted by descending size. Using method `merge_sequences`, users can shuffle reference sequences and merge them into one. Method `replace_Ns` replaces any `N`s in the reference sequence with random nucleotides, optionally sampled with weights provided by the user. Reference genomes can be written to FASTA files using the `write_fasta`

function.

The `mevo` class

The `mevo` class stores molecular evolution information for use in the `create_variants` function, and is created using the `make_mevo` function. Molecular evolution can include substitutions, indels (insertions and deletions) of lengths up to $2^{31} - 1$, and variation in mutation rates among sites.

The following substitution models can be employed: TN93 (Tamura & Nei, 1993), JC69 (Jukes & Cantor, 1969), K80 (Kimura, 1980), F81 (Felsenstein, 1981), HKY85 (Hasegawa, Kishino, & Yano, 1985; Hasegawa, Yano, & Kishino, 1984), F84 (Thorne, Kishino, & Felsenstein, 1992), GTR (Tavaré, 1986), and UNREST (Z. B. Yang, 1994). If using the UNREST model, equilibrium nucleotide frequencies (π) are calculated by solving for $\pi\mathbf{Q} = 0$, where $\mathbf{Q}$ is the substitution rate matrix. This is done by finding the left eigenvector of $\mathbf{Q}$ that corresponds to the eigenvalue closest to zero.

Insertions and deletions are generated the same and have the same requirements for inputs in the `make_mevo` function. They first require an overall rate parameter, which is for the sum among all nucleotides; indel rates do not differ among nucleotides. Function `make_mevo` also requires information about the relative rates of indels of different sizes, which can be provided in 3 different ways. First, rates can be proportional to $\exp(-u)$ for indel length u from 1 to the maximum possible length, M (Albers et al., 2010). Second, rates can be generated from a Lavalette distribution, where the rate for length u is proportional to $[uM/(M - u + 1)]^{-a}$ (Fletcher & Yang, 2009). Third, relative rates can be specified directly by providing a length-M numeric vector of positive values.

Among-site variation in mutation rates is included either by generating gamma distances (γ) from a distribution or by passing them manually. The overall mutation rate is $\gamma\ q_0$, where q_0 is the base mutation rate determined only by the nucleotide. Gamma distances are generated from a Gamma distribution with a fixed mean of 1 and with a shape parameter provided

by the user. Users can also pass a list of matrices, one for each reference sequence, with a gamma distance and end point for each sequence region. The gamma distances can optionally be written to a BED file.

The `variants` class

Variants from the reference genome are represented as haploid genomes in the `variants` class. Similarly to `ref_genome`, this R6 class wraps a pointer to a C++ object that stores all the information, and it was designed to prevent copying of large objects in memory. The underlying C++ class also does not store whole variant genomes, but rather just their mutation information—this dramatically reduces memory usage. Variants can be created in five different ways, all of which are encompassed in the `create_variants` function.

The first two methods directly specify numbers and locations of mutations and therefore do not require any phylogenetic methods in `jackalope`. First, a variant call format (VCF) file can directly specify mutations for each variant. This method works using the `vcfR` package (Knaus & Grünwald, 2016, 2017) and is the only method that does not require a `mevo` object. Second, matrices of segregating sites from coalescent output can provide the locations of mutations. A `mevo` object then provides the type of mutation at each site. The segregating-site information can take the form of (1) a coalescent-simulator object from the `scrm` (Paul R. Staab, Sha Zhu, Dirk Metzler, & Gerton Lunter, 2015) or `coala` (Paul R. Staab & Dirk Metzler, 2016) package, or (2) a file containing output from a coalescent simulator in the format of the `ms` (Hudson, 2002) or `msms` (Ewing & Hermisson, 2010) programs.

The last three methods require simulations along phylogenetic trees, which is outlined in the following paragraph. In the first of these phylogenetic methods, phylogenetic tree(s) can be directly input from either `phylo` object(s) or NEWICK file(s). One tree can be used for all genome sequences, or each sequence can use a separate tree. Alternatively, users can pass an estimate for θ (the population-scaled mutation rate). A random coalescent tree is first generated using the `rcoal` function in the `ape` package (Paradis & Schliep, 2018). Then, its total tree length is scaled to be $\theta/\mu \sum_{i=1}^{n-1} 1/i$ for n variants and an equilibrium mutation rate

of μ. The last method allows for simulation of recombination by simulating along gene trees that can differ both within and among reference sequences. As for coalescent segregating sites, gene trees can be from `scrm` or `coala` objects, or from `ms`-style output files.

Variants are simulated along tree branches by generating 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 (Z. Yang, 2006). At each jump, a position on the sequence is sampled with a probability proportional to the mutation rate for that nucleotide. To sample positions, I use weighted reservoir sampling with exponential jumps (Efraimidis & Spirakis, 2006). To optionally improve performance for sampling long sequences, a number of positions can be first uniformly sampled using "Algorithm D" by Vitter (1984), followed by weighted sampling by rates. After sampling a position, a mutation type is sampled with probabilities proportional to the rate of each mutation type for the nucleotide present at the sampled position. I used alias sampling (Kronmal & Peterson, 1979; Walker, 1974) for sampling mutation types. Jumps are performed until the summed length of all jumps is greater than the branch length.

Methods in `variants` allow the user to view the number of sequences/variants, variant sequence sizes, sequence/variant names, and individual variant-sequence strings. Users can also edit variant names, remove one or more variants by name, and manually add mutations. Variant information can be written to VCF files using the `write_vcf` function, where each variant can optionally be considered one of multiple haplotypes for samples with ploidy levels > 1.

Simulate sequencing data

Both R6 classes `ref_genome` and `variants` can be input to the sequencing functions `illumina` and `pacbio`. If providing a `variants` object, you can specify sampling weights for each variant to simulate the library containing differing amounts of DNA from each. Both methods also allow for a probability of read duplication, which might occur due to PCR in either method and from optical duplicates in Illumina sequencing. Reads can be written using

multiple threads by having a read "pool" for each thread and having pools write to file only when they are full. This reduces conflicts that occur when multiple threads attempt to write to disk at the same time. The size of a "full" pool can be adjusted, and larger sizes should increase both speed and memory usage. Reads are output to FASTQ files, optionally with `gzip` compression.

Function `illumina` simulates single, paired-ended, or mate-pair Illumina reads, while `pacbio` simulates reads from the Pacific Biosciences (PacBio) platform. Illumina read simulation is based on the ART program (Huang, Li, Myers, & Marth, 2011), and PacBio read simulation is based on SimLoRD (Stöcker, Köster, & Rahmann, 2016). Function inputs emulate the program they were based on.

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

Conclusion

`jackalope` 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 `jackalope` will help develop more specific sequencing goals in funding applications and estimate the power of a given sequencing design. Furthermore,

`jackalope` 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

Chang, W. (2019). *R6: Encapsulated classes with reference semantics*. Retrieved from https://CRAN.R-project.org/package=R6

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

Fletcher, W., & Yang, Z. (2009). INDELible: A flexible simulator of biological sequence evolution. *Molecular Biology and Evolution*, *26*(8), 1879–1888. https://doi.org/10.1093/molbev/msp098

Forner, K. (2018). *RcppProgress: An interruptible progress bar with openmp support for c++ in r packages*. Retrieved from https://CRAN.R-project.org/package=RcppProgress

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

Huang, W., Li, L., Myers, J. R., & Marth, G. T. (2011). ART: A next-generation sequencing read simulator. *Bioinformatics*, *28*(4), 593–594. https://doi.org/10.1093/bioinformatics/btr708

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.

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

Knaus, B. J., & Grünwald, N. J. (2016). VcfR: An r package to manipulate and visualize VCF format data. *BioRxiv*. Retrieved from http://dx.doi.org/10.1101/041277

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

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., & Schliep, K. (2018). Ape 5.0: An environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics*, *35*, 526–528.

Paul R. Staab, & Dirk Metzler. (2016). Coala: An R framework for coalescent simulation. *Bioinformatics*. https://doi.org/10.1093/bioinformatics/btw098

Paul R. Staab, Sha Zhu, Dirk Metzler, & Gerton Lunter. (2015). scrm: Efficiently simulating long sequences using the approximated coalescent with recombination. *Bioinformatics*, *31*(10), 1680–1682. Retrieved from http://bioinformatics.oxfordjournals.org/content/31/10/1680

R Core Team. (2019). *R: a language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Stöcker, B. K., Köster, J., & Rahmann, S. (2016). SimLoRD: Simulation of long read data. *Bioinformatics*, *32*(17), 2704–2706. https://doi.org/10.1093/bioinformatics/btw286

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.

Tavaré, S. (1986). Some probabilistic and statistical problems in the analysis of DNA sequences. *Lectures on Mathematics in the Life Sciences*, *17*(2), 57–86.

Thorne, J. L., Kishino, H., & Felsenstein, J. (1992). Inching toward reality: an improved likelihood model of sequence evolution. *Journal of Molecular Evolution*, *34*(1), 3–16.

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

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

Yang, Z. B. (1994). Estimating the pattern of nucleotide substitution. *Journal of Molecular Evolution*, *39*(1), 105–111.