Two Locus General Statistics in tskit

Lloyd Kirk

March 21, 2023

1 Background

The intention of this document is to describe the changes to tskit to provide a generalized framework for computing two-locus statistics for branches and sites.

Much of the design proposal is taken from this PR There has been a bit of discussion here as well.

2 Goal/Scope

Ultimately, we intend to deprecate the LD calculator that is currently implemented in tskit and implement a general framework for two-locus statistics. This framework will create an interface for implementing new summary statistics and implement the following features:

- Sites with more than 1 mutation (multiallelic sites)
- Polarization, where applicable
- Branch Statistics
- LD Stats beyond r^2

3 Implementation

We have been trying to follow the general design pattern laid out in the C api. Most of the initial design documentation will revolve around creating the necessary C interfaces to consume from the python api.

Some of this documentation is based on my (LK's) interpretation of the design patterns in the existing code. I have no insight into how things might change apart from the scattered TODOs in the code. In short, please feel free to correct my assumptions/understandings.

3.1 Outer Layer

We propose to add more functionality to `tsk_treeseq_general_stat`, which is the main entrypoint for computing stats from tree sequences. Within `tsk_treeseq_general_stat`, we will add some more conditionals in the form of `tsk_flags_t`.

```
  bool stat_two_site = !!(options & TSK_STAT_TWO_SITE);
  bool stat_branch = !!(options & TSK_STAT_BRANCH);
```

These flags would dispatch our entrypoints for computing two site/branch statistics.

```
static int
tsk_treeseq_two_branch_general_stat(
        const tsk_treeseq_t *self,
        tsk_size_t state_dim,
        const double *sample_weights,
        tsk_size_t result_dim,
        general_stat_func_t *f,
        void *f_params,
        tsk_size_t num_windows,
        const double *windows,
        tsk_flags_t options,
        double *result
)
```

4 Two Site Statistics

We would provide a similar interface to computing `site_general_stats`, but the tree traversal algorithm will differ, providing haplotype counts instead of site counts.

```
static int
tsk_treeseq_two_site_general_stat(
        const tsk_treeseq_t *self,
        tsk_size_t state_dim,
        const double *sample_weights,
        tsk_size_t result_dim,
        general_stat_func_t *f,
        void *f_params,
        tsk_size_t num_windows,
        const double *windows,
        tsk_flags_t options,
        double *result
)

static int
compute_general_two_stat_site_result(
        tsk_site_t *site_a,
        tsk_site_t *site_b,
        double *state,
        tsk_size_t state_dim,
        tsk_size_t result_dim,
        general_stat_func_t *f,
        void *f_params,
        double *total_weight,
        bool polarised,
        double *result,
)
```

5 Branch Statistics

6 Summary Functions

Statistic	Polarization	Normalization	Equation
D	Polarized	Total	$D = f_{ab} - f_a f_b$
D'	Polarized	Haplotype Weighted	$D' = \frac{D}{D_{max}}$
D^2	Unpolarized	Total	$D^2 = D^2$
D_z	Unpolarized	Total	$D_z = D(1 - 2f_a)(1 - 2f_b)$
π_2	Unpolarized	Total	$\pi_2 = f_a f_b (1 - f_a)(1 - f_b)$
r	Polarized	Haplotype Weighted	$r = \frac{D}{\sqrt{f_a f_b (1-f_a)(1-f_b)}}$
r^2	Unpolarized	Haplotype Weighted	$r^2 = \frac{D^2}{f_a f_b (1-f_a)(1-f_b)}$

Where D_{max} is defined as:

$$D_{max} = \begin{cases} \min\{f_a(1 - f_b), f_b(1 - f_b)\} & \text{if } D >= 0 \\ \min\{f_a f_b, (1 - f_b)(1 - f_b)\} & \text{otherwise} \end{cases}$$

6.1 Summary Function Signature

Two locus statistics need to know the number of AB, Ab, and aB haplotypes. They also need to know the total number of haplotypes being considered in order to properly convert the counts of each haplotype to proportions.

```
static int
summary_func(int w_AB, int w_Ab, int w_aB, int n)

two_site_summary_func(
        tsk_size_t state_dim,
        const double *state,
        tsk_size_t TSK_UNUSED(result_dim),
        double *result,
        void *params
)
```

6.2 Normalization

In our testing of summary functions, we found that the appropriate normalization procedure can vary depending on the summary function. We've settled on two normalization procedures: “Haplotype Weighted” and “Total”.

6.2.1 Hapltype Weighted

$$\sum_{i=1}^{n}\sum_{j=1}^{m} p(A_iB_j)F_{ij}$$

where F is the summary function and $p(A_iB_j)$ is the frequency of haplotype A_iB_j. This method can be found in [1]. We apply this

6.2.2 Total

In the “Total” normalization method, we simply divide by the number of haplotypes that we’ve visited. If we’re

$$\frac{1}{(n-\mathbb{1}_p)(m-\mathbb{1}_p)}\sum_{i=1}^{n}\sum_{j=1}^{m} F_{ij}$$

where $\mathbb{1}_p$ is an indicator function conditioned on whether or not our statistic is polarized.

6.3 Evaluation

To ensure the correctness of our implementation, we have devised a number of test scenarios that will produce data at the theoretical limits of the statistics we’ve implemented.

6.3.1 Test cases

Table

6.3.2 Polarized

We’ll begin with the polarized statistics. D and D' sum to zero when they are unpolarized, so we are only providing a polarized method to compute them. We can test the correctness of D by computing D with two alleles that are in full repulsion:

Name	$\begin{pmatrix} SiteA \\ SiteB \end{pmatrix}$
Correlated	$\begin{pmatrix} 0 & 1 & 1 & 0 & 2 & 2 & 1 & 0 & 1 \\ 1 & 2 & 2 & 1 & 0 & 0 & 2 & 1 & 2 \end{pmatrix}$
Uncorrelated	$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \end{pmatrix}$
Correlated Biallelic	$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$
Uncorrelated Biallelic	$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$
Repulsion Biallelic	$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$

Table 1: **Test cases for validating statistics.** In each case, we have an A and B site, representing the two sites under consideration for computaiton of our statistics.

6.3.3 Notation

In the following section, we use pairs of 1-d row vectors to describe the terminal allelic state in a given pair of trees. For example, if we're given a tree like the one shown in Figure 1.

In this example, the state matrix is a 2×8 matrix, representing the terminal state of each sample in the tree. For Figure 1, our state matrix looks like:

$$\begin{pmatrix} 2 & 2 & 3 & 3 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 2 & 2 & 2 & 0 \end{pmatrix}$$

References

[1] Honghua Zhao, D Nettleton, and Jack CM Dekkers. Evaluation of linkage disequilibrium measures between multi-allelic markers as predictors of linkage disequilibrium between single nucleotide polymorphisms. *Genetics Research*, 89(1):1–6, 2007.

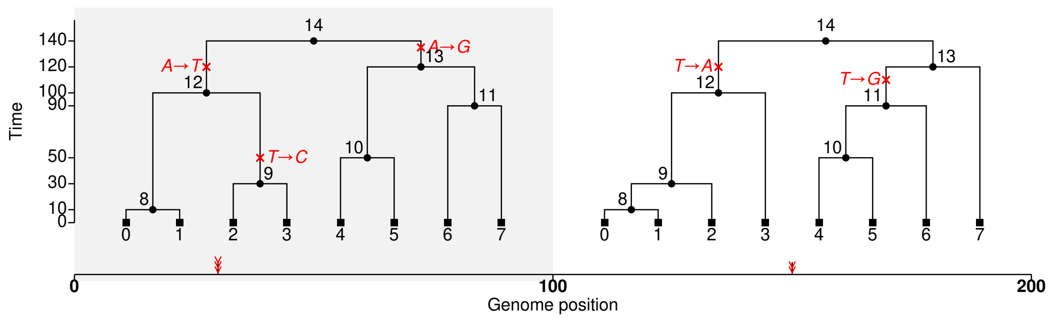

Figure 1: **An example of a tree sequence used to compute two-locus statistics.** The left tree has one site with four states and the right tree has one site with three states.