Memoization on Shared Subtrees Accelerates Computations on Genealogical Forests

Lukas Hübner
Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany
Heidelberg Institute for Theoretical Studies, Germany

Alexandros Stamatakis
Institute of Computer Science, Foundation for Research and Technology-Hellas, Heraklion, Greece
Heidelberg Institute for Theoretical Studies, Germany
Institute of Theoretical Informatics, Karlsruhe Institute of Technology, Germany

Abstract

The field of population genetics attempts to advance our understanding of evolutionary processes. It has applications, for example, in medical research, wildlife conservation, and – in conjunction with recent advances in ancient DNA sequencing technology – studying human migration patterns over the past few thousand years. The basic toolbox of population genetics includes genealogical trees, which describe the shared evolutionary history among individuals of the same species. They are calculated on the basis of genetic variations. However, in recombining organisms, a single tree is insufficient to describe the evolutionary history of the whole genome. Instead, a collection of correlated trees can be used, where each describes the evolutionary history of a consecutive region of the genome. The current corresponding state of-the-art data structure, tree sequences, compresses these genealogical trees via edit operations when moving from one tree to the next along the genome instead of storing the full, often redundant, description for each tree. We propose a new data structure, genealogical forests, which compresses the set of genealogical trees into a DAG. In this DAG identical subtrees that are shared across the input trees are encoded only once, thereby allowing for straight-forward memoization of intermediate results. Additionally, we provide a C++ implementation of our proposed data structure, called `gfkit`, which is 2.1 to 11.2 (median 4.0) times faster than the state-of-the-art tool on empirical and simulated datasets at computing important population genetics statistics such as the Allele Frequency Spectrum, Patterson's f, the Fixation Index, Tajima's D, pairwise Lowest Common Ancestors, and others. On Lowest Common Ancestor queries with more than two samples as input, `gfkit` scales asymptotically better than the state-of-the-art, and is thus up to 990 times faster. In conclusion, our proposed data structure compresses genealogical trees by storing shared subtrees only once, thereby enabling straight-forward memoization of intermediate results, yielding a substantial runtime reduction and a potentially more intuitive data representation over the state-of-the-art. Our improvements will boost the development of novel analyses and models in the field of population genetics and increases scalability to ever-growing genomic datasets.

2012 ACM Subject Classification Applied computing → Molecular sequence analysis; Applied computing → Bioinformatics; Applied computing → Population genetics; Applied computing → Computational genomics

Keywords and phrases bioinformatics, population genetics, algorithms

Digital Object Identifier 10.4230/LIPIcs.WABI.2024.5

Related Version *Previous Version*: `https://www.biorxiv.org/content/10.1101/2024.05.23.595533v1`

Supplementary Material
Software (Source Code): `https://github.com/lukashuebner/gfkit` [28]
archived at `swh:1:dir:bffece502c3a579271ac3d91d66b051d089d02f2`
Dataset (Simons Genome Diversity Project gfkit): `https://doi.org/10.5281/zenodo.11241730` [25]
Dataset (Thousand Genome Project gfkit): `https://doi.org/10.5281/zenodo.11241619` [26]

Software (Unified (Wohns et al 2022) gfkit): `https://doi.org/10.5281/zenodo.11241788` [27]
Dataset (640k Samples Simulated gfkit): `https://doi.org/10.5281/zenodo.11241938` [29]

Funding This project received funding from **(a)** the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 882500) **(b)** the European Union (EU) under Grant Agreement No 101087081 (Comp-Biodiv-GR) **(c)** the Ministry of Science, Research, and the Arts of Baden-Württemberg (Az: 33-7533.-9-10/20/2) to Peter Sanders and Alexandros Stamatakis.

1 Introduction

Charles Darwin famously wrote that living beings share a common evolutionary history and organize in a tree [6, 14]. Since then, hand-drawn trees based on morphological features [14, 21, 23] have been replaced by trees computed using mathematical and statistical models taking into account variations among the genomic data of the observed species [50, 22].

Next to analyzing the evolutionary history among distinct species (*phylogeny*), the question of how living and past individuals and populations of a single species, for example humans[1], have emerged, migrated, and how they relate to each other (*genealogy*) is also of interest. Population genetics is the sub-field of evolutionary biology addressing these questions and advances our understanding of evolutionary processes such as mutations, genetic drift, gene flow, and natural selection [41]. Its downstream applications include for example medical research [63, 47, 39, 3], wildlife conservation [24, 41, 59, 54], and – in conjunction with recent advances in ancient DNA sequencing technology [48, 44, 52, 12] – studying human migration patterns in the past few thousand years [2, 40, 53, 48].

1.1 Genealogical Trees, Tree Sequences, and Forests

Genealogical trees depict shared evolutionary history among genomes and are based the genetic variation across sampled individuals [36]. Genetic loci (positions) which are identical across all sampled genomes lack evolutionary signal and are therefore omitted. Hence, in this work, *genomic sequence* or *genome* often refers to only those loci of the genome with variation across the samples. In genealogical trees, *tips* (out-degree 0 nodes) represent biological samples, that is, the genetic material of a single individual. Non-tip nodes, or *inner nodes*, represent one or multiple sequence variations inherited from a single parent. These inner nodes might be multifurcating, that is, they can have more than two children. Further, we consider all genealogical trees to be rooted, that is, they have a node with an in-degree of 0 (i.e., no parent), which we call the *root*, and directed edges going away from this root. This root is the hypothetical common ancestor individual of all tips in the tree.

Single trees cannot adequately describe processes in which individuals pass on different parts of the genome independently [45]. This is seen, for example, in sexually reproducing species with multiple genome copies (e.g., two in humans) [30]: Here, a process called *recombination* causes the genetic material, organized in a DNA strand, to break and reconnect.

[1] In this work, human always refers to *homo sapiens*.

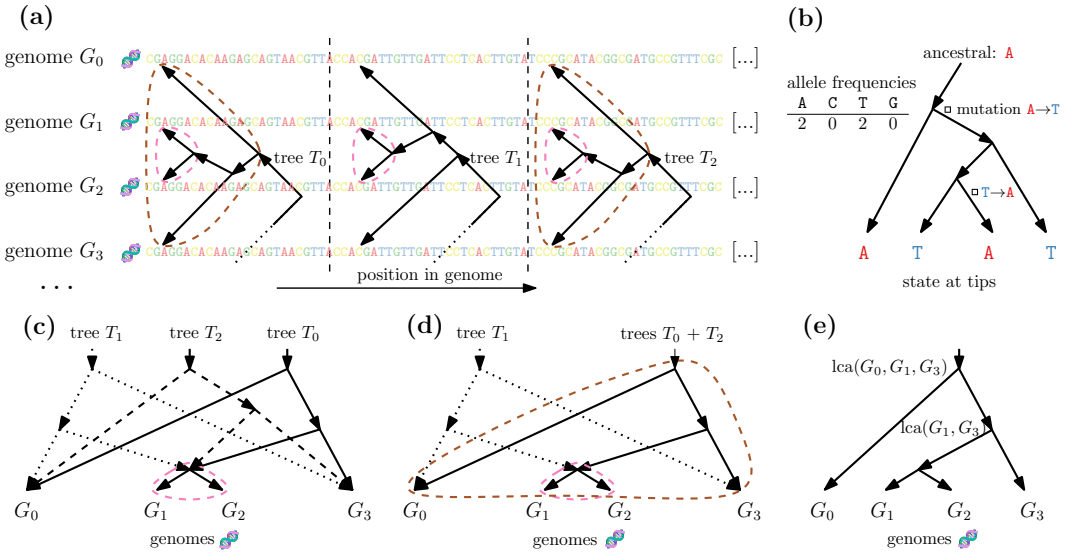

Figure 1 **(a)** Different, but correlated, trees each describe distinct regions of multiple aligned genomes. The subtree `(`G_1`,`G_2`)` (pink) appears in all displayed trees, whereas the subtree `(`G_0`,((`G_1`,`G_2`),`G_3`))` (bronze) appears only in T_0 and T_2, but not in T_1. **(b)** A tree describes the genomic states of genomes at its tips by storing ancestral states and corresponding mutations annotated at the tree edges (we show only a single genomic site). **(c)** `tskit` encoding of the three trees from (a). `tskit` re-uses `(`G_1`,`G_2`)` but not `(`G_0`,((`G_1`,`G_2`),`G_3`))`, as the latter does not re-occur in *consecutive* trees. **(d)** Our proposed encoding (`gfkit`) of the trees from (a). `gfkit` re-uses both, `(`G_0`,`G_1`)` when describing T_1, and `(`G_0`,((`G_1`,`G_2`),`G_3`))` when describing T_2. **(e)** Lowest Common Ancestors of two (lca(G_1, G_3)) or more (lca(G_0, G_1, G_3)) selected samples.

Thus, the closer two genetic positions are along the genome, the higher the chance for them to have been inherited from the same parent [58]. Therefore, for recombining organisms, modelling the (correlated) history of distinct regions along the genome using distinct (correlated) trees captures the evolutionary history more accurately than a single tree [30]. Based on this observation, Kelleher et al. [35, 36, 51] introduce (succinct) *tree sequences* (Figure 1.a), which are collections of correlated genealogical trees, each describing the evolutionary history of a range of adjacent sites along the genome. In the genealogical trees stored in tree sequences, tips represent biological samples, edges represent lines of descent, and mutations are meta-data associated with these edges (Figure 1). In tree sequences, the edges are valid for a specified region of the genome, as they describe its evolutionary history. Statistics over the topologies of genealogical trees (Section 1.2) and differences in the genetic code between the observed genomes (Section 1.3) are frequently deployed in population genetics to conduct quantitative assessments, e.g. how genetically diverse a population is.

1.2 Lowest Common Ancestor (LCA) Queries

The Lowest Common Ancestor (LCA)[2] [1] of two or more tips in a tree is the lowest (farthest from the root) inner node on *all* paths from the root to the tips (Figure 1.e). Population geneticist employ LCA queries on genealogical trees for example to answer questions like "Did the evolutionary history of genomes of group A and B separate at the time this land bridge

[2] Also called the Most Recent Common Ancestor (MRCA).

vanished?". This involves computing the LCA of all genomes in A and B and estimating the time of the resulting ancestor[3] using methods beyond the scope of this paper. Note that here, we extend the definition of LCA to also include more than two tips.

1.3 Differences in The Genetic Code Among Genomes

Alleles are concrete variants of genetic variation between genomes and encompass one or multiple (related) genetic loci. This work considers only alleles consisting of single-loci differences. For instance, a set of genomic sequences of human individuals might have the alleles `A` and `C` at a specific genome site. We call a site with more than one allele *polyallelic* (e.g., `A`, `C`, or `T`), or *biallelic* if it has exactly two alleles. The *allele frequency* is the proportion of alleles at a given site, e.g., 80 `A`s and 5 `C`s (Figure 1.b). We often select only a subset of the samples (tips in the genealogical tree) for a given query; which we call the *selected* samples or *sample set*. The exponential number of possible sample sets yields pre-computing all statistics of interest infeasible, necessitating algorithmic improvements.

Many statistics in population genetics are functions of allele frequencies [51]: For example sequence *diversity* [46] measures the average genetic differences among the genomes of individual samples in a single sample set (drawn from a single population). Other statistics work on two or more sample sets, for example to assess the degree of sequence *divergence* [46] between them. While we define these statistics per site, we often average them across many or all sites in the input genome or within a sliding window over all sites of the genome.

1.4 The Need for Efficient Algorithms

Storing and processing the full genomes of all samples in contemporary datasets on a base-by-base basis is infeasible as collections of hundreds of thousands of enomes exist for humans alone [5, 42, 11, 61, 34, 53], with even larger datasets being sequenced and assembled.[4] In fact, the growth of genomic data outpaces the growth of computational power and storage per unit of money [8]. Thus, we can not rely on increasing (sequential or parallel) processing speeds to analyze these data. Instead, we require algorithms performing no more work than minimally required by the problem[5] (i.e., are compute-efficient) as well as highly optimized implementations exploiting modern hardware features and leveraging data locality.

1.5 State of the Art and Contribution

Tree sequences allow for reducing the storage space required for a set of related genomes by modelling the evolutionary history of the included samples, and reusing computations in statistical queries [35, 36, 51]. For example, the human chromosome 20 in the Thousand Genomes Project collection (Section 5) contains 5008 sequences with 860 thousand variant sites; each sequence being in one of four potential states (`A`, `C`, `T`, or `G`) at each site [36]. This results in a theoretical space requirement of 1 GiB if stored on a base-by-base basis using 2 bit per sequence and site – compared to the tree sequence file size of 283 MiB [36].

Additionally, a tree sequence stores the evolutionary history of each part of the genome via the corresponding genealogical trees and allows reusing partial results between adjacent, yet topologically distinct trees, thereby avoiding some – yet not all possible – redundant

[3] `Gfkit` currently does not store coalescent times, but we plan on adding them (Section 8).
[4] `https://digital-strategy.ec.europa.eu/en/policies/1-million-genomes`
[5] For some problems, the minimum work required might not be known.

computations [35, 51]. This is because tree sequences reuse only intermediate results for shared subtrees among *adjacent* trees along the genome [35, 51]. They do not allow reusing intermediate results across trees further apart, if the respective subtree is not part of all intermediate trees (Figure 1.c). Tree sequences use edit operations (edge insertions and removals) to describe changes/differences in the tree topology from one tree to the next along the genome; and thus across recombination events. Therefore, determining if a given subtree was already present in an earlier tree is not trivial. For example, if an edge from node a to node b ($a \rightarrow b$) is removed and a (sans meta-data) identical edge $a \rightarrow b$ is added back a few trees further down the genome, other edges might have changed in the subtree below node b, thus invalidating the intermediate results for b.

The core concept of our data structure is to encode and store shared subtree across *all* (not just adjacent) trees exactly once, enabling queries on the trees topologies (Section 3.3) and genetic sequences (Section 3.4) to reuse intermediate results across *all* trees (*memoization*).

1.6 Overview of the Paper Structure

After providing an overview of related work (Section 2), we describe the core idea of our proposed data structure `gfkit` (Section 3), as well as the implications of our design decisions. Further, we describe `gfkit`'s query algorithm for computing the LCA (Section 3.3) and other common statistics used in population genetics (Section 3.4). We also describe the conversion of the `tskit` to the `gfkit` data structure (Section 3.6) as well as two variants of our data structure (Section 3.5 and Section 3.7). Further, we evaluate the performance of `gfkit` (Section 6), report speedups (Section 6.1 and Section 6.2) and analyze the created DAGs (Section 6.3 and Section 6.4). Next, we discuss the space usage of `gfkit` vs `tskit` (Section 6.6) and describe our qualitative observations concerning numerical stability (Section 7). Finally, we conclude, and highlight possible direction of further work (Section 8).

2 Related Work

The genetic code of related species or individuals of the same species is highly redundant, even when considering only variant sites [51]. Ané and Sanderson [4] thus propose compressing related genomic sequences using a phylogenetic tree. Here, the tips of the tree represent the genomic sequences, which are fully described by storing the ancestral state for each site at the root and the respective mutations along the edges of the tree (Figure 1.b). Ané and Sanderson primarily use a biologically reasonable evolutionary tree with an optimized parsimony score in order to attain a good compression ratio but not for explicitly modelling evolutionary history. Kelleher et al. [35, 36, 51] introduce so-called *tree sequences*, which model the evolutionary history of a set of related genomes, allow reusing some intermediate results when computing statistics on them, and enable space-efficient storage of these sequences (Section 1.5).

Matthews and Williams [43] compress a collection of trees by encoding each *bipartition* exactly once. Here, a bipartition is a separation of a tree's tips into two disjoint sets. Note, that each edge of a tree induces a bipartition and thus the set of all bipartitions fully describes a tree. However, this does not allow for direct access to the trees' topologies, which we require, for example, to compute the Lowest Common Ancestor (Section 3.3).

Directed Acyclic Graphs (DAGs) contain only directed edges and no path of length $\gneq 0$ from a node to itself. The tree terminology introduced in Section 1.1 generalizes to DAGs: We call out-degree 0 nodes *tips*, which represent samples and their associated genomes. Contrary to trees, DAGs may have multiple *root* nodes, that is, nodes with an in-degree of 0.

Theoretical computer science has studied the compression of single trees via DAG-compression [55, 10], tree grammars [17], and top-trees [9]. DAG-based compression reuses identical (topology and label) subtrees, when occurring multiple times in the same tree. In a tree, each node also induces/represents the subtree containing it and all of its descendants. Thus, instead of encoding a subtree a second time, we add an edge to the node representing the already encoded subtree, resulting in a DAG. In a single genealogical tree, all tips (representing samples) are distinct and thus not compressible using DAG-based compression. However, one can extend the idea of representing each unique subtree only once to forests by reusing subtrees that are part of multiple trees (Section 3). Ingels [32, 31] implements this idea, encoding a forest as a DAG. However, they focus on the enumeration of trees instead of on reusing intermediate results during computations. Tree grammars and top-tree based compression reuse tree patterns, that is any identically-connected subgraph (not necessarily including all descendants) [9]. It remains an open question if top-trees[6] can be used to encode the set of related genealogical trees while supporting the required queries efficiently.

To the best of our knowledge, two phylogenetic tools use DAGs to memoize intermediate results: ASTRAL-III, a tool for inferring a species tree from a set of topologically distinct gene trees, represents and processes unique tip bipartitions only once [67]. Further, Larget [38] uses memoization to compute the conditional clade probability only once per unique subtree.

Gene Recombination Graphs (GRGs) [15] encode the mutations of a collection of related genomes as a DAG in order to re-use computations. GRG are conceptually similar to our bipartition-based DAG (Section 3.5) and hence do not support LCA queries. Further, GRGs do neither model the evolutionary history of the described genomes nor encode mutations back to the ancestral state or recurrent mutations.

3 Design of the Genealogical Forest Data Structure

In recombining organisms, the collection of genealogical trees used to describe the related evolutionary histories of different parts of the genome share common subtrees (a node including all of its descendants).[7] Thus, we propose a data structure which encodes each unique subtree across *all* trees exactly once (Figure 1.d). We construct such a data structure by contracting the set of unconnected trees to a Directed Acyclic Graph (DAG) where each root node represents a tree and each non-root node represents a unique subtree in the input set of trees. Here, we consider each sample to be a subtree containing only itself. If two or more input trees share an identical subtree, the associated node in the DAG has multiple incoming edges; see for example `(B,C)` in Figure 1.d. These resulting DAGs are called *multitrees* [19]. In them, each subgraph and all nodes reachable from it induce a tree. Additionally, there is exactly one path from each root to each tip. In analogy to tree sequences (implemented in `tskit`), we call this data structure a *genealogical forest* and provide an implementation called `gfkit`[8]. We choose this naming to highlight that the trees describe genealogies and use the established term "forest" to describe a collection of trees.

We encode the (closely related) genomic sequences represented by the tips of the genealogical forest analogously to `tskit` (Figure 1.b). For this, we make the biologically realistic assumption that the evolutionary history of each genomic position is described by exactly one tree. Several adjacent positions along the genome can and will often share the same

[6] Minimal tree grammars are $\mathcal{NP}$-hard to construct, and they do not support efficient navigation [9].
[7] Branch lengths are irrelevant for genetic variation based statistics and the LCA, thus we omit them.
[8] `https://github.com/lukashuebner/gfkit`; LGPL

tree. This assumption can potentially be relaxed - see future work (Section 8). For each site with genetic variation[9], we store the ancestral state at the root and the respective mutations along the edges of the associated tree. Note, that each edge in a tree might thus be annotated by multiple mutations, as it describes the evolutionary history of multiple genomic sites. Additionally, note that there might be multiple mutations per genomic site, including recurring mutations and mutations back to the ancestral state.

3.1 Implications of Our Design

Nodes in a genealogical forest represent unique subtrees (Section 3). Thus, queries using the trees' topologies or the genetic sequences may easily reuse intermediate results on subtrees shared by *all* trees (*memoization*). In contrast, tree sequences allow for reusing intermediate results only among *adjacent* trees along the genome (see Section 1.5 for an explanation). We implement this memoization by storing the intermediate results in a lookup table, using the ID of the respective DAG node as the key. This memoization speeds up post-order traversals on the DAG, which we use to compute the LCAs (Section 3.3) and allele frequencies (Section 3.4), from which we derive numerous statistics in population genetics (Section 1.3).

3.2 Computing the Number of Selected Samples in a Subtree

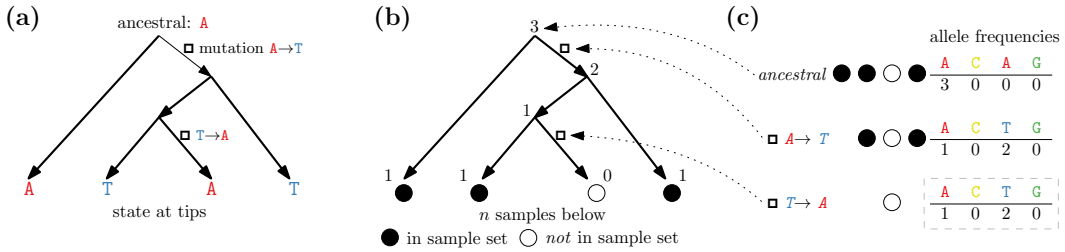

Figure 2 **(a)** Encoding a set of four sequences (here: one site) using a tree, an ancestral state, and mutations along the edges. **(b)** We compute the number of samples in the input sample set below each node using a post-order traversal. We need this information in order to **(c)** compute the allele frequencies (number of samples per genomic state) of each site by iterating over its mutations.

For each query to the genealogical forest data structure, the user selects a subset of the forest's samples to be considered (Section 1.3). Counting the number of selected samples which are part of each subtree of the genealogical forest (represented by a node in DAG) is an important building block for computing allele frequencies and the LCA. We define a post-order traversal on a DAG as iterating over all of its nodes such that all children of a node are processed before the node itself, thereby allowing us to reuse the intermediate result for these children using memoization (Section 3.1). Thus, to count the number of selected samples in each subtree, we assign $v(t) = 1$ for all tips t representing samples selected in the user query and $v(t) = 0$ to all other tips. Next, we perform a post-order traversal on the DAG by assigning to each node n_k the sum over the values assigned to its children $c(n_k)$, that is $v(n_k) = \sum_{n_c \in c(n_k)} v(n_c)$ (Figure 2.b).

[9] Invariant sites do not carry evolutionary information and are thus filtered out during pre-processing.

3.3 Computing the Lowest Common Ancestor

We compute the Lowest Common Ancestor (LCA, Section 1.2) of two or more tips in all trees represented by a genealogical forest using a variation of the algorithm described in Section 3.2. For this, we exploit that in a genealogical forest DAG, there exists exactly one path from each tip to each root. We chose this approach, as unrolling the DAG into a set of trees and answering LCA queries using the well studied techniques for trees (e.g., Schieber and Vishkin [56]) would require memory linear in the number of nodes per tree times the number of trees. Further, using a LCA algorithm developed for general DAGs (e.g., Kowaluk and Lingas [37]) is asymptotically slower as they cannot exploit the above property. First, we assign each tip t selected as part of the query the tuple $n(t) = (1, \varnothing)$ and all tips not selected the tuple $n(t) = (0, \varnothing)$. Here, the first element of the tuple counts the number of selected samples which are part of the subtree represented by the node. We accumulate this counter up the trees (bottom up from the tips) using a post-order traversal on the DAG. Additionally, if for any node the sample counter is equal to the number of selected samples for the first time, this node must be the LCA for this subtree. Thus, we store the ID of this node in the second element of the tuple and propagate this value up the tree, instead of the sample count. Note, that we might select multiple nodes in the DAG as LCAs in a single query. However, as noted above, there is exactly one path from each tip to each root. This ensures that for each root in the DAG (representing a genealogical tree), we pick *exactly one* node that is reachable from this root as the LCA.

The runtime of this algorithm depends on the number of edges in the DAG, but not on the number of samples in the input sample set. Thus, computing the LCA of a large subset of the samples takes no longer than computing the LCA of only two samples (Section 6.2).

Note, that we could modify this algorithm to compute an almost-LCA- the lowest node under which "almost all" samples are located. This could increase the robustness of the biological interpretation against single "rogue" samples which were erroneously included in the input sample set and cause the LCA to be much higher than it would be without them.

3.4 Computing the Allele Frequencies and Derived Statistics

Most population genetics statistics depend on the allele frequencies [51], that is, how many `A`s, `C`s, `T`s, and `G`s we observe at a given site. We are interested only in variant sites, where at least two of these counts are > 0. Also, the allele frequencies at different genomic sites are evidently independent of one another. Again, we allow for selecting a subset of samples for a query. Given a query, we first count the number of selected samples contained in each subtree represented by the nodes in the genealogical forest DAG (Section 3.2). Let $S = \{\texttt{A}, \texttt{C}, \texttt{T}, \texttt{G}\}$ be the set of alleles (here: possible genomic states) and $n(*)$ be the number of selected samples. For each site, we intend to compute the number of selected samples $n(x)$ which exhibit allele $x \in S$. First, we set all samples to be in the ancestral state, that is $n(x_{\text{ancestral}}) = n(*)$ and $n(x) = 0$ for all other x (Figure 2.c). Next, we iterate over the mutations at this site: Each mutation is associated with a subtree in the DAG and changes the state $x_i \in S$ of all samples contained in this subtree to the state $x_j \in S$. We thus successively decrement $n(x_i)$ and increment $n(x_j)$ by the number of selected samples contained in the respective subtree.

From the allele frequencies, we now derive numerous population genetics statistics. The sequence *diversity* [46], for instance, reflects the probability that two random sequences differ at a random site (both chosen uniformly at random). *Segregating Sites* are sites exhibiting more than one allele (i.e., polyallelic sites). The *Allele Frequency Spectrum (AFS)* [18] is a histogram over the allele frequencies. Some statistics also work on multiple sample sets. For

example the sequence *divergence* [46] is the probability that two sequences – each chosen uniformly at random from its own sample set – differ at any given site. More elaborate statistics can be derived from the above basic statistics, using up to four disjoint sample sets. Examples include *Patterson's* $f_{2,3,4}$ [49], *Tajima's D* [60], and the *Fixation Index* (F_{ST}) [66].

3.5 Memoizing on Shared Bipartitions

In contrast to computing the LCA, computing the allele frequencies is independent of the actual tree topologies as it requires only the sample bipartitions induced by the subtrees associated with the mutations. Remember, that these bipartition separate all tips of a tree into two disjoint sets (Section 2). For example, two identical mutations in two distinct subtrees $((G_0,G_1),G_2)$ and $(G_0,(G_1,G_2))$ will identically affect the respective allele frequencies. Memoizing on shared *bipartitions* allows reusing (slightly) more intermediate results than memoizing on shared *subtrees* (Section 6.5). Next, we detail the construction of the regular (unique subtrees) genealogical forest and this variant (unique bipartitions).

3.6 Constructing the Genealogical Forest DAG

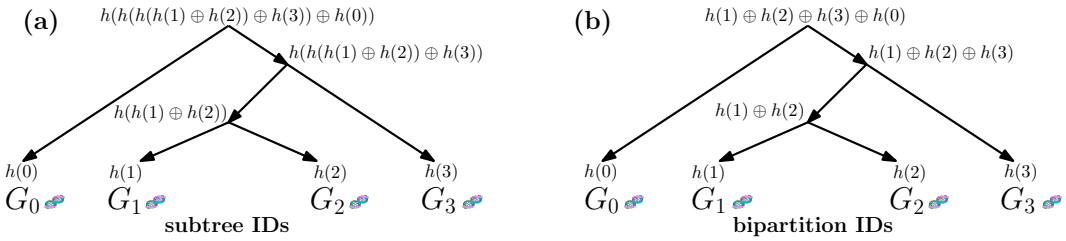

Figure 3 **(a)** Four sample genomes mapped to $t_i \in 0,1,2,3$. The unique subtree IDs of the tips (i.e., samples) are $h(t_i)$ where h is a pseudorandom hash function. We compute the unique subtree IDs for non-tip nodes by applying h to the exclusive or ($\oplus$) of all of its children. The resulting ID uniquely identifies a *subtree* including its topology. **(b)** Not breaking the linearity of the $\oplus$ results in a unique ID per set of samples in a subtree (i.e., not including its topology; called *bipartition*).

In order to construct the genealogical forest DAG, we assign unique IDs to all subtrees of the genealogical trees forming the input. For this, let h be a pseudorandom hash-function. We start by assigning unique subtree IDs to the tips $T = \{t_0, t_1, \ldots, t_{|T|-1}\}$ of the tree: $\mathrm{id}(t_0) = h(0), \mathrm{id}(t_1) = h(1), \ldots, \mathrm{id}(t_{|T|-1}) = h(t_{|T|-1})$. We then compute the unique subtree ID of each non-tip node by applying the bitwise exclusive or ($\oplus$) over all of its children, followed by hashing the result using h (Figure 3.a). That is, for a node n with children $c(n)$, we compute $\mathrm{id}(n) = h(\oplus_{j \in c(n)} \mathrm{id}(j))$. By using a symmetric function – i.e., $\mathrm{id}(j) \oplus \mathrm{id}(i) = \mathrm{id}(i) \oplus \mathrm{id}(j)$ – we ensure, that the order by which we process the children does not affect the resulting subtree ID. In order for the actual topology of the subtree to influence its ID, we break the linearity of the $\oplus$ operator using a hash function. Otherwise, the ID would solely be determined by the bipartition induced by the respective subtree. For example, the following two subtrees would have the same subtree ID: `((A,B)C)` & `(A(B,C))`.

Using a 128 bit hash function, ensures a collision probability of 10×10^{-16} when computing $\approx 3.8 \times 10^{11}$ hashes [7]. This corresponds approximately to the probability of a single bit-flip occurring at any given second in the 128 bit needed to output a single subtree ID [57].

As noted in Section 3.5, the allele frequencies are invariant w.r.t. the actual topology of a subtree, and solely by the samples contained in it. For computing a unique *bipartition* ID, we therefore remove the linearity-breaking hash function h from the computation of the non-tip IDs, i.e., $\mathrm{id}(n_i) = \oplus_{j \in c(n)} \mathrm{id}(j)$ for any non-tip node n.

Construction Algorithm

We construct the genealogical forest DAG from a given set of trees, by iterating over the input trees[10] and processing the nodes of each tree in post-order. For each tree node (representing the subtree induced by it and its descendants) encountered during this post-order traversal, we compute the unique subtree ID and add the subtree to the DAG if it is not already present. For the sake of simplicity, we ensure that there exists exactly one distinct root in the DAG for each tree in the collection of input trees. Even if two trees are topologically exactly identical, each is represented by a separate root node in the DAG. They of course still share all subtrees below their respective, distinct, root node.

Recent versions of `tskit` provide information on which edges were inserted or removed when moving from one tree to the next along the genome. We exploit this information in order to recompute only those subtree IDs that might have changed, that is, all subtrees induced by nodes which are on the path from a changed edge to the tree root. Using this information results in speedups of 2 to 5, depending on the dataset.

Analogously to `tskit`, we encode the genomic sequence by storing the ancestral state as well as the respective mutations annotated at *nodes* of the DAG for each genomic site.

3.7 Balanced-Parenthesis Encoding of a Forest

Instead of encoding a genealogical forest as a DAG using explicit nodes and edges, an encoding extending the balanced parenthesis encoding for trees [33] with back-references is possible. This encoding is space-efficient and a post-order traversal over a balanced parenthesis encoded tree is a simple linear scan. Exploring the full design space of this approach and providing an efficient implementation represents an interesting challenge. However, initial experiments did not yield substantial speedups, and hence we do not repeat them here.

4 Experimental Setup

We implement our algorithms in C++20 and build our tool using `CMake` 3.25.1, `gcc` 12.1, and `ld` 2.38. We run our experiments on an AMD EPYC 7551P processor running at 2 GHz with 64 MiB of shared L3, 512 KiB of core-local L2, 32 KiB of core-local L1 data, and 64 KiB core-local L1 instruction cache. We use 8 banks of 32 GiB DDR4 RAM running at 2667 MT s^{-1}. As our experiments are single threaded, only a single socket is being used. We compare `tskit` git rev `77faade5` and `gfkit` version `fbd2740`.[11]

5 Datasets

We evaluate our method on three freely-available empirical human (GRCh38) tree sequence collections (see Appendix for details): Thousand Genomes Project (TGP) [5] phase 3 autosomes, Simons Genome Diversity Project (SGDP) [42] autosomes, and the collection inferred by Wohns et al. [64] ("Unified"). We use all 22 autosomes for each of these collections in our experiments. We choose these specific tree sequence collections because to the best of our knowledge they constitute the only publicly-available empirical collections at present.

[10] The order of trees in not relevant.
[11] `https://github.com/tskit-dev/tskit` and `https://github.com/lukashuebner/gfkit`

Additionally, we simulate a human dataset containing 640 000 samples (see Appendix for details). As we do not observe substantial runtime differences between distinct chromosomes of the empirical genomic data collections, we limit our simulated data benchmarks to chromosome 20 to conserve computation resources and reduce our CO2 footprint.[12]

6 Evaluation

We evaluate the implementation of our proposed data structure "genealogical forests" (`gfkit`) regarding query speed (Section 6.1 and Section 6.2) and storage space used (Section 6.6). Additionally, we assess the algorithmic reasons for the obtained speedups (Section 6.4 and Section 6.3) and report the time required to convert `tskit` tree sequences into the `gfkit` data structure (Section 6.7).

6.1 Speedup for Computing Statistics Based on the Allele Frequencies

We evaluate the runtime of our proposed genealogical forest data structure and its associated implementation `gfkit` by comparing it to the state-of-the-art reference implementation of tree sequences, `tskit`. We benchmark the runtimes of various important statistics in population genetics (Section 1.3), including statistics that are based on the allele frequencies (e.g., AFS) as well as on the topology (LCA). We use 10 repeats for each runtime measurement on each of the 22 autosomal chromosomes of each empirical collection (TGP, SGDP, and Unified; see Section 5) and on chromosome 20 of the simulated dataset. The mean standard deviation of runtimes across the 10 repeats of each statistic, collection, and chromosome is below 1.3 %. We report a median speedup of 4.0 of `gfkit` (ours) over `tskit` (state-of-the-art) for computing various allele frequency-based statistics (Figure 4). The absolute runtimes range from 775 to 1770 ms for `tskit` and 152 to 340 ms for `gfkit`.

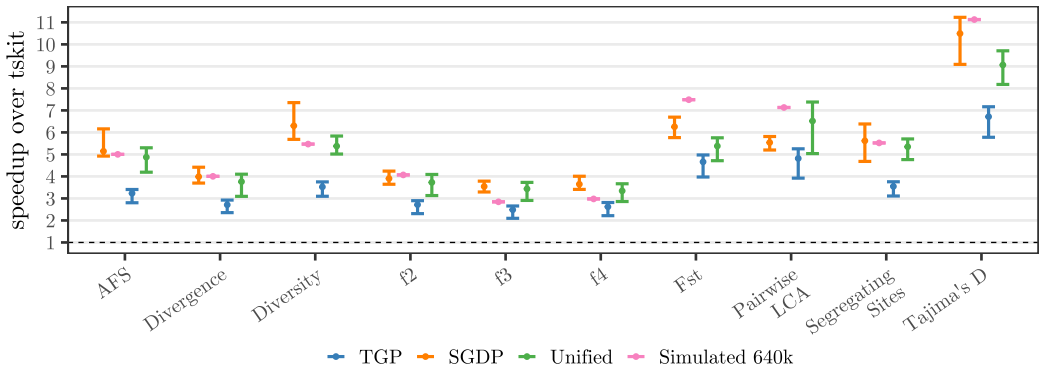

Figure 4 Speedup of `gfkit` (ours) over `tskit` (state-of-the-art) for computing statistics on three empirical and one simulated dataset (human). The bars indicate the speedup range and the dot the median speedup across all chromosomes of the respective collection. We use all 22 autosomal chromosomes from the Thousand Genome Project (TGP), Simons Genome Diversity Project (SGDP), and the Unified collection. We use chromosome 20 of a simulated human collection containing 640 000 samples. AFS: Allele Frequency Spectrum. f{2,3,4}: Patterson's f. Fst: Fixation Index.

[12] Wohns et al. [64] use chromosome 20, as they consider it representative for genome-wide patterns.

Profiling using Intel VTune shows that `gfkit` spends over 90 % of its runtime in the post-order traversal during the computation the allele frequencies. The numerical computations for the actual statistics thus amount for less than 10 % of runtime. Therefore, the number of nodes in the genealogical forest DAG, as well as the in-degree of those nodes appear to constitute the dominating runtime factors. These values correspond to the number of unique subtrees in the genealogical trees used to construct the genealogical forest, and the number of times that an intermediate result can be reused during the post-order traversal, respectively. Thus, we provide more details on these measurements (Section 6.3 and Section 6.4).

6.2 Speedup for Computing the Lowest Common Ancestor

We compare `gfkit`'s and `tskit`'s runtimes for computing the Lowest Common Ancestor (LCA) of two ("pairwise") or more samples. The asymptotic runtime of `gfkit`'s LCA-algorithm (Section 3.3) does not depend on the number of selected samples. In contrast, given three or more samples, `tskit` performs an analogous number of LCA-queries: First, it computes the LCA of two arbitrary samples, which it subsequently uses as input for the next LCA query; together with another sample from the input. After processing all samples, `tskit` obtains the overall LCA. Therefore, `tskit`'s LCA-algorithm scales linearly with the number of samples in the input.

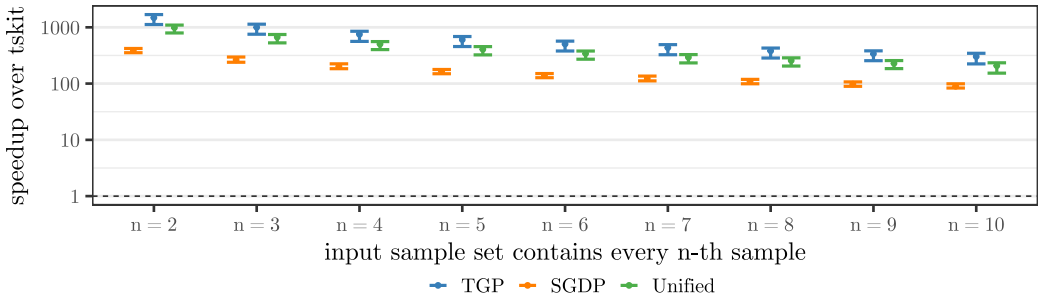

Figure 5 Speedup of `gfkit` (ours) over `tskit` (state-of-the-art) for computing the Lowest Common Ancestor (LCA) of every n-th sample in the dataset. The runtime of `gfkit`'s LCA-algorithm does not depend on the number of samples in the sample set. In contrast, the runtime of `tskit`'s LCA-algorithm depends linearly on the number samples in the sample set.

We report the speedups of `gfkit` over `tskit` when computing the LCA (Figure 4 and Figure 5). In order to save computational resources and reduce our environmental footprint, we perform only 3 repeats when computing the LCA of more than two samples using `tskit` (runtime up to 34 min). We observe speedups of `gfkit` over `tskit` of 5.5 (median) for pairwise queries, 208 (median) when selecting 10 % of the samples, and 990 (median) when selecting 50 % of the samples as input sample set. Experiments on the simulated dataset took 64 min for `gfkit` (median 155 ms per query) but did not finish for `tskit` in a week.

6.3 Proportion of Subtrees that are Unique

Each query spends the majority of its time in the post-order traversal. Thus, the number of nodes and edges in the `gfkit` DAG is *the* determining runtime factor. The number of unique subtrees in the input tree set determines the number of nodes in the DAG. We report the number of overall subtrees in the input as well as the absolute and relative number of unique subtrees (Table 1). For example, in the TGP collection, 0.48 % (mean) of the subtrees are

unique per chromosome. Additionally, we report that the absolute number of unique subtrees in a simulated dataset (human, chromosome 20) with 640 000 samples is not substantially higher than for the shown empirical collections containing 554 to 7508 samples, respectively.

Table 1 Number of overall and unique (i.e, distinct) subtrees and the proportion of subtrees that are unique. The ranges given cover all 22 autosomal chromosomes of the respective collection. We report the arithmetic mean and standard deviation across all chromosomes of a collection. TODO Check new Trees column.

Collection	Chr.	Samples	Trees $\times 10^6$	Subtrees $\times 10^6$	Uniq. Subtrees $\times 10^6$	Uniq. Subtrees / Subtrees
SGDP	all	554	0.1 to 0.7	84 to 587	1.8 to 11	0.020 ± 0.001
TGP	all	5008	0.3 to 2.1	2150 to 14 916	11 to 69	0.0048 ± 0.0003
Unified	all	7508	0.04 to 0.2	698 to 2708	3.2 to 18	0.006 ± 0.002
Sim. 640k	20	640 000	0.5	695 185	14	0.000 02

In conclusion, only 0.002 to 2 % of the subtrees in the tested collections are unique. This proportion decreases as we add more samples (and thus subtrees) from the same species. This is due to the absolute number of unique subtrees not increasing substantially for larger collections, thus, neither does the runtime of the queries. The number of mutations per subtree is 3 to 4 orders of magnitude smaller[13] for the 640 000-sample dataset compared to the collections with $\leq$ 7508 samples. Therefore, there is less signal to resolve the evolutionary history of the samples, possibly leading to larger unresolved subtrees. It remains an open question how other data characteristics (for example the species when not considering human populations) influence the number of unique subtrees.

6.4 Reusing Shared Subtrees

Apart from to the number of nodes in the `gfkit` DAG, the number of edges also influences the runtime of the post-order traversal. However, the proportion of edges to nodes also serves as an indicator for memoization performance. During the post-order traversal, the in-degree of a node is equal to the number of times its result is used (and reused). We report the mean in-degree across all non-root nodes in all trees across all chromosomes of each collection (Appendix). Thus, each intermediate result on a unique subtree is reused on average 4.12 (SGDP), 5.48 (TGP), 5.59 (Unified), or 5.10 (Simulated 640k) times.

6.5 Speedups when Memoizing on Shared Bipartitions

We also benchmark queries on the bipartition-DAG described in Section 3.5. Here, we observe a median speedup of 4.7 over `tskit` across all empirical collections (Appendix). Queries on the bipartition DAG are on average 1.14 ± 0.09 (mean $\pm$ sd) times faster than on the subtree-DAG. There are 5 to 20 % fewer unique bipartitions than there are unique subtrees; corresponding to fewer nodes in the bipartition-DAG. The average in-degree in the bipartition-DAG, and thus number of times we reuse an intermediate result, ranges from 4.44 to 5.20 (Appendix), compared to 4.12 to 5.49 on the subtree-DAG. These two measurements explain the (moderately) faster runtimes of the bipartition-DAG compared to the subtree-DAG. However, as the bipartition-DAG does not support topology-aware queries (e.g, LCA), we do not consider the tradeoff worthwhile with respect to the increased code complexity and storage unless additional optimizations further reduce the runtimes.

[13] Mutations per subtree: $\approx$ Sim. 640k: 0.000 000 7, SGDP: 0.002, TGP: 0.0002, and Unified: 0.003

6.6 Storage Space Needed for Encoding the Forest

In a sense, the genealogical forest DAG factors-out all unique subtrees that a tree sequence describes using edge insertions and removals. Concerning the space usage, there are two contrary effects at play here: On the one hand, tree sequences store some identical subtrees multiple times, but using distinct edges and/or nodes. On the other hand, a single edge insertion or removal can induce multiple new subtrees along the path from the insertion/removal point up to the tree root. In the first case, genealogical forest are the more space-efficient representation, in the second case, tree sequences are.

In order to quantify the trade-off between the two effects, we compare the size of the genealogical forest DAG against the size of the respective tree sequence. However, the tree sequence implementation `tskit` stores a variety of additional meta-data, to support operations which we currently do not implement in `gfkit`. We thus perform a theoretical, instead of an empirical space requirement comparison.

A tree sequence as well as a genealogical forest could minimally be described by its (directed) edges plus sequence information. Let ι be the number bits needed to encode a node ID and ϕ be the number of bits needed to encode a position in the genome. Thus, choosing an edge list (see below) we need $2 \cdot \iota$ bit for each edge in a genealogical forest. In tree sequences, an edge is valid for a specified region of the genome, requiring $2 \cdot \phi$ additional bits per edge to store this information. Note, however, that the number of nodes and edges differs between `tskit` and `gfkit` and these numbers are thus not directly comparable. Additionally, we cannot arbitrarily sort `tskit`'s edges without incurring a performance penalty, as we require the insertion order in order to efficiently move from one tree to the next along the genome.[14] In `gfkit`, all outgoing edges of a node could be stored consecutively in an edge list, which would allow omitting some of the `from` node IDs, conceptually creating an adjacency array.

Both, `tskit` and `gfkit`, describe the genomic sequences using one ancestral state and a list of mutations per genome site. We can encode each mutation using a site ID, the node ID at which the mutation occurs, plus the derived state. Let s be the number of sites, σ be the number of bits per site ID, γ be the number of bits per genomic state, and m be the overall number of mutations. The sequence information thus (theoretically) requires $s \cdot \gamma + m \cdot (\sigma + \phi + \gamma)$ bit. In practice, `tskit` and `gfkit` store additional information: For example, in order to avoid traversing up the tree to the parent mutation (or root node), `tskit` stores a pointer to the parent mutation (64 bit) and `gfkit` stores the parent mutation's state (2 bit). However, we omit these and other implementation-specific constants here.

Table 2 Theoretical space usage of `tskit` vs `gfkit`. The description of the trees is substantially larger than the description of the sequence (ancestral states and mutations). `tskit` encodes the trees as a tree sequence (Section 1.5), whereas `gfkit` encodes them using a genealogical forest (Section 3).

Collection	Chr.	`tskit`	`gfkit`	Sequence
SGDP	all	1076 MiB	3891 MiB	123 MiB
TGP	all	4577 MiB	32 134 MiB	310 MiB
Unified	all	1684 MiB	7843 MiB	874 MiB
Sim. 640k	20	53 MiB	198 MiB	3.74 MiB

[14] We could determine the deletion order on the fly by using a priority queue containing only the currently active edges.

For our calculations (Table 2), we assume $\iota = \phi = \sigma = 32$ bit integers for node IDs, site IDs, and genomic positions[15] and four possible genomic states, thus $\gamma = 2$ bit. We conclude that the edge removals and insertions used to store the set of genealogical trees by `tskit` are more-space efficient than `gfkit`'s method of storing a node for each unique subtree. However, we argue that, as the differences are in the same order of magnitude than implementation-specific constants (e.g., choice of data-types or explicitly storing IDs), one could mitigate against this effect. Therefore, it remains a tradeoff between query-speed and space usage. It is subject of future work to evaluate the genealogical forest variants using the compressed edge list (above) and the balanced parenthesis representation (Section 3.7).

6.7 Converting Tree Sequences to Genealogical Forests

To the best of our knowledge, there is no fundamental reason why tools could not output genealogical forests instead of tree sequences once we add support for storing coalescent times (Section 8). Currently, however, these tools output the established `tskit` format – and will probably continue doing so for a while, since the current implementation of our data structure, does not support all features of `tskit` yet. While we provide corresponding `gfkit` files (Appendix) for the openly available `tskit` collections (Section 5), the time taken to convert `tskit` to `gfkit` input files is not negligible in general.

Kelleher et al. [36] report tree sequence inference times of 5 min for chromosome 20 of SGDP (Section 5) and 2 h for chromosome 20 of TGP using 40 cores. We convert their output files from the `tskit` to the `gfkit` format on a single core in 6.5 s and 123 s respectively. We thus argue that one can convert empirical collections in a negligible amount of time compared to the time required for inferring them. Additionally, in principle, tree sequence inference could directly output the `gfkit` format, thereby entirely eliminating the conversion step.

7 Numerical Stability

Basic operations such as additions or multiplications on IEEE 754 floating point numbers always induce rounding errors [20]. The magnitude of these errors directly depends on the difference order of magnitude between the two operands. Thus, computing running sums, where we sum many small values to an ever-growing sum, is particular rounding-error prone.

We often compute statistics in population genetic per site (Section 1.3) and then average over all sites. Using a running sum, as currently implemented in `tskit` and `gfkit`, leads to increasing round-off errors the more variant genetic sites there are. Some statistics, like Patterson's f_4 entail multiplying the number of samples in four different sample sets. With more than 262 143 samples[16], the product might exceed a 64 bit integer. We (for the sake of result verification), as well as `tskit`, thus choose to represent these products as 64 bit IEEE 754 floating point numbers, potentially introducing substantial numerical errors.

It is currently unknown if errors introduced by these simplifications influence the biological interpretation of the results. Further, rounding errors imply that the order of operations influences the result, even though the input operands are bit-identical, thus impacting reproducibility. Even when the software version and the random seeds are fixed, different algorithms, compiler optimizations, or degrees of parallelism influence the order of operations. Reports on these effects influencing the results of scientific computations exist for example in the fields of phylogenetics [13], sheet metal forming [16], and fluid simulations [62].

[15] This is true for `gfkit`. However, `tskit` uses $\phi = 64$ bit IEEE 754 floating points for genomic positions.
[16] When the four sample sets have the same size: $(262144/4)^4 = 2^{64}$

An analysis of the impact of numerical errors on population genetics statistics, their downstream analyses, and result reproducibility is needed. Ideally, a standardized order of calculations would lead to all software in the field outputting comparable and reproducible results with well-understood numerical inaccuracies. Further, results should be invariant under addition of further, but ignored in the query at hand, genomes and mutations to the dataset. Given numerical calculations require less than 10 % of overall runtime, we are optimistic that introducing arbitrary precision calculations could have negligible overhead.

8 Conclusion and Future Work

In recombining organisms, a set of genealogical trees is often used to describe the evolutionary history of the studied samples. While tree sequences (`tskit`) compress these trees using edit operations from one tree to the next along the genome, genealogical forest (`gfkit`) compresses them into a DAG where each node represents a unique subtree of the input (Section 3). While the genealogical forest encoding (theoretically) requires a factor 3.6 to 7.0 more space than tree sequences (Section 6.6), it also yields speedups by a factor of 2.1 to 11.2 (median 4.0) when computing pairwise LCA queries and important statistics in population genetics (e.g., sequence diversity). In contrast to `tskit`'s LCA-algorithm, the runtime of `gfkit`'s LCA-algorithm does not depend on the number of samples selected in the query. Thus, `gfkit`'s LCA queries are substantially faster, e.g. a speedup of 208 (median) when querying 10 % of samples and a speedup of 990 (median) when querying 50 % of samples. Additionally, we describe, implement, and benchmark an alternative data structure, which represents the set of genealogical trees as a DAG in which each node represents a unique bipartition in the input. This variant is slightly faster but does not support topology-aware queries (e.g., LCA). To explain this, we show, that subtrees containing the same samples but having a different topology are actually rare in the analyzed datasets.

As more genomes are being sequenced, future datasets will contain a multiple of today's samples. However, we observe speedups (and runtimes) comparable to the largest existing empirical datasets for a simulated human dataset with 640 000 samples (Section 6.1). We show, that this might be due to the number of unique subtrees being comparable to those of current empirical datasets with at most 7508 samples, thus resulting in DAGs of about the same size. As the post-order traversal on the genealogical forest DAG takes up over 90 % of a query's runtime, the size of this DAG is the determining runtime factor. Which characteristics of the input influence the number of unique subtrees remains an open question.

`Tskit` was not build in a day and neither was `gfkit`. Currently, `gfkit` does not support all `tskit` features, yet most of these other features should be straightforward to implement. For example, `gfkit` currently supports only sample weights of 0 and 1 (a sample is in the sample set, or it is not), as these are sufficient for implementing the allele frequencies and derived statistics as well as LCA queries. Further, `gfkit` currently supports only site statistics on the entire genome, that is, neither branch- nor node-based statistics as well as no sliding windows over the genomic sites. We plan on adding support for storing coalescent times at inner nodes, enabling branch-length based statistics and the application of LCAs described in Section 1.2. Additionally, `gfkit` assumes that for each genomic site, a single tree describes the evolutionary history of all tips, while `tskit` allows for multiple (partial) trees. Tree sequences are an instantiation of Ancestral Recombination Graphs and can be augmented with nodes representing a recombination event between to individuals [65]. In a genealogical forest each tree describes the evolutionary history of *a part of the* genome. Thus, the full genetic code of an individual is described by a subset of nodes and a recombination event can be annotated between these subsets. This is, however, not implemented yet.

Next to lifting these limitations, additional improvements include evaluating a top-tree based genealogical forest (Section 2) and extending LCA queries with more than two samples to also report ancestors which are common to "almost all" selected samples, thus making the query robust against single samples erroneously included in the query (Section 3.3).

Tree sequences opened up new possibilities of storing and processing large sets of genealogical trees and sequences used in population genetics. Genealogical forests provide a substantial reduction in runtime for site-based statistics and LCA calculations over the state-of-the-art. Additionally, they make developing efficient algorithms more intuitive as these algorithms can be implemented for example as a post-order traversal with automatic reuse of intermediate results. We believe that these improvements will boost the development of new analyses in the field of population genetics. For instance, they might be used for developing appropriate optimization criteria and techniques for automatically determining subpopulations.

References

1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. On finding lowest common ancestors in trees. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, STOC '73, pages 253–265. ACM, 1973. doi:10.1145/800125.804056.

2 Morten E. Allentoft, Martin Sikora, Anders Fischer, Karl-Göran Sjögren, Andrés Ingason, et al. 100 ancient genomes show repeated population turnovers in neolithic denmark. *Nature*, 625(7994):329–337, January 2024. doi:10.1038/s41586-023-06862-3.

3 A. C. Allison. Polymorphism and natural selection in human populations. *Cold Spring Harbor Symposia on Quantitative Biology*, 29(0):137–149, January 1964. doi:10.1101/sqb.1964.029.01.018.

4 Cécile Ané and Michael J. Sanderson. Missing the forest for the trees: Phylogenetic compression and its implications for inferring complex evolutionary histories. *Systematic Biology*, 54(1):146–157, February 2005. doi:10.1080/10635150590905984.

5 Adam Auton, Gonçalo R. Abecasis, David M. Altshuler, Richard M. Durbin, Gonçalo R. Abecasis, et al. A global reference for human genetic variation. *Nature*, 526(7571):68–74, September 2015. doi:10.1038/nature15393.

6 Gautrey Barrett PH, Herbert PJ, Kohn S, and Smith S D, editors. *Charles Darwin's Notebooks, 1836-1844*. British Museum (Natural History), 1987.

7 Mihir Bellare and Philipp Rogaway. Introduction to modern cryptography, 2005.

8 Bonnie Berger, Noah M. Daniels, and Y. William Yu. Computational biology in the 21st century: scaling with compressive algorithms. *Commun. ACM*, 59(8):72–80, July 2016. doi:10.1145/2957324.

9 Philip Bille, Inge Li Gørtz, Gad M. Landau, and Oren Weimann. Tree compression with top trees. *Inf. Comput.*, 243:166–177, August 2015. doi:10.1016/j.ic.2014.12.012.

10 Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 373–389. SIAM, January 2011. doi:10.1137/1.9781611973082.30.

11 Clare Bycroft, Colin Freeman, Desislava Petkova, Gavin Band, Lloyd T. Elliott, et al. The uk biobank resource with deep phenotyping and genomic data. *Nature*, 562(7726):203–209, October 2018. doi:10.1038/s41586-018-0579-z.

12 Meredith L. Carpenter, Jason D. Buenrostro, Cristina Valdiosera, Hannes Schroeder, Morten E. Allentoft, et al. Pulling out the 1%: Whole-genome capture for the targeted enrichment of ancient dna sequencing libraries. *The American Journal of Human Genetics*, 93(5):852–864, November 2013. doi:10.1016/j.ajhg.2013.10.002.

13 Diego Darriba, Tomáš Flouri, and Alexandros Stamatakis. The state of software for evolutionary biology. *Molecular Biology and Evolution*, 35(5):1037–1046, January 2018. doi:10.1093/molbev/msy014.

14 Charles Darwin. *On the Origin of Species by Means of Natural Selection, or the PReservation of Favoured Races in the Struggle for Life*. John Murray, 1859.

15 Drew DeHaas, Ziqing Pan, and Xinzhu Wei. Genotype representation graphs: Enabling efficient analysis of biobank-scale data. *bioRxiv*, April 2024. doi:10.1101/2024.04.23.590800.

16 Kai Diethelm. The limits of reproducibility in numerical simulation. *Comput. Sci. Eng.*, 14(1):64–72, January 2012. doi:10.1109/mcse.2011.21.

17 Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, October 1980. doi:10.1145/322217.322228.

18 R. A. Fisher. Xvii.–the distribution of gene ratios for rare mutations. *Proceedings of the Royal Society of Edinburgh*, 50:204–219, 1931. doi:10.1017/s0370164600044886.

19 George W. Furnas and Jeff Zacks. Multitrees: enriching and reusing hierarchical structure. In Beth Adelson, Susan T. Dumais, and Judith S. Olson, editors, *Conference on Human Factors in Computing Systems, CHI 1994, Boston, Massachusetts, USA, April 24-28, 1994, Proceedings*, CHI '94, pages 330–336. ACM, 1994. doi:10.1145/191666.191778.

20 David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv.*, 23(1):5–48, March 1991. doi:10.1145/103162.103163.

21 Ernst Haeckel. *Allgemeine Anatomie der Organismen*. Georg Reimer, 1866.

22 Cody E. Hinchliff, Stephen A. Smith, James F. Allman, J. Gordon Burleigh, Ruchi Chaudhary, et al. Synthesis of phylogeny and taxonomy into a comprehensive tree of life. *Proceedings of the National Academy of Sciences*, 112(41):12764–12769, September 2015. doi:10.1073/pnas.1423041112.

23 Edward Hitchcock. *Elementary Geology*. 1840.

24 Paul A. Hohenlohe, W. Chris Funk, and Om P. Rajora. Population genomics for wildlife conservation and management. *Molecular Ecology*, 30(1):62–82, November 2020. doi:10.1111/mec.15720.

25 Lukas Hübner and Alexandros Stamatakis. Genealogical Forest Files for Simons Genome Diversity Project. Dataset (visited on 2024-08-13). URL: https://doi.org/10.5281/zenodo.11241730.

26 Lukas Hübner and Alexandros Stamatakis. Genealogical Forest Files for Thousand Genome Project. Dataset (visited on 2024-08-13). URL: https://doi.org/10.5281/zenodo.11241619.

27 Lukas Hübner and Alexandros Stamatakis. Genealogical Forest Files for Unified Genome (Wohns 2022). Software (visited on 2024-08-13). URL: https://doi.org/10.5281/zenodo.11241788.

28 Lukas Hübner and Alexandros Stamatakis. gfkit. Software, swhId: swh:1:dir:bffece502c3a579271ac3d91d66b051d089d02f2 (visited on 2024-08-13). URL: https://github.com/lukashuebner/gfkit.

29 Lukas Hübner and Alexandros Stamatakis. Tree Sequence and Genealogical Forest Files for a Simulated Human Chromosome 20. Dataset (visited on 2024-08-13). URL: https://doi.org/10.5281/zenodo.11241938.

30 Richard R. Hudson. Properties of a neutral allele model with intragenic recombination. *Theoretical Population Biology*, 23(2):183–201, April 1983. doi:10.1016/0040-5809(83)90013-8.

31 Florian Ingels. *On the similarities of trees: the interest of enumeration and compression methods. (Sur la similarité des arbres : l'intérêt des méthodes d'énumération et de compression)*. PhD thesis, École normale supérieure de Lyon, France, 2022. URL: https://tel.archives-ouvertes.fr/tel-03908078.

32 Florian Ingels and Romain Azaïs. A reverse search method for the enumeration of unordered forests using dag compression. *WEPA 2020 - Fourth International Workshop on Enumeration Problemsand Applications,*, 2020.

33 Guy Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 549–554. IEEE Computer Society, 1989. doi:10.1109/sfcs.1989.63533.

34 Konrad J. Karczewski, Laurent C. Francioli, Grace Tiao, Beryl B. Cummings, Jessica Alföldi, et al. The mutational constraint spectrum quantified from variation in 141,456 humans. *Nature*, 581(7809):434–443, May 2020. doi:10.1038/s41586-020-2308-7.

35 Jerome Kelleher, Alison M. Etheridge, and Gilean McVean. Efficient coalescent simulation and genealogical analysis for large sample sizes. *PLoS Comput. Biol.*, 12(5):e1004842, May 2016. doi:10.1371/journal.pcbi.1004842.

36 Jerome Kelleher, Yan Wong, Anthony W. Wohns, Chaimaa Fadil, Patrick K. Albers, et al. Inferring whole-genome histories in large population datasets. *Nature Genetics*, 51(9):1330–1338, September 2019. doi:10.1038/s41588-019-0483-y.

37 Miroslaw Kowaluk and Andrzej Lingas. LCA queries in directed acyclic graphs. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 241–248. Springer, 2005. doi:10.1007/11523468_{2}{0}.

38 Bret Larget. The estimation of tree posterior probabilities using conditional clade probability distributions. *Systematic Biology*, 62(4):501–511, April 2013. doi:10.1093/sysbio/syt014.

39 R C Lewontin and J L Hubby. A molecular approach to the study of genic heterozygosity in natural populations. ii. amount of variation and degree of heterozygosity in natural populations of drosophila pseudoobscura. *Genetics*, 54(2):595–609, August 1966. doi:10.1093/genetics/54.2.595.

40 Mark Lipson, Elizabeth A. Sawchuk, Jessica C. Thompson, Jonas Oppenheimer, Christian A. Tryon, et al. Ancient dna and deep population structure in sub-saharan african foragers. *Nature*, 603(7900):290–296, February 2022. doi:10.1038/s41586-022-04430-9.

41 Gordon Luikart, Marty Kardos, Brian K. Hand, Om P. Rajora, Sally N. Aitken, et al. *Population Genomics: Advancing Understanding of Nature*, pages 3–79. Springer International Publishing, 2018. doi:10.1007/13836_2018_60.

42 Swapan Mallick, Heng Li, Mark Lipson, Iain Mathieson, Melissa Gymrek, et al. The simons genome diversity project: 300 genomes from 142 diverse populations. *Nature*, 538(7624):201–206, September 2016. doi:10.1038/nature18964.

43 Suzanne J. Matthews, Seung-Jin Sul, and Tiffani L. Williams. A novel approach for compressing phylogenetic trees. In Mark Borodovsky, Johann Peter Gogarten, Teresa M. Przytycka, and Sanguthevar Rajasekaran, editors, *Bioinformatics Research and Applications, 6th International Symposium, ISBRA 2010, Storrs, CT, USA, May 23-26, 2010. Proceedings*, volume 6053 of *Lecture Notes in Computer Science*, pages 113–124. Springer, 2010. doi:10.1007/978-3-642-13078-6_13.

44 Craig D. Millar, Leon Huynen, Sankar Subramanian, Elmira Mohandesan, and David M. Lambert. New developments in ancient genomics. *Trends in evology & evolution*, 23(7):386–393, July 2008. doi:10.1016/j.tree.2008.04.002.

45 David A. Morrison. Genealogies: Pedigrees and phylogenies are reticulating networks not just divergent trees. *Evolutionary Biology*, 43(4):456–473, February 2016. doi:10.1007/s11692-016-9376-5.

46 M Nei and W H Li. Mathematical model for studying genetic variation in terms of restriction endonucleases. *Proceedings of the National Academy of Sciences*, 76(10):5269–5273, October 1979. doi:10.1073/pnas.76.10.5269.

47 Atsuko Okazaki, Satoru Yamazaki, Ituro Inoue, and Jurg Ott. Population genetics: past, present, and future. *Human Genetics*, 140(2):231–240, July 2020. doi:10.1007/s00439-020-02208-5.

48 M. Parks, S. Subramanian, C. Baroni, M. C. Salvatore, G. Zhang, et al. Ancient population genomics and the study of evolution. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 370(1660):20130381, January 2015. doi:10.1098/rstb.2013.0381.

49 Nick Patterson, Priya Moorjani, Yontao Luo, Swapan Mallick, Nadin Rohland, et al. Ancient admixture in human history. *Genetics*, 192(3):1065–1093, November 2012. doi:10.1534/genetics.112.145037.

50 David Penny. Inferring phylogenies.–joseph felsenstein. 2003. sinauer associates, sunderland, massachusetts. *Systematic Biology*, 53(4):669–670, August 2004. doi:10.1080/10635150490468530.

51 Peter Ralph, Kevin Thornton, and Jerome Kelleher. Efficiently summarizing relationships in large samples: A general duality between statistics of genealogies and genomes. *Genetics*, 215(3):779–797, July 2020. doi:10.1534/genetics.120.303253.

52 UMA RAMAKRISHNAN and ELIZABETH A. HADLY. Using phylochronology to reveal cryptic population histories: review and synthesis of 29 ancient dna studies. *Molecular Ecology*, 18(7):1310–1330, March 2009. doi:10.1111/j.1365-294x.2009.04092.x.

53 David Reich. *Who we are and how we got here*. Oxford University Press, Oxford, 2019.

54 O. P. Roja. *Population Genomics: Concepts, Approaches, and Applications*. Sprinter Nature Switzerland AG, 2019.

55 Sherif Sakr. XML compression techniques: A survey and comparison. *J. Comput. Syst. Sci.*, 75(5):303–322, August 2009. doi:10.1016/j.jcss.2009.01.004.

56 Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. Comput.*, 17(6):1253–1262, 1988. doi:10.1137/0217079.

57 Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. Dram errors in the wild: a large-scale field study. *ACM SIGMETRICS Performance Evaluation Review*, 37(1):193–204, June 2009. doi:10.1145/2492101.1555372.

58 Patrick Sung and Hannah Klein. Mechanism of homologous recombination: mediators and helicases take on regulatory functions. *Nature Reviews Molecular Cell Biology*, 7(10):739–750, August 2006. doi:10.1038/nrm2008.

59 Megan A. Supple and Beth Shapiro. Conservation of biodiversity in the genomics era. *Genome Biology*, 19(1), September 2018. doi:10.1186/s13059-018-1520-3.

60 F Tajima. Statistical method for testing the neutral mutation hypothesis by dna polymorphism. *Genetics*, 123(3):585–595, November 1989. doi:10.1093/genetics/123.3.585.

61 Daniel Taliun, Daniel N. Harris, Michael D. Kessler, Jedidiah Carlson, Zachary A. Szpiech, et al. Sequencing of 53,831 diverse genomes from the nhlbi topmed program. *Nature*, 590(7845):290–299, February 2021. doi:10.1038/s41586-021-03205-y.

62 Matthias Wiesenberger, Lukas Einkemmer, Markus Held, Albert Gutierrez-Milla, Xavier Saez, and Roman Iakymchuk. Reproducibility, accuracy and performance of the feltor code and library on parallel computer architectures. *Comput. Phys. Commun.*, 238:145–156, May 2019. doi:10.1016/j.cpc.2018.12.006.

63 George C. Williams and Randolph M. Nesse. The dawn of darwinian medicine. *The Quarterly Review of Biology*, 66(1):1–22, March 1991. doi:10.1086/417048.

64 Anthony Wilder Wohns, Yan Wong, Ben Jeffery, Ali Akbari, Swapan Mallick, et al. A unified genealogy of modern and ancient genomes. *Science*, 375(6583), February 2022. doi:10.1126/science.abi8264.

65 Yan Wong, Anastasia Ignatieva, Jere Koskela, Gregor Gorjanc, Anthony W. Wohns, et al. A general and efficient representation of ancestral recombination graphs. *arxiv*, November 2023. doi:10.1101/2023.11.03.565466.

66 Sewall Wright. Genetical structure of populations. *Nature*, 166(4215):247–249, August 1950. doi:10.1038/166247a0.

67 Chao Zhang, Maryam Rabiee, Erfan Sayyari, and Siavash Mirarab. ASTRAL-III: polynomial time species tree reconstruction from partially resolved gene trees. *BMC Bioinform.*, 19-S(6):15–30, 2018. doi:10.1186/s12859-018-2129-y.

A Dataset Details

Tree sequence datasets were inferred using `tsinfer` [36] version `0.2.1` and dated using `tsdate` [64] version `0.1.4`. The description on Zenodo is incorrect; see: `https://tskit-dev.slack.com/archives/C010834D669/p1713340040404519`. We simulate the "Sim. 640k" dataset containing 640 000 samples using `stdpopsim 0.2.0`[17] and the `HapMapII_GRCh38` genetic map. We convert tree sequence to genealogical forest files via `gfkit` version `fbd2740`.

Collection	Chr.	Format	Author	Link
SGDP	all	`tskit`	Wohns et al. [64]	`https://doi.org/10.5281/zenodo.3052359`
SGDP	all	`gfkit`	us	`https://doi.org/10.5281/zenodo.11241730`
TGP	all	`tskit`	Wohns et al. [64]	`https://doi.org/10.5281/zenodo.3051855`
TGP	all	`gfkit`	us	`https://doi.org/10.5281/zenodo.11241619`
Unified	all	`tskit`	Wohns et al. [64]	`https://doi.org/10.5281/zenodo.5495535`
Unified	all	`gfkit`	us	`https://doi.org/10.5281/zenodo.11241788`
Sim. 640k	20	`tskit`	us	`https://doi.org/10.5281/zenodo.11241938`
Sim. 640k	20	`gfkit`	us	`https://doi.org/10.5281/zenodo.11241938`

B Reusing Shared Bipartitions

Table 3 Average in-degree of non-root nodes in the `gfkit` DAG. The in-degree of a node in the DAG is equal to the number of times its intermediate result is reused during the post-order traversal.

Dataset	Chr.	Mean In-degree Bipartition	Mean In-degree Subtree
SGDP	all	4.44	4.12
TGP	all	5.89	5.48
Unified	all	5.20	5.49
Sim. 640k	20	—	5.10

[17] `https://github.com/popsim-consortium/stdpopsim`

C Speedups Using the Bipartition-DAG

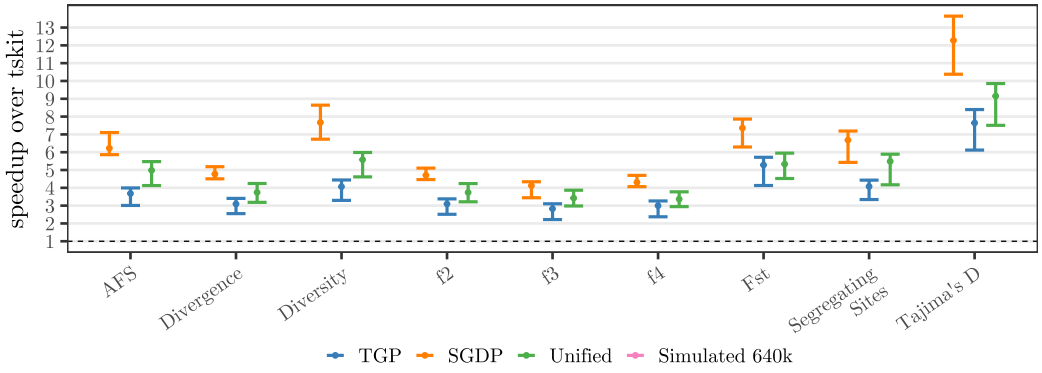

Figure 6 Speedup of `gfkit` (ours) over `tskit` (state-of-the-art) when computing various statistics on three empirical and one simulated dataset (human). The bars indicate the range of speedups and the dot the median speedup across all chromosomes of the respective collection. We use all 22 autosomal chromosomes for Thousand Genome Project (TGP), Simons Genome Diversity Project (SGDP), and the Unified collection. We use chromosome 20 of a simulated human dataset with 640 000 samples.