jackalope: a swift, versatile phylogenomic and high-throughput sequencing simulator

Lucas A. Nell[1]

[1]Department of Integrative Biology, University of Wisconsin–Madison

Abstract

High-throughput sequencing (HTS) is central to the study of population genomics and has an increasingly important role in constructing phylogenies. Choices in research design for sequencing projects can include a wide range of factors, such as sequencing platform, depth of coverage, and bioinformatic tools. Simulating HTS data better informs these decisions. However, current standalone HTS simulators cannot generate genomic variants under even somewhat complex evolutionary scenarios, which greatly reduces their usefulness for fields such as population genomics and phylogenomics. Here I present the R package `jackalope` that simply and efficiently simulates (i) variants from reference genomes and (ii) reads from both Illumina and Pacific Biosciences (PacBio) platforms. Genomic variants can be simulated using phylogenies, gene trees, coalescent-simulation output, population-genomic

summary statistics, and Variant Call Format (VCF) files. `jackalope` can simulate single, paired-end, or mate-pair Illumina reads, as well as reads from Pacific Biosciences. These simulations include sequencing errors, mapping qualities, multiplexing, and optical/PCR duplicates. It can read reference genomes from FASTA files and can simulate new ones, and all outputs can be written to standard file formats. `jackalope` is available for Mac, Windows, and Linux systems.

Keywords: sequencing simulator, population genomics, phylogenomics, 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). The combination of massive datasets, sequencing errors, and potentially complex evolutionary histories 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. Simulation of HTS data is needed to test these tools. Although there are many sequence simulators currently available (reviewed in Escalona, Rocha, & Posada, 2016), most have only rudimentary ways to generate variants from a reference genome. For example, both `Mason` (Holtgrewe, 2010) and `GemSIM` (McElroy, Luciani, & Thomas, 2012) place mutations randomly throughout genome sequences, the former based on per-site mutation

rates and the latter on a set number of mutations per variant haplotype. In either case, these simulators are not useful for researchers interested in simulating HTS data to test more complex evolutionary questions, such as (a) how demographic changes and selection interact to affect genome-wide diversity or (b) how incongruence between gene trees might affect reconstructed phylogenies.

Recently, pipelines have been developed that use multiple programs to simulate complex evolutionary histories and HTS on the resulting populations or species. `TreeToReads` (McTavish et al., 2017) simulates sequences along a single phylogenetic tree using `Seq-Gen` (Rambaut & Grass, 1997) and generates Illumina reads using `ART` (Huang, Li, Myers, & Marth, 2011). The newer pipeline `NGSphy` (Escalona, Rocha, & Posada, 2018) uses `INDELible` (Fletcher & Yang, 2009) for more mutation options than `Seq-Gen` and can simulate along multiple gene trees via `SimPhy` (Mallo, Oliveira Martins, & Posada, 2015). These pipelines are quite powerful. However, they both require the installation of multiple programs to use them, and the vast array of features creates a steep learning curve when getting started. Additionally, the lack of integration between inner programs means that the entire process is not as computationally efficient as a single standalone program.

In the present paper I introduce `jackalope`, an R (R Core Team, 2019) package that combines the functionality of an HTS simulator with that of a phylogenomics simulator. I chose the R platform because of its simple integration with C++ code via the `Rcpp` package (Eddelbuettel & François, 2011). The interface with R allows the user greater flexibility, while the underlying C++ code improves memory use and speed. `jackalope` can read

genomes from FASTA files or simulate them in silico. It can also create variants from the reference genome based on basic population-genomic summary statistics, phylogenies, gene trees, Variant Call Format (VCF) files, or matrices of segregating sites. These variants can be simulated based on any of several popular molecular-evolution models. `jackalope` simulates single, paired-ended, or mate-pair reads on the Illumina platform, as well as Pacific Biosciences (PacBio) reads. All information generated by `jackalope` can be output to standard file formats.

After outlining the methods, I compare the performance of `jackalope` to that of other popular programs. Lastly, I demonstrate the usefulness of `jackalope` through three 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). All input and output files can have `gzip` or `bgzip` compression, using the `zlib` and `htslib` (Li et al., 2009) libraries. Access to these libraries uses the R packages `zlibbioc` (Morgan, 2019) and `Rhtslib` (Hayden & Morgan, 2019) to improve portability. An overview of the methods is show in Figure 1.

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. If a reference genome is not available, the `create_genome` function creates a reference genome of given equilibrium nucleotide distributions. Sequence lengths are drawn from a gamma distribution (Li et al., 2011) with a mean and standard deviation provided by the user.

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, individual sequence strings, and nucleotide proportions (GC or otherwise). Users can also edit sequence names, remove one or more sequences by name, and add sequences manually. 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 nucleotides that are sampled with weights proportional to their equilibrium distributions (provided by the user). Reference genomes can be written to FASTA files using

the `write_fasta` function.

Creating variants

Haploid variants from the reference genome are generated using the `create_variants` function. To organize a potentially large amount of information that can be input to this function, I added helper functions that handle parts of the input. There are helper functions for the higher-level method information (information from phylogenies, coalescent simulations, etc.) and for various aspects of the molecular evolution (substitutions, indels, and among-site variability in mutation rates). The first subsection below outlines the higher-level information functions, and the second subsection details the molecular evolution functions.

Higher-level method information

There are five ways to generate variants from the reference genome, and each has a function associated with it. The names of these functions follow the form `vars_X` for method `X`, and information on all of the function can be found in the `vars_functions` documentation. The outputs of these functions are meant to be passed to the `vars_info` argument in `create_variants`.

The first two methods directly specify numbers and locations of mutations and therefore do not require any phylogenomic methods in `jackalope`. First, a variant call format (VCF) file can directly specify mutations for each variant (function `vars_vcf`). This method works

using the `vcfR` package (Knaus & Grünwald, 2017) and is the only method that does not require any molecular-evolution information. Second, matrices of segregating sites from coalescent output can provide the locations of mutations (function `vars_ssites`). Molecular evolution information passed to `create_variants` then informs the sampling for the type of mutation at each site. The segregating-site information can take the form of (1) a coalescent-simulator object from the `scrm` (Staab, Zhu, Metzler, & Lunter, 2015) or `coala` (Staab & 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) program.

The last three methods simulate sequences along phylogenies. In the first of these methods, phylogenetic tree(s) can be directly input from either `phylo` object(s) or NEWICK file(s) (function `vars_phylo`). One tree can be used for all genome sequences, or each sequence can use a separate tree. In the second method, users can pass an estimate for θ, the population-scaled mutation rate (function `vars_theta`). 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 (function `vars_gtrees`). Similarly to simulations using coalescent segregating sites, gene trees can be from `scrm` or `coala` objects, or from `ms`-style output files.

For the phylogenomic methods, 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 all nucleotides in the sequence (Yang, 2006). At each jump, a position on the sequence is sampled with a probability proportional to the mutation rate for that position. A position's mutation rate is the product of its "gamma distance" (determined by among-site variability in mutation rates) and the overall mutation rate for the nucleotide at that position (determined by the summed indel and substitution 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. Jumps are performed until the summed length of all jumps is greater than the branch length.

Because sampling of mutation positions incorporates both among-site and among-nucleotide variability in mutation rates, users can separately specify rate variation that occurs due to (1) where the nucleotides are positioned on the sequence molecule and (2) the chemical structures of the nucleotides themselves. To accomplish this, `jackalope` first splits each sequence into smaller regions, where each region contains the same gamma distance. The nucleotide-level rates are then summed by region and multiplied by the region's gamma distance to get the region's overall rate. I used these total rates as weights to sample regions, and I incorporated a binary search tree to speed this sampling. Simple inversion sampling is used to sample within a region.

Mutation-type sampling is performed by creating 4 vectors (one for each nucleotide), each containing all possible mutation types: substitutions to other nucleotides and insertions/deletions of all specified lengths. I then use alias sampling (Kronmal & Peterson,

1979; Walker, 1974) to sample within this vector.

Molecular evolution

Molecular evolution information is provided to the `create_variants` function through organizing functions that are specific to the type of information: substitutions, indels (insertions and deletions), and variation in mutation rates among sites. Substitutions use the `sub_models` group of functions, indels use the `indels` function, and among-site variation uses the `site_var` function. A table of rates from Sung et al. (2016) is included in the package as a guide for reasonable rates to use.

Each substitution model uses its own function of the pattern `sub_M` for model `M`, and the provided `sub_models` documentation includes information to help users choose among them. 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 (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.

Providing insertion and deletion information is organized in the `indels` function. It first requires an overall rate parameter, which is for the sum among all nucleotides; indel rates do

not differ among nucleotides. The `indels` function 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. Indels up to 1 Mb are allowed.

Among-site variation in mutation rates, specified in the `site_var` function, is included either by generating gamma distances from a distribution or by passing them manually. Gamma distances are generated from a Gamma distribution with a fixed mean of 1 and with a shape parameter provided by the user; a proportion of these regions can optionally be invariant. 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

Haploid variant information is stored 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 does not store whole variant genomes, but rather just their mutation information—this dramatically reduces

memory usage. 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, add blank variants, duplicate existing variants, 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. Alternatively, variant genomes can be written to separate FASTA files using `write_fasta`.

Simulate sequencing data

Both R6 classes `ref_genome` and `variants` can be input to the sequencing functions `illumina` and `pacbio`. Sequences from which reads are derived are sampled with weights proportional to their length. If a `variants` object is provided, individual variants are sampled with equal probabilities by default. Alternatively, the user 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. `jackalope` can create reads 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.

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 et al., 2011), and PacBio read simulation is based on `SimLoRD` (Stöcker, Köster, & Rahmann, 2016). Each was re-coded in C++ to more seemlessly integrate into `jackalope`. Function inputs emulate the program they were based on, and users are encouraged in each function's documentation to also cite `ART` or `SimLoRD`.

Validation and performance

I validated `jackalope` by conducting a series of simulations for variant creation and Illumina and PacBio sequencing. Using known inputs, I compared predicted to observed values of outputs, and `jackalope` conformed closely to expectations (Supporting Information Figures S1–S12).

I compared the performance of `jackalope` to existing programs on a MacBook Pro running macOS Mojave (version 10.14.4) with a 2.6GHz Intel Core i5 processor and 8 GB RAM. Elapsed time and maximum memory used were measured by using the GNU `time` program (`/usr/bin/time` from the terminal). For each test, 10 runs of each program were tested in random order.

Creating variants

For creating variants, I compared `jackalope` to `Seq-Gen` (Rambaut & Grass, 1997). I chose `Seq-Gen` because it is used by `TreeToReads` and because it can generate sequences based on gene trees. It is relatively simple and written in C, so it should provide a conservative estimate of how well `jackalope` performs against similar programs. I tested each program by having them simulate a 2 Mb, 20 Mb, and 200 Mb genome split evenly among 20 sequences, then generate 8 variants from that genome using gene trees produced by the following `scrm` command:

```
scrm 8 20 -T -seed 1829812441 253160851 137767610 > scrm.tree
```

I used the HKY85 substitution model with an overall mutation rate of 0.001 substitutions per site per unit of time (where one unit of time is equal to a branch length of 1). I did not include indels because `Seq-Gen` cannot simulate them. Among-site variability was generated using a Gamma distribution with a shape parameter of 1; in `jackalope`, the region size was 100 bp. Output was written to either PHYLIP format (for `Seq-Gen`) or FASTA files (for `jackalope`). `jackalope` was tested for 1 and 4 threads, but `Seq-Gen` for only 1 because it cannot use multiple threads.

`jackalope` consistently outperforms `Seq-Gen` in terms of speed, and the effect of using 4 threads is most pronounced at the larger genome sizes (Figure 2). This makes sense, since the overhead associated with multithreading becomes more worthwhile when there are more

tasks to accomplish. Memory usage is greater in `jackalope` at low genome sizes due to the overhead associated with loading R: An R script that simply printed an empty string used $\sim$ 55 MB memory, and another that only loaded `jackalope` used $\sim$ 70 MB. At the 20 Mb genome size, however, `jackalope` used slightly less memory than `Seq-Gen`, and for a 200 Mb genome size, `jackalope` used 38% less memory.

Generating sequencing reads

For both Illumina and PacBio read generation in `jackalope`, I compared their performance to the programs they were based on: `ART` (Huang et al., 2011) and `SimLoRD` (Stöcker et al., 2016), respectively. `ART` has the added benefit of being used for both `NSGphy` and `TreesToReads`. All these tests consisted of reading a 2 Mb genome from a FASTA file, simulating reads, and writing them to FASTQ files. No ALN or SAM/BAM files were written by any of the programs. For Illumina sequencing, I generated 2×150 reads from the HiSeq 2500 platform, and tests were conducted for 100 thousand, 1 million, and 10 million total reads. For PacBio sequencing, I generated reads using default parameters, and tests were conducted for 1, 10, and 100 thousand total reads. `jackalope` was tested for 1 and 4 threads, but neither `ART` nor `SimLoRD` allowed the use of more than 1.

Again, `jackalope` outperformed both `ART` and `SimLoRD` in terms of speed (Figure 3). Using multiple threads was most useful when many reads were generated: For at least 10M Illumina reads and at least 1M PacBio reads, using 4 threads reduced the elapsed time for jobs by

$\sim 50\%$. All three programs used very little memory, and the amount did not depend strongly on the number of reads generated. `ART` used 9.4 MB and `SimLoRD` used 54.0–61.2 MB. `jackalope` used 76.5–79.4 MB for Illumina reads and 101.3–110.1 MB for PacBio reads.

Example usage

This section provides brief examples of how `jackalope` can be used to generate sequencing data that can inform some common sampling decisions for HTS studies. These examples only show how to produce the output from `jackalope`, as a review of different bioinformatic programs that would process these data is beyond the scope of the present manuscript.

For an example reference assembly, I used the *Drosophila melanogaster* assembly (version 6.27) downloaded from `flybase.org` (Thurmond et al., 2018). After downloading, I read the compressed FASTA file, filtered out scaffolds by using a size threshold, and manually removed the sex chromosomes using the following code:

```
ref <- read_fasta("dmel-6.27.fasta.gz", cut_names = TRUE)
ref$filter_seqs(1e6, method = "size")
ref$rm_seqs(c("X", "Y"))
```

This resulted in the following `ref_genome` object:

```
## < Set of 5 sequences >
## # Total size: 110,338,337 bp
```

```
##   name                          sequence                              length
## 3R         ACGGGACCGAGTATAGTACCAGTAC...CGTGACTGTTCGCATTCTAGGAATTC  32079331
## 3L         TAGGGAGAAATATGATCGCGTATGC...TTCTATGTTATTCCATGTTATTCTAT  28110227
## 2R         CTCAAGATACCTTCTACAGATTATT...TGGAGGTACGGAAACAGAATGAATTC  25286936
## 2L         CGACAATGCACGACAGAGGAAGCAG...AACAGAGAACAGAGAACAGAGAAGAG  23513712
## 4          TTATTATATTATTATATTATTATAT...GCAGCCGTCGATTTGAGATATATGAA   1348131
```

For molecular-evolution information, I used the TN93 model, an insertion rate of `2e-5` for lengths from 1 to 10, and a deletion rate of `1e-5` for lengths from 1 to 40.

```
sub <- sub_TN93(pi_tcag = c(0.1, 0.2, 0.3, 0.4),
                alpha_1 = 0.0001, alpha_2 = 0.0002,
                beta = 0.00015)
ins <- indels(rate = 2e-5, max_length = 10)
del <- indels(rate = 1e-5, max_length = 40)
```

Assembling a genome

The example here produces FASTQ files from the known reference assembly that could test strategies for how to assemble a similar genome using HTS data.

The first strategy is to use only PacBio sequencing. The PacBio Sequel system produces up to 500,000 reads per Single Molecule, Real-Time (SMRT) cell, so you could run the following

for two cells (with the file `pacbio_R1.fq` as output):

```
pacbio(ref, out_prefix = "pacbio", n_reads = 2 * 500e3)
```

An alternative, hybrid strategy uses 1 SMRT cell of PacBio sequencing and 1 lane ($\sim$ 500 million reads) of 2×100bp Illumina sequencing on the HiSeq 2500 system (the default Illumina system in `jackalope`):

```
pacbio(ref, out_prefix = "pacbio", n_reads = 500e3)
illumina(ref, out_prefix = "illumina", n_reads = 500e6, paired = TRUE,
         read_length = 100)
```

The last strategy combines 1 lane of 2×100bp Illumina HiSeq 2500 sequencing with 1 flow cell of 2×250bp mate-pair sequencing on an Illumina MiSeq v3. The mate-pair sequencing uses longer fragments (defaults are mean of 400 and standard deviation of 100) to better cover highly repetitive regions.

```
illumina(ref, out_prefix = "ill_pe", n_reads = 500e6, paired = TRUE,
         read_length = 100)
illumina(ref, out_prefix = "ill_mp", seq_sys = "MSv3",
         read_length = 250, n_reads = 50e6, matepair = TRUE,
         frag_mean = 3000, frag_sd = 500)
```

Estimating divergence between populations

Here, I will demonstrate how to generate population-genomic data of a type that might be used to estimate the divergence between two populations. I first use the `scrm` package (Staab et al., 2015) to conduct coalescent simulations that will generate segregating sites for 40 haploid variants from the reference genome. Twenty of the variants are from one population, twenty from another. The symmetrical migration rate is 100 individuals per generation. To generate many mutations, I set the population-scaled mutation rate ($\theta = 4N_0\mu$) to `1000` for this example.

```
ssites <- scrm(paste("40", ref$n_seqs(), "-t 1000 -I 2 20 20 100"))
```

Using the previously created objects for molecular evolution information (`sub`, `ins`, and `del`), I create variants from the reference genome:

```
vars <- create_variants(ref, vars_ssites(ssites), sub, ins, del)
```

This results in the following set of variants:

```
##                              << Variants object >>
## # Variants: 40
## # Mutations: 440,203
##
##                            << Reference genome info: >>
```

```
## < Set of 5 sequences >
## # Total size: 110,338,337 bp
##   name                          sequence                          length
## 3R         ACGGGACCGAGTATAGTACCAGTAC...CGTGACTGTTCGCATTCTAGGAATTC  32079331
## 3L         TAGGGAGAAATATGATCGCGTATGC...TTCTATGTTATTCCATGTTATTCTAT  28110227
## 2R         CTCAAGATACCTTCTACAGATTATT...TGGAGGTACGGAAACAGAATGAATTC  25286936
## 2L         CGACAATGCACGACAGAGGAAGCAG...AACAGAGAACAGAGAACAGAGAAGAG  23513712
## 4          TTATTATATTATTATATTATTATAT...GCAGCCGTCGATTTGAGATATATGAA   1348131
```

For a file of true divergences from the reference genome, the `write_vcf` function writes the `variants` object to VCF file `variants.vcf`:

```
write_vcf(vars, "variants")
```

Lastly, I simulate 1 lane of 2×100bp Illumina HiSeq 2500 sequencing. In this case, individuals within a population are pooled, and the population sequences are derived from are identified by barcodes.

```
illumina(vars, out_prefix = "vars_illumina", n_reads = 500e6, paired = TRUE,
         read_length = 100, barcodes = c(rep("AACCGCGG", 20),
                                         rep("GGTTATAA", 20)))
```

The below example instead has each individual variant's reads output to separate FASTQ files:

```
illumina(vars, out_prefix = "vars_illumina", n_reads = 500e6, paired = TRUE,
         read_length = 100, sep_files = TRUE)
```

Constructing a phylogeny

From one phylogenetic tree

This section shows how `jackalope` can generate variants from a phylogeny, then simulate sequencing data from those variants to test phylogeny reconstruction methods. First, I generated a random coalescent tree of 10 species using the `ape` package (Paradis & Schliep, 2018).

```
tree <- ape::rcoal(10)
```

Then I input that to the `create_variants` function alongside the molecular evolution information.

```
vars <- create_variants(ref, vars_phylo(tree), sub, ins, del)
```

This results in the following `variants` object:

```
##                              << Variants object >>
## # Variants: 10
## # Mutations: 301,710
##
```

```
##                          << Reference genome info: >>
## < Set of 5 sequences >
## # Total size: 110,338,337 bp
##   name                          sequence                              length
## 3R         ACGGGACCGAGTATAGTACCAGTAC...CGTGACTGTTCGCATTCTAGGAATTC  32079331
## 3L         TAGGGAGAAATATGATCGCGTATGC...TTCTATGTTATTCCATGTTATTCTAT  28110227
## 2R         CTCAAGATACCTTCTACAGATTATT...TGGAGGTACGGAAACAGAATGAATTC  25286936
## 2L         CGACAATGCACGACAGAGGAAGCAG...AACAGAGAACAGAGAACAGAGAAGAG  23513712
## 4          TTATTATATTATTATATTATTATAT...GCAGCCGTCGATTTGAGATATATGAA   1348131
```

Now I can generate data for 1 flow cell of 2×250bp sequencing on an Illumina MiSeq v3.

```
illumina(vars, out_prefix = "phylo_tree", seq_sys = "MSv3",
         read_length = 250, n_reads = 50e6)
```

From gene trees

Similar to the section above, the ultimate goal here is to test phylogeny reconstruction methods. The difference in this section is that instead of using a single, straightforward phylogeny, I use multiple gene trees per sequence. In the species used in these simulations, species 3 diverged from 1 and 2 at $t = 1.0$, where t indicates time into the past and is in units of $4N_0$ generations. Species 1 and 2 diverged at $t = 0.5$. I assume a recombination rate of $1/(4N_0)$ recombination events per sequence per generation. Because each sequence is a

different length and the length is required for including a recombination rate, I had to run `scrm` separately for each sequence. There are 4 diploid individuals sampled per species.

```
# Run scrm for one sequence size:
one_seq <- function(size) {
    sims <- scrm(
        paste("24 1",
              # Output gene trees:
              "-T",
              # Recombination:
              "-r 1", size,
              # 3 species with no ongoing migration:
              "-I 3", paste(rep("8", 3), collapse = " "), "0",
              # Species 3 derived from 1 (and 2) at time 1.0:
              "-ej 1.0 3 1",
              # Species 2 derived from 1 at time 0.5:
              "-ej 0.5 2 1"
        ))
    return(sims$trees[[1]])
}
# For all sequences:
gtrees <- list(trees = lapply(ref$sizes(), one_seq))
```

```
vars <- create_variants(ref, vars_gtrees(gtrees), sub, ins, del)
```

This results in the following `variants` object:

```
##                              << Variants object >>
## # Variants: 24
## # Mutations: 613,142
##
##                           << Reference genome info: >>
## < Set of 5 sequences >
## # Total size: 110,338,337 bp
##   name                            sequence                              length
## 3R         ACGGGACCGAGTATAGTACCAGTAC...CGTGACTGTTCGCATTCTAGGAATTC  32079331
## 3L         TAGGGAGAAATATGATCGCGTATGC...TTCTATGTTATTCCATGTTATTCTAT  28110227
## 2R         CTCAAGATACCTTCTACAGATTATT...TGGAGGTACGGAAACAGAATGAATTC  25286936
## 2L         CGACAATGCACGACAGAGGAAGCAG...AACAGAGAACAGAGAACAGAGAAGAG  23513712
## 4          TTATTATATTATTATATTATTATAT...GCAGCCGTCGATTTGAGATATATGAA   1348131
```

To store mutation information by diploid sample, the `write_vcf` function writes the `variants` object to a VCF file. It assigns haplotypes to diploid samples using a matrix for the `sample_matrix` argument, where each row represents the variants for a particular sample. Ploidy level is not limited by `jackalope`, so polyploidy is easily simulated by having more columns in the matrix. The matrix for this set of diploid samples might look something like

this (only showing the first 5 rows below):

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
## [4,]    7    8
## [5,]    9   10
##          ...
```

Using the matrix above assigns variants 1 and 2 to sample 1, variants 3 and 4 to sample 2, and so on. If the matrix is named `samp_mat`, we would use it to write the VCF file as follows:

```
write_vcf(vars, out_prefix = "var_gtrees",
          sample_matrix = samp_mat)
```

Now I can generate data for 1 flow cell of 2×250bp sequencing on an Illumina MiSeq v3.

```
illumina(ref, out_prefix = "phylo_gtrees", seq_sys = "MSv3",
         read_length = 250, n_reads = 50e6)
```

Conclusion

`jackalope` outperforms popular stand-alone programs for phylogenomic and HTS simulation and combines their functionalities into one cohesive package. Although it does not provide the in-built power of full pipelines like `NGSphy`, simulations using `jackalope` are much simpler to implement. Moreover, the flexibility of `jackalope` allows power users to manually provide inputs in many instances where the in-built functionality may not suffice. This package should inform research design for projects employing HTS, particularly those focusing on population genomics or phylogenomics. 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 evolutionary histories than most 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. doi: 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. doi: 10.18637/jss.v040.i08

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. doi: 10.1038/nrg.2016.57

Escalona, M., Rocha, S., & Posada, D. (2018). NGSphy: Phylogenomic simulation of next-generation sequencing data. *Bioinformatics*, *34*(14), 2506–2507. doi: 10.1093/bioinformatics/bty146

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. doi: 10.1093/bioinformatics/btq322

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

Fletcher, W., & Yang, Z. (2009). INDELible: A flexible simulator of biological sequence evolution. *Molecular Biology and Evolution*, *26*(8), 1879–1888. doi: 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. doi: 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. doi: 10.2183/pjab.60.95

Hayden, N., & Morgan, M. (2019). *Rhtslib: HTSlib high-throughput sequencing library as an r package*.

Holtgrewe, M. (2010). Masona read simulator for second generation sequencing data. *Technical Report Freie Universität Berlin*.

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

Hudson, R. R. (2002). Generating samples under a wright-fisher neutral model of genetic variation. *Bioinformatics*, *18*(2), 337–338. doi: 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. doi: 10.1007/bf01731581

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.

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. doi: 10.1080/00031305.1979.10482697

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., ... and, R. D. (2009). The sequence alignment/map format and SAMtools. *Bioinformatics*, *25*(16), 2078–2079. doi: 10.1093/bioinformatics/btp352

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. doi: 10.1093/molbev/msr011

Mallo, D., Oliveira Martins, L. de, & Posada, D. (2015). SimPhy: Phylogenomic simulation of gene, locus, and species trees. *Systematic Biology*, *65*(2), 334–344.

McElroy, K. E., Luciani, F., & Thomas, T. (2012). GemSIM: General, error-model based simulator of next-generation sequencing data. *BMC Genomics*, *13*(1), 74.

McTavish, E. J., Pettengill, J., Davis, S., Rand, H., Strain, E., Allard, M., & Timme, R. E. (2017). TreeToReads - a pipeline for simulating raw reads from phylogenies. *BMC Bioinformatics*, *18*(1). doi: 10.1186/s12859-017-1592-1

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

Morgan, M. (2019). *Zlibbioc: An r packaged zlib-1.2.5*. Retrieved from http://bioconductor.org/packages/release/bioc/html/Zlibbioc.html

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.

Rambaut, A., & Grass, N. C. (1997). Seq-gen: An application for the monte carlo simulation of DNA sequence evolution along phylogenetic trees. *Bioinformatics*, *13*(3), 235–238. doi:

10.1093/bioinformatics/13.3.235

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

Staab, P. R., & Metzler, D. (2016). Coala: An R framework for coalescent simulation. *Bioinformatics*. doi: 10.1093/bioinformatics/btw098

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.

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

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. doi: 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.

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.

Thurmond, J., Goodman, J. L., Strelets, V. B., Attrill, H., Gramates, L. S., Marygold, S. J., ... and, P. B. (2018). FlyBase 2.0: The next generation. *Nucleic Acids Research*, *47*(D1), D759–D765. doi: 10.1093/nar/gky1003

Walker, A. (1974). New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, *10*(8), 127. doi: 10.1049/el:19740097

Yang, Z. (2006). *Computational molecular evolution*. doi: 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.

Data Accessibility

`jackalope` is open source, under the MIT license. Its code is available on GitHub (https://github.com/lucasnell/jackalope), and the documentation can be found at https://jackalope.lucasnell.com. The package has been submitted to CRAN. The version used in this manuscript was 0.1.0. Code for the example usage, validation, and performance is available on GitHub at https://github.com/lucasnell/jlp_ms.

Author Contributions

L.A.N. conceived and designed the project, wrote the code, and wrote the manuscript.

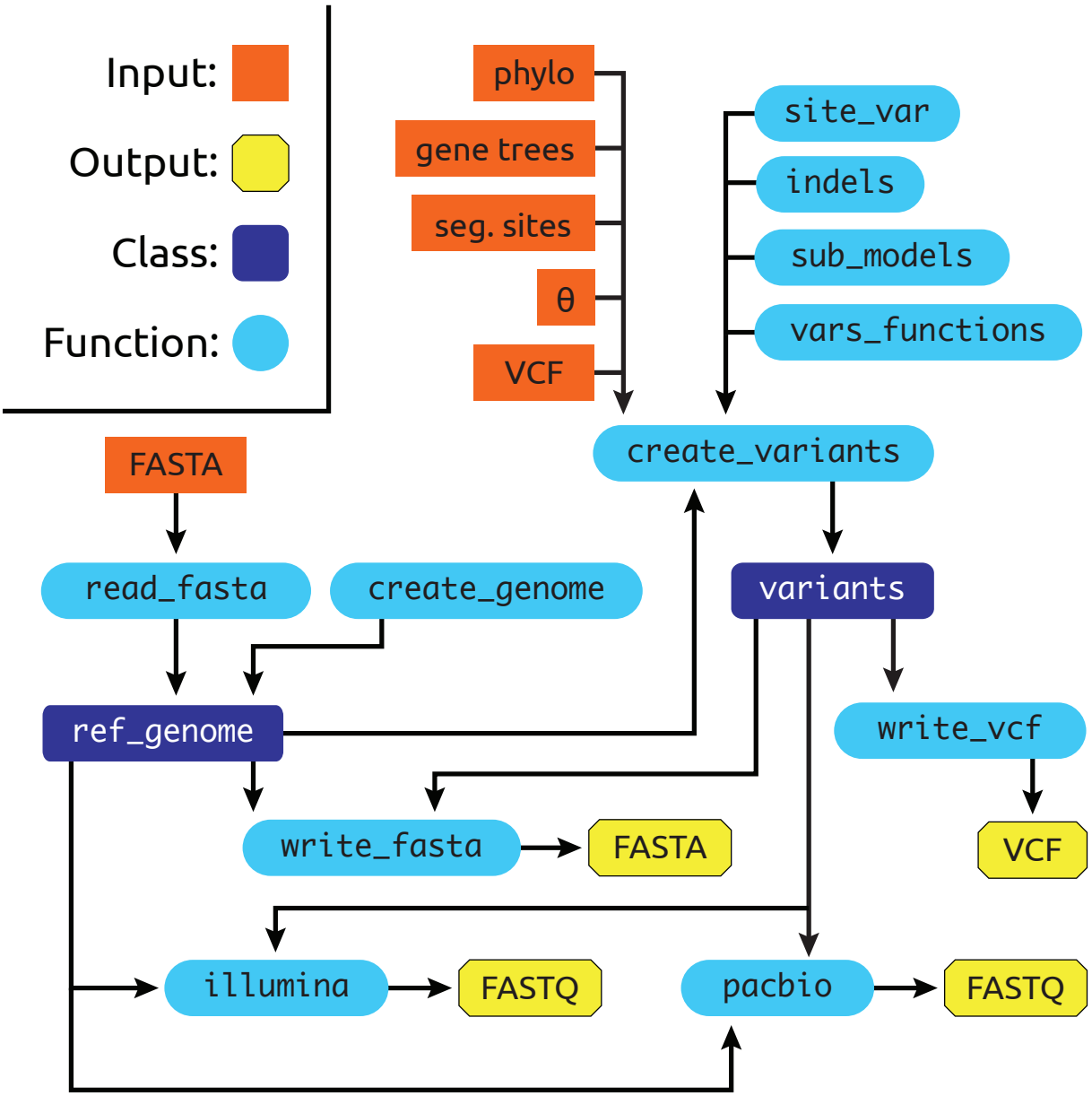

Figure 1: Overview of primary `jackalope` functions, classes, inputs, and outputs. Circles `sub_models` and `vars_functions` refer to multiple functions; see text for details. θ indicates the population-scaled mutation rate.

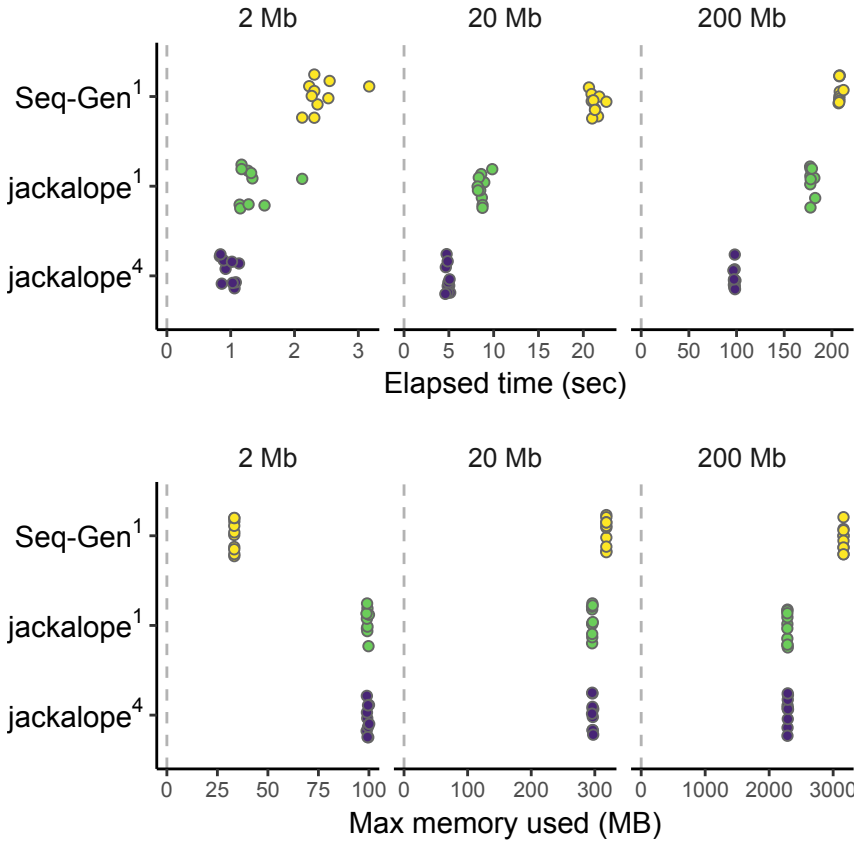

Figure 2: Performance comparison between `jackalope` and `Seq-Gen` in generating variants from a reference genome. Sub-panel columns indicate the size of the genome from which variants were generated. Superscripts in the y-axis labels indicate the number of threads used.

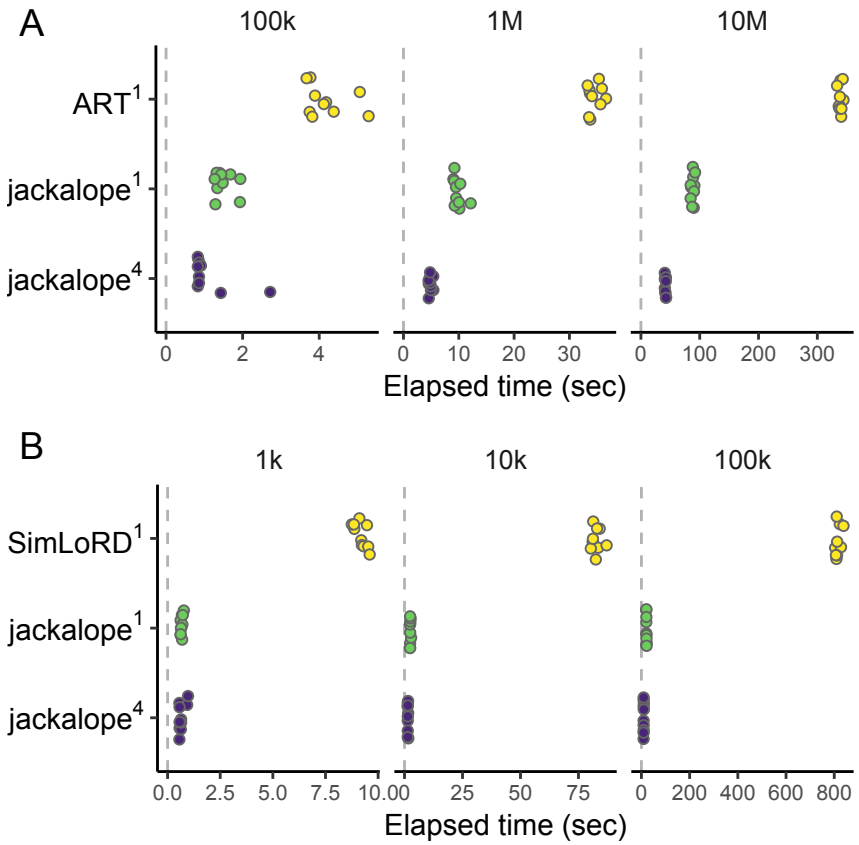

Figure 3: Performance comparison between `jackalope` and (A) `ART` in generating Illumina reads and (B) `SimLoRD` in generating PacBio reads. Sub-panel columns indicate the number of reads generated. Superscripts in the y-axis labels indicate the number of threads used.