Recording pedigrees for fast genome simulation and analysis

Peter Ralph[‡]

‡ *Mathematics and Biology, University of Oregon*

code: `https://github.com/tskit-dev`

preprint: `https://www.biorxiv.org/content/early/2018/01/26/248500`

Tree sequences: all the genealogies

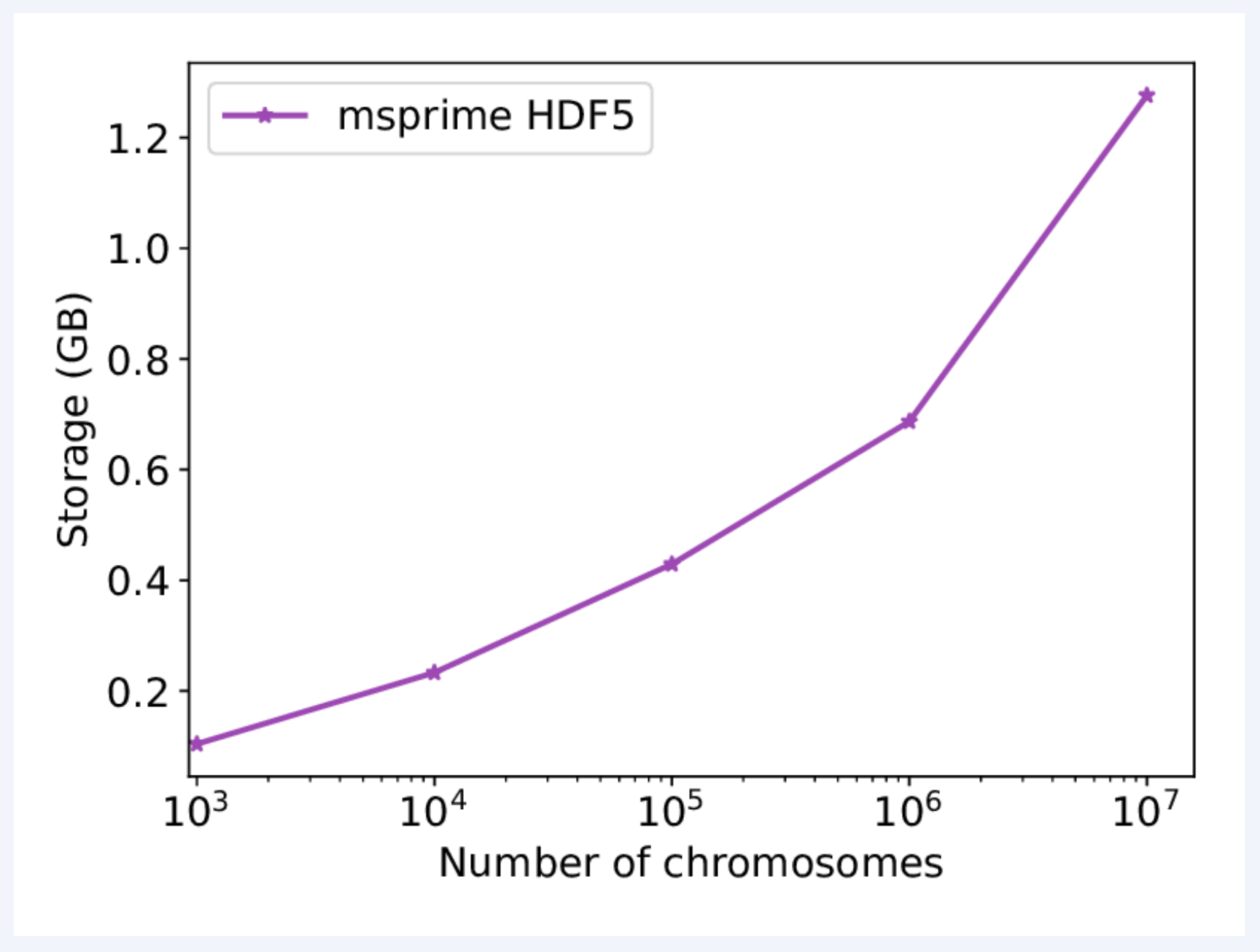
For a set of sampled chromosomes, at each position along the genome there is a genealogical tree that says how they are related.
A **tree sequence** describes this sequence of trees.
Observations:

- The *pedigree* (parental relationships) plus crossover locations would give us the tree sequence for *everyone, ever.*
- Much less can fully describe the history relevant to a *sample* of genomes.
- This information is equivalent to the Ancestral Recombination Graph (ARG).

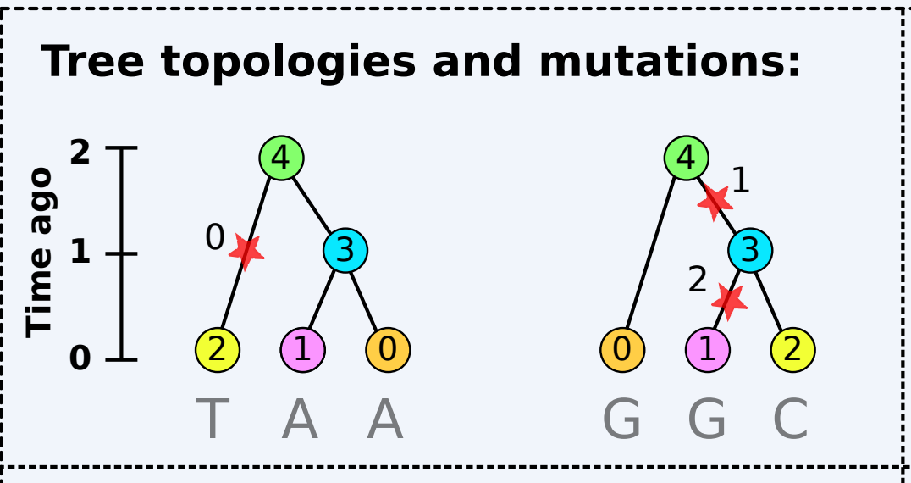
(Kelleher et al., 2016) introduced the **tree sequence** data structure for a fast coalescent simulator, `msprime`.

Tables: a data structure for tree sequences

- **Edges:** Who inherits from who; only *necessary* for coalescent events. *Records:* interval (left, right); parent node; child node.
- **Nodes:** The ancestors those happen in. *Records:* time ago (of birth); ID (implicit).
- **Mutations:** When state changes along the tree. *Records:* site it occurred at; node it occurred in; derived state.
- **Sites:** Where mutations fall on the genome. *Records:* genomic position; ancestral (root) state; ID (implicit).



Example

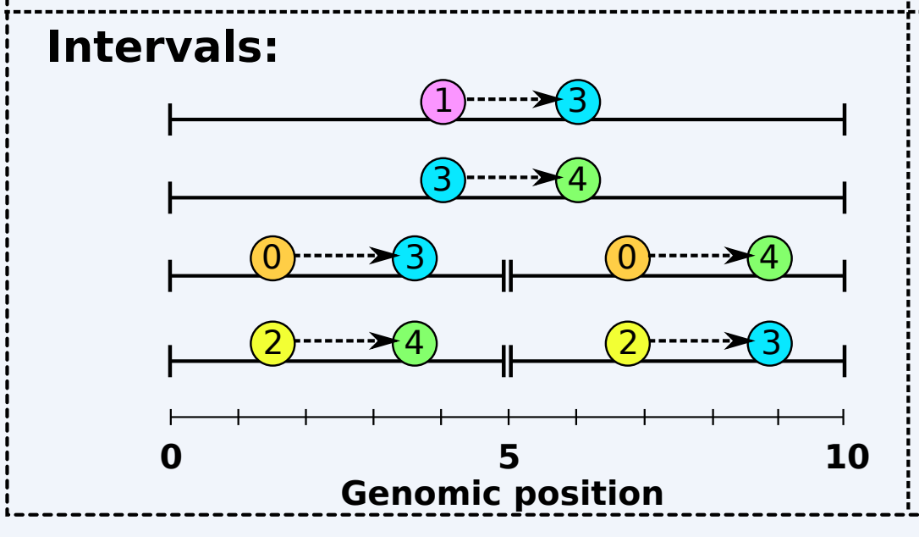


Nodes:

ID	time
0	0.0
1	0.0
2	0.0
3	1.0
4	2.0

Edges:

left	right	parent	child
0	10	3	1
0	10	4	3
0	5	3	0
0	5	4	2
5	10	3	2
5	10	4	0



Sites:

ID	position	ancestral state
0	2.5	A
1	7.5	G

Mutations:

ID	site	node	derived state
0	0	2	T
1	1	3	C
2	1	1	G

Another example

Nodes:

id	time
0	0.00
1	0.00
2	0.00
3	0.00
4	0.00
5	0.00
6	0.00
7	0.00
8	0.09
9	0.31
10	0.39
11	0.41
12	0.43
13	0.97
14	1.04
15	1.21
16	1.36
17	1.45
18	2.41
19	2.46
20	2.85
21	3.84
22	4.45

Edges:

left	right	parent	child
0.00	1.00	8	2
0.00	1.00	8	6
0.00	1.00	9	5
0.00	1.00	9	7
0.00	1.00	10	3
0.00	1.00	10	8
0.00	1.00	11	0
0.00	1.00	11	1
0.00	1.00	12	9
0.00	1.00	12	10
0.99	1.00	13	4
0.99	1.00	13	12
0.00	0.19	14	11
0.00	0.19	14	12
0.93	0.99	15	4
0.93	1.00	15	11
0.99	1.00	15	13
0.30	0.54	16	4
0.91	0.93	16	4
0.30	0.54	16	11
0.91	0.93	16	11
0.54	0.69	17	4

cont'd

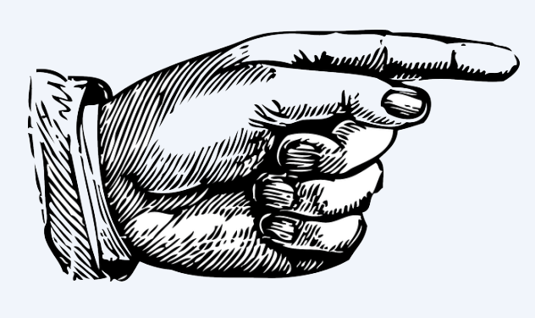
left	right	parent	child
0.54	0.69	17	12
0.69	0.91	18	4
0.69	0.91	18	11
0.00	0.30	19	4
0.69	0.69	19	11
0.19	0.46	19	12
0.69	0.99	19	12
0.00	0.19	19	14
0.93	0.99	19	15
0.30	0.46	19	16
0.91	0.93	19	16
0.69	0.69	19	17
0.69	0.91	19	18
0.19	0.30	20	11
0.19	0.30	20	19
0.60	0.69	21	11
0.60	0.69	21	17
0.54	0.60	22	11
0.46	0.54	22	12
0.46	0.54	22	16
0.54	0.60	22	17

Contributions

- Jerome Kelleher – algorithms, data structures, and core development
- Kevin Thornton – implementation, profiling with `fwdpp`,
- Jaime Ashander – implementation, profiling with `simuPOP`
- Ben Haller – implementation in `SLiM`
- Jared Galloway – implementation in `SLiM`

Record history in forwards simulations?

- Coalescent simulations are *much faster* than forwards-time, individual-based simulations because they don't have to keep track of *everyone*, only the ancestors of your sample.
- **But:** selection, or sufficient geographic structure, break the assumptions of coalescent theory.
- If we *record the tree sequence* that relates everyone to everyone else, after the simulation is over we can put neutral mutations down on the trees.
- Since neutral mutations don't affect demography, this is *equivalent* to having kept track of them throughout.
- This means recording the entire genetic history of **everyone** in the population, **ever**.



This may be a bad idea.

How to do it

1. add each gamete to the Node Table,
2. add entries to the Edge Table recording which parent each gamete inherited each bit of genome from, and
3. add any new selected mutations to the Mutation Table and (if necessary) their locations to the Site Table.

But, we won't end up needing the entire history of everyone ever.

Simplification

Given an input tree sequence and a subset of its nodes (the *samples*), we want a new tree sequence for which:

1. All marginal trees match the corresponding subtree in the input tree sequence.
2. Every non-sample node in marginal trees has at least two children.
3. All nodes and edges ancestral to at least one sample.
4. No adjacent redundant edges (e.g., $(\ell, x, p, c) + (x, r, p, c) \rightarrow (\ell, r, p, c)$).

To simplify a tree sequence to the history of the *samples*, we:

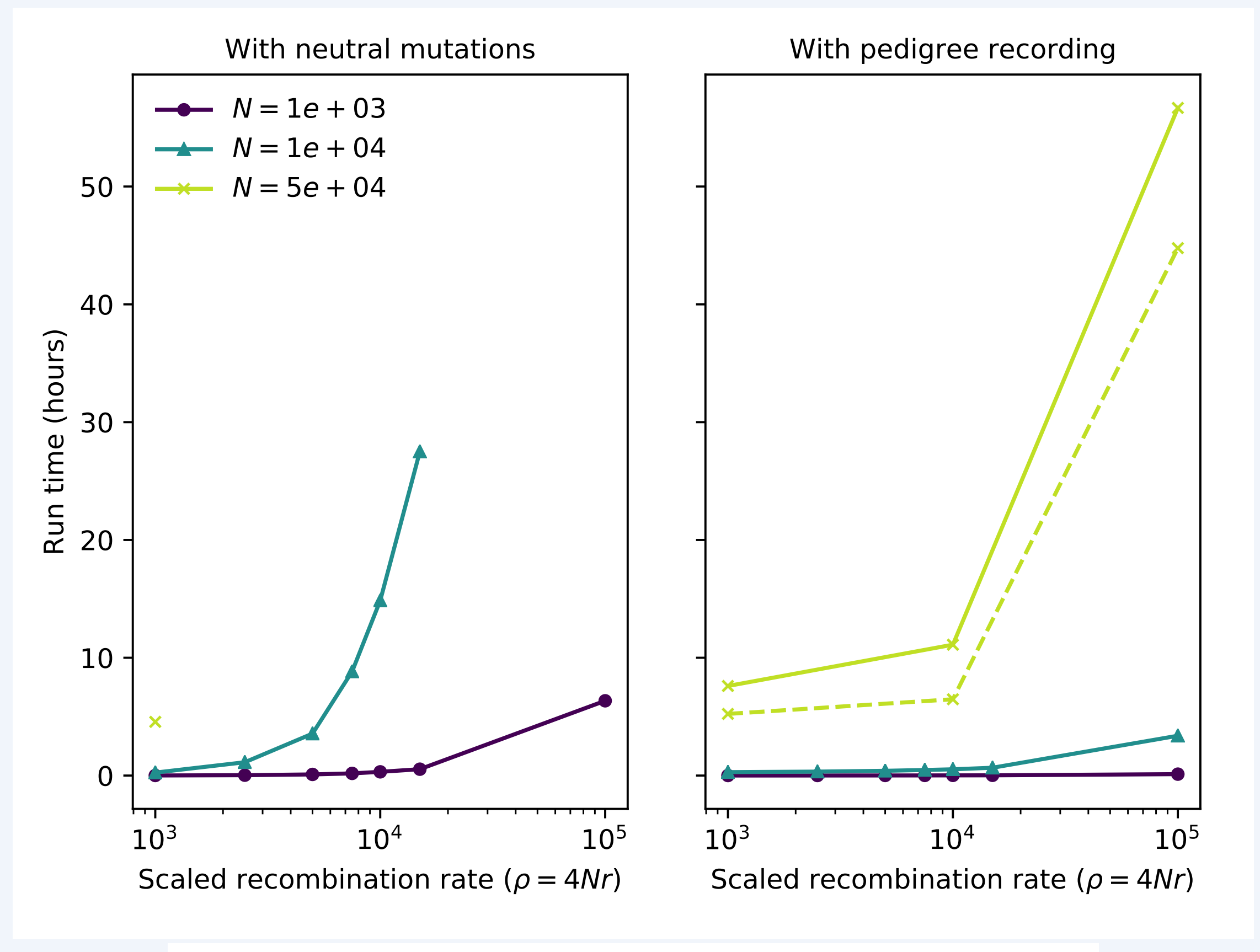
- Paint each *sampled* chromosome a distinct color.
- Moving back up the tree sequence, copy colors of each chromosome to the parental chromosomes they inherited from.
- If two colors go in the same spot (*coalescence*), replace with a new color (unique to that ancestor). Output a node for the ancestor and an edge for the coalescence.
- Once all colors have coalesced in a given segment, stop propagating it.

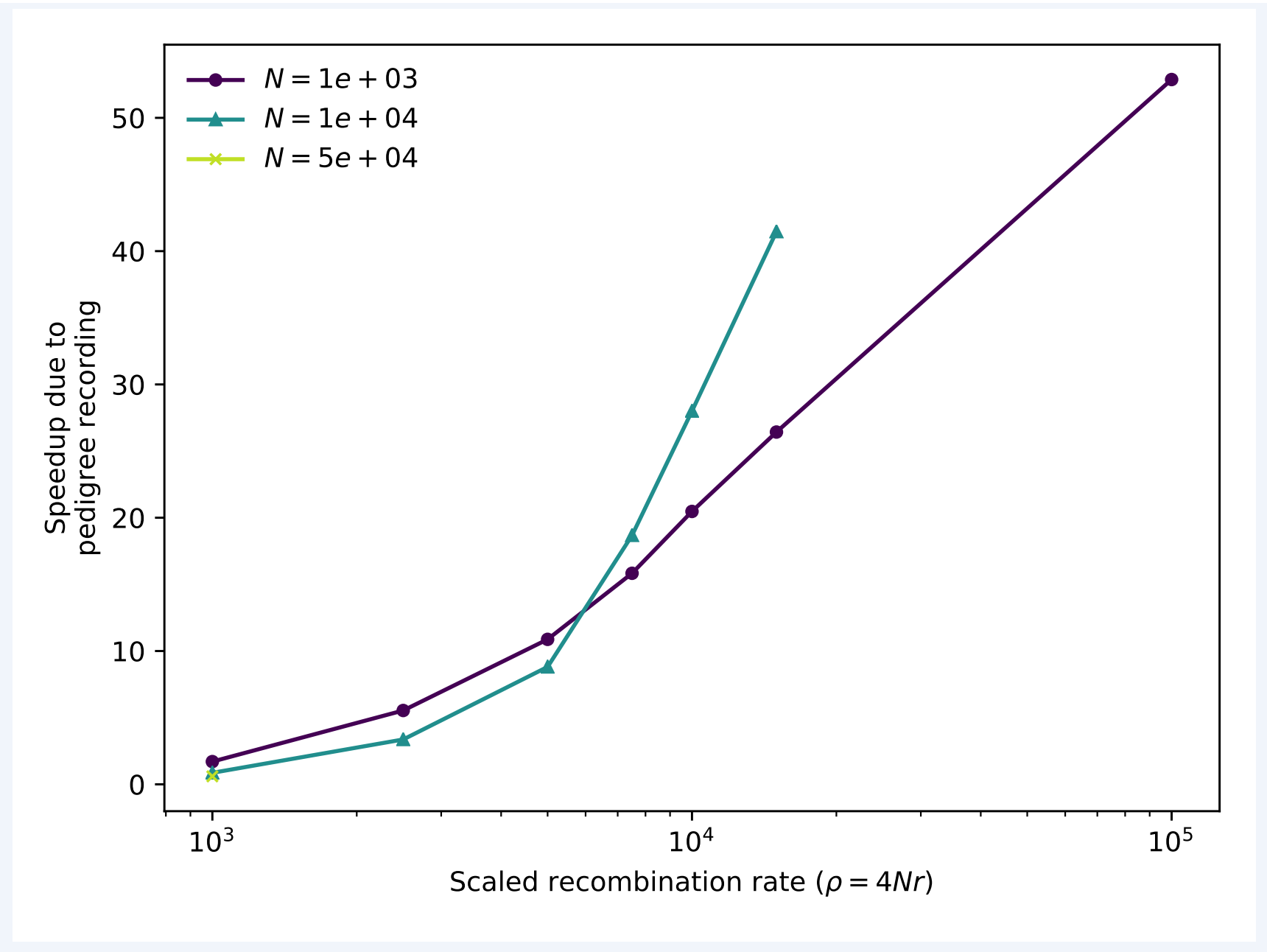
Simulations, $50\times$ faster

Recording, simplifying, and output of tables, was done with `C` code in `tskit`. The simulation was done in `fwdpp` (C++ by Kevin Thornton), connected with `pybind11` and `numpy`.
Machine: Ubuntu / 2x 2.6 GHz Intel E5-2650 CPU.
Simulation parameters:

- Wright-Fisher population of size N, for $10N$ generations
- neutral mutation rate μ equal to recombination rate r per gamete
- many, weakly deleterious mutations: rate $\mu/100$ with s exponentially distributed with mean $2.5/N$.

Note: simulations that recorded tree sequences ("pedigree recording") had neutral mutation rate was zero, but neutral mutations were added *afterwards.*





Fast statistics computation

A general class of statistics: any polynomial $f(p_1, \ldots, p_k)$ gives a statistic on k populations, by setting p_i to the allele frequency in population i and averaging f across loci. *(Example: divergence is* $p_1(1-p-2)+p-2(1-p_1)$*.)*
The polynomial defines a weighting function on edges of the trees in the tree sequence, and the statistic can be calculated by summing across mutations weighted by the edge they occur on. *(Example: proportion of pairs from the two populations on opposite sides of the edge.)*
Computation on a tree sequence: If we compute weights on the first tree, then we only need update those weights above nodes that change in the second tree.

Summary

Tree sequences

- ...can make your simulations much faster
- *because* they record genealogical history.
- (And, then you have genealogies!)
- ...can store genomic data *very* efficiently,
- and compute with genomic data *very* quickly.

Why are they so useful? It is a compression scheme, provided by the process itself that produced the data. *(think: optimal compression into shared haplotypes)*