Extracting Two-Locus Statistics From Simulated Coalescent Trees

Michael Huang, 260739391

COMP 401 - Project in Biology and Computer Science
McGill University
August 23, 2019

Contents

1 Introduction

Simulation studies play an important role in better understanding population genetics. There are three primary simulation methods: backward-time, forward-time, and resampling [5]. Backward-time approach, or coalescent simulation, is a model that looks back in time and traces lineages from samples to a common ancestor. A coalescent can be represented as a tree, where the leaf nodes are the sampled individuals, and the internal nodes represent common ancestors.

Msprime is a python library that efficiently simulates coalescent trees under different evolutionary scenarios [1]. Trees generated by msprime are sparsely encoded as a set of coalescence records allowing msprime to outperform other backwards-time simulation methods. This efficiency is critical when performing simulations of regions hundreds of megabases long with hundreds of thousands of individuals. The trees generated by msprime contain underlying statistics which may be of interest to population genetics researchers.

2 Background

Expected two-locus statistics can be extracted from trees simulated in msprime by taking pairs of trees in a simulated tree sequence and comparing TMRCA's (time to most recent ancestor).

Linkage disequilibrium is the measure of nonrandom association of alleles at different loci, and is widely used to infer properties of population history (for example, historical population bottlenecks [2]). One common measure of LD is r^2, and is the squared correlation coefficient between alleles A and B. Where f_A and f_B are the allele frequencies at the left and right loci.

$$r^2 = \frac{D^2}{f_A(1-f_A)f_B(1-f_B)}$$

Because there isn't a simple analytical expression for the expectation of r^2, we can compute the ratio of expectations, σ_d^2 which approximates r^2 given intermediate allele frequencies [3].

$$\sigma_d^2 = \frac{\mathbb{E}[D^2]}{\mathbb{E}[f_A(1-f_A)f_B(1-f_B)]}$$

This has been shown to converge with $\mathbb{E}[r^2]$ as the population scaled recombination rate approaches infinity [4].

We can also express σ_d^2 in terms of covariances of TMRCA's (as shown by McVean, 2002 [2], where $t_{x(ij)}$ refers to the TMRCA of samples i and j at the left tree x and $t_{y(ik)}$ refers to the TMRCA of samples i and k at the right tree y, giving us the formulation:

$$\sigma_d^2 = \frac{\mathrm{Cov}[t_{x(ij)},t_{y(ij)}]-2\mathrm{Cov}[t_{x(ij)}t_{y(ik)}]+\mathrm{Cov}[t_{x(ij)}t_{y(kl)}]}{\mathbb{E}[t]^2+\mathrm{Cov}[t_{x(ij)},t_{y(kl)}]}$$

3 Methods

Using McVean's formulation, we can express the Covariances in the numerator in terms of Expectations, for example with $t_{x(ij)}t_{y(ij)}$:

$$\begin{aligned}\mathrm{Cov}[t_{x(ij)}, t_{y(ij)}] &= \mathbb{E}[(t_{x(ij)} - \mathbb{E}[t_{x(ij)}])(t_{y(ij)} - \mathbb{E}[t_{y(ij)}])] \\ &= \mathbb{E}[t_{x(ij)}t_{y(ij)}] - \mathbb{E}[t_{x(ij)}]\mathbb{E}[t_{y(ij)}]\end{aligned}$$

Applied to McVean's numerator, we get:

$$\begin{aligned}&= \mathbb{E}[t_{x(ij)}t_{y(ij)}] - \mathbb{E}[t_{x(ij)}]\mathbb{E}[t_{y(ij)}]- \\ &2(\mathbb{E}[t_{x(ij)}t_{y(ik)}] - \mathbb{E}[t_{x(ij)}]\mathbb{E}[t_{y(ik)}])+ \\ &\mathbb{E}[t_{x(ij)}t_{y(kl)}] - \mathbb{E}[t_{x(ij)}]\mathbb{E}[t_{y(kl)}]\end{aligned}$$

We rearrange to:

$$\begin{aligned}= \mathbb{E}[t_{x(ij)}t_{y(ij)}] - 2\mathbb{E}[t_{x(ij)}t_{y(ik)}] + \mathbb{E}[t_{x(ij)}t_{y(kl)}] \\ -\mathbb{E}[t_{x(ij)}]\mathbb{E}[t_{y(ij)}] + 2\mathbb{E}[t_{x(ij)}]\mathbb{E}[t_{y(ik)}]) \\ -\mathbb{E}[t_{x(ij)}]\mathbb{E}[t_{y(kl)}]\end{aligned}$$

We can make the assumption that samples are exchangeable (as sampling ij means sampling two different samples, thus sampling ik is the

same, sampling two different samples, and how we name them doesn't matter), such that:

$$\mathbb{E}[t_{x(ij)}] = \mathbb{E}[t_{x(ik)}] = \mathbb{E}[t]$$

This allows us to cancel terms ans simplify the numerator to:
$= \mathbb{E}[t_{x(ij)}t_{y(ij)}] - 2\mathbb{E}[t_{x(ij)}t_{y(ik)}] + \mathbb{E}[t_{x(ij)}t_{y(kl)}]$

Now we can consider the denominator:

$$= \mathbb{E}[t]^2 + \text{Cov}[t_{x(ij)}, t_{y(kl)}]$$
$$= \mathbb{E}[t]^2 + \mathbb{E}[t_{x(ij)}t_{y(kl)}] - \mathbb{E}[t_{x(ij)}]\mathbb{E}[t_{y(kl)}]$$

Likewise for the numerator, we can simplify the denominator to:

$$= \mathbb{E}[t_{x(ij)}t_{y(kl)}]$$

Combining these two equations we get:

$$\sigma_d^2 = \frac{\mathbb{E}[t_{x(ij)}t_{y(ij)}] - 2\mathbb{E}[t_{x(ij)}t_{y(ik)}] + \mathbb{E}[t_{x(ij)}t_{y(kl)}]}{\mathbb{E}[t_{x(ij)}t_{y(kl)}]}$$

3.1 Naive Method of Calculating Expected σ_d^2

In order to calculate the expected σ_d^2 for a given tree sequence and recombination distance, we take expectations for that recombination distance. We do this by considering all pairs of sites genome-wide that are separated by that distance. Using coalescence times, we can consider the three components of the following equation for D^2 from the numerator of σ_d^2 (also note that the third term is also the denominator for σ_d^2):

$$D^2 = \mathbb{E}[t_{x(ij)}t_{y(ij)}] - 2\mathbb{E}[t_{x(ij)}t_{y(ik)}] + \mathbb{E}[t_{x(ij)}t_{y(kl)}]$$

For brevity I will refer to the three terms as $\text{E}_{\text{ij,ij}}$ $\text{E}_{\text{ij,ik}}$ and $\text{E}_{\text{ij,kl}}$ respectively. We can calculate these three terms for a given sequence of coalescent trees by multiplying TMRCA's between different samplings.

Trees are defined over non-overlapping segments of the simulated genome. So for a given recombination rate, there will be some length of the genome that the recombination distance window being considered falls on both trees. $\text{E}_{\text{ij,ij}}$ refers to when the two samples on the left tree are the same as the two samples on the right tree. Below, $l_{x,y}$ refers to the length of the segment over trees x and y. When normalized over all pairings we get:

$$E_{ij,ij} = \sum_{x,y} l_{x,y} \frac{1}{\binom{n}{2}} \sum_{i,j} t_{x(ij)}t_{y(ij)}$$
$$E_{ij,ik} = \sum_{x,y} l_{x,y} \frac{1}{n(n-1)(n-2)} \sum_{i,j,k} t_{x(ij)}t_{y(ik)}$$
$$E_{ij,kl} = \sum_{x,y} l_{x,y} \frac{1}{\binom{n}{2}\binom{n-2}{2}} \sum_{i,j,k,l} t_{x(ij)}t_{y(kl)}$$

If we hold the number of pairs x, y constant, and only consider the inner summations for complexity, we get: $O(n^2)$, $O(n^3)$, and $O(n^4)$ respectively.

If we consider the number of comparisons as we traverse the pairs of trees in the tree sequence to be some set of tree pairings, let T represent the number of trees in the tree sequence and α be some value relating the recombination distance and the number of trees, the overall computational complexity of calculating σ_d^2 using the naive method is: $O(\alpha T n^4)$.

The pseudocode for this algorithm can be found in Appendix A.

3.2 Efficiently Calculating Expected LD Using Nodes

It is far too computationally expensive to calculate these statistics using the larger sequence lengths and reasonably large sample sizes, thus it was necessary to use a more efficient approach.

Due to the nature of these simulated trees, many TMRCA multiplications are repeated. Additionally, these TMRCA's are encoded in the nodes themselves. This enables a more efficient method to calculate these statistics by iterating over the nodes of trees. For a sample size n, each tree has $n-1$ internal nodes,

meaning if we iterate comparing nodes between trees x and y, we can operate in $O(n^2)$ for every pair of trees compared, giving us the overall computation complexity $O(\alpha T n^2)$.

Given an internal node n from tree x and an internal node k from tree y, we can view the leaves of their children as sets n_l, n_r, k_l, k_r, where n_l refers to all the leaves under the left child under node n. Consider the 4 combinations of intersections of the sets:

$$\begin{aligned} m_l &= n_l \cap k_l \\ m_r &= n_r \cap k_r \\ c_1 &= n_l \cap k_r \\ c_2 &= n_r \cap k_l \end{aligned}$$

We know the number of paired combinations under each node is:

$$\begin{aligned} \text{pairs}_n &= C(n_l) \times C(n_r) \\ \text{pairs}_k &= C(k_l) \times C(k_r) \end{aligned}$$

And the total number of possible paired combinations between the two nodes is:

$$T = \text{pairs}_n \times \text{pairs}_k$$

There exist exactly 3 types of paired combinations between the two nodes, both pairs are the same samples, both pairs share exactly one sample, or the pairs share no samples (i, j, k, l) all unique. Let us define the count of leaves in a set to be $\text{count}(ij, ij) = C(ij, ij)$:

$$T = C(ij, ij) + C(ij, ik) + C(ij, kl)$$

The number of ij, ij pairings can be found the counting the number of combinations we can make between the left matches and right matches, as well as the number of combinations we can make between the two cross matches:

$$C(ij, ij) = C(m_l) \times C(m_r) + C(c_1) \times C(c_2)$$

The counting of ij, ik pairings is more involved. We can fix the match between the two trees to one set of matches. Once fixed, we need to count the number of combinations for the opposite pairing to mismatch, which can be counted as the total number of pairings minus those that match.

$$C(ij, ik) = C(ij, ik)_{m_l} + C(ij, ik)_{m_r} + C(ij, ik)_{c_1} + C(ij, ik)_{c_2}$$

We can define the 4 fixed counts by:

$$\begin{aligned} C(ij, ik)_{m_l} &= (C(n_r) \times C(k_r) - C(m_r))C(m_l) \\ C(ij, ik)_{m_r} &= (C(n_l) \times C(k_l) - C(m_l))C(m_r) \\ C(ij, ik)_{c_1} &= (C(n_r) \times C(k_l) - C(c_2))C(c_1) \\ C(ij, ik)_{c_2} &= (C(n_l) \times C(k_r) - C(c_1))C(c_2) \end{aligned}$$

Finally, we are able to extract the remaining number of ij, kl pairings:

$$C(ij, kl) = \text{pairs}_n \times \text{pairs}_k - C_{ij,ij} - C_{ij,ik}$$

Since we know that all pairings of left leaves and right leaves beneath node n and node k respectively have TMRCA's, for any left leaf i, m and any right leaf j, n:

$$t_{x(ij)} = t_x(n) \quad t_{y(mn)} = t_y(k)$$

We are thus able to simplify all comparisons for pairs under nodes n and k by weighting based on the number of combinations of pairs we can get:

$$\begin{aligned} E(x_n, y_k)_{(ij,ij)} &= C_{ij,ij} \times t_x(n) \times t_y(k) \\ E(x_n, y_k)_{(ij,ik)} &= C_{ij,ik} \times t_x(n) \times t_y(k) \\ E(x_n, y_k)_{(ij,kl)} &= C_{ij,kl} \times t_x(n) \times t_y(k) \end{aligned}$$

The pseudocode for this algorithm can be found in Appendix B.

3.3 Reducing Redundent Calculations in the Simple Node Based Approach

Additional optimization can be made because of the nature of Coalescent Trees. As we traverse the Tree Sequence, we tend to only change a few nodes each time we change to a new tree. And in doing so, we only actually need to recalculate node comparisons for the nodes that have changed.

As we slide our window across the tree sequence, we will have to change either left or right trees, but only one at a time. The nodes that have to be calculated at this position are the nodes that have changed from the old to the new tree, for which the interval they are valid lies between the position that node was last updated, and the end of the old tree.

Assuming that we have an adequately large sample size, on average, the trees will tend to be balanced binary trees. Moving a branch among the lowest nodes will cause all nodes above the affected node to require recalculation. Since we assume the trees, are balanced, the height is thus $O(log(n))$. We need to recompute with all nodes on the previous tree, and need to compute all nodes at the end to catch the last interval, we get an overall computational complexity of $O(n^2 + Tnlog(n))$.

The pseudocode for this algorithm can be found in Appendix C.

3.4 Efficient Internal Representation and Traversing Trees

Msprime encodes the changes from tree to tree as a set of Edge Differences (Edge Diffs). These Edge Diffs report which edges have been removed from the previous tree, and which new edges were added and include the parent and child nodes, which is sufficient to update internal representations.

Beyond the first tree, there are, on average, three edges removed and three edges added while traversing between trees. These can be events where a branch was moved between nodes (4), a node was substituted (3), or the root was changed (2).

Rather than accessing the trees statistics themselves repeatedly, we can keep lists containing the information we need, and update them as we traverse the trees. This approach is based on the Allele Frequency Spectrum calculator originally implemented in tskit, where the parents and number of children for each node were tracked. This was done by having the index of the list represent the node id.

The information tracked for each node (each is tracked twice, once for the left and once for the right tree) is:

- The nodes parent (-1 for the root and nodes not in the present tree)
- Direct children (2 distinct children for internal nodes in the present tree)
- Leaves under the node (sets of leaves)
- Update position (the last time the node was recalculated, also is the left bound for the relevant interval)

The leaves are implemented in python as a list of sets to make propagation of changes up the tree quicker. The parents and update positions are lists of integers and floats respectively. The children are implemented in lists of lists so children can be appended and removed quickly. Additionally, the time for each node in the tree is stored in a list, allowing for quicker retrieval from a list rather than getting the time from the tree sequence repeatedly.

4 Results

In order to verify the correctness of my method, I computed McVean's [2] expected σ_d^2 and Ohta and Kimura's [3] expected σ_z^2:

$$\rho = 4N_e r_{dist}$$
$$\sigma_d^2 = \frac{10 + \rho}{22 + 13\rho + \rho^2} = \frac{D^2}{\pi_2}$$
$$\sigma_z^2 = \frac{8}{22 + 13\rho + \rho^2} = \frac{D_z}{\pi_2}$$

To compare, 5000 randomized tree sequences were simulated and the expected statistics were calculated using (where the mutation rate cancels out):

$$D^2 = u^2(E_{ij,ij} - 2E_{ij,ik} + E_{ij,kl})$$
$$D_z = 4u^2(E_{ij,ik} - E_{ij,kl})$$
$$\pi_2 = u^2 E_{ij,kl}$$

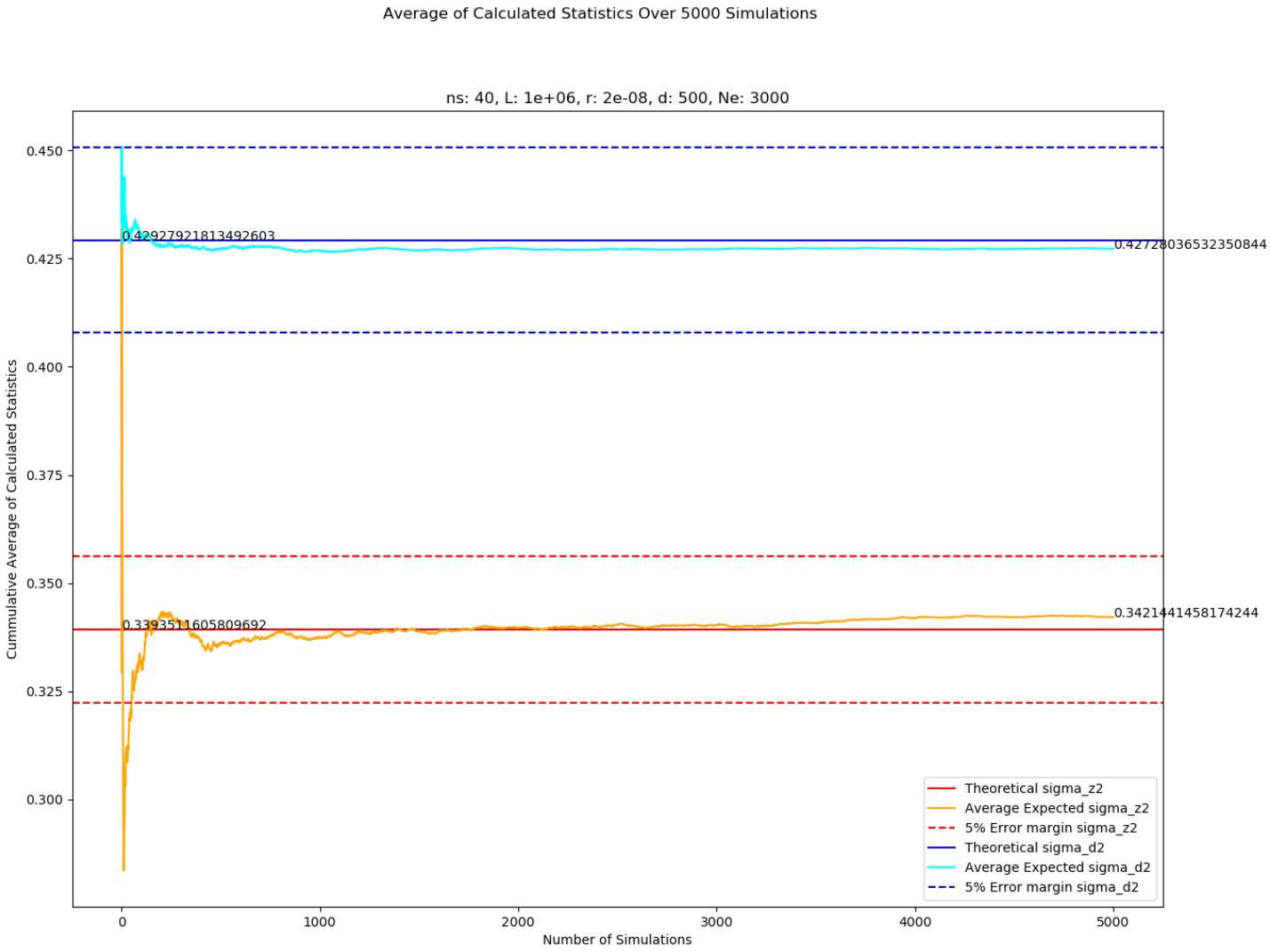

Figure 1: Moving average of calculated expected statistics over many simulated trees

Over 5000 simulations, the average calculated expected σ_d^2 falls under 0.5% error. The average calculated expected σ_d^2 falls under 0.9% error.

In order to compare the speeds of the three algorithms, I ran 100 simulations with randomized tree sequences with different parameters for: sequence length (L), recombination rate (r), recombination distance in base pairs (d), effective population size (Ne), and sample size (ns).

The average time across 100 simulations was plotted against either sequence length, effective population size, and sample size.

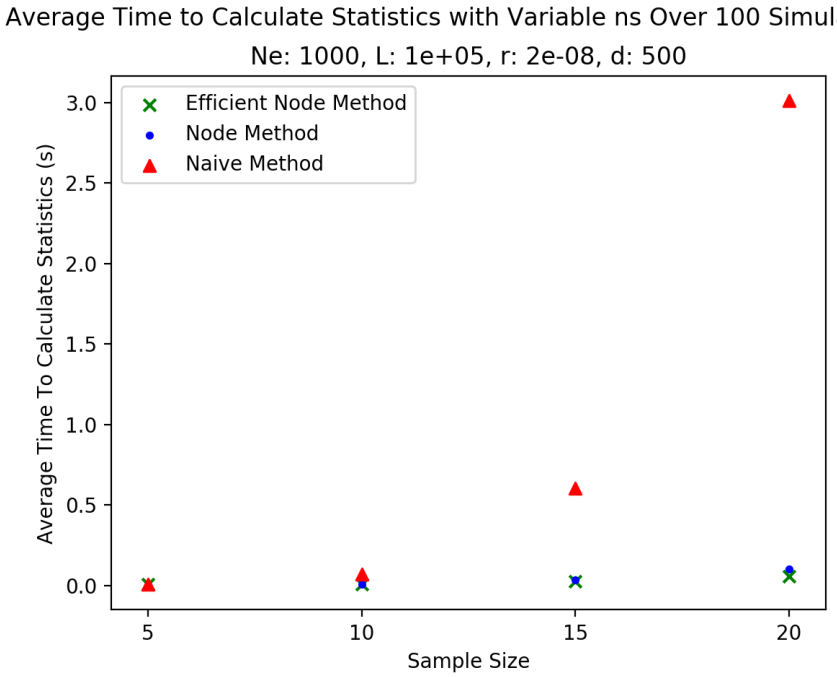

Figure 2: Average time with all 3 Algorithms at low sample sizes.

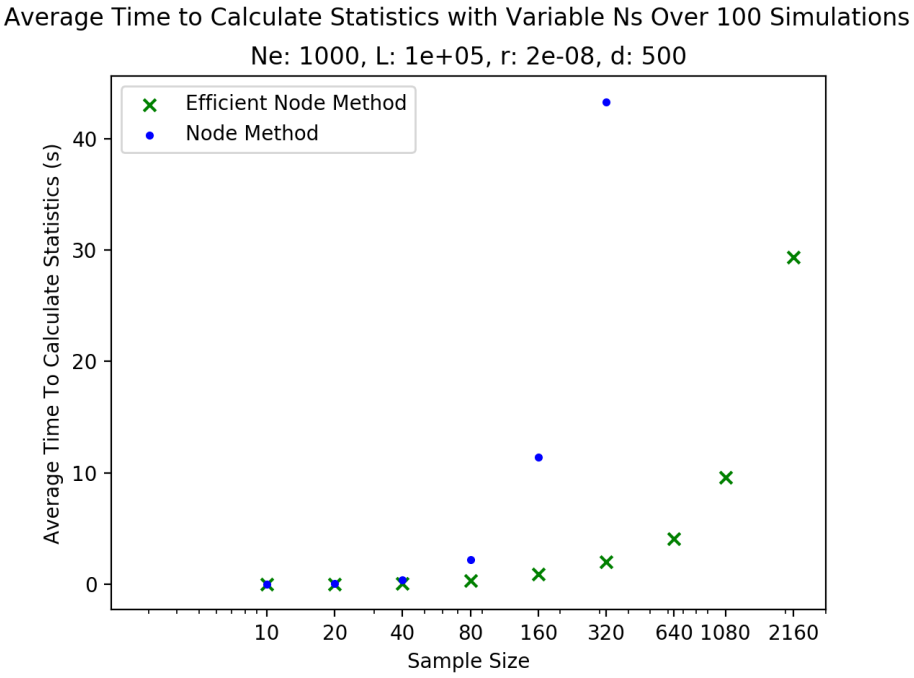

Figure 3: Average time for larger sample sizes without Naive algorithm. The Simple Node Method wasn't computed for sample sizes past 320 due to time constraints.

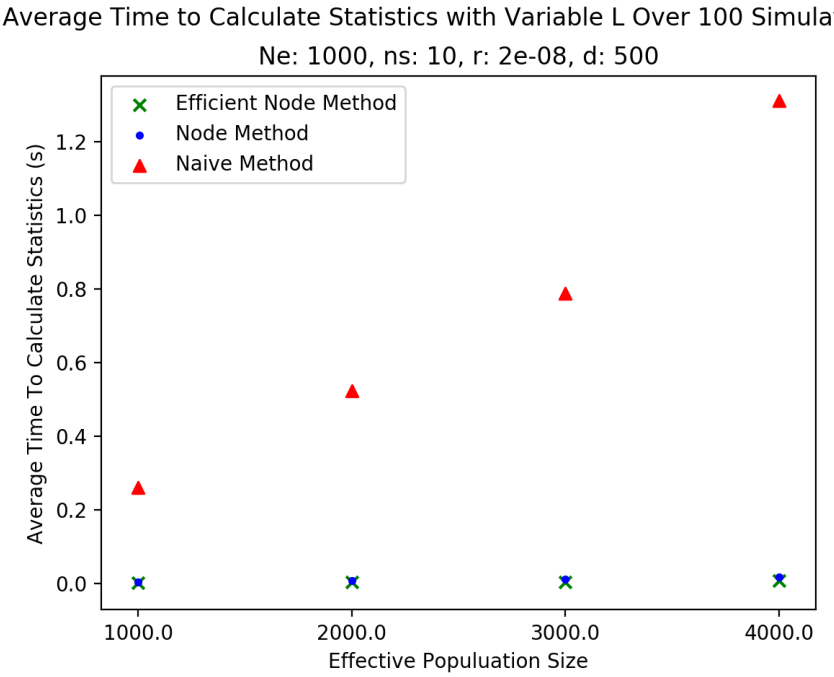

Figure 4: Average time for all 3 Algorithms with variable Effective Population Sizes at low sample size.

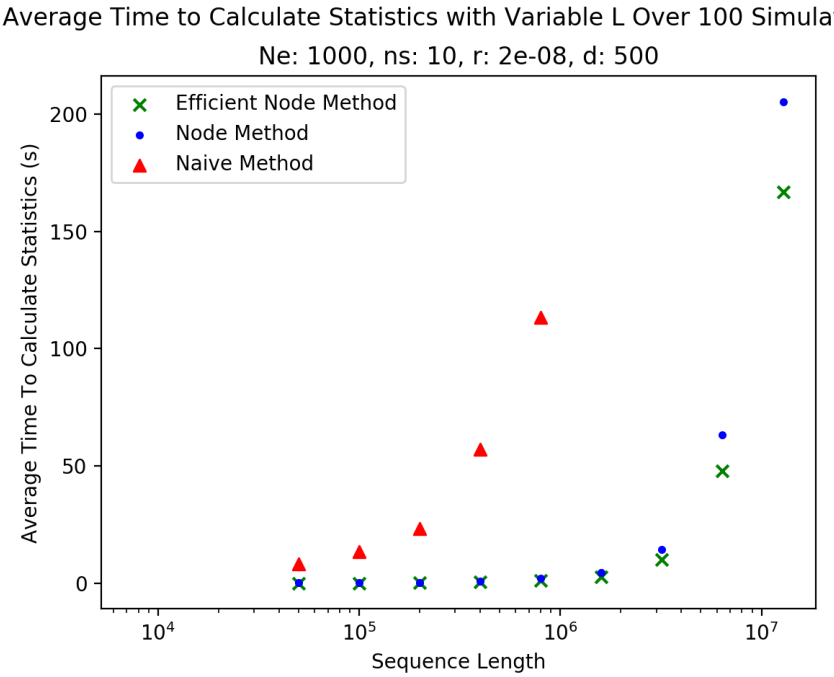

Figure 5: Average time for all 3 Algorithms with variable sequence length at low sample size. The Naive Method wasn't computed at past sequence length 8e5 due to time constraints

5 Discussion

From Fig. 1 we can see that the average calculated expected statistics tend to converge to the theoretical results within 1% error (Note: σ_z^2 tends to be a noisier statistic than σ_d^2 overall). We can also note a tendency for the average calculated σ_d^2 to be slightly lower than the theoretical result and the average calculated σ_z^2 to be slightly higher than the theoretical result.

In Fig. 2, comparing average time to calculate the statistics, it is evident that the Naive Method is far too computationally expensive to use for any reasonably large sample size. Fig. 3 illustrates the significance of the improvement between the Simple Node Based Method and the Efficient Node Based Method, where with the Efficient Node Method, computing the expected statistics for sample sizes of 2160 occurs faster than computing the expected statistics with sample size 320.

In Fig. 4, as we increase the effective population size for the simulation, we increase the time for computation for all 3 methods. The Simple Node Method outperforms the Efficient Node Method in these simulations due to the low sample size used in this test.

In Fig. 5, we can see the effect sequence length has on computation time. Note the log scale for sequence length. In contrast to Fig 4., even with the low sample size, the Efficient Node Method outperforms the Simple Node Method.

The data highlights how much the Efficient Node Method outperforms the other two algorithms, especially with increasing sample size.

- Naive: $O(\alpha T n^4)$
- Simple Node: $O(\alpha T n^2)$
- Efficient Node: $O(n^2 + \alpha T n log(n))$

The size of α depends on the recombination distance, and will be smaller for larger distances as there are fewer tree pairs to be computed. The size of T is modified by the effective population size, sample size, recombination rate, and sequence length.

6 Conclusion

The goal of this project was to design a method to efficiently extract two-locus statistics that were relevant to population genetics researchers. With the Efficient Node Based Algorithm, researchers will be able to quickly calculate expected σ_d^2 and σ_z^2 statistics for any msprime simulated tree sequence, with reasonably large sample sizes.

By leveraging the nature of coalescent trees and the ability to summarize information under nodes, an improvement of the order $O(n^2)$ was made. By leveraging the Edge Diffs given by msprime, a further improvement by the order of $O(T)$. This algorithm lays the groundwork for future algorithms targetting other multi-locus statistics. Future work includes python conversion to C and implementation in the open source msprime/tskit library.

A Naive σ_d^2 Algorithm

Algorithm 1 Naive σ_d^2 Algorithm

procedure CALCULATE PAIRWISE STATISTICS(ts, d)
  $n \leftarrow ts.samplesize$
  $e_{ijij} \leftarrow 0$
  $e_{ijik} \leftarrow 0$
  $e_{ijkl} \leftarrow 0$
  $L \leftarrow$ FIND PAIRWISE WINDOWS(ts, d)
  for each $p \in L$ **do**
    $S \leftarrow ts.AtIndex(p[0])$
    $T \leftarrow ts.AtIndex(p[1])$
    $c \leftarrow p[2]$ ▷ c represents the length for which this window exists
    $acc_{ijij} \leftarrow 0$
    $acc_{ijik} \leftarrow 0$
    $acc_{ijkl} \leftarrow 0$
    for $i \leftarrow 0, n-1$ **do**
      for $j \leftarrow i+1, n$ **do**
        $acc_{ijij} \leftarrow acc_{ijij} + S.tmrca(i,j) \times T.tmrca(i,j)$
        for $k \leftarrow 0, n$ **do**
          if $k = i$ or $k = j$ **then**
            continue
          else
            $acc_{ijik} \leftarrow acc_{ijik} + S.tmrca(i,j) \times T.tmrca(i,k) + S.tmrca(i,j) \times T.tmrca(j,k)$
            for $l \leftarrow 0, n$ **do**
              if $l \leq k$ or $l = i$ or $l = k$ **then**
                continue
              else
                $acc_{ijkl} \leftarrow acc_{ijkl} + S.tmrca(i,j) + T.tmrca(k,l)$
              end if
            end for
          end if
        end for
      end for
    end for
    $e_{ijij} \leftarrow e_{ijij} + acc_{ijij} \times c$
    $e_{ijik} \leftarrow e_{ijik} + acc_{ijik} \times c$
    $e_{ijkl} \leftarrow e_{ijkl} + acc_{ijkl} \times c$
  end for
  $e_{ijij} \leftarrow e_{ijij}/(ts.length \times n(n-1)/2)$ ▷ Normalize the over all possible pairs
  $e_{ijik} \leftarrow e_{ijik}/(ts.length \times n(n-1)(n-2)$
  $e_{ijkl} \leftarrow e_{ijkl}/(ts.length \times n(n-1)(n-2)(n-3)/4)$
  return $e_{ijij}, e_{ijik}, e_{ijkl}$ ▷ Sufficient to calculate stats, ie D^2, Dz
end procedure

Algorithm 2 Calculating Overlapping Pairwise Windows

```
1: procedure FIND PAIRWISE WINDOWS(ts, d)
2:     S ← ts.first()                          ▷ S, T are the left and right trees respectively
3:     T ← ts.at(d)
4:     l ← 0
5:     r ← d
6:     L ← Empty List
7:     while r < ts.length do
8:         ld ← IntervalRightBound(S) − l
9:         rd ← IntervalRightBound(T) − r
10:        L.append([S.index, T.index, min(ld, rd)])
11:        if ld > rd then
12:            l ← l + rd
13:            if r + rd = ts.length then
14:                break while
15:            end if
16:            T ← T.next
17:            r = IntervalLeftBound(T)
18:        else
19:            r ← r + ld
20:            S ← S.next
21:            l ← IntervalLeftBound(S)
22:        end if
23:    end while
24:    return L
25: end procedure
```

B Simple Node Based σ_d^2 Algorithm

Algorithm 3 Simple Node Based σ_d^2 Algorithm

procedure CALCULATE PAIRWISE STATISTICS(ts, d)
  $L \leftarrow$ FIND PAIRWISE WINDOWS(ts, d)
  $e_{ijij} \leftarrow 0$
  $e_{ijik} \leftarrow 0$
  $e_{ijkl} \leftarrow 0$
  for each $p \in L$ **do**
    $S \leftarrow ts.AtIndex(p[0])$
    $T \leftarrow ts.AtIndex(p[1])$
    $c \leftarrow p[2]$ ▷ c represents the length for which this window exists
    for each $l \in S.InternalNodes$ **do**
      for each $r \in T.InternalNodes$ **do**
        $V \leftarrow$ CALCULATE BETWEEN NODES(S, T, l, r, c)
        $e_{ijij} \leftarrow e_{ijij} + V[0]$
        $e_{ijik} \leftarrow e_{ijik} + V[1]$
        $e_{ijkl} \leftarrow e_{ijkl} + V[2]$
      end for
    end for
  end for
  $e_{ijij} \leftarrow e_{ijij}/(ts.length \times n(n-1)/2)$ ▷ Normalize the over all possible pairs
  $e_{ijik} \leftarrow e_{ijik}/(ts.length \times n(n-1)(n-2)$
  $e_{ijkl} \leftarrow e_{ijkl}/(ts.length \times n(n-1)(n-2)(n-3)/4)$
end procedure

Algorithm 4 Simple Node Based σ_d^2 Algorithm Helpers

procedure CALCULATE PAIRING COUNTS(S, T, l, r)
  $K \leftarrow l_l l_r r_l r_r$ ▷ K is the total number of pairing combinations
  $l_l \leftarrow S.leaves(S.children(l).left)$
  $l_r \leftarrow S.leaves(S.children(l).right)$
  $r_l \leftarrow T.leaves(T.children(r).left)$
  $r_r \leftarrow T.leaves(T.children(r).right)$

  $m_l \leftarrow l_l \cap r_l$
  $m_r \leftarrow l_r \cap r_r$
  $c_1 \leftarrow l_l \cap r_r$
  $c_2 \leftarrow l_r \cap r_l$

  $k_{ijij} \leftarrow len(m_l) \times len(m_r) + len(c_1) \times len(c_2)$ ▷ counting ij, ij pairings
  $a \leftarrow len(m_l) \times (len(l_r) \times len(r_r) - len(m_r))$
  $b \leftarrow len(m_r) \times (len(l_l) \times len(r_l) - len(m_l))$
  $c \leftarrow len(c_1) \times (len(l_r) \times len(r_l) - len(c_2))$
  $d \leftarrow len(c_2) \times (len(l_l) \times len(r_r) - len(c_1))$

  $k_{ijik} \leftarrow a + b + c + d$ ▷ counting ij, ik pairings
  $k_{ijkl} \leftarrow K - k_{ijij} - k_{ijkl}$ ▷ remaining pairs must fall under ij, kl
  return $k_{ijij}, k_{ijik}, k_{ijkl}$
end procedure

procedure CALCULATE BETWEEN NODES(S, T, l, r, c)
  $c_{ijij}, c_{ijik}, c_{ijkl} \leftarrow$ CALCULATE PAIRING COUNTS(S, T, l, r)
  $t \leftarrow l.time \times r.time \times c$
  $L \leftarrow [tc_{ijij}, tc_{ijik}, tc_{ijkl}]$
  return L
end procedure

C Efficient Node Based σ_d^2 Algorithm

Algorithm 5 Efficient Node Based σ_d^2 Algorithm

procedure CALCULATE PAIRWISE STATISTICS(ts, d)
  $L \leftarrow$ FIND CHECKPOINTS(ts, d)
  $S \leftarrow ts.first$
  $T \leftarrow ts.at(d)$
  $e_{ijij} \leftarrow 0$
  $e_{ijik} \leftarrow 0$
  $e_{ijkl} \leftarrow 0$
  for each $c \in L$ **do**
    $p \leftarrow ts.AtIndex(c[0])$
    $s \leftarrow ts.AtIndex(p[1])$
    if $s =$ "LEFT" **then** ▷ Left tree has been changed at the checkpoint position
      $N \leftarrow$ CHANGED NODES($S, S.next$)
      $L_n \leftarrow (N \cap S.InternalNodes)$
      $R_n \leftarrow T.InternalNodes$
    else
      $N \leftarrow$ CHANGED NODES($T, T.next$)
      $R_n \leftarrow (N \cap T.InternalNodes)$
      $L_n \leftarrow S.InternalNodes$
    end if
    for each $l \in L_n$ **do**
      for each $r \in R_n$ **do**
        $x \leftarrow min(p + d - U_r[r], p - U_l[l])$ ▷ x is interval length for this node pairing
        $V \leftarrow$ CALCULATE BETWEEN NODES(S, T, l, r, x)
        $e_{ijij} \leftarrow e_{ijij} + V[0]$
        $e_{ijik} \leftarrow e_{ijik} + V[1]$
        $e_{ijkl} \leftarrow e_{ijkl} + V[2]$
      end for
    end for
    if $s =$ "LEFT" **then**
      $S \leftarrow S.next$
    else
      $T \leftarrow T.next$
    end if
    for each $n \in N$ **do**
      if $s =$ "LEFT" **then**
        $U_l[n] \leftarrow p$
      else
        $U_r[n] \leftarrow p + d$
      end if
    end for
  end for
  $p \leftarrow ts.SequenceLength - d$
  for each $l \in S.InternalNodes$ **do** ▷ Catches all remaining intervals at the end
    for each $\in T.InternalNodes$ **do**

$x \leftarrow min(p + d - U_r[r], p - U_l[l])$ ▷ x is interval length for this node pairing
$V \leftarrow$ CALCULATE BETWEEN NODES(S, T, l, r, x)
$e_{ijij} \leftarrow e_{ijij} + V[0]$
$e_{ijik} \leftarrow e_{ijik} + V[1]$
$e_{ijkl} \leftarrow e_{ijkl} + V[2]$
end for
end for
$e_{ijij} \leftarrow e_{ijij}/(ts.length \times n(n-1)/2)$ ▷ Normalize the over all possible pairs
$e_{ijik} \leftarrow e_{ijik}/(ts.length \times n(n-1)(n-2)$
$e_{ijkl} \leftarrow e_{ijkl}/(ts.length \times n(n-1)(n-2)(n-3)/4)$
end procedure

References

[1] Jerome Kelleher, Alison M Etheridge, and Gilean A. T. McVean. Efficient coalescent simulation and genealogical analysis for large sample sizes. *PLOS Computational Biology*, 12(5), 2016.

[2] Gilean A. T. McVean. A genealogical interpretation of linkage disequilibrium. *Genetics*, 162(2):987–991, 2002.

[3] Tomoko Ohta and Motoo Kimura. Linkage disequilibrium between two segregating nucleotide sites under the steady flux of mutations in a finite population. *Genetics*, 68(4):571–580, 1971.

[4] Yun S. Song and Jun S. Song. Analytic computation of the expectation of the linkage disequilibrium coefficient. *Theoretical Population Biology*, 71(1):49–60, Sep 2007.

[5] Xiguo Yuan, David J. Miller, Junying Zhang, David Herrington, and Yue Wang. An overview of population genetic data simulation. *Journal of Computational Biology*, 19(1):42–54, 2012.