Create structural variants from reference genome

Lucas Nell

03 April 2017

Contents

1 Data on which to base simulations

Reference:

Bickel, R. D., J. P. Dunham, and J. A. Brisson. 2013. Widespread selection across coding and noncoding DNA in the pea aphid genome. *G3: Genes|Genomes|Genetics* **3**:993–1001. Available from http://www.g3journal.org/content/3/6/993

The main points are below (all quotes are from p 996):

- "We sequenced 21 genetically distinct lines of pea aphids…"
- "…we calculated F_{st} levels across the genome, comparing 11 pea aphid lines from the Northeast US (New York and Massachusetts) and 10 from California. We observed no structure, with an overall F_{st} value of -0.021. We conclude that pea aphid populations in the United States function as a single, panmictic population."
- "[θ_w and θ_π] for all sites across the genome were 0.0050 and 0.0045, respectively"

2 Initial information

From the paper's information above, we have…

```
theta_w <- 0.0050
theta_pi <- 0.0045
```

I'm going to simulate a sample size of 10. (The period is prepended to avoid conflicts with other object names.)

```
.n <- 10
```

3 Calculating parameters from paper data

The two main pieces of information I want for the calculations are (1) the proportion of segregating sites and (2) some measure of how different individuals are at segregating sites.

3.1 Segregating sites

Watterson's (1975) estimator (θ_w) is as follows:

$$\theta_w = \frac{K}{a_n} \tag{1}$$

where K is the proportion of segregating sites. Variable a_n is below:

$$a_n = \sum_{i=1}^{n-1} \frac{1}{i} \tag{2}$$

where n is the number of individuals sampled. Thus the the proportion of segregating sites is simply $K = \theta_w a_n$.

For simplicity, I'm going to first make a function to compute a_n for a given value or values of n.

```
a_n <- function(n) {
    # Inner function to get a single "harmonic number"
    harm_n <- function(inner_n) {
        harm_n_vec <- 1 / 1:inner_n
        return(sum(harm_n_vec))
    }
    if (any((n %% 1) != 0)) stop("n must be entirely integers")
    sapply(n - 1, harm_n)
}
```

Now to compute the proportion of segregating sites for my chosen sample size of 10, I simply multiply θ_w and a_{10}.

```
(seg_sites <- round(theta_w * a_n(10), 5))
```

```
## [1] 0.01414
```

3.2 Diversity at segregating sites

Nei and Li's (1979) measure of nucleotide diversity, θ_π, is calculated using the following equation:

$$\theta_\pi = \sum_{ij} x_i x_j \pi_{ij} \tag{3}$$

Here, x_i and x_j represent the frequencies of the ith and jth unique sequences respectively and π_{ij} represents the proportion of divergent sequence between the ith and jth unique sequences.

If I assume that all lines will be unique sequences—a safe assumption if whole genomes are considered—then the above equation can be expressed as follows:

$$\theta_\pi = \frac{1}{n^2} \sum_{ij} \pi_{ij} \tag{4}$$

Then, since the number of total pairwise combinations between n sequences is $\binom{n}{2}$, we can calculate $\bar{\pi}$, the mean proportional sequence divergence between any two sequences, as such:

$$\bar{\pi} = \frac{\sum_{ij} \pi_{ij}}{\binom{n}{2}} \tag{5}$$

Some simple arithmetic gives us...

$$\sum_{ij} \pi_{ij} = \binom{n}{2} \bar{\pi} \tag{6}$$

Now I insert this into equation (4):

$$\theta_\pi = \frac{1}{n^2} \binom{n}{2} \bar{\pi} \tag{7}$$

Solving for $\bar{\pi}$ yields the following:

$$\bar{\pi} = \frac{\theta_\pi n^2}{\binom{n}{2}} \tag{8}$$

Since I've already calculated the proportion of segregated sites, I want the mean divergence at segregated sites only. (This improves computational and coding efficiency because I only have to worry about segregating sites.) To only consider segregated sites, I divide the whole expression by the proportion of segregated sites. This leaves me with the proportional nucleotide divergence between two sequences at segregating sites.

```r
(seg_div <- round({theta_pi * .n^2 / choose(.n, 2)} / seg_sites, 4))
```

```
## [1] 0.7072
```

The remainder of this document outlines code to carry out variant creation.

4 Functions to create variants

Loading packages:

```r
suppressPackageStartupMessages({
    library(magrittr)
    library(ggplot2)
    library(purrr)
    library(dplyr)
    library(ShortRead)
    library(gtools)
    library(parallel)
    library(Rcpp)
    library(RcppArmadillo)
})
```

Some of this code is written in C++, so I need to load that file.

```r
sourceCpp('variants.cpp')
```

4.1 Pairwise comparisons and making sequences

These functions (1) do pairwise comparisons for a vector of sequences (each sequence containing one nucleotide for each sample) and (2) create a sequence from a specified number of each nucleotide.

```r
pw_comp <- function(seq_vec) {
    seq_list <- .Internal(strsplit(seq_vec, '', FALSE, FALSE, FALSE))
    output <- sapply(seq_list,
                     function(.s) {
                         pw_mat <- t(combn(.s, 2))
                         diffs <- ifelse(pw_mat[,1] == pw_mat[,2], 0, 1)
                         return(mean(diffs))
                     })
    return(output)
}
make_seq <- function(a, c, g, t, return_vec = FALSE) {
    seq_vec <- c(rep('A', as.integer(a)), rep('C', as.integer(c)),
                 rep('G', as.integer(g)), rep('T', as.integer(t)))
    if (return_vec) return(seq_vec)
    return(paste(seq_vec, collapse = ''))
}
```

4.2 Create nucleotide frequency matrix

This function creates a nucleotide frequency matrix containing, in each row, nucleotide frequencies that (1) coincides closest with specified divergence at segregated sites and (2) sum to the number of samples. The order of the frequencies is not considered important, since I intend rows in this frequency matrix to be shuffled when used.

This takes ~1 min for `N=100`, ~2 sec for `N=50`, and << 1 sec for `N=10`.

```r
nt_freq <- function(N, divergence) {
    freq_mat <- combinations(N + 1, 4, 0:N, set = FALSE, repeats.allowed = TRUE)
    freq_sums <- rowSums(freq_mat)
    freq_mat <- freq_mat[freq_sums == N,]
    pw_divs <- pw_comp(apply(freq_mat, 1, function(x) do.call(make_seq, as.list(x))))
    min_diff_inds <- which(abs(pw_divs - divergence) == min(abs(pw_divs - divergence)))
    cat(sprintf('Minimum absolute difference from divergence = %s\n',
                format(min(abs(pw_divs - divergence)), scientific = TRUE, digits = 4)))
    return(freq_mat[min_diff_inds,])
}
```

4.3 Construct sequence objects

This function returns a `seq_obj` object containing an innner character vector, matrix, and numeric value. The first 2 inner objects contain information on each input sequence, and positions in output objects coincide with positions in the input `DNAStringSet`. So information in row `i` of `freq_len` and position `i` in `seqs` contains info for the `i`th sequence in the input `DNAStringSet`. The matrix `freq_len` contains two columns, one for the number of segregating sites and one for the length of the sequence. The character vector `seqs` contains the sequences themselves. The numeric value (`N`) contains the total number of sequences.

These `seq_obj` objects are used for downstream processes, and this function is not intended to be used outside another function.

```
setClass(Class = 'seq_obj', representation(seqs = 'character', freq_len = 'matrix',
                                           N = 'numeric'))
constr_objs <- function(dna_ss, seg_prop) {
    seqs <- as.character(dna_ss)
    seq_lens <- nchar(seqs)
    total_seg <- round(sum(seq_lens) * seg_prop, 0)
    rand_seqs <- sort(.Internal(sample(length(seq_lens), total_seg, replace = TRUE,
                                       prob = seq_lens)))
    freq_len <- table(factor(rand_seqs, levels = 1:length(seq_lens))) %>%
        as_data_frame %>%
        mutate(len = seq_lens) %>%
        select(n, len) %>%
        as.matrix
    if (any(freq_len[,1] > freq_len[,2])) {
        stop('Retry with different seed.')
    }
    seq_obj <- new('seq_obj', freq_len = freq_len, seqs = seqs, N = length(seqs))
}
```

The inner objects can be accessed as such, for a `seq_obj` named `Z`:

```
Z@freq_len
Z@seqs
Z@N
```

4.4 Choose random locations in a sequence

This function randomly choses sites for one sequence As input, it takes a row in a `seq_obj@freq_len` matrix. It outputs an integer vector.

```
one_sites <- function(.freq_len_row) {
    n_samps <- .freq_len_row[1]
    if (n_samps == 0) {
        return(integer(0))
    }
    seq_length <- .freq_len_row[2]
    out_locs <- .Internal(sample(seq_length, n_samps, FALSE, NULL))
    return(out_locs)
}
```

4.5 Changing sequences

This function changes one sequence to `N` sequences, where `N` is the number of samples. It also introduces variants at the supplied positions. The variants will conform to the nucleotide frequency matrix that should be created beforehand (see above for its description). Because this function will be run many times, I wrote it in C++ (function `cpp_change` in file `variants.cpp`). The R version is kept here to show the general process.

```
change_seqs <- function(seq, positions, freq_mat, n_samps) {

if (length(positions) == 0) return(rep(seq, n_samps))

    seq_vec <- unlist(.Internal(strsplit(seq, '', FALSE, FALSE, FALSE)))

    seq_mat <- matrix(rep(seq_vec, n_samps), nrow = n_samps, byrow = TRUE)

    ran_freq <- freq_mat[sample(nrow(freq_mat), 1), sample(4)]
    new_nucs <- sapply(positions,
                       function(i) do.call(make_seq, as.list(c(ran_freq, TRUE))))

    seq_mat[,positions] <- new_nucs

    seq_out <- apply(seq_mat, 1, paste0, collapse = '')

    return(seq_out)
}
```

4.6 Putting everything together

It creates a new, variant-filled `DNAStringSet` from one without variants. As inputs it takes a `DNAStringSet`, an estimate of divergence at segregating sites (like `seg_div` above), an estimate of the proportion of sites that are segregating, and the number of samples to output. I used `parallel::mcapply` to run this in parallel if desired. This part will have to be re-coded if parallel processing is desired on a Windows PC.

Using 3 cores, processing the entire aphid genome took ~8.5 min, and processing a digested, size-filtered aphid genome took ~6 sec.

```
make_variants <- function(dna_ss, divergence, seg_prop, n_samps, cores = 1) {

    freq_mat <- nt_freq(n_samps, divergence)
    seq_obj <- constr_objs(dna_ss, seg_prop)

    .one <- function(i) {
        freq_len_row <- seq_obj@freq_len[i,]
        seq <- seq_obj@seqs[i]
        sites <- one_sites(freq_len_row)
        new_seqs <- cpp_change(seq, sites - 1, freq_mat, n_samps)
        return(new_seqs)
    }
    if (cores > 1) {
        new_sites <- mclapply(1:seq_obj@N, .one, mc.cores = cores)
```

```r
    } else {
        new_sites <- lapply(1:seq_obj@N, .one)
    }
    new_sites <- c(new_sites, recursive = TRUE)
    dna_out <- DNAStringSet(new_sites)
    return(dna_out)
}
```

A version of this function is in the file `wr_variants.R`.

5 Session info

```
## Session info --------------------------------------------------------------
##  setting  value
##  version  R version 3.3.3 (2017-03-06)
##  system   x86_64, darwin13.4.0
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  tz       America/Chicago
##  date     2017-04-03
## Packages ------------------------------------------------------------------
##  package              * version     date       source
##  assertthat             0.1         2013-12-06 CRAN (R 3.3.0)
##  backports              1.0.5       2017-01-18 CRAN (R 3.3.2)
##  Biobase              * 2.34.0      2016-10-18 Bioconductor
##  BiocGenerics         * 0.20.0      2016-10-18 Bioconductor
##  BiocParallel         * 1.8.1       2016-10-30 Bioconductor
##  Biostrings           * 2.42.1      2016-12-01 Bioconductor
##  bitops                 1.0-6       2013-08-17 CRAN (R 3.3.0)
##  colorspace             1.3-2       2016-12-14 CRAN (R 3.3.2)
##  DBI                    0.6         2017-03-09 CRAN (R 3.3.2)
##  devtools               1.12.0      2016-06-24 CRAN (R 3.3.0)
##  digest                 0.6.12      2017-01-27 CRAN (R 3.3.2)
##  dplyr                * 0.5.0       2016-06-24 CRAN (R 3.3.0)
##  evaluate               0.10        2016-10-11 CRAN (R 3.3.0)
##  GenomeInfoDb         * 1.10.3      2017-02-07 Bioconductor
##  GenomicAlignments    * 1.10.1      2017-03-18 Bioconductor
##  GenomicRanges        * 1.26.4      2017-03-18 Bioconductor
##  ggplot2              * 2.2.1       2016-12-30 CRAN (R 3.3.2)
##  gtable                 0.2.0       2016-02-26 CRAN (R 3.3.0)
##  gtools               * 3.5.0       2015-05-29 CRAN (R 3.3.0)
##  htmltools              0.3.5       2016-03-21 CRAN (R 3.3.0)
##  hwriter                1.3.2       2014-09-10 CRAN (R 3.3.0)
##  IRanges              * 2.8.2       2017-03-18 Bioconductor
##  knitr                  1.15.1      2016-11-22 CRAN (R 3.3.2)
##  lattice                0.20-35     2017-03-25 CRAN (R 3.3.2)
##  latticeExtra           0.6-28      2016-02-09 CRAN (R 3.3.0)
##  lazyeval               0.2.0       2016-06-12 CRAN (R 3.3.0)
##  magrittr             * 1.5         2014-11-22 CRAN (R 3.3.0)
##  Matrix                 1.2-8       2017-01-20 CRAN (R 3.3.3)
##  memoise                1.0.0      2016-01-29 CRAN (R 3.3.0)
##  munsell                0.4.3      2016-02-13 CRAN (R 3.3.0)
##  plyr                   1.8.4      2016-06-08 CRAN (R 3.3.0)
##  purrr                * 0.2.2      2016-06-18 CRAN (R 3.3.0)
##  R6                     2.2.0      2016-10-05 CRAN (R 3.3.0)
##  RColorBrewer           1.1-2      2014-12-07 CRAN (R 3.3.0)
##  Rcpp                 * 0.12.10    2017-03-19 CRAN (R 3.3.2)
##  RcppArmadillo        * 0.7.700.0.0 2017-02-08 CRAN (R 3.3.2)
##  RCurl                  1.95-4.8   2016-03-01 CRAN (R 3.3.0)
##  rmarkdown              1.4        2017-03-24 CRAN (R 3.3.2)
##  rprojroot              1.2        2017-01-16 CRAN (R 3.3.2)
##  Rsamtools            * 1.26.1     2016-10-22 Bioconductor
##  S4Vectors            * 0.12.2     2017-03-18 Bioconductor
##  scales                 0.4.1      2016-11-09 CRAN (R 3.3.2)
##  ShortRead            * 1.32.1     2017-03-18 Bioconductor
##  stringi                1.1.3      2017-03-21 CRAN (R 3.3.2)
##  stringr                1.2.0      2017-02-18 CRAN (R 3.3.2)
##  SummarizedExperiment * 1.4.0      2016-10-18 Bioconductor
##  tibble                 1.2        2016-08-26 CRAN (R 3.3.0)
##  withr                  1.0.2      2016-06-20 CRAN (R 3.3.0)
##  XVector              * 0.14.1     2017-03-18 Bioconductor
##  yaml                   2.1.14     2016-11-12 CRAN (R 3.3.2)
##  zlibbioc               1.20.0     2016-10-18 Bioconductor
```