Genotype Representation Graphs: Enabling Efficient Analysis of Biobank-Scale Data

Drew DeHaas[1, +], Ziqing Pan[1, +], and Xinzhu Wei[1, *]

[1]*Department of Computational Biology, Cornell University, Ithaca, NY*
[+]Contributed equally
[*]Corresponding author: aprilwei@cornell.edu

Abstract

Computational analysis of a large number of genomes requires a data structure that can represent the dataset compactly while also enabling efficient operations on variants and samples. Current practice is to store large-scale genetic polymorphism data using tabular data structures and file formats, where rows and columns represent samples and genetic variants. However, encoding genetic data in such formats has become unsustainable. For example, the UK Biobank polymorphism data of 200,000 phased whole genomes has exceeded 350 terabytes (TB) in Variant Call Format (VCF), too large to fit into hard drives in uncompressed form. To mitigate the computational burden, we introduce the Genotype Representation Graph (GRG), an extremely compact data structure to losslessly present phased whole-genome polymorphisms. A GRG is a fully connected hierarchical graph that exploits variant-sharing across samples, leveraging on ideas inspired by Ancestral Recombination Graphs. Capturing variant-sharing in a graph format compresses biobank-scale data to the point where it can fit in a typical server's RAM (5-26GB per chromosome), and enables graph-traversal algorithms to trivially reuse computed values, both of which can significantly reduce computation time. We have developed a command-line tool and a library usable via both C++ and Python for constructing and processing GRG files which scales to a million whole genomes. It takes 160GB disk space to encode the information in 200,000 UK Biobank phased whole genomes as a GRG, more than 2000 times smaller than the size of VCF. Moreover, the size of GRG increases sublinearly with the number of samples stored, making it a sustainable solution to the increasing number of samples in large datasets. We show that summaries of genetic variants can be computed on GRG via graph traversal that runs 230 times faster than on VCF. We anticipate that GRG-based algorithms will improve the scalability of various types of computation and generally lower the cost of analyzing large genomic datasets.

Introduction

The rapid progress in sequencing technologies and the growing interest in genetic association with diseases have led to the collection of large datasets comprising hundreds of thousands of sequenced human whole genomes (Kaiser, 2021). At the end of 2023, the UK Biobank released ~500,000 whole genomes on their cloud computing platform, 200,000 of which have already been phased (Browning and Browning 2023; Hofmeister et al., 2023). To store a large number of phased whole genomes, the formats VCF (sometimes called pVCF, Danecek et al., 2011, **Fig. 1a**), and BGEN (Band and Marchini, 2018) are the most commonly used. These two formats vary in whether they store genotypes as hard calls (i.e., variant allele or not a variant) or probabilities of being a variant, and whether they use text or binary format. Despite the differences, genetic polymorphism data are generally organized either by individuals or by variants, to be easily converted into a 2-D matrix (**Fig. 1b**) or a 1-D vector when being loaded into computer memory for downstream computations. We hereafter refer to such formats as tabular formats.

The convenience and ease of use of tabular data structures have made them the predominant and default data structures for encoding genotypes. However, the underlying biological generative model of DNA sequences is based on graphs, not tables. For example, in the absence of incomplete lineage sorting (Kapli, Yang, and Telford, 2020), the genetic differences among species generated by spontaneous mutations can be seen as sampling random mutations based on the mutation rate on each branch of a true phylogenetic tree. Therefore, assuming DNA sequences can be accurately ascertained, a phylogenetic tree can be inferred from DNA sequences at the species level (Fitch and Margoliash, 1967; Kapli, Yang, and Telford, 2020), and these sequences can be efficiently stored with a tree data structure (**Fig. S1**). In the absence of recombination, a tree structure fully describes the evolutionary relationship among a set of genomes. In the presence of recombination, different sections of the genomes trace back to different common ancestors. As a result, the evolutionary relationship among a set of genomes can no longer be described by a single tree structure, but a graph. A graph of sequences of closely related species is called a phylogenetic network (Huson and Bryant, 2006), and a graph of sequences within species is called an Ancestral Recombination Graph (ARG) (Marjoram and Wall, 2006; Speidel et al., 2019; Kelleher et al., 2019; Brandt et al., 2022). Considering that individual haplotypes in a population are related through a true ARG (Kingman, 1982; Hudson, 1983), the genetic variants among haplotypes that arise from spontaneous mutations can be seen as a random sampling process on the ARG (Zhang et al., 2023). When an ARG is inferred from a set of *phased* whole genomes, the resulting graph can be used to store information about these genomes (**Fig. 1c**). However, mutations only occur very sparsely on an ARG, so using an ARG to encode genetic variants for downstream computation can be costly. Moreover, the state-of-the-art ARG inference methods can only scale to 1000 to 10,000 whole-genomes (Speidel et al., 2019; Kelleher et al., 2019; Brandt et al., 2022). As a result, ARG data structures are rarely adopted for biobank-scale whole-genome statistical analyses.

In this study, we introduce the Genotype Representation Graph (GRG) (**Fig. 1d**), a graph representation of whole genomes, and implement a scalable method that can infer a GRG for a million whole genomes at a very low cost. GRG is designed to faithfully encode the hard-call information in phased whole genomes and it uses a fraction of the space that would be required by existing alternatives. Moreover, GRG can be traversed efficiently which enables algorithms in the style of dynamic programming (Eddy, 2004), facilitating scalable genome-wide statistical analyses.

Results

Overview of GRG

We use the term *mutation* to refer to a (genomic position, alternate allele) pair, such that each genomic position could have more than one mutation. In general, a single haplotype can be represented as the full DNA sequence or as a sparse list of mutations. Such a list contains only the variations from the reference sequence: a haplotype that exactly matches the reference at all polymorphic sites is represented by an empty list. Similarly, a GRG encodes mutation lists of phased haplotypes in the graph: only deviations from the reference sequence are explicitly represented.

A GRG is a directed acyclic graph (DAG) (Greenland, Pearl, and Robins, 1999) with the following properties:

I. The nodes with no successors (leaf nodes) are *sample nodes*. There is one sample node for each haploid genome in the dataset, and a diploid individual will be represented by two sample nodes.

II. Any node (including sample nodes) can contain zero or more mutations. A node with at least one mutation is called a *mutation node*. A node without any mutation is called an *empty node*.

III. A node that has at least one successor and one predecessor is called an *internal node*. An internal node represents the union of samples beneath it, and the union of mutations above it. We say such a node *covers* these samples and mutations.

IV. The roots (nodes with no predecessors) must be mutation nodes.

V. There exists at most one path from any given root node to any given sample node (i.e., there are no "diamond patterns" in a GRG).

VI. There exists a path from the mutation node containing mutation M_i to the sample node S_j iff the genotype for sample S_j contains mutation M_i.

A GRG is defined by its *down edges*: the edges that are directed from the roots to the samples. We also refer to this as the parent-child direction. It also symmetrically contains *up edges*, which are child-parent edges and form the reverse DAG.

The worst-case GRG is equivalent to representing the dataset as a list of mutations: each mutation node has a direct edge to every sample that contains it. In contrast, a compact GRG utilizes internal nodes to cover sample subsets that are shared by many mutations, thus compressing the mutation-to-sample relationship (see **Fig. S2**).

A Scalable Algorithm for Inferring GRG

The goal of GRG inference is to create a graph representation of the relationship among samples such that 1) each mutation is correctly represented by a mutation node and 2) the number of edges in the graph is minimized. To this end, GRG inference proceeds in four high-level steps (**Fig. 2a**). The first step can be conceptualized as separating the genotype matrix into T non-overlapping continuous segments. For each of these segments, a local GRG is inferred through two additional steps: (2) building a tree GRG (tGRG) that approximates the genetic relationship among all samples based on the genotypes in the current segment (the *BuildShape* step), and (3) mapping mutations from the current segment to the tGRG (the *MapMutation* step). Finally, the T tGRGs are merged into a single global GRG that represents the

genome-wide polymorphisms. This final GRG is then simplified and serialized to disk (see **Supplement**). Experimentally, it is optimal to split the genome into segments of size 50-150Kbp for most non-simulated datasets (**Fig. S3**).

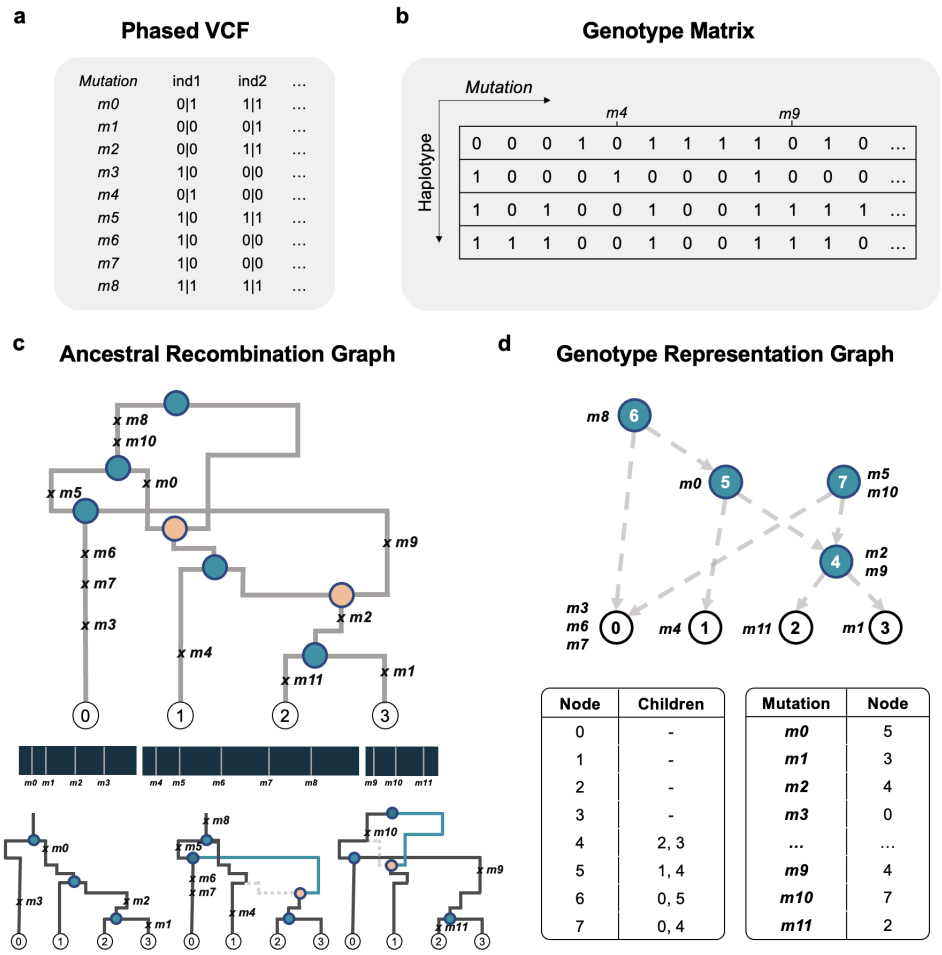

Figure 1: Data encodings for phased polymorphisms. Phased diploid genotypes in four different formats. **a**: A phased VCF file. Rows are genetic polymorphisms, columns are individual genotypes. The reference allele is labeled as 0 and alternative alleles are non-zero. A vertical bar separates an individual's genotype into two haplotypes. **b**: Encoding the data as a matrix. This is essentially a rotation of the VCF data, with columns representing polymorphisms and rows representing haplotypes. **c**: An Ancestral recombination graph (ARG) that represents both the genealogies among four haplotypes and the genetic polymorphisms. The full ARG is shown in the upper panel, where leaf nodes represent the present-day haplotypes, the dark teal nodes represent the ancestral haplotypes resulting from coalescence, and the orange nodes represent the ancestral recombination events. Lower panel **c** shows an alternate ARG representation: a collection of marginal coalescent trees along the genome (Rasmussen et al., 2014). Recombination breakpoints along the genomic segment (the gaps in the rectangle bar) define where marginal trees begin. The edges (i.e., branches) from the first marginal tree are colored black. In subsequent trees, the dashed lines represent the left-side ancestral lineage in the recombination, and the dark teal edges represent the right-side ancestral lineage in the recombination. Mutations are mapped to edges based on the mutational time and genetic coordinate. **d**: A GRG representation of the same data. Each node represents a unique subset of samples, and each mutation is mapped to one node. The node IDs are labeled inside nodes, and the mutation IDs are marked next to the mutation nodes. Samples are reachable from a mutation node iff the mutation is present in the sample haplotype. This GRG can be described as the tables in the lower panel **d**.

The *BuildShape* and *MapMutations* steps are tightly linked. When a mutation's sample list can be fully represented by an existing node in the graph, the mutation is directly mapped to the node (e.g., mutation

m0 in **Fig. 2a**). Otherwise, a new node is created and connected to existing nodes in the graph such that the union of samples of the new node can exactly represent the sample list of the mutation (e.g., mutation *m5* in **Fig. 2a**). In the worst case, a new node is added for the mutation, and edges are added from the new node to all samples that contain the mutation. *BuildShape* is designed to avoid this worst-case scenario by generating nodes that are highly reusable across many mutations.

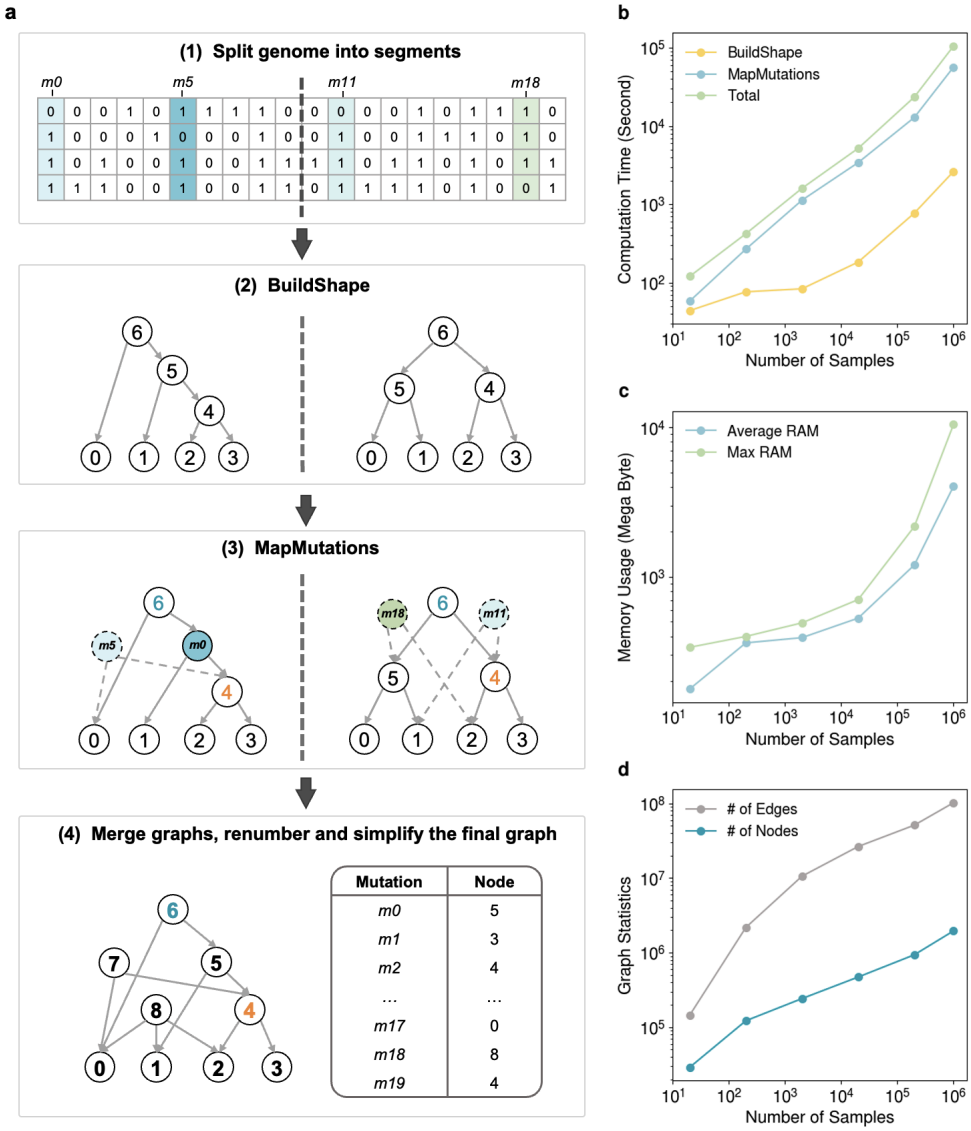

Figure 2: Overview and performance of the GRG inference algorithm. a: GRG inference includes four high-level steps: (1) segmenting the genome, (2) building an initial tGRG for each segment, (3) mapping mutations from each segment to the corresponding tGRG, and (4) merging all tGRGs into a single GRG. Dashed lines represent edges/nodes newly created in the current step. **b**, **c**, and **d** summarize the performance of GRG inference against various sample sizes using simulated data of a 100Mbp region. **b**: The total inference time (in green) and the time breaking down into BuildShape (in yellow) and MapMutations (in blue) against the number of samples. **c**: Memory usage in terms of average RAM (in blue) and maximum RAM (in green) against the number of samples. **d**: Graph size of the resulting GRGs breaking down into the number of edges (in grey) and number of nodes (in teal) against the number of samples.

In each segment separated in step (1), *BuildShape* creates a tGRG by repeatedly modifying and merging pairs of subtrees from a set until a single tree is formed. The initial set is simply the sample nodes, serving

as the roots of the subtrees. An arbitrary root node from the subtree set is selected, and a nearest-neighbor search is performed to find another root node of the highest similarity measured by the Hamming distance (Waggener, 1995) of the mutation lists associated with the nodes. The Hamming distance is against the mutation lists (haplotypes) for the associated samples, when the sample nodes are root nodes. As new root nodes are added, Hamming distance is against the consensus haplotype (Miller, Koren, and Sutton, 2010) represented by the root, i.e., the intersection of their descendants' mutation lists. After the nearest-neighbor search, the two subtrees are merged by constructing a new root node and attaching the previous root nodes to it. These steps are repeated until a single tree is formed (only one root node remains in the set), which is the tGRG. The mutations are then mapped, either directly to the tGRG or to a GRG composed of multiple tGRGs that have been merged.

We first tested the performance characteristics of GRG inference using simulated data. We used *msprime* (Baumdicker et al., 2022) to simulate 100Mbp sequences for sample sizes of 10, 100, 1000, 10000, 100000, and 500000 diploid individuals with a human-like parameter setting (Ne = 10^5, recombination rate = 10^{-8} / (bp × generation), mutation rate = 10^{-8} / (bp × generation)). While *msprime* emits compact tree sequence output, we converted the output to a tabular format (VCF files) (Danecek et al., 2011) to test GRG inference. The inference time, memory usage, and node/edge counts of the resulting GRGs are shown in **Fig. 2b-d**. Time to infer a GRG from tabular format scales roughly linearly as sample size increases. Importantly, the size of the graph (**Fig. 2d**) scales sub-linearly with respect to sample size. The benchmarking was performed on a server with two AMD EPYC 7532 32-core CPUs (a total of 64 cores, no hyperthreading) and 1024GB of RAM.

Applications to UK Biobank Whole-Genome Sequencing Data

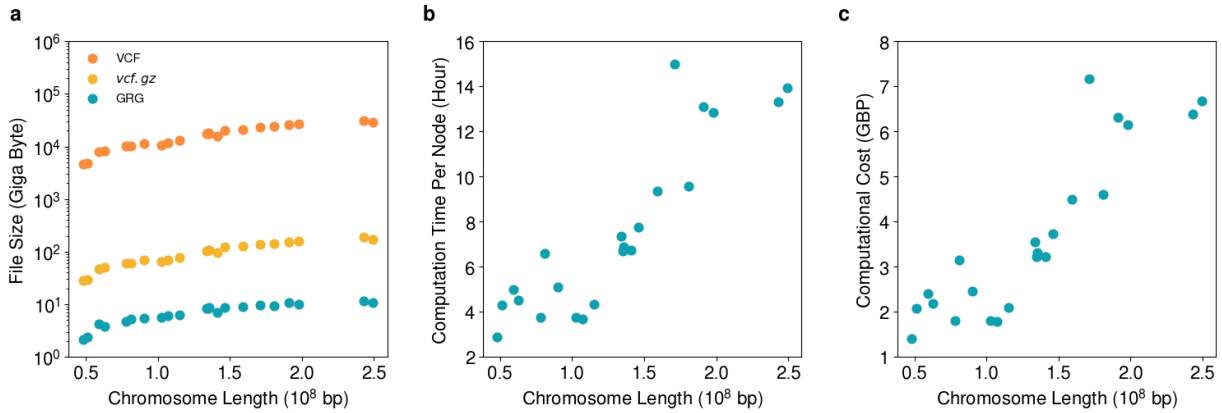

Figure 3: Inferencing GRG from the UK Biobank whole-genome sequencing data. a: Comparisons of GRG against existing data encodings for storing the genetic polymorphisms in the UK Biobank. Each chromosome is represented by one dot. **b**: The inference time and **c:** computation cost of GRG with UK Biobank whole-genome sequencing data. A cloud node with 70 CPU threads is used for the computation.

We applied GRG inference to all chromosomes of the UK Biobank dataset containing 200,011 phased individual genomes (Hofmeister et al., 2023). First, we converted the *vcf.gz* files to a fast/compact intermediate representation (see **Supplement**) and then inferred GRGs. Results are shown in **Fig. 3**. All GRG inference was performed on a cloud node with 72 CPU cores and 2GB RAM per core. We used 70 of these cores during parallel GRG inference. The slowest chromosome took 14 hours to convert to GRG,

resulting in a file 16 times smaller than the *vcf.gz* and more than 200 times faster for allele frequency computation. We estimate the total uncompressed VCF size to be 350 terabytes; large enough that decompressing it to disk is impractical. We do not filter any data; hence, the GRG represents all of the genotype information that is in the phased VCF files. The GRG files for chromosomes 1-22 store the data for more than 500 million variant sites of 200,011 phased genomes in less than 160GB. Sample identifiers are stored in the GRG file, but all other metadata for the samples and SNPs must be stored separately. A more detailed comparison of different file formats is provided in **Table S1** and **Fig. S4**.

GRG as a Computational Data Structure

Dynamic programming is an algorithmic technique that exploits the storage and re-use ("memoization") of computed values from overlapping sub-problems, and has been applied to many problems in computer science and computational biology (e.g., sequence alignment) (Giegerich, 2000; Eddy, 2004; Cormen et al., 2022). The GRG graph structure resembles dynamic programming in that internal nodes represent "overlapping sub-problems": in a compact GRG, these nodes capture the mutations-to-samples and samples-to-mutations relationship that are partially shared by many different samples and mutations. The node-sharing structure that compresses a dataset in GRG also facilitates the reuse of computed values. Computations involving multiple samples and mutations can store computed values at internal nodes, having computed those values from the graph's parent/child edges, resulting in the same recursive overlapping sub-problems approach as dynamic programming. Dynamic programming and GRG both leverage a trade-off between storage and computation: memoization uses more storage (typically RAM) but can significantly reduce the number of computations needed.

Genome-wide summary statistics can be computed from GRG using graph traversals. The most natural traversal of a GRG is encoded in its node ID ordering: the bottom-up topological ordering of nodes in a GRG ensures that each node n is visited after all of its children, such that those children have node IDs less than n. The bottom-up ordering can also transform a recursive dynamic programming approach into an iterative approach, thus potentially accelerating computation and conserving memory in practical application. Here, we use the allele frequency computation as an example to demonstrate this approach. For tabular data, this could be done element by element (**Fig. 4a**). In a GRG, the sample frequency at any node n can be computed as the sum of the sample frequencies for each child node (**Fig. 4b**). Since the children of node n have IDs less than n, we can simply compute sample frequencies in the order of nodes $0...n$ and store each frequency. The frequency values of all mutation nodes can then be extracted to obtain the allele frequencies for the dataset. This process guarantees that mutations containing the same subset of samples (S_1, ..., S_k) captured by an internal node will all reuse this internal node's frequency.

To benchmark the performance of GRG-based computation, we compute the allele frequencies of SNPs in each of the 22 autosomes of 200,000 phased whole-genome sequences in the UK Biobank. We use either bed files or VCF files as input in plink v.1.90b6.26 (Chang et al., 2015) and compare the total computation time and cost against GRG (**Fig. 4cd**). Notably, GRG achieves the fastest computation efficiency, computing allele frequencies on average more than 7-fold and 200-fold faster than plink with bed files and *vcf.gz* files, respectively. For longer chromosomes and larger sample sizes, the discrepancy of efficiency between GRG and plink is expected to be more pronounced, as suggested by our simulated results (**Fig. S5**). To avoid extra cost, we predict the runtime of bed files and VCF files for longer chromosomes using the results from shorter chromosomes. The total projected cost of allele frequency

computation with bed files and VCF files is 141 British pounds. In contrast, it takes only 0.8 pounds to compute genome-wide allele frequencies using GRG in the UK Biobank Research Analysis Platform (**Fig. 4d**). This analysis showcases the potential of GRG in enabling efficient computation for whole-genome summary statistics in large-scale datasets.

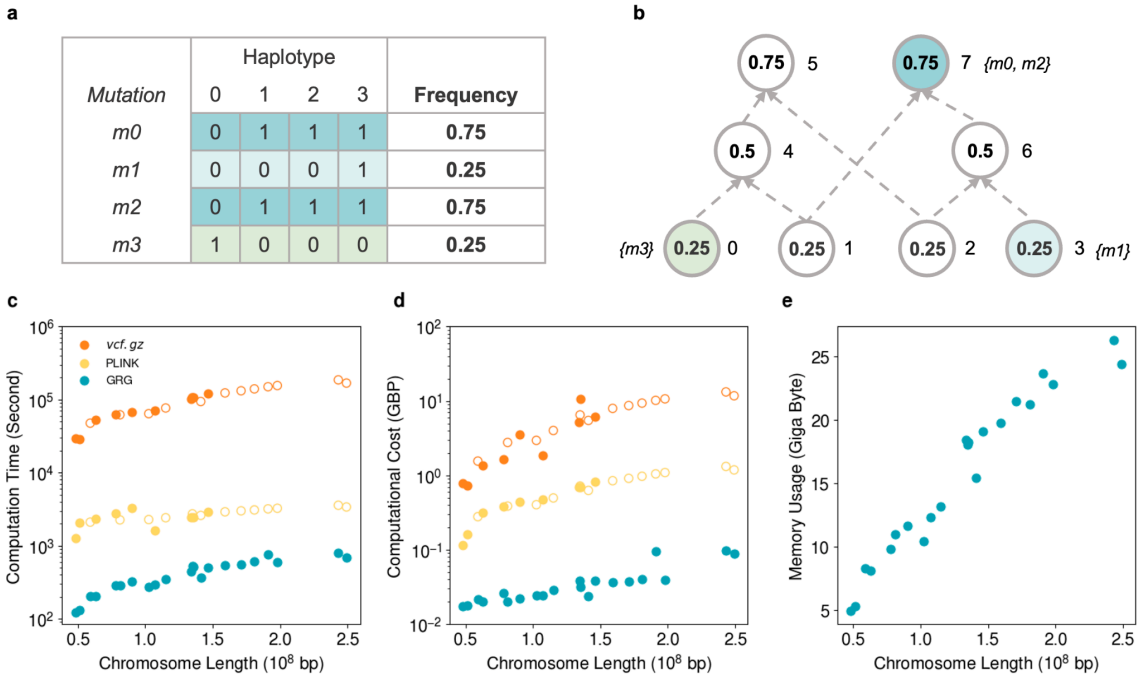

Figure 4: GRG accelerates genome-wide summary statistic computation. a: Computation of allele frequencies using standard tabular methods. Genotypes of each variant are read row by row from disk and allele frequencies are computed independently for each variant in the tabular structure. **b**: Computation of allele frequencies using GRG. In a GRG, the sample frequencies of nodes are firstly computed and then mapped to the mutations based on the node-mutation mapping. The frequency of leaf nodes is set as 1/n (e.g., node0-3), where n is the number of haplotypes in the population. For other nodes (e.g., internal nodes and root nodes), frequencies are computed sequentially based on the node IDs (e.g., from node4 to node7) as the sum of frequencies of each node's children nodes (e.g., node4 = node0 + node1). **c**: Comparisons of GRG-based computation efficiency with standard tubular methods for genome-wide allele frequencies in the UK Biobank dataset. For standard tubular methods, allele frequencies were computed on 9 chromosomes (chromosomes 8, 10, 12, 14, 16, 18, 20, 21, 22, labeled as solid dots). These results were used to predict the expected computation time for the remaining chromosomes (presented as open dots) with a linear projection. **d**: Comparisons of computational costs in Pound Sterling (GBP) for calculating genome-wide allele frequencies in the UK Biobank Research Analysis Platform. Similar to **c**, solid dots represent the results from our experiments while open dots represent the projected computational cost for the remaining chromosomes. **e**: Maximum memory usage for computing genome-wide allele frequencies with GRGs.

GRG as a Reduction from ARG

The term ARG has been used to describe several different things in the literature (Wong et al., 2023). Related to our study, *true* ARGs are equivalent to the underlying real genealogical history of a sample set. In simulations (Baumdicker et al., 2022), the paths of inheritance and coalescence times are often simulated first, and then neutral mutations are overlaid on top of these paths to generate haplotypes. Here we also refer to the actual paths of inheritance and coalescence as true ARG, and the generated haplotype data as *true* data. Mutation events, by definition, can always be overlaid to a true ARG to encode the true data, but not all parts of a true ARG bear mutation events. The GRG could be regarded as a more compact ARG-like graph that only captures the paths used for the actual "inheritance" of mutations to a set of

haplotypes. The *optimal* GRG could be defined as a reduction from the true ARG in four aspects (**Fig. S6**). First, in order to record the coalescence time, the same topological structure differing only by coalescence times must be duplicated in an ARG. These duplicated structures are omitted in a GRG because GRG does not encode coalescence times. Second, paths in a true ARG that do not serve the purpose of passing down any mutations can be removed. Third, when two or more mutations are inherited by the same set of haplotypes through different paths, these paths could be merged. Fourth, edges that are never used as the first edge in any mutation path could be removed. Based on the mutation and recombination rate of human genomes, more than half of the structures in an ARG can be removed. After doing this reduction, we get a new graph that compactly encodes the true data. If we encode this new graph using the GRG data structure, we get an *optimal* GRG.

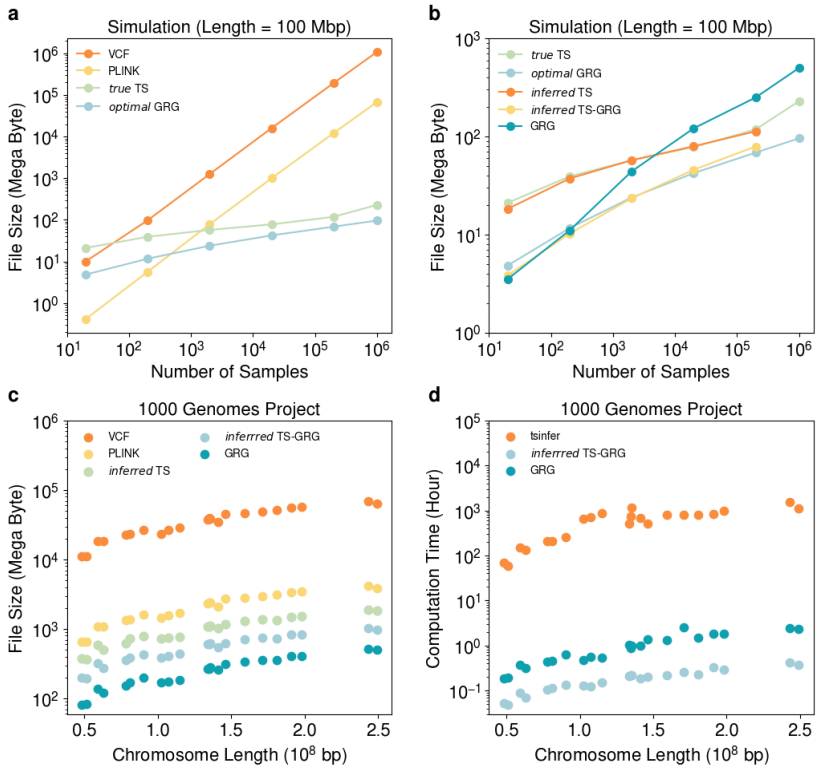

Figure 5: Investigating the optimality of GRG inference. A simulated genetic sequence of 100Mbp is used in the benchmark for panels **a** and **b**. **a:** Comparisons of the file sizes among VCFs (VCF), bed files (PLINK), true tree sequences from *msprime* (*true* TS), and GRGs inferred from true tree sequences (*optimal* GRG). **b**: Comparisons of file sizes of *true* TS, *optimal* GRG, inferred tree sequences from *tsinfer* (*inferred* TS), GRGs inferred from inferred tree sequences (*inferred* TS-GRG) and GRG inferred from true genotypes of varied sample sizes (GRG). The phased whole-genome sequences from the Phase III 1000 Genomes Project are used for benchmarking for panels **c** and **d**. **c**: Comparisons of the file sizes using data from the 1000 Genomes Project. Each chromosome is represented as a data point. **d**: Runtime of GRG and *tsinfer* inference. The inference time of *tsinfer* is in orange. The inference time of GRG is in teal. The light blue dots show the conversion time from the inferred tree sequences to inferred TS-GRGs.

It is important not to confuse GRG being a reduction from ARG with GRG being a reduction from tree sequence (TS), which is a particular ARG *representation* (Kelleher et al., 2019). In practice, GRG can be constructed from tree sequences (if available). The tree sequence representation of an ARG reuses nodes such that the subtree topology below a mutation node is only valid when the genomic position of the mutation and the genomic range of the tree overlap. In contrast, when constructing GRG from tree

sequences, nodes from a tree sequence might be duplicated such that the topology below a mutation is unique and globally valid.

The Robustness of GRG Inference

To test the optimal size of GRG, we performed *msprime* simulations (Baumdicker et al., 2022) to generate *true* ARGs and *true* data (see **Methods**). ARGs can be compactly encoded with the tree sequence format (Kelleher et al., 2018). We construct GRGs in two different ways. First, we construct GRGs from true ARGs to investigate the size of the *optimal* GRG. To this end, we find that the optimal GRG is consistently the most compact data structure across sample sizes. Moreover, the size of optimal GRG grows sublinearly with sample size (**Fig. 5ab**).

Next, we investigate whether we can obtain a more compact GRG by first inferring an ARG. In this experiment, true data are generated by *msprime* simulation (see **Methods**). While ARG-Needle (Zhang et al., 2023) is one of the more scalable ARG inference techniques, the mutation events are not inferred, thus we cannot use it directly for GRG construction. To date, *tsinfer* (Kelleher et al., 2019) is the only ARG inference method that is scalable to at least 10,000 whole genomes that still maps mutation events. Applying *tsinfer* to true data, we find that the optimal GRG is smaller than the TS-GRG, which is smaller than the *inferred* GRG (**Fig. 5b**).

To investigate whether the TS-GRG is smaller than the inferred GRG in *real* data, we applied *tsinfer* to 1000 Genomes Project data (The 1000 Genomes Project Consortium et al., 2015). Interestingly, we find that the TS-GRG is about two times larger than the inferred GRG (**Fig. 5c**). When genetic data are not accurately ascertained, the true ARG is no longer able to encode the observed data. Therefore, we hypothesize that real data, such as those collected by the UK Biobank and the 1000 Genomes Project, are subject to sequencing and variant calling errors, and ARG inference methods might not perform robustly in the presence of such errors, whereas the similarity search algorithm used in *BuildShape* tolerate such errors better. To this end, we performed a simple test to compare the sizes of the TS, TS-GRG, and GRG using simulated data that has 0.1% added noise, which roughly mimics the variant calling error in the UK Biobank (Saunders, Brohede, and Hannan, 2007; Browning and Yu, 2009; Halldorsson et al., 2022). Although this noise simulation procedure is oversimplified, we find that the sizes of the inferred TS, inferred TS-GRG, and GRG all increase significantly as the noise level increases, suggesting that noise can indeed have a sizable influence on graph sizes. Interestingly, the size of GRG becomes consistently smaller than inferred TS and inferred TS-GRG across a range of sample sizes even when there is only 0.1% of added noise (**Fig. 6**), suggesting that our GRG inference algorithm is more robust to errors in the data ascertainment. This robustness to noise is likely because our GRG inference decouples the BuildShape and MapMutations steps, hence not making any explicit assumptions about mutation models.

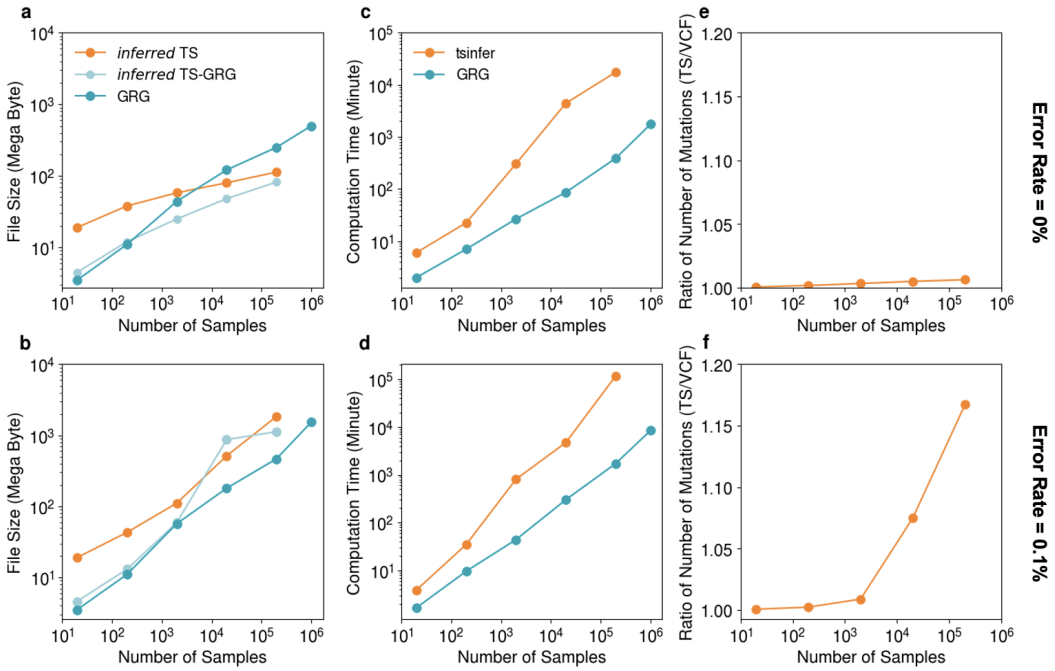

Figure 6: Robustness of GRG inferences with noisy simulated data. The file sizes (**a-b**) and inference efficiency (**c-d**) of inferred tree sequences and GRGs in simulated data: (**a, c**) without error and (**b, d**) with 0.1% error. The error is introduced by manually flipping individuals' genotypes according to the specified error rate to mimic the sequencing or genotyping error in empirical data. We compare the file sizes and inference times of the inferred tree sequences from *tsinfer* (*inferred* TS), the GRG inferred from the inferred TS (*inferred* TS-GRG) and the GRG inferred directly from the genotypes (GRG). **e**, **f:** the ratio of the number of mutations in inferred tree sequences to the number of input mutations in VCF with different error rates in the data.

Discussion

We presented a new data structure and file format termed genotype representation graph and demonstrated it to be smaller than all other existing formats on non-simulated, phased genetic datasets. Inferring GRG can be done for Biobank-scale datasets in a matter of days. GRG inference is much faster than ARG inference, and it is more robust to the ascertainment errors in the sequencing data generating process. We showed that calculations can be performed much more efficiently via GRG than other file formats. This speed for calculation also makes GRG a useful representation for simulated population genetic data, especially since converting from tree sequence format to GRG is extremely fast - on the order of a few minutes for very large datasets.

Compared to other file formats (Purcell et al., 2007; Durbin, 2014) such as bed files which prohibit multi-allelic polymorphisms, GRG provides a unified way of presenting both bi-allelic and multi-allelic polymorphisms as mutation nodes. Similarly, GRG can encode insertions and deletions with mutation nodes. Missing data can be handled by GRG as separate mutation nodes that are marked specially. Whereas a typical mutation is a pair (genomic position, alternate allele), a missing data mutation is a pair (genomic position, *missing data flag*). When there is a lot of random missingness in the dataset, the graph size can increase non-trivially, since it needs to encode these additional mutation-to-sample relationships. In the UK Biobank ShapeIt phased whole genomes (Ribeiro et al., 2023), there is no missingness, as missing genotype imputation is usually coupled into the phasing pipeline (Howie, Donnelly, and Marchini, 2009; Hofmeister et al., 2023).

Generally speaking, the purpose of the GRG data structure is to compactly and faithfully encode haplotypes as mutations being passed down via paths in the graph, to allow easy access to both mutation information and haplotype information. In contrast, ARG data structures are designed to fully record the true or inferred inheritance history of the genomes, including coalescence times (Hudson, 1990; Griffiths and Marjoram, 1996; Griffiths and Marjoram, 1997; Rasmussen et al., 2014). Although GRG encodes "less" information compared to a *true* ARG, it supports random queries of variant-to-sample relationships and sample-to-variant relationships – querying a variant-to-sample relationship simply requires traversing downward from the node bearing the variant to collect sample nodes, and querying a sample haplotype simply requires traversing upward from the sample node to collect mutations (**Fig. S7**). In contrast, none of the existing ARG data structures support such random data queries. Tabular data formats allow easy access to either variant-to-sample relationships or sample-to-variant relationships but not both. This unique feature of GRG data structure makes it well-suited for different types of computational studies. Different from conventional vectorized algebraic operations, computations against the GRG benefit from graph traversals.

Interestingly, the GRG data structure satisfies the definition of a multitree or strongly unambiguous graph, where there is at most one direct path between any two nodes (Furnas and Zacks, 1994). Coincidentally to what we have observed with the UK Biobank data (**Fig. 3**), in astronomy, multitree methods typically yield orders of magnitude in speedup over the alternatives, making millions of data points tractable on desktop workstations (Gray et al., 2004). We, therefore, anticipate that the GRG data structure can also broadly improve the scalability of data analyses involving large genetic datasets. To facilitate the adoption of GRG in other studies, our library's API provides visitor and iteration patterns for depth-first, breadth-first, and topological order traversals. The API is available in Python for ease-of-use, and C++ for users needing higher performance.

For representing the true ARG, the tree sequence format is the most compact format, and it can be more compact than inferred GRG (i.e., with simulated error-free data, inferred tree sequence can be slightly smaller than inferred GRG). However, the tree sequence format does not provide the same memoization benefits as GRG because nodes in a tree sequence are reused for different sample subsets. Furthermore, tree sequence inference by *tsinfer* appears to be less robust than GRG inference to noise in the data (**Fig. 6**), which can obscure the underlying genetic ancestry. In real datasets, sequencing and genotyping errors are orders of magnitude larger than mutation rates, with many error rate estimates near 0.1% (Saunders, Brohede, and Hannan, 2007; Browning and Yu, 2009; Halldorsson et al., 2022) (see **Methods**). We suspect that there is room for further reducing GRG sizes by generally improving robustness to noise, or by identifying and "ignoring" instances of mutations that were caused by errors. In general, sequencing errors appear to pose a challenge for any technique that is trying to identify the true inheritance relationships between haplotypes, whether for the purpose of compression (GRG) or genealogy inference (ARG).

Although we primarily use the UK Biobank data as an example, the utility of GRG is not limited to human data. Indeed, data from other species are also rapidly growing. We converted two sets of inferred tree sequences for the SARS-CoV-2 genomes (Zhan et al., 2023) into GRGs, one for 1.27 million SARS-CoV-2 genomes sampled up to June 30, 2021, and the other for sparsely sampled 657 thousand

samples sampled up to June 30, 2022, both originally from GISAID (Shu and McCauley, 2017) (**Table S2**). For the tree sequence with 1.27 million SARS-CoV-2 genomes, the resulting size of the GRG is 37MB, compared to 891MB in the tree sequence format. For the tree sequence of 657 thousand samples (480MB), the resulting GRG is 31MB. It takes only 5.4 and 3.6 seconds to convert these tree sequences to GRG. As ARGs are being inferred on more datasets, GRG can be readily available for computational analyses "for free". In the future, GRG could also serve as a better input file for ARG inference and refinement.

When storing large datasets, tabular file formats use standard compression algorithms such as zlib (Gailly and Adler, 2004) (VCF, BGEN) and zstandard (Collet and Kucherawy, 2021) (BGEN, Plink BIM/FAM). This type of compression requires decompressing during data processing, which imposes a computational burden, but also means that the size of the underlying decompressed data still dictates the number of operations needed to process it. For compressed text-based formats like VCF, the uncompressed text - terabytes of it, in the case of biobank-scale data - still needs to be parsed before it can be used for computations. Binary formats like BGEN (Band and Marchini, 2018) can be faster than text formats (**Fig. S4**), both compressed and uncompressed, but still need to traverse $O(NM)$ items (where N is the number of samples and M is the number of variants). In contrast, GRG does not require decompression as the compression is inherent to the graph structure, and strictly fewer than $O(NM)$ items (edges) need to be traversed. GRG is more compact than traditionally compressed formats (*vcf.gz*), and using traditional compression to further compress a GRG does not provide much reduction in file size: GRGs for UK Biobank only further compressed by approximately 40% with GZIP (Deutsch, 1996) using the default compression level.

Generative models in machine learning have demonstrated exceptional performance in tasks like image synthesis, text generation, and data augmentation (Gm et al., 2020). Extending their utility, these models can be applied to lossy data compression by generating compressed data representations (Kingma, Abbeel, and Ho, 2019; Yang, Mandt, and Theis, 2022). However, such compressed representations cannot support the same computation as with uncompressed data. In population genetics, researchers have long employed models motivated by the mechanisms in the biological world to closely resemble the true genetic data-generating process to gain insights into genetic polymorphism. The inference of GRG can be seen as an approximation to this data-generating process to losslessly compress the observed genetic polymorphism. In addition to achieving a highly compressed data representation, GRG maintains or even improves our ability to support computations on the original data. Our data structure and algorithm, inspired by population genetics principles, addresses computationally challenging problems beyond the traditional scope of population genetics.

Methods

BuildShape

Given a dataset, let $S = \{0 \ldots N\text{-}1\}$ be the set of all samples and $M = \{M_0 \ldots M_V\}$ be the set of all mutations. Let S_i refer to the subset of samples from S that contain the i-th mutation, M_i. We'll use $\mathcal{G}$ to refer to a corresponding GRG. The GRG inference problem is to search from the space of all possible internal

nodes to find an optimal set of internal nodes such that 1) all mutations in M are correctly mapped to a node and 2) the total number of edges in $\mathcal{G}$ is minimized.

BuildShape constructs an initial graph $\mathcal{G}$ by iterating over a set of nodes S'. Initially $S' = S$, i.e. the set of all sample nodes. For each sample i, we find the sample j with the most similar haplotype. We create the union of these samples by constructing a new node N_k and making it the parent of nodes i and j. Nodes i and j are removed from S', and node N_k is added (i.e., $S' = S' \setminus \{i, j\} \cup \{k\}$).

Building the initial GRG shape requires $O(N^2)$ haplotype comparisons, where N is the number of samples. We speed up the Hamming distance calculation and reduce memory footprint by reducing the dimensionality of each haplotype. Instead of exactly representing mutation lists, we use bloom filters (Bloom, 1970) with a single hash function ($k = 1$) and bit-size $m = A \times B$ where A is the average number of mutations per sample and B is a parameter specifying how many bits to use for each mutation (see **Supplement**). We also use a BK-tree (Burkhard and Keller, 1973) nearest-neighbor index to speed up the similarity search.

The T GRGs that cover the genome are merged into $P <= T$ sub-GRGs prior to *MapMutations*. For simplicity, **Fig. 2a** shows the case where $P = T$. P determines how parallelizable GRG construction is: each merged sub-GRG can have mutations in the corresponding genomic region mapped onto it in parallel. In practice, P is expected to be larger than the number of cores available for GRG construction. An increase in P only marginally increases the size of the final GRG, and larger P can reduce overall runtime even when inference is run on a single core (see **Fig. S8**).

Determining Parameters T and P

T is the number of GRG-trees to construct for the genome, an important hyperparameter that determines the inference efficiency. Varying this parameter can significantly affect graph size (**Fig. S8**). If T is too large (segment size is small), there may not be enough variation in the distribution of mutations in that region to accurately ascertain the genealogy among samples. If T is too small (segment size is large), the important sample relationships cannot be well-described by a single tree, due to recombination within the region. We split the genome evenly, according to base pairs, between the T trees. We've found that for real data the optimal segment length is in the range 50 - 150Kbp, which means that the optimal T varies depending on chromosome length. The lower range (50Kbp) works best for data that is dense in mutations-per-sample (such as the 1000 Genomes Project). The upper range (150Kbp) works better for sample-rich data (such as the UK Biobank). A method for finding the best parameter for a new dataset is to simply infer multiple GRGs for a small 500Kbp segment of the genome: one for each segment length in the set {50, 75, 100, 125, 150}. Once a reasonable segment length for a particular chromosome in a dataset is found, it can usually be reused across all chromosomes.

In our experiments, simulated data performs better with much smaller T values than non-simulated data. Adding noise to the simulated data did not have a significant impact on the optimal T value, so it is not clear what causes this difference between real and simulated data.

In contrast to T, the P parameter has very little effect on the final graph size, but has a significant effect on runtime (**Fig. S8**). P must be less than or equal to T and should be greater than or equal to the number of cores available for GRG inference.

MapMutations

MapMutations modifies each sub-GRG such that each M_i in M is assigned to a node that covers exactly S_i. In the worst case, a new node N_k is added for every M_i, and edges are added from N_k to every sample node in S_i. For each mutation M_i, MapMutations does an upward graph search starting at S_i and collects all nodes that have *only* samples in S_i beneath them. In the best case, it finds an existing node that exactly covers S_i and just assigns M_i to that node. More typically, a new node is created for the mutation and multiple nodes N_j are discovered such that the union of samples covered by N_j is a subset of S_i and the intersection of samples covered by N_j is empty. The empty intersection ensures we maintain GRG property **V** (no diamond patterns).

Merging GRGs

When two GRGs are merged the result is simply the union of the nodes and edges, except that internal nodes containing the *exact same* covered sample set are combined into a single node. We perform this merge by mapping the second GRG onto the first. The first GRG is traversed and a one-way hash is computed of the samples covered by each node. Then the second GRG is traversed in bottom-up topological order and the same hash is computed for each node. Each node and its edges are copied to the first GRG *unless* the node has a matching one-way hash in the first GRG. Regardless of whether the hashes match, the mutations are always copied from the second GRG to the first.

Node renumbering, graph simplification, Compressed Sparse Row (CSR) encoding (Duff, 1984; Buluç et al. 2009) (**Fig. S9**), and serialization to disk are then performed simultaneously. A depth-first search, using arbitrary edge ordering, is performed over the entire graph to produce a topological ordering such that the sample nodes take the node orders $0 \ldots n\text{-}1$ and all other nodes are ordered behind them. The nodes are iterated in this order, and a *node position vector* and an *edge vector* are constructed. The node position vector value at index $i+1$ holds the index e_i in the edge vector that contains the first edge for node i. If node i has c down edges, then positions $e_i \ldots (e_i + c)$ of the edge vector contain those edges, where each edge is represented by the node ID j for the edge (i, j).

GRG is both an in-memory data structure and a binary file format. The node ordering and CSR encoding mean that data for all nodes and edges are stored sequentially (**Fig. S9**), such that traversing the graph from the samples to the mutations can be done easily (simply iterate over the node identifiers) and cache-efficiently. Using this custom format we can store huge datasets much more compactly and efficiently than using a standard graph file format like GraphML (Brandes et al., 2002).

Simulating Ground Truth Tree Sequences

To investigate the performance of different data encodings, we simulated population genetic polymorphisms under different conditions. Msprime v.1.2.0 (Baumdicker et al., 2022) is used to run a series of backward-time coalescent simulations to generate the true ARGs with varied sample sizes ranging from 10 to 500,000. For simulated data presented in the main text, we set a constant effective

population size (Ne = 10^4), a flat recombination rate of 10^{-8} / (bp × generation), and a mutation rate of 10^{-8} / (bp × generation). The random seed is 42 for all simulations. Tree sequences are the output (Kelleher et al., 2018). More complicated models including human demography, gene conversion, and variable recombination rate are also tested (see **Supplement**).

We convert the tree sequences with population genetic polymorphisms to phased VCF files with tskit v.0.5.5 (Kelleher et al., 2018), and convert the phased VCF files into bed files with plink v.1.90b6.26 (Chang et al., 2015).

Inferring GRG from non-Simulated Data

We used two datasets for GRG inference: the 1000 Genomes Project and the UK Biobank data. For the 1000 Genomes Project (The 1000 Genomes Project Consortium, 2015), we used parameters T = 1600 and P = 50. All of our experiments used 20 CPU threads during GRG inference. For the UK Biobank data (Hofmeister et al., 2023), we used various T values from 280 (chromosome 22) to 1260 (chromosome 1), and P from 70 (chromosomes 22-13) to 140 (chromosomes 12-1). We used 70 CPU threads during GRG inference for chromosomes 8-22, and 96 CPU threads for chromosomes 1-7. All UK Biobank inference was performed in the cloud using the Research Analysis Platform, and the SHAPEIT phased VCFs from the "GATK and GraphTyper WGS" release containing genotypes for 200,011 individuals (Ribeiro et al., 2023; McKenna et al., 2010; Eggertsson et al., 2017).

Inferring Tree Sequences from Genotype Matrices

To infer tree sequences from genotype matrices or true tree sequences, we use tsinfer v.0.2.3 (Kelleher et al., 2019), a scalable tree inference method. True tree sequences can be easily converted to the *.sample* data format required by *tsinfer* using the command *tsinfer.SampleDate.from_tree_sequence* in the python API. To convert genotype matrices (phased VCF files) to *.sample* files, a python script is implemented to load the genetic variants in VCF files with the cyvcf2 package (Pedersen and Quinlan, 2017) and convert them into the sample object. Per the guideline of *tsinfer*, only biallelic sites can be used for inference. For positions with multiple allelic variants, only the reference allele and the first alternative allele are retained in the *.sample* files. Non-single nucleotide polymorphisms are also removed. Once the *.sample* files are prepared, the *tsinfer.infer* function is used to estimate the genealogies with the full inference pipeline. Default settings are applied for the recombination rate and mismatch ratio.

For the 1000 Genomes Project, we used 20 threads during *tsinfer* inference for chromosomes 1-3, 10, 13-15, and 21; 10 threads for chromosomes 4-9, 11-12, and 22; and 5 threads for chromosomes 16-20. For the simulated data, we used 5 threads for simulations with sample sizes from 10 to 10000, and 10 threads for 100000 samples.

Tree Sequence to GRG Conversion

Tree sequences are converted to GRG by copying and duplicating nodes and edges from the tree sequence into a new GRG. Every recombination event that occurs between trees $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$ in a tree sequence can be represented as a set of edge insertions and deletions to $\mathcal{T}_k$ in order to produce $\mathcal{T}_{k+1}$. Consider a single edge $E_1 = (n_i, n_j)$ being deleted and another edge $E_2 = (n_r, n_p)$ being added (**Fig. S10**). n_i and n_r are the parent nodes for these edges. The nodes above n_r now potentially have more samples beneath them

(because n_p is a new child, and it has samples beneath it). The nodes above n_i now potentially have *fewer* samples beneath them (because n_j is no longer a child, and it had samples beneath it). Now consider the most recent common ancestor (MRCA) (Kingman, 1982b; 1982a) between n_i and n_r. All nodes at and above the MRCA have no change to their reached samples: those samples were moved from one subtree of the MRCA to another subtree of the MRCA. This implies an efficient tree sequence to GRG conversion algorithm: nodes from $\mathcal{T}_0$ are copied into GRG $\mathcal{G}$ to produce the initial graph. The MRCA between added/removed edges for subsequent trees $\mathcal{T}_i$ is used to determine a list of nodes that need their reached samples modified. In constructing the GRG, we duplicate a node whenever its reached sample set changes, and copy the relevant graph edges.

Even for large inferred tree sequences, converting from a tree sequence to a GRG takes on the order of minutes. Converting simulated tree sequences to GRG takes seconds even for hundreds of thousands of samples.

Effects of Noise

We explore the effects of noise in the dataset by randomly flipping alleles in VCF files produced from *msprime* simulations. For a given percentage of noise, P_n, and a dataset with N individuals and V variants, we uniformly sample $(P_n / 100) \times N \times V$ pairs of (individuals, sites) and flip each allele from its current value in the set {0|1, 1|0, 1|1, 0|0} to one of the other values in that set. We chose our experimental value of $P_n = 0.1\%$ to be in line with common error estimates on medium to high coverage DNA sequencing (i.e., 20-30×) data.

Data Availability

The phased *vcf.gz* files used in this study from the Phase III 1000 Genomes Project can be downloaded through https://hgdownload.cse.ucsc.edu/gbdb/hg19/1000Genomes/phase3/.

The UK Biobank data is available to approved researchers through http://dnanexus.com/. The latest SHAPEIT phased VCFs from the field 20279 are used for analysis and the details about it are documented at https://biobank.ndph.ox.ac.uk/showcase/refer.cgi?id=1910. The inferred GRG from the SHAPEIT phased data will be returned to the UK Biobank for access by approved projects.

The inferred tree sequences of SARS-CoV-2 genomes from Zhan et al., 2023 are available by request from the original authors with approved GISAID data access. The original genomes of the SARS-CoV-2 are available on GISAID, via gisaid.org/EPI_SET_230329cd and included in the Supplemental Table (GISAID EPI SET PDF). The inferred GRG of SARS-CoV-2 genomes are available by request with approved GISAID data access.

Code Availability

We have released an open-source, lightweight VCF parser picovcf v2.0 through https://github.com/aprilweilab/picovcf/releases/tag/v2.0. The Genotype Representation Graph Library (GRGL), which implements the inference of GRGs and downstream computations, will be released

through https://github.com/aprilweilab/grgl. A detailed description of the software is provided in the **Supplement**.

External softwares used in this study include:
msprime v.1.2.0, https://pypi.org/project/msprime/1.2.0/;
tskit v.0.5.5, https://github.com/tskit-dev/tskit/releases/tag/0.5.5;
tsinfer v.0.2.3, https://github.com/tskit-dev/tsinfer/releases/tag/0.2.3;
plink v.1.90b6.26, https://www.cog-genomics.org/plink/.

Supplementary Materials

Supplementary Text, Algorithm S1-S3, Table S1-S2, and Figure S1-S14, and a Supplementary Table for GISAID SARS-CoV-2 data.

Acknowledgments

We gratefully acknowledge all data contributors, i.e., the Authors and their Originating laboratories responsible for obtaining the specimens, and their Submitting laboratories for generating the genetic sequence and metadata and sharing via the GISAID Initiative, on which this research is based. We thank Jerome Kelleher for generously sharing the inferred SARS-CoV-2 tree sequences with us. This work is partly supported by NIH R35-GM150579 to X.W. This research is conducted under the UK Biobank application 97908.

References

Band, Gavin, and Jonathan Marchini. 2018. "BGEN: A Binary File Format for Imputed Genotype and Haplotype Data." Preprint. Bioinformatics. https://doi.org/10.1101/308296.

Baumdicker, Franz, Gertjan Bisschop, Daniel Goldstein, Graham Gower, Aaron P Ragsdale, Georgia Tsambos, Sha Zhu, et al. 2022. "Efficient Ancestry and Mutation Simulation with Msprime 1.0." Edited by S Browning. *Genetics* 220 (3): iyab229. https://doi.org/10.1093/genetics/iyab229.

Bloom, Burton H. 1970. "Space/Time Trade-Offs in Hash Coding with Allowable Errors." *Communications of the ACM* 13 (7): 422–26. https://doi.org/10.1145/362686.362692.

Browning, Brian L., and Sharon R. Browning. 2023. "Statistical Phasing of 150,119 Sequenced Genomes in the UK Biobank." *The American Journal of Human Genetics* 110 (1): 161–65. https://doi.org/10.1016/j.ajhg.2022.11.008.

Browning, Brian L., and Zhaoxia Yu. 2009. "Simultaneous Genotype Calling and Haplotype Phasing Improves Genotype Accuracy and Reduces False-Positive Associations for Genome-Wide Association Studies." *The American Journal of Human Genetics* 85 (6): 847–61. https://doi.org/10.1016/j.ajhg.2009.11.004.

Buluç, Aydin, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. 2009. "Parallel Sparse Matrix-Vector and Matrix-Transpose-Vector Multiplication Using Compressed Sparse Blocks." In Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures, 233–44. Calgary AB Canada: ACM. https://doi.org/10.1145/1583991.1584053.

Burkhard, W. A., and R. M. Keller. 1973. "Some Approaches to Best-Match File Searching." *Communications of the ACM* 16 (4): 230–36. https://doi.org/10.1145/362003.362025.

Chang, Christopher C, Carson C Chow, Laurent Cam Tellier, Shashaank Vattikuti, Shaun M Purcell, and James J Lee. 2015. "Second-Generation PLINK: Rising to the Challenge of Larger and Richer Datasets." *GigaScience* 4 (1): 7. https://doi.org/10.1186/s13742-015-0047-8.

Collet, Y., and M. Kucherawy. 2021. "Zstandard Compression and the 'application/Zstd' Media Type." RFC8878. RFC Editor. https://doi.org/10.17487/RFC8878.

Cormen, Thomas H., Charles Eric Leiserson, Ronald L. Rivest, and Clifford Stein. 2022. *Introduction to Algorithms*. Fourth edition. Cambridge, Massachusett: The MIT Press.

Danecek, P., A. Auton, G. Abecasis, C. A. Albers, E. Banks, M. A. DePristo, R. E. Handsaker, et al. 2011. "The Variant Call Format and VCFtools." *Bioinformatics* 27 (15): 2156–58. https://doi.org/10.1093/bioinformatics/btr330.

Deutsch, P. 1996. "GZIP File Format Specification Version 4.3." RFC1952. RFC Editor. https://doi.org/10.17487/rfc1952.

Duff, Iain S. 1984. "Computer Solution of Large Sparse Positive Definite Systems (Alan George and Joseph W. Liu." *SIAM Review* 26 (2): 289–91. https://doi.org/10.1137/1026055.

Durbin, Richard. 2014. "Efficient Haplotype Matching and Storage Using the Positional Burrows–Wheeler Transform (PBWT)." *Bioinformatics* 30 (9): 1266–72. https://doi.org/10.1093/bioinformatics/btu014.

Eddy, Sean R. 2004. "What Is Dynamic Programming?" *Nature Biotechnology* 22 (7): 909–10. https://doi.org/10.1038/nbt0704-909.

Eggertsson, Hannes P, Hakon Jonsson, Snaedis Kristmundsdottir, Eirikur Hjartarson, Birte Kehr, Gisli Masson, Florian Zink, et al. 2017. "Graphtyper Enables Population-Scale Genotyping Using Pangenome Graphs." *Nature Genetics* 49 (11): 1654–60. https://doi.org/10.1038/ng.3964.

Fitch, W. M., and E. Margoliash. 1967. "Construction of Phylogenetic Trees." *Science (New York, N.Y.)* 155 (3760): 279–84. https://doi.org/10.1126/science.155.3760.279.

Furnas, George W., and Jeff Zacks. 1994. "Multitrees: Enriching and Reusing Hierarchical Structure." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems Celebrating Interdependence - CHI '94*, 330–36. Boston, Massachusetts, United States: ACM Press. https://doi.org/10.1145/191666.191778.

Gailly, Jean-loup, and Mark Adler. 2004. "Zlib Compression Library." *Other*. http://www.dspace.cam.ac.uk/handle/1810/3486.

Giegerich, Robert. 2000. "A Systematic Approach to Dynamic Programming in Bioinformatics." *Bioinformatics* 16 (8): 665–77. https://doi.org/10.1093/bioinformatics/16.8.665.

Gm, Harshvardhan, Mahendra Kumar Gourisaria, Manjusha Pandey, and Siddharth Swarup Rautaray. 2020. "A Comprehensive Survey and Analysis of Generative Models in Machine Learning." *Computer Science Review* 38 (November): 100285. https://doi.org/10.1016/j.cosrev.2020.100285.

Gray, Alexander G., Andrew W. Moore, Robert C. Nichol, Andrew J. Connolly, Christopher Genovese, and Larry Wasserman. 2004. "Multi-Tree Methods for Statistics on Very Large Datasets in Astronomy." https://doi.org/10.48550/ARXIV.ASTRO-PH/0401121.

Greenland, S., J. Pearl, and J. M. Robins. 1999. "Causal Diagrams for Epidemiologic Research." *Epidemiology (Cambridge, Mass.)* 10 (1): 37–48.

Griffiths, R.C., and P. Marjoram. 1996. "Ancestral Inference from Samples of DNA Sequences with Recombination." *Journal of Computational Biology* 3 (4): 479–502. https://doi.org/10.1089/cmb.1996.3.479.

Griffiths, Robert C., and Paul Marjoram. 1997. "An Ancestral Recombination Graph." *In Progress in Population Genetics and Human Evolution*, 257–70.

Halldorsson, Bjarni V., Hannes P. Eggertsson, Kristjan H. S. Moore, Hannes Hauswedell, Ogmundur Eiriksson, Magnus O. Ulfarsson, Gunnar Palsson, et al. 2022. "The Sequences of 150,119 Genomes in the UK Biobank." *Nature* 607 (7920): 732–40. https://doi.org/10.1038/s41586-022-04965-x.

Hofmeister, Robin J., Diogo M. Ribeiro, Simone Rubinacci, and Olivier Delaneau. 2023. "Accurate Rare Variant Phasing of Whole-Genome and Whole-Exome Sequencing Data in the UK Biobank." *Nature Genetics* 55 (7): 1243–49. https://doi.org/10.1038/s41588-023-01415-w.

Howie, Bryan N., Peter Donnelly, and Jonathan Marchini. 2009. "A Flexible and Accurate Genotype Imputation Method for the Next Generation of Genome-Wide Association Studies." Edited by Nicholas J. Schork. *PLoS Genetics* 5 (6): e1000529. https://doi.org/10.1371/journal.pgen.1000529.

Hudson, Richard R. 1990. "Gene Genealogies and the Coalescent Process." *Oxford Surveys in Evolutionary Biology* 7: 1–44.

Huson, Daniel H., and David Bryant. 2006. "Application of Phylogenetic Networks in Evolutionary Studies." *Molecular Biology and Evolution* 23 (2): 254–67. https://doi.org/10.1093/molbev/msj030.

Kapli, Paschalia, Ziheng Yang, and Maximilian J. Telford. 2020. "Phylogenetic Tree Building in the Genomic Age." *Nature Reviews Genetics* 21 (7): 428–44. https://doi.org/10.1038/s41576-020-0233-0.

Kelleher, Jerome, Kevin R. Thornton, Jaime Ashander, and Peter L. Ralph. 2018. "Efficient Pedigree Recording for Fast Population Genetics Simulation." Edited by Sergei L. Kosakovsky Pond. *PLOS Computational Biology* 14 (11): e1006581. https://doi.org/10.1371/journal.pcbi.1006581.

Kelleher, Jerome, Yan Wong, Anthony W. Wohns, Chaimaa Fadil, Patrick K. Albers, and Gil McVean. 2019. "Inferring Whole-Genome Histories in Large Population Datasets." *Nature Genetics* 51 (9): 1330–38. https://doi.org/10.1038/s41588-019-0483-y.

Kingma, Friso H., Pieter Abbeel, and Jonathan Ho. 2019. "Bit-Swap: Recursive Bits-Back Coding for Lossless Compression with Hierarchical Latent Variables." https://doi.org/10.48550/ARXIV.1905.06845.

Kingman, J. F. C. 1982a. "On the Genealogy of Large Populations." *Journal of Applied Probability* 19 (A): 27–43. https://doi.org/10.2307/3213548.

Kingman, J.F.C. 1982b. "The Coalescent." *Stochastic Processes and Their Applications* 13 (3): 235–48. https://doi.org/10.1016/0304-4149(82)90011-4.

Marjoram, Paul, and Jeff D Wall. 2006. "Fast 'Coalescent' Simulation." *BMC Genetics* 7 (1): 16. https://doi.org/10.1186/1471-2156-7-16.

McKenna, Aaron, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, et al. 2010. "The Genome Analysis Toolkit: A MapReduce Framework for Analyzing next-Generation DNA Sequencing Data." *Genome Research* 20 (9): 1297–1303. https://doi.org/10.1101/gr.107524.110.

Miller, Jason R., Sergey Koren, and Granger Sutton. 2010. "Assembly Algorithms for Next-Generation Sequencing Data." *Genomics* 95 (6): 315–27. https://doi.org/10.1016/j.ygeno.2010.03.001.

Pedersen, Brent S., and Aaron R Quinlan. 2017. "Cyvcf2: Fast, Flexible Variant Analysis with Python." Edited by John Hancock. *Bioinformatics* 33 (12): 1867–69. https://doi.org/10.1093/bioinformatics/btx057

Purcell, Shaun, Benjamin Neale, Kathe Todd-Brown, Lori Thomas, Manuel A.R. Ferreira, David Bender, Julian Maller, et al. 2007. "PLINK: A Tool Set for Whole-Genome Association and Population-Based Linkage Analyses." *The American Journal of Human Genetics* 81 (3): 559–75. https://doi.org/10.1086/519795.

Rasmussen, Matthew D., Melissa J. Hubisz, Ilan Gronau, and Adam Siepel. 2014. "Genome-Wide Inference of Ancestral Recombination Graphs." Edited by Graham Coop. *PLoS Genetics* 10 (5): e1004342. https://doi.org/10.1371/journal.pgen.1004342.

Ribeiro, Diogo, Robin Hofmeister, Simone Rubinacci, and Oliver Delaneau. 2023. "Phasing of the UK Biobank Whole Genome Sequencing Data Interim Release of 200,031 Samples," July. https://biobank.ndph.ox.ac.uk/showcase/ukb/docs/PhasingUKB200k_report_SHAPEIT.pdf.

Saunders, Ian W., Jesper Brohede, and Garry N. Hannan. 2007. "Estimating Genotyping Error Rates from Mendelian Errors in SNP Array Genotypes and Their Impact on Inference." *Genomics* 90 (3): 291–96. https://doi.org/10.1016/j.ygeno.2007.05.011.

Shu, Yuelong, and John McCauley. 2017. "GISAID: Global Initiative on Sharing All Influenza Data – from Vision to Reality." *Eurosurveillance* 22 (13). https://doi.org/10.2807/1560-7917.ES.2017.22.13.30494.

Speidel, Leo, Marie Forest, Sinan Shi, and Simon R. Myers. 2019. "A Method for Genome-Wide Genealogy Estimation for Thousands of Samples." *Nature Genetics* 51 (9): 1321–29. https://doi.org/10.1038/s41588-019-0484-x.

The 1000 Genomes Project Consortium. 2015. "A Global Reference for Human Genetic Variation." *Nature* 526 (7571): 68–74. https://doi.org/10.1038/nature15393.

Waggener, William N. 1995. *Pulse Code Modulation Techniques: With Applications in Communications and Data Recording*. New York: Van Nostrand Reinhold.

Y. C. Brandt, Débora, Xinzhu Wei, Yun Deng, Andrew H Vaughn, and Rasmus Nielsen. 2022. "Evaluation of Methods for Estimating Coalescence Times Using Ancestral Recombination Graphs." Edited by N Barton. *Genetics* 221 (1): iyac044. https://doi.org/10.1093/genetics/iyac044.

Yang, Yibo, Stephan Mandt, and Lucas Theis. 2022. "An Introduction to Neural Data Compression." https://doi.org/10.48550/ARXIV.2202.06533.

Zhan, Shing H., Anastasia Ignatieva, Yan Wong, Katherine Eaton, Benjamin Jeffery, Duncan S. Palmer, Carmen Lia Murall, Sarah P. Otto, and Jerome Kelleher. 2023. "Towards Pandemic-Scale Ancestral Recombination Graphs of SARS-CoV-2." https://doi.org/10.1101/2023.06.08.544212.

Zhang, Brian C., Arjun Biddanda, Árni Freyr Gunnarsson, Fergus Cooper, and Pier Francesco Palamara. 2023. "Biobank-Scale Inference of Ancestral Recombination Graphs Enables Genealogical Analysis of Complex Traits." *Nature Genetics* 55 (5): 768–76. https://doi.org/10.1038/s41588-023-01379-x.