Pangenome-based genome inference using integer programming

Ghanshyam Chandra, Md Helal Hossen, Stephan Scholz, et al.

Genome Res. 2025 35: 2661-2670 originally published online August 21, 2025
Access the most recent version at doi:10.1101/gr.280567.125

Method

Pangenome-based genome inference using integer programming

Ghanshyam Chandra,[1] Md Helal Hossen,[2] Stephan Scholz,[3,4] Alexander T. Dilthey,[3,4] Daniel Gibney,[2] and Chirag Jain[1]

[1]*Department of Computational and Data Sciences, Indian Institute of Science, Bangalore, Karnataka 560012, India;* [2]*Department of Computer Science, The University of Texas at Dallas, Dallas, Texas 75080, USA;* [3]*Institute of Medical Microbiology and Hospital Hygiene, Heinrich Heine University Düsseldorf, 40225 Düsseldorf, Germany;* [4]*Center for Digital Medicine, Heinrich Heine University Düsseldorf, 40225 Düsseldorf, Germany*

Affordable genotyping methods are essential in genomics. Commonly used genotyping methods primarily support single-nucleotide variants and short indels but neglect structural variants. Additionally, accuracy of read alignments to a reference genome is unreliable in highly polymorphic and repetitive regions, further impacting genotyping performance. Recent works highlight the advantage of pangenome graphs in addressing these challenges. Building on these developments, we propose a rigorous alignment-free genotyping method. Our optimization framework identifies a path through the pangenome graph that maximizes the matches between the path and substrings of sequencing reads (e.g., *k*-mers) while minimizing recombination events (haplotype switches) along the path. We prove that this problem is NP-hard and develop efficient integer-programming solutions. We benchmark the algorithm using downsampled short-read data sets from homozygous human cell lines with coverage ranging from 0.1× to 10×. Our algorithm accurately estimates complete major histocompatibility complex (MHC) haplotype sequences with small edit distances from the ground-truth sequences, providing a significant advantage over existing methods on low-coverage inputs.

[Supplemental material is available for this article.]

Many initiatives are in progress for building haplotype-resolved pangenome references of human and non-human species (Gao et al. 2023; Liao et al. 2023; Smith et al. 2023). Among many applications, pangenome graphs can enable cost-effective genotyping and imputation of a wide spectrum of variant classes beyond single-nucleotide polymorphisms (SNPs) and short insertions and deletions (indels) (Harris et al. 2024). Pangenome graphs represent sequence alignment of high-quality fully phased genome assemblies of individuals from diverse populations (Baaijens et al. 2022). A pangenome graph can be represented as either a cyclic or acyclic directed graph in which the vertices are labeled with sequences. Paths in this graph spell the reference haplotype sequences and their recombinations. The graph-based representation is flexible enough to incorporate SNPs, indels, large structural variants (SVs), nested variants, gene absence/presence, etc. (Computational Pan-Genomics Consortium 2018).

Recent works propose the use of pangenome references to improve genotyping accuracy from short-read sequencing data (Eggertsson et al. 2017; Sibbesen et al. 2018; Hickey et al. 2020; Letcher et al. 2021; Bradbury et al. 2022; Ebler et al. 2022; Grytten et al. 2022; Mun et al. 2023). Especially for SVs, these methods are an effective alternative to the conventional genotyping methods that are based on aligning reads to a single reference because short-read alignments can be inaccurate for the reads originating from SVs (Mahmoud et al. 2019; Ebert et al. 2021). Methods such as PRG (Dilthey et al. 2015), Pangenie (Ebler et al. 2022) and KAGE (Grytten et al. 2022) utilize *k*-mer statistics to infer paths in the graph that correspond to the target genome. These methods compare the *k*-mers surrounding a variant site in the graph with the *k*-mer counts in the sequencing data to calculate likelihoods of reference and alternative alleles. Pangenie and KAGE also use the long-range haplotype information available in the haplotype-resolved pangenome references. The other approach used in methods such as Giraffe (Sirén et al. 2021) and Graphtyper (Eggertsson et al. 2017) involves aligning reads to a pangenome graph.

There have been efforts on improving the accuracy of read alignments to pangenome graphs as well. A large combinatorial search space in terms of the number of candidate paths in a pangenome graph increases ambiguity during read alignment. This issue has motivated methods that impute a personalized reference genome (Vaddadi et al. 2023) or sample variants (Pritt et al. 2018; Jain et al. 2021; Tavakoli et al. 2022) to obtain a smaller graph or that prioritize the use of reference haplotypes in the graph during alignment (Sirén et al. 2021; Chandra et al. 2024; Mustafa et al. 2024). Our previous work proposed haplotype-aware sequence alignment to graphs by introducing penalties for haplotype switches in an alignment (Chandra et al. 2024). A recent feature added to VG allows sampling of reference haplotypes and their recombinations from the graph that are most relevant to the target genome using a *k*-mer-based greedy heuristic (Sirén et al. 2024).

Low-coverage sequencing, combined with genotyping and phasing, is a cost-effective approach to conduct large-scale genetic

Corresponding author: chirag@iisc.ac.in
Article published online before print. Article, supplemental material, and publication date are at https://www.genome.org/cgi/doi/10.1101/gr.280567.125.

studies (Davies et al. 2021; Li et al. 2021; Martin et al. 2021; Rubinacci et al. 2021). In this paper, we develop a rigorous formulation and algorithms for genotyping using pangenome references. Our framework is also applicable to low-coverage short-read sequencing data (coverage 0.1×–1×). Similar to the standard Li and Stephens model (Li and Stephens 2003), we view the target genome as an imperfect mosaic of the reference haplotypes. Our contributions are as follows:

- We introduce a novel optimization framework to estimate the complete haplotype sequence of a haploid genome by determining an appropriate path in the pangenome graph. The objective is to maximize the number of shared substrings (e.g., *k*-mers or minimizers) between the sequencing data and the sequence spelled by the path. We permit recombinations in the path, subject to a fixed penalty per recombination. We refer to this problem as the *path inference problem* (formally defined in the Methods).
- We prove that the path inference problem is NP-hard.
- To solve this problem, we develop two integer-programming solutions that involve linear and quadratic constraints, respectively. The two solutions involve a tradeoff between runtime and memory usage.
- We demonstrate the utility of this framework by testing it on downsampled short-read data sets from five human haploid cell lines (coverage 0.1×–10×). For these five samples, complete major histocompatibility complex (MHC) haplotype sequences have been previously determined using long-read assembly (Houwaart et al. 2023). As our pangenome reference, we used a haplotype-resolved pangenome directed acyclic graph (DAG) of 49 MHC haplotype sequences (Li 2022). We chose the MHC region for evaluation because it is the most polymorphic and gene-rich region of the human genome (Dilthey 2021). The length of this region is ∼5 Mbp.
- Using data sets with 0.1× coverage, our algorithm outputs MHC sequences that are up to 99.96% identical to the ground-truth sequences. It compares favorably to the existing methods.

Methods

Overview

Our method, pangenome-based haplotype inference (PHI), takes as input a pangenome graph reference and short-read sequencing data from a target haploid genome. We assume that the given pangenome graph has been constructed using high-quality, haplotype-resolved genome assemblies. This graph is modeled as a DAG, in which each node is labeled with a DNA sequence, and edges represent the adjacency of these sequences along known haplotypes. Any path through this graph can represent an existing haplotype or a recombinant mosaic of segments from multiple haplotypes.

In the Li–Stephens haplotype model (Li and Stephens 2003), a new haplotype is formed by copying segments from reference haplotypes, occasionally switching between them. Inspired by this model, we aim to identify a path through the graph that best explains the observed sequencing reads by maximizing the number of shared *k*-mers between the reads and the sequence encoded by the chosen path. To account for recombination, our method allows switching between haplotypes in the graph but imposes a fixed penalty for each recombination event. We refer to this computational task as the path inference problem (illustrated in Fig. 1).

We show that solving this optimization task exactly is computationally intractable (NP-hard). To address this, we develop two efficient integer programming algorithms that leverage structural properties of the problem. A key component in our method is the construction of an *expanded graph*, which is a modified version of the pangenome graph in which each reference haplotype is represented as a separate path, and all haplotype paths start and end at common *source* and *sink* vertices, respectively. Within this expanded graph, the path inference problem can be reinterpreted as a constrained network flow problem: identifying a unit flow from source to sink that minimizes the combined penalties for recombinations and unmatched *k*-mers. In the following subsections, we describe our algorithms formally.

Notations and problem formulation

Let $G(V, E, \sigma, \mathcal{H})$ denote a DAG representing a haplotype-resolved pangenome reference. Function σ assigns a string label over alphabet $\Sigma = \{A, C, G, T\}$ to each vertex. A path $(u_1, u_2, \ldots, u_n)$ in G spells string $\sigma(u_1) \circ \sigma(u_2) \circ \ldots \circ \sigma(u_n)$, where $s_1 \circ s_2$ denotes the concatenation of strings s_1 and s_2. $\mathcal{H} = \{h_1, h_2, \ldots, h_{|\mathcal{H}|}\}$ denotes a set of paths in G such that each of these paths spells a reference haplotype sequence used in the pangenome reference. We refer to these paths as haplotype paths. We assume that each haplotype path is described by an array; that is, $h_i[1]$ is the first vertex in h_i, $h_i[2]$ is the second vertex in h_i, etc. The length of a haplotype path h_i, that is, the count of vertices in h_i, is denoted as $|h_i|$. The set of haplotype paths covering vertex $v \in V$ is denoted as $haps(v)$. We assume that, for each edge $(u, v) \in E$, there exists a haplotype path $h_i \in \mathcal{H}$ such that u and v are consecutive vertices in h_i. In other words, each edge is supported by at least one haplotype path. Functions $N^+(v)$ and $N^-(v)$ denote the sets of out-neighbors and in-neighbors of vertex v, respectively.

Genotyping a haploid genome sample can be framed as finding a path in the pangenome graph that contains the sample's variants (Paten et al. 2017). Such a path can be modeled as a mosaic path formed by recombinations between the haplotype paths of the graph. Based on this intuition, we define an *inferred path* $\mathcal{P}$ of length n as an ordered set $(a_1, a_2, \ldots, a_n)$, where each a_i is a two-tuple (u, h), $u \in V$, $h \in \mathcal{H}$. For a tuple a_i, we use $a_i.u$ to denote the respective vertex and $a_i.h$ to denote the respective haplotype path. By using this notation, we keep track of the haplotype path being used alongside the vertex indices. In an inferred path, $(a_i.u, a_{i+1}.u) \in E$ for all $i \in [1, n)$. If $a_i.h = a_{i+1}.h$, then we also require that $a_i.u$ and $a_{i+1}.u$ must be consecutive vertices in haplotype path $a_i.h$. The string spelled by path $\mathcal{P}$ is $\sigma(a_1.u) \circ \sigma(a_2.u) \circ \ldots \circ \sigma(a_n.u)$. We denote this string as $\sigma(\mathcal{P})$, with a slight abuse of notation.

We say a *recombination*, or a haplotype switch, occurs between two consecutive vertices $a_i.u$ and $a_{i+1}.u$ in $\mathcal{P}$ if $a_i.h \neq a_{i+1}.h$. We use $\gamma(\mathcal{P})$ to denote the count of recombinations in $\mathcal{P}$. Because an inferred path must be a path in the pangenome graph by our definition, this setup models homologous recombination events during meiosis. Our goal is to determine the best path in the pangenome graph that is consistent with the set of strings (e.g., *k*-mers) present in the sequencing reads. We define the problem as follows.

Problem 1 (path inference problem)

– **Input:** a haplotype-resolved pangenome DAG $G = (V, E, \sigma, \mathcal{H})$, a set of strings $\mathcal{S}$ from the target genome, and a nonnegative integer c indicating recombination penalty.
– **Output:** an inferred path $\mathcal{P}$ such that

$$Cost(\mathcal{P}) = c \cdot \gamma(\mathcal{P}) + \sum_{r \in \mathcal{S}} \bar{\chi}(r, \sigma(\mathcal{P}))$$

is minimized, where $\bar{\chi}(r, \sigma(\mathcal{P})) = 0$ if string r occurs as a substring of string $\sigma(\mathcal{P})$ and 1 otherwise.

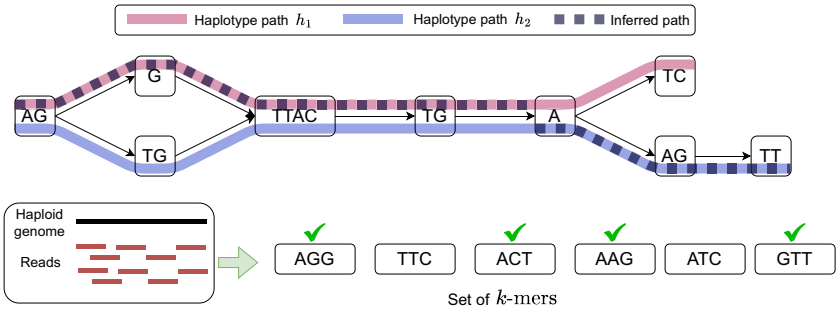

Figure 1. A schematic to illustrate the path inference problem. At the *top*, we show a haplotype-resolved pangenome graph containing two haplotype paths, h_1 and h_2. These paths are shown in pink and blue, respectively. An inferred path containing a single recombination is illustrated using a thick dashed line. At the *bottom*, we show a set of k-mers observed in the sequencing reads assuming $k=3$. A k-mer is marked with a green tick if it is a substring of the sequence spelled by the inferred path AGGTTACTGAAGTT. Two k-mers (TTC and ATC) are not present in this sequence. Our optimization framework identifies an optimal inferred path with minimum cost given user-defined penalties for recombinations and for missing k-mers.

The intuition behind our formulation is to maximize the number of string matches along the inferred path while minimizing the number of recombinations. This approach yields an inferred path that incorporates the majority of strings from $\mathcal{S}$ as a substring with a finite number of recombinations, constrained by a recombination penalty c. Set $\mathcal{S}$ in the above formulation can be a set of either k-mers or minimizers observed in the sequencing reads. The optimization is robust to sequencing errors. Errors in reads typically introduce novel k-mers that are not present anywhere in the graph. Because such erroneous k-mers cannot be matched to any path in the graph, the associated cost is unavoidable. As a result, they do not affect the optimization outcome.

We assume a constant cost per recombination. Ideally, the model should use variable, locus-specific recombination costs based on known recombination rates (Petes 2001). However, recombination maps analogous to those used on linear reference genomes are not yet available for pangenome graphs.

Next, we consider the complexity of path inference problem. We show that the problem is NP-hard, even when restricted to binary alphabets.

Theorem 1. *Problem 1 is NP-hard. This holds for any value of* $c = |V|^{\Theta(1)}$ *and even when* $\Sigma = \{0, 1\}$.

We refer the reader to Supplemental Note S1 for a proof (see also Supplemental Fig. S1). This result suffices to establish the theoretical hardness of finding an optimal inferred path under the given cost function. The hardness result holds specifically for our formulation (Problem 1) and does not apply to other possible formulations for genome inference on pangenome graphs. Note that there exist many versions of Problem 1 that could be practically useful and whose computational complexity remains open, for example, when a small (e.g., polylogarithmic) bound is given on the number of allowed recombinations in the inferred path or when the set of haplotype paths in the pangenome graph is small. We leave this investigation for future research.

Construction of an expanded graph

Before developing our integer programming solutions to Problem 1, it is first helpful to define an additional graph representation, which we call as *expanded graph*. In pangenome graphs, multiple haplotype paths share vertices if the sequences are conserved, whereas in the expanded graph, we will split all haplotypes into separate paths (Fig. 2A,B). The expanded graph enables us to model Problem 1 as a sort of network flow problem. In particular, the inferred path will be reconstructed from a flow of value one in the expanded graph. We will assign weights to edges to account for recombination penalty. Additional constraints will be used to capture how many strings in $\mathcal{S}$ occur in the resulting inferred path.

In an inferred path $\mathcal{P} = (a_1, \ldots, a_n)$, we call a recombination $a_i.h \neq a_{i+1}.h$ *useful* except when $a_i.u$ and $a_{i+1}.u$ are consecutive vertices in haplotype path $a_i.h$. Lemma 1 allows us to only consider useful recombinations when finding an optimal solution to Problem 1.

Lemma 1. *There exists an optimal inferred path* $\mathcal{P} = (a_1, \ldots, a_n)$ *for Problem 1 such that all recombinations in* $\mathcal{P}$ *are useful.*

Proof. Suppose there is an optimal inferred path $\mathcal{P} = (a_1, \ldots, a_n)$ for Problem 1 in which for some a_i, $a_i.h \neq a_{i+1}.h$ such that $a_i.u$ and $a_{i+1}.u$ are consecutive vertices in haplotype path $a_i.h$. Furthermore, suppose we start with the smallest i where this holds. We then change the haplotype path for a_{i+1} to equal $a_i.h$. This does not increase the overall cost because the number of strings in $\mathcal{S}$ occurring in $\sigma(\mathcal{P})$ has not changed, and the number of recombinations either decreases or stays the same. Continuing this process from the next $j > i$, such that $a_j.h \neq a_{j+1}.h$ and $a_j.u$ and $a_{j+1}.u$ are consecutive vertices in $a_j.h$, we achieve an inferred path satisfying the conditions stated in the lemma after at most n iterations.

Next, we present a definition of the expanded graph in which we will consider only the useful recombinations. For technical reasons, we preprocess each edge in E, splitting it and adding a new vertex labeled with the empty string ε. Each added vertex inherits the haplotype paths that supported the edge it was formed from. This added step is to prevent recombinations from a haplotype to itself when we build our expanded graph.

We use $G_E = (V_E, E_E, \sigma_E)$ to denote the expanded graph. In G_E, vertices are string-labeled and edges are weighted. Vertex set V_E is defined as

$$V_E = \{s\} \cup \{t\} \cup \{(h_j[i])^j | 1 \le j \le |\mathcal{H}|, 1 \le i \le |h_j|\}, \tag{1}$$

where $h_j[i]$ is a vertex in V (which may appear within multiple paths in $\mathcal{H}$), and $(h_j[i])^j$ is a newly created vertex in V_E. We refer to the ordered vertex set $(h_j[1])^j \cdots (h_j[|h_j|])^j$ as a haplotype path for h_j in G_E. Note that the superscript is used to indicate to which haplotype path the vertex is designated. This results in disjoint vertices for each haplotype path in $\mathcal{H}$. The vertex set also contains a source and sink vertex, s and t, respectively (Fig. 2A,B).

We denote weighted edges in E_E as tuples of the form (*start*, *end*, *weight*). The weighted edge set is

$$E_E = \{(s, (h_j[1])^j, 0) \mid 1 \le j \le |\mathcal{H}|\} \tag{2}$$

$$\cup \{((h_j[|h_j|])^j, t, 0) \mid 1 \le j \le |\mathcal{H}|\} \tag{3}$$

$$\cup \{((h_j[i])^j, (h_j[i+1])^j, 0) \mid 1 \le j \le |\mathcal{H}|, 1 \le i < |h_j|\} \tag{4}$$

$$\cup \{((h_j[i])^j, (h_{j'}[y])^{j'}, c) \mid 1 \le j, j' \le |\mathcal{H}|, 1 \le i \le |h_j|, 1 \le y \le |h_{j'}|, \exists (h_j[i], h_{j'}[y]) \in E, i = |h_j| \text{ or } h_j[i+1] \neq h_{j'}[y]\} \tag{5}$$

Next, we give some intuition for each line (2)–(5) in the above construction of E_E.

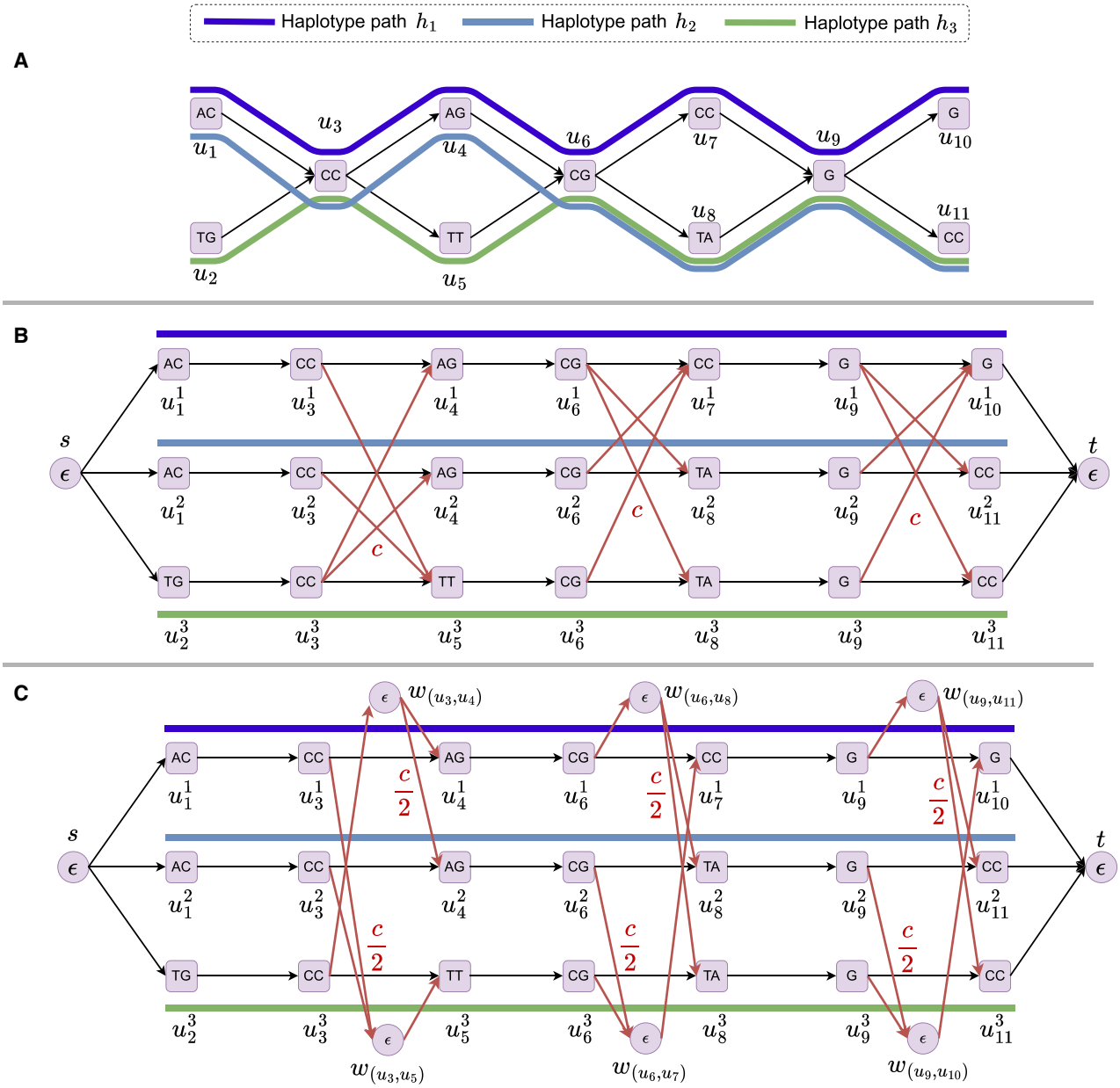

Figure 2. Illustration of an expanded graph. (*A*) A pangenome graph with three haplotype paths: h_1, h_2, and h_3. (*B*) The corresponding expanded graph, which includes three disjoint paths, one for each haplotype path. The recombination edges are shown in purple; these edges have a weight of c. We consider only the useful recombinations (Lemma 1). The edges that are not recombination edges in the expanded graph have a weight of 0. (*C*) The corresponding optimized expanded graph.

(2) Weight zero edges are created from s to the start of each haplotype path in G_E.
(3) Weight zero edges are created from the end of each haplotype path in G_E to t.
(4) Weight zero edges are created between adjacent vertices in each haplotype path. That is, in the path for h_j, an edge is created from $(h_j[i])^j$ to $(h_j[i+1])^j$.
(5) Weight c edges are used to represent the useful recombinations described in Lemma 1. We call these *recombination edges*.

We use ε to denote the empty string. The vertex labels are defined as follows:

$$\sigma_E((h_j[i])^j) = \sigma(h_j[i]) \text{ for } 1 \le j \le |\mathcal{H}|,\ 1 \le i \le |h_j| \quad (6)$$

$$\sigma_E(s) = \sigma_E(t) = \epsilon. \quad (7)$$

(6) The vertices in a haplotype path are labeled according to the corresponding vertex label in G. These labels will be used to identify matches.
(7) The source and sink do not require vertex labels and are hence labeled with the empty string ϵ.

Optimizing the expanded graph

One issue with the above construction is that the number of recombination edges for a given potential recombination can be $O(|\mathcal{H}|^2)$ in the worst case. This occurs because we maintain $|haps(v)|$ copies of each vertex $v \in V$. For every edge $(u, v) \in E$ allowing a recombination, we add $O(|haps(u)| \cdot |haps(v)|)$ edges to the edge set E_E. Because both $|haps(u)|$ and $|haps(v)|$ can be at most $|\mathcal{H}|$, any potential recombination can result in $O(|\mathcal{H}|^2)$ recombination edges in the worst case. We observe this issue in practice as well. An improvement is to represent a recombination by having an intermediate vertex w_e that represents the edge $e \in E$, allowing for the recombination. We then create an edge to w_e from every vertex in a haplotype path that the recombination would start from, and edges from w_e to every vertex in a haplotype path to which the recombination would lead (Fig 2C). More formally, the modified vertex set becomes

$$V_E = \{s\} \cup \{t\} \cup \{(h_j[i])^j \mid 1 \le j \le |\mathcal{H}|,\ 1 \le i \le |h_j|\} \cup \{w_e \mid e \in E\}. \quad (8)$$

We also replace Equation 5 in the construction of E_E with the following:

$$\bigcup_{\substack{1 \le j, j' \le |\mathcal{H}|,\\ 1 \le i \le |h_j|,\ 1 \le y \le |h_{j'}|,\\ e=(h_j[i], h_{j'}[y]) \in E,\\ i=|h_j| \text{ or } h_j[i+1] \ne h_{j'}[y]}} \{((h_j[i])^j, w_e, c/2), (w_e, (h_{j'}[y])^{j'}, c/2)\}. \quad (9)$$

We now call these edges recombination edges. After creating the edges in E_E, we delete any w_e vertex that is isolated in G_E. Finally, for any remaining w_e vertices, we define $\sigma_E(w_e) = \epsilon$. Observe that the above modification allows for the same set of useful recombinations as our initial expanded graph construction. However, per potential useful recombination, the number of edges remains $O(|\mathcal{H}|)$ rather than $O(|\mathcal{H}|^2)$. Before giving the integer programming solutions, we require one additional definition.

Definition 1 (Hits). *For a string $r \in \mathcal{S}$, assuming $\max_{u \in V_E} |\sigma_E(u)| < |r|$, a path in G_E, denoted as an ordered edge set $((u, v), (v, w), (w, x) \ldots, (y, z))$, matches r if $r = \sigma_E(u)' \circ \sigma_E(v) \circ \sigma_E(w) \circ \sigma_E(x) \circ \sigma_E(y) \circ \sigma_E(z)'$, where $\sigma_E(u)'$ is a suffix of $\sigma_E(u)$ and $\sigma_E(z)'$ is a prefix of $\sigma_E(z)$. We use hits(r) to represent the set of paths matching string r in G_E.*

Integer linear programming solution

We assume that the maximum length of any vertex label is upper bounded by the length of any string in $\mathcal{S}$; that is, $\max_{u \in V_E} |\sigma_E(u)| < \min_{r \in \mathcal{S}} |r|$. This condition can be easily enforced in the input graph by adjusting the lengths of vertex labels, for example, by splitting a vertex with a long label into two, while ensuring that the graph's topology is preserved. We assume $\min_{r \in \mathcal{S}} |r| > 1$.

The basis for our solution is to find an *st*-flow with a flow of one through the expanded graph G_E. Our integer programs will utilize binary decision variable x_{uv} for each edge. The variable x_{uv} will take the value one if edge $(u, v) \in E_E$ is part of the solution flow and zero otherwise. Because these are binary variables, the flow will always be a path. From the solution path in G_E, it is straight forward to recover the corresponding inferred path $\mathcal{P}$. We use binary decision variable z_r for each string $r \in \mathcal{S}$ such that z_r will take the value one if the solution flow includes a subpath from $hits(r)$. We also use variable $z_{r\omega}$ for each $\omega \in hits(r)$, $r \in \mathcal{S}$.

Letting $weight(u, v)$ denote the weight of an edge $(u, v) \in E_E$, our integer linear programming (ILP) formulation is as follows:

$$\min \sum_{(u,v) \in E_E} weight(u, v) \cdot x_{uv} + \sum_{r \in \mathcal{S}} (1 - z_r), \quad (10)$$

subject to

$$\sum_{v \in \mathcal{N}^+(u)} x_{uv} - \sum_{v \in \mathcal{N}^-(u)} x_{vu} = \begin{cases} 1 & \text{if } u = s, \\ -1 & \text{if } u = t, \\ 0 & \text{otherwise,} \end{cases} \quad \forall u \in V_E, \quad (11)$$

$$\sum_{(u,v) \in \omega} x_{uv} \ge |\omega| \cdot z_{r\omega}, \quad z_{r\omega} \in \{0, 1\}, \quad \forall \omega \in hits(r), \forall r \in \mathcal{S}, \quad (12)$$

$$\sum_{\omega \in hits(r)} z_{r\omega} = z_r, \quad z_r \in \{0, 1\}, \quad \forall r \in \mathcal{S}, \quad (13)$$

$$x_{uv} \in \{0, 1\}, \quad \forall (u, v) \in E_E \quad (14)$$

In the ILP formulation, the Equation 10 models $Cost(\mathcal{P})$. The summation over $weight(u, v) \cdot x_{uv}$ imposes penalty c for each recombination. This is because of the two $c/2$ weighted recombination edges that must traversed when the path switches between haplotype paths in G_E (Fig. 2C). In the second summation, the term $(1 - z_r)$ adds a penalty of one to the objective for every $r \in \mathcal{S}$, where $\bar{\chi}(r, \sigma(\mathcal{P})) = 1$. Constraint 11 enforces flow conservation, allowing a unit flow from the source vertex s to the sink vertex t, ensuring that the ILP formulation selects a single path in the expanded graph.

To explain the function of Constraint 12, termed as linear string-hit constraint, and Constraint 13, observe that in an optimal solution, whenever possible the variable z_r is set to one. This is because the term $(1 - z_r)$ in the objective function adds a penalty of zero whenever $z_r = 1$. However, this is only possible when $z_{r\omega}$ is equal to one for some $\omega \in hits(r)$. This, in turn, is only possible if $\sum_{(u,v) \in \omega} x_{uv} = |\omega|$, meaning r occurs as a substring in the inferred path. Also note that at most one $z_{r\omega}$ variable can equal one in Constraint 13. Other $z_{r\omega'}$ variables, where $\omega, \omega' \in hits(r)$ and $\omega \ne \omega'$, can have a value of zero, even if $\sum_{(u,v) \in \omega'} x_{uv} = |\omega'|$, justifying the use of equality in Constraint 13.

A weakness of the proposed ILP formulation is that the number of string-hit constraints equals the total number of string matches, that is, $\sum_{r \in \mathcal{S}} hits(r)$. We design another formulation with quadratic constraints in which fewer constraints are needed.

Integer quadratic programming solution

In our integer quadratic programming (IQP) formulation, Equation 10, and Constraints 11, 13, and 14 remain unchanged from the ILP formulation. Constraints in Equation 12 are replaced by quadratic constraints defined as

$$\sum_{w \in hits(r)} (1 - |\omega| + \sum_{(u,v) \in \omega} x_{uv}) \cdot z_{r\omega} = z_r, \quad \forall r \in \mathcal{S}, \quad (15)$$

which is the quadratic string-hit constraint. Again, because of Constraint 13 at most one $z_{r\omega}$ variable can be one. The expression $1 - |\omega| + \sum_{(u,v) \in \omega} x_{uv}$ sums to one when the subpath ω is contained in the flow. In this case z_r will take the value one, and no penalty is paid in the objective. Conversely, if some of the edges for ω are not in the flow, the expression will sum to zero or less. If this is the case for each $\omega \in hits(r)$, then Constraint 15 can only be satisfied by setting $z_r = 0$ and $z_{r\omega} = 0$ for each $\omega \in hits(r)$. Because $z_r = 0$, a penalty is paid in the objective. The total number of quadratic string-hit constraints is $|\mathcal{S}|$. In our experiments, we observe that IQP formulation solves the problem faster, albeit while requiring more memory.

As a further improvement, we relax the variables x_{uv} for all $(u, v) \in E_E$ to continuous values $x_{uv} \in [0, 1]$ in Constraint 14, following Lemma 2.

Lemma 2. *An optimal solution ϕ_{cont} to the IQP (or ILP) with relaxed Constraint 14 where variables x_{uv} lie within the continuous interval* [0, 1] *can be transformed in polynomial time to an optimal solution ϕ satisfying $x_{uv} \in \{0, 1\}$ for all $(u, v) \in E_E$.*

Proof. First, observe that $z_r = 1$ if and only if all edges in some $\omega \in hits(r)$ have their corresponding variables set to one. This follows from Constraints 12 and 15, and the fact that at most one $z_{r\omega}$ can be one for a given r, by Constraint 13.

If $z_r = 0$ for all $r \in \mathcal{S}$ in ϕ_{cont}, then ϕ can be trivially obtained as a single haplotype path in G_E without recombination penalties. In such a case, all edge variables are assigned either zero or one.

For the remaining cases, we introduce the following terms:

- $\omega \in hits(r)$ is a *used hit-subpath* if $z_{r\omega} = 1$.
- A flow between vertices u and v can be decomposed into uv-paths each assigned some positive flow and called *flow subpaths*.
- ω is the *first used hit-subpath* if there is a flow subpath from vertex s to the first vertex of ω without passing through another used hit-subpath.
- ω is the *last used hit-subpath* if there is a flow subpath from the last vertex of ω to vertex t without passing through another used hit-subpath.
- ω and ω' are *consecutive used hit-subpaths* if there is a flow subpath between them without passing through a third used hit-subpath, where $\omega' \neq \omega$ and $\omega' \in hits(r)$.

Now, if $z_r = 1$ in ϕ_{cont} for some $r \in \mathcal{S}$, there exists a used hit-subpath. We obtain ϕ as following. The flow used to reach the first hit-subpath avoids recombination penalties by following a single haplotype path. Similarly, the flow from the end vertex on the last used hit-subpath to t avoids recombinations penalties by staying on a single haplotype path. Next, consider two consecutive used hit-subpaths ω and ω', with u and v as their respective end and start vertices. If u and v are on different haplotype paths, any flow subpaths between u and v must minimize the recombination penalty. The same minimum recombination cost can be achieved by replacing the potentially multiple fractional flow subpaths with a single path that incurs the same recombination penalty. We can select any flow subpath from u to v and assign its edge variables to one. Edge variables on edges used on the flow from u to v and not on this selected path are set to zero.

Results

Implementation details

We implemented our ILP and IQP solutions in C++ using Gurobi (v11.0.2) solver. We call our software as PHI. The user can provide a pangenome reference as either a graph (GFA format) or as a list of phased variants (VCF format). Given short-read or long-read sequencing data of either a haploid or a homozygous genome, PHI outputs the haplotype sequence associated with the optimal inferred path from the graph in FASTA format.

Given a set of reads, we compute (w, k) window minimizers (Roberts et al. 2004) for identifying our *hits* (Definition 1). By default, we use $w = 25$ and $k = 31$. This choice of k strikes a balance, as longer k-mers have a higher likelihood of including sequencing errors, whereas shorter k-mers can increase the number of false-positive matches. The set of window minimizers corresponds to the input set of strings $\mathcal{S}$ in Problem 1. In practice, most minimizers will have a match with two or more haplotypes. We expect that a small fraction of the minimizers will match to a single haplotype owing to haplotype-specific variation. Such minimizers will help steer the optimization toward identifying the correct haplotype paths.

Computing minimizer matches between two strings is faster than computing minimizer matches on a pangenome graph. For this reason, we find minimizer matches between reads and the sequences spelled by all the haplotype paths in the graph. This means $hits(r)$ includes only those subpaths that are completely contained in some haplotype path in G_E (Definition 1). This restriction to $hits(r)$ also prevents us from needing to perform the additional edge splitting step described in the Methods. In PHI, we set the default value of recombination penalty parameter c as 100. We discuss the effect of this parameter later. We ran all our experiments on AMD EPYC 7763 processors with 512 GB RAM. We used 32 threads in all experiments.

Data sets

We evaluated our algorithm by estimating MHC sequences of five haplotypes (APD, DBB, MANN, QBL, SSTO) from homozygous human cell lines. Recently, Houwaart et al. (2023) and Scholz (2024) published complete assemblies of these MHC sequences using long- and short-read sequencing. The average length of these assemblies is 4.99 Mbp. We downloaded the five short-read sequencing data sets available from this study. To evaluate our algorithm using varying sequencing coverage, we downsampled each short-read data set to obtain coverage of 0.1×, 0.5×, 1×, 2×, 5×, and 10×. We also used the full data sets for evaluation (coverage, 12.9× – 18.2×). We used the complete assemblies of five MHC haplotypes as ground truth to evaluate the accuracy of our estimated sequences. To quantify the accuracy, we measured edit distance between each estimated sequence and the corresponding ground-truth sequence.

We built a haplotype-resolved pangenome graph of 49 complete MHC sequences (Li 2022) using Minigraph-Cactus (Hickey et al. 2024). These sequences were extracted from phased assemblies of 24 diploid human samples (Liao et al. 2023) and the CHM13 reference (Nurk et al. 2022). Using Minigraph-Cactus, we obtained the pangenome reference in a VCF format file. We subjected this file to further simplification steps (https://github.com/eblerjana/genotyping-pipelines/tree/main/prepare-vcf-MC) to ensure compatibility with various tools. We show sequence similarity statistics between the complete MHC assemblies of five haplotypes (APD, DBB, MANN, QBL, SSTO) and the 49 pangenome reference haplotypes in Supplemental Table S1.

Other methods

We compared PHI with two existing pangenome-based genotyping tools: VG (v1.60) (Sirén et al. 2024) and PanGenie (v3.1) (Ebler et al. 2022). VG supports sampling of relevant haplotypes from a pangenome graph by comparing k-mer counts in the reads and k-mers of a reference haplotype. The selection of haplotypes is done locally in fixed-length nonoverlapping blocks. Recombinations may be introduced to create contiguous haplotypes across the blocks. The number of samples can be specified by the user. Accordingly, VG's haplotype sampling feature can be adapted for haplotype sequence estimation by simply setting the number of desired samples to one. Next, PanGenie supports short-read genotyping using a haplotype-resolved pangenome graph. PanGenie uses a hidden Markov model, which is similar to the standard Li and Stephens model (Li and Stephens 2003). PanGenie compares k-mer counts in the reads with the k-mers present in the graph to compute genotype likelihoods. PanGenie

exhibited better genotyping accuracy and speed than other genotyping tools (Ebler et al. 2022). Our sequencing data sets are derived from homozygous cell lines; therefore, we ignored the heterozygous genotype calls made by PanGenie (Supplemental Table S3). We incorporated PanGenie's predicted genotypes in the reference sequence to obtain the haplotype sequence. We list our commands to run PHI, VG, and PanGenie in Supplemental Table S2.

Genotyping performance

We evaluated the PHI, VG, and PanGenie methods in their ability to infer the MHC sequences from short-read data sets of varying coverage (see Fig. 3A–E). Using low-coverage data sets (0.1×–2×), PHI exhibits significantly higher accuracy. The VG and PanGenie methods may not be suitable for low-coverage sequencing. For example, the distribution of *k*-mer counts at low coverage can be unreliable. Distinguishing *k*-mers originating from unique versus repetitive regions, as required by PanGenie and VG, is also challenging at low-coverage. Using coverage of ≥5×, the results of VG and PHI are comparable. PanGenie also produces comparable results using full data sets. We note that the IQP approach used in PHI requires more time and memory compared with the methods used in VG and PanGenie. PHI used up to 1.5 h and 137 GB RAM in a single experiment. In contrast, VG and PanGenie required <5 min and <50 GB memory. It may be possible to optimize PHI by incorporating efficient heuristics. We show detailed performance statistics for PHI, including its runtime and memory usage in Supplemental Table S4.

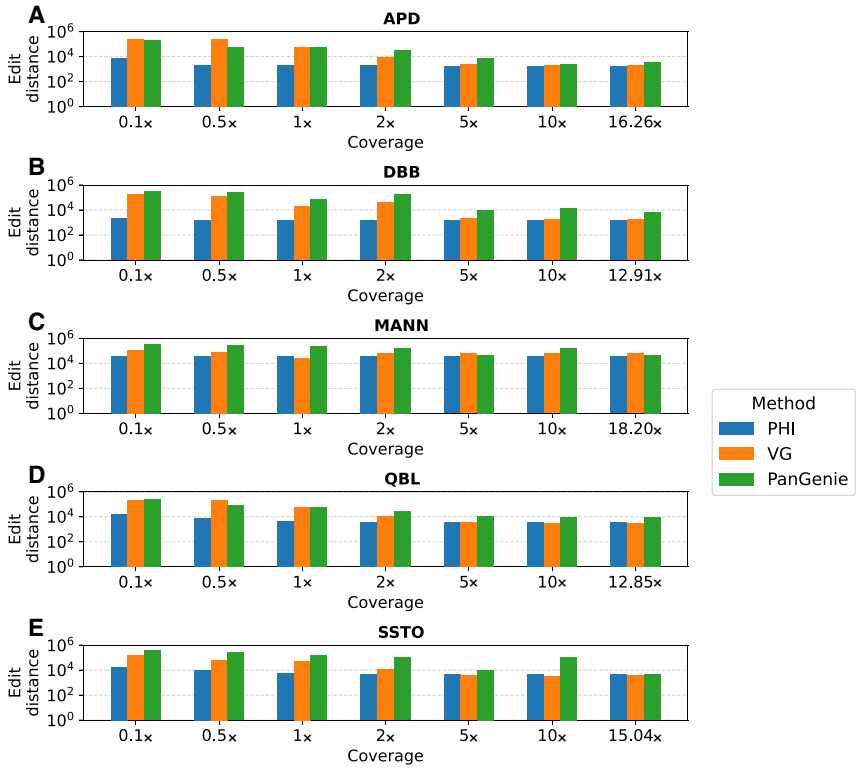

Figure 3. Accuracy of the haplotype sequences estimated by PHI, VG, and PanGenie. We used short-read data sets from MHC sequences of five haplotypes: (*A*) APD, (*B*) DBB, (*C*) MANN, (*D*) QBL, (*E*) SSTO. The *x*-axes indicate the coverage of short-read data. The *y*-axes indicate the edit distance between the estimate haplotype sequence and the ground-truth sequence on a logarithmic scale.

Effect of parameter *c*

We further explored the effect of parameter *c* in PHI. The user-specified parameter *c* determines the cost per recombination event. Having a good balance between allowing too many recombination events versus not allowing recombinations is important. By default, PHI uses $c = 100$. We arrived at this default value by testing three different settings: $c = 10$, $c = 100$, and $c = 1000$. For this experiment, we used the five full short-read data sets as input to PHI. We present the edit distances, counts of recombination events, and the lengths of the estimated sequences in Table 1. We observe that using $c = 100$ results in the lowest edit distances consistently across all five haplotypes. In the following, we argue why having $c = 10$ or $c = 1000$ results in less accurate outputs.

A large value of *c* such as 1000 can be interpreted as significantly limiting the flexibility to recombine, thus forcing the

Table 1. Effect of modifying parameter *c* on PHI's output

	Edit distance			No. of recombinations			Haplotype length (Mbp)			
							Estimated sequence by PHI			
Haplotype	$c = 10$	$c = 10^2$	$c = 10^3$	$c = 10$	$c = 10^2$	$c = 10^3$	$c = 10$	$c = 10^2$	$c = 10^3$	Ground truth
APD	144,835	1810	9994	36	10	4	5.07	4.93	4.92	4.93
DBB	67,290	1377	2111	13	4	2	5.11	5.05	5.05	5.05
MANN	131,564	35,940	38,872	37	14	5	5.15	5.04	5.04	5.03
QBL	125,655	3343	13,034	46	17	4	5.03	4.90	4.90	4.90
SSTO	125,770	4637	13,600	68	24	5	5.17	5.04	5.05	5.05

The table presents output statistics obtained by running PHI using three different choices of the recombination penalty parameter *c*: 10, 100, and 1000. We calculated (1) accuracy, that is, edit distance values between the estimated haplotype sequences and the ground-truth assemblies; (2) count of recombinations in the estimated haplotype sequences, and (3) lengths of the estimated sequences. The last column shows the length of the ground-truth sequences.

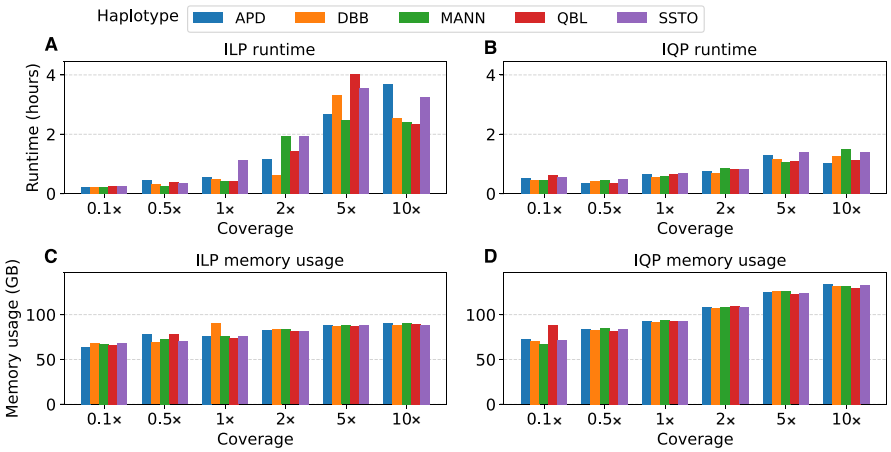

Figure 4. Performance comparison between the ILP and IQP solutions implemented in PHI. We compared their runtime (*A*,*B*) and memory usage (*C*,*D*) using short-read sequencing data sets sampled from five haplotypes.

optimal output to use fewer recombinations than required. Accordingly, we find that the count of recombinations using $c = 1000$ reduces using all five data sets (Table 1). A large value of c is appropriate in those scenarios in which the target haplotype is known to closely resemble a haplotype included in the pangenome graph. Otherwise, it will result in low output accuracy, as we see in this experiment. On the other hand, we find that a very low value of c such as 10 allows an excessive number of recombination events. In such situations, the algorithm greedily chooses paths in the graph to maximize minimizer matches. This greedy choice affects the accuracy when there are false-positive minimizer matches owing to repeats and sequencing errors. Because of this reason, we find that having $c = 10$ results in longer output sequences than expected. Overall, this experiment also justifies the design of our problem formulation and the importance of maintaining a balance between the count of minimizer matches and the count of recombination events.

Effect of our optimizations

In PHI, we implemented both ILP-based and IQP-based solutions to solve the optimization problem. Using either solution, Gurobi solves Problem 1 to optimality. We benchmarked our ILP and IQP solutions to compare their runtime and memory usage (see Fig. 4A–D). On low-coverage data sets (0.1×–1×), the runtimes are comparable. At higher coverage, the IQP solution runs faster, which is likely owing to fewer string-hit constraints used (Methods), although it requires approximately 1.5 times more memory. This may be because Gurobi requires additional storage to handle quadratic constraints. Accordingly, while using PHI, the user can choose between ILP and IQP using a command line argument based on the available memory. If no choice is provided, the IQP solution is used by default. We also evaluated the advantage of relaxing edge variables to continuous values (Lemma 2) by comparing it to another version of our code in which we set the edge variables to be discrete. Relaxation of variables deceases runtime of the IQP solution by a factor of 1.6 on average (Supplemental Fig. S2). Not much effect on the runtime is observed in the ILP solution (Supplemental Fig. S3).

Impact of graph expansion with the addition of more genomes

We evaluated the impact of pangenome graph expansion on PHI's genotyping accuracy as well as runtime. To do this, we created five versions of our pangenome graph, each containing an increasing number of reference haplotypes, added progressively. The first graph comprises a single diploid sample (chosen randomly from 24 diploid samples) plus the CHM13 reference; therefore, it has three reference haplotypes in total. The second graph includes two more diploid samples (chosen randomly from the remaining 23); therefore, it has seven reference haplotypes in total. Similarly, the third, fourth, and fifth graphs contain 13, 25, and 49 reference haplotypes, respectively. The fifth graph is equivalent to the graph used in previous experiments as well. This results in five different graphs that have three, 7, 13, 25, and 49 reference haplotypes, respectively.

We repeated our experiments with full short-read data sets using these five graphs and present the results in Figure 5. We observe that edit distances between the estimated sequences and the ground-truth sequences decrease with the increasing number of reference haplotypes. This is expected because more haplotypes are available to choose from when we compute our inferred path in the graph. We also observe an increase in runtime and memory usage. Runtime appears to increase superlinearly and memory appears to increase linearly with the number of reference haplotypes. This is because the size of expanded graph and the number of minimizer matches increase, leading to more variables and constraints in our integer program.

Discussion

Genotyping using pangenome graphs is equivalent to finding a walk in the graph that contains the sample's variants (Paten et al. 2017). If the sample is diploid, this becomes equivalent to finding a pair of paths. Drawing inspiration from this idea, we proposed a rigorous framework to infer a path through the graph, such

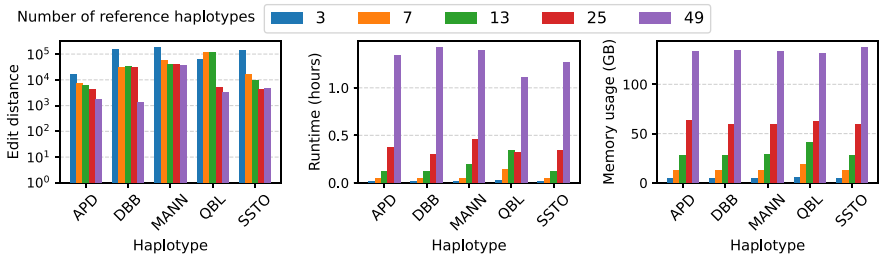

Figure 5. Assessment of PHI's performance with the increasing number of genomes in pangenome graph. The *left* figure shows the accuracy in terms of edit distance between the output sequences and ground-truth sequences. The *middle* and *right* figures show the runtime and memory usage, respectively.

that the sequence spelled by the path is consistent with the sequencing data in terms of the shared *k*-mers between them, while permitting a limited number of recombinations in the path, each incurring a fixed penalty. This optimization problem requires considering all possible paths in the graph. We proved that this problem is NP-Hard and subsequently gave efficient integer programming solutions. As part of our methodology, we introduced the expanded graph data structure on which we could compute an appropriate *st*-flow of one. Experimental results demonstrate the advantage of the proposed ILP/IQP approaches for accurate genome inference, especially with low-coverage data (coverage, 0.1×–1×). Thus, our algorithm can facilitate affordable genotyping and association studies of complex and repeat-rich regions of the genome.

Although our approach is currently tailored to haploid samples, it could generalize to diploid samples. This may be accomplished by finding an *st*-flow of two through the expanded graph and modifying some constraints. How well this approach genotypes and phases the genome would be interesting to explore. Another limitation of this work is that we do not capture uncertainty. For example, there may be multiple inferred paths with minimum cost. Other future directions to explore include obtaining a better runtime bound for the bounded recombination case. This could be achieved by modifying Problem 1 to have an upper limit on the count of recombinations, effectively limiting the number of recombinations in the inferred haplotype. This problem version is yet to be explored.

Our problem formulation employs a simple cost model for penalizing recombination events by applying a fixed penalty for each event. In PHI, we set this parameter to 100 by default based on our empirical observations on the MHC pangenome graph. But on other pangenome graphs, users may need to tune the parameter using the procedure described in the Results (see section "Effect of parameter *c*").

Further refinement of the proposed optimization framework is necessary to align it with established statistical models in population genetics and evolutionary biology. Rosen et al. (2017) developed an efficient algorithm for estimating haplotype likelihoods on graphs, extending the Li–Stephens model to graph-based representations. Lastly, pangenome graphs are expected to grow in the number of genomes; therefore, scaling the current approach to a large number of haplotype paths is important. We leave these extensions to future work.

Software availability

The software PHI is available at GitHub (https://github.com/at-cg/PHI) and as Supplemental Code. The scripts to reproduce the results are available at GitHub (https://github.com/at-cg/PHI/tree/master/data) and as Supplemental Scripts.

Competing interest statement

The authors declare no competing interests.

Acknowledgments

This research is funded in part by the DBT/Wellcome Trust India Alliance Fellowship (grant no. IA/I/23/2/506979), the Intel India Research Fellowship, the National Institutes of Health of the USA (NIH-NIAID U01 AI090905), and the Jürgen Manchot Foundation. We utilized computing resources available at the Indian Institute of Science and the U.S. National Energy Research Scientific Computing Center.

Author contributions: All authors contributed to the development of the methods and proofs. G.C. implemented the software and conducted the experiments.

References

Baaijens JA, Bonizzoni P, Boucher C, Della Vedova G, Pirola Y, Rizzi R, Sirén J. 2022. Computational graph pangenomics: a tutorial on data structures and their applications. *Nat Comput* **21:** 81–108. doi:10.1007/s11047-022-09882-6

Bradbury PJ, Casstevens T, Jensen SE, Johnson L, Miller Z, Monier B, Romay M, Song B, Buckler ES. 2022. The practical haplotype graph, a platform for storing and using pangenomes for imputation. *Bioinformatics* **38:** 3698–3702. doi:10.1093/bioinformatics/btac410

Chandra G, Gibney D, Jain C. 2024. Haplotype-aware sequence alignment to pangenome graphs. *Genome Res* **34:** 1265–1275. doi:10.1101/gr.279143.124

Computational Pan-Genomics Consortium. 2018. Computational pan-genomics: status, promises and challenges. *Brief Bioinformatics* **19:** 118–135. doi:10.1093/bib/bbw089

Davies RW, Kucka M, Su D, Shi S, Flanagan M, Cunniff CM, Chan YF, Myers S. 2021. Rapid genotype imputation from sequence with reference panels. *Nat Genet* **53:** 1104–1111. doi:10.1038/s41588-021-00877-0

Dilthey AT. 2021. State-of-the-art genome inference in the human MHC. *Int J Biochem Cell Biol* **131:** 105882. doi:10.1016/j.biocel.2020.105882

Dilthey A, Cox C, Iqbal Z, Nelson MR, McVean G. 2015. Improved genome inference in the MHC using a population reference graph. *Nat Genet* **47:** 682–688. doi:10.1038/ng.3257

Ebert P, Audano PA, Zhu Q, Rodriguez-Martin B, Porubsky D, Bonder MJ, Sulovari A, Ebler J, Zhou W, Serra Mari R, et al. 2021. Haplotype-resolved diverse human genomes and integrated analysis of structural variation. *Science* **372:** eabf7117. doi:10.1126/science.abf7117

Ebler J, Ebert P, Clarke WE, Rausch T, Audano PA, Houwaart T, Mao Y, Korbel JO, Eichler EE, Zody MC, et al. 2022. Pangenome-based genome inference allows efficient and accurate genotyping across a wide spectrum of variant classes. *Nat Genet* **54:** 518–525. doi:10.1038/s41588-022-01043-w

Eggertsson HP, Jonsson H, Kristmundsdottir S, Hjartarson E, Kehr B, Masson G, Zink F, Hjorleifsson KE, Jonasdottir A, Jonasdottir A, et al. 2017. Graphtyper enables population-scale genotyping using pangenome graphs. *Nat Genet* **49:** 1654–1660. doi:10.1038/ng.3964

Gao Y, Yang X, Chen H, Tan X, Yang Z, Deng L, Wang B, Kong S, Li S, Cui Y, et al. 2023. A pangenome reference of 36 Chinese populations. *Nature* **619:** 112–121. doi:10.1038/s41586-023-06173-7

Grytten I, Dagestad Rand K, Sandve GK. 2022. KAGE: fast alignment-free graph-based genotyping of SNPs and short indels. *Genome Biol* **23:** 209. doi:10.1186/s13059-022-02771-2

Harris L, McDonagh EM, Zhang X, Fawcett K, Foreman A, Daneck P, Sergouniotis PI, Parkinson H, Mazzarotto F, Inouye M, et al. 2024. Genome-wide association testing beyond SNPs. *Nature Reviews Genetics* **26:** 156–170. doi:10.1038/s41576-024-00778-y

Hickey G, Heller D, Monlong J, Sibbesen JA, Sirén J, Eizenga J, Dawson ET, Garrison E, Novak AM, Paten B. 2020. Genotyping structural variants in pangenome graphs using the vg toolkit. *Genome Biol* **21:** 35. doi:10.1186/s13059-020-1941-7

Hickey G, Monlong J, Ebler J, Novak AM, Eizenga JM, Gao Y, Marschall T, Li H, Paten B. 2024. Pangenome graph construction from genome alignments with Minigraph-Cactus. *Nat Biotechnol* **42:** 663–673. doi:10.1038/s41587-023-01793-w

Houwaart T, Scholz S, Pollock NR, Palmer WH, Kichula KM, Strelow D, Le DB, Belick D, Hülse L, Lautwein T, et al. 2023. Complete sequences of six major histocompatibility complex haplotypes, including all the major MHC class II structures. *HLA* **102:** 28–43. doi:10.1111/tan.15020

Jain C, Tavakoli N, Aluru S. 2021. A variant selection framework for genome graphs. *Bioinformatics* **37:** i460–i467. doi:10.1093/bioinformatics/btab302

Letcher B, Hunt M, Iqbal Z. 2021. Gramtools enables multiscale variation analysis with genome graphs. *Genome Biol* **22:** 259. doi:10.1186/s13059-021-02474-0

Li H. 2022. Sample graphs and sequences for testing sequence-to-graph alignment. https://zenodo.org/records/6617246

Li N, Stephens M. 2003. Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics* **165:** 2213–2233. doi:10.1093/genetics/165.4.2213

Li JH, Mazur CA, Berisa T, Pickrell JK. 2021. Low-pass sequencing increases the power of GWAS and decreases measurement error of polygenic risk

scores compared to genotyping arrays. *Genome Res* **31:** 529–537. doi:10.1101/gr.266486.120
Liao WW, Asri M, Ebler J, Doerr D, Haukness M, Hickey G, Lu S, Lucas JK, Monlong J, Abel HJ, et al. 2023. A draft human pangenome reference. *Nature* **617:** 312–324. doi:10.1038/s41586-023-05896-x
Mahmoud M, Gobet N, Cruz-Dávalos DI, Mounier N, Dessimoz C, Sedlazeck FJ. 2019. Structural variant calling: the long and the short of it. *Genome Biol* **20:** 246. doi:10.1186/s13059-019-1828-7
Martin AR, Atkinson EG, Chapman SB, Stevenson A, Stroud RE, Abebe T, Akena D, Alemayehu M, Ashaba FK, Atwoli L, et al. 2021. Low-coverage sequencing cost-effectively detects known and novel variation in underrepresented populations. *Am J Hum Genet* **108:** 656–668. doi:10.1016/j.ajhg.2021.03.012
Mun T, Vaddadi NSK, Langmead B. 2023. Pangenomic genotyping with the marker array. *Algorithms Mol Biol* **18:** 2. doi:10.1186/s13015-023-00225-3
Mustafa H, Karasikov M, Mansouri Ghiasi N, Rätsch G, Kahles A. 2024. Label-guided seed-chain-extend alignment on annotated de Bruijn graphs. *Bioinformatics* **40:** i337–i346. doi:10.1093/bioinformatics/btae226
Nurk S, Koren S, Rhie A, Rautiainen M, Bzikadze AV, Mikheenko A, Vollger MR, Altemose N, Uralsky L, Gershman A, et al. 2022. The complete sequence of a human genome. *Science* **376:** 44–53. doi:10.1126/science.abj6987
Paten B, Novak AM, Eizenga JM, Garrison E. 2017. Genome graphs and the evolution of genome inference. *Genome Res* **27:** 665–676. doi:10.1101/gr.214155.116
Petes TD. 2001. Meiotic recombination hot spots and cold spots. *Nat Rev Genet* **2:** 360–369. doi:10.1038/35072078
Pritt J, Chen NC, Langmead B. 2018. FORGe: prioritizing variants for graph genomes. *Genome Biol* **19:** 220. doi:10.1186/s13059-018-1595-x
Roberts M, Hayes W, Hunt BR, Mount SM, Yorke JA. 2004. Reducing storage requirements for biological sequence comparison. *Bioinformatics* **20:** 3363–3369. doi:10.1093/bioinformatics/bth408
Rosen Y, Eizenga J, Paten B. 2017. Modelling haplotypes with respect to reference cohort variation graphs. *Bioinformatics* **33:** i118–i123. doi:10.1093/bioinformatics/btx236
Rubinacci S, Ribeiro DM, Hofmeister RJ, Delaneau O. 2021. Efficient phasing and imputation of low-coverage sequencing data using large reference panels. *Nat Genet* **53:** 120–126. doi:10.1038/s41588-020-00756-0
Scholz S. 2024. Complete sequences of six major histocompatibility complex haplotypes rev2. https://zenodo.org/records/13889312
Sibbesen JA, Maretty L, Danish Pan-Genome Consortium, Krogh A. 2018. Accurate genotyping across variant classes and lengths using variant graphs. *Nat Genet* **50:** 1054–1059. doi:10.1038/s41588-018-0145-5
Sirén J, Monlong J, Chang X, Novak AM, Eizenga JM, Markello C, Sibbesen JA, Hickey G, Chang PC, Carroll A, et al. 2021. Pangenomics enables genotyping of known structural variants in 5202 diverse genomes. *Science* **374:** abg8871. doi:10.1126/science.abg8871
Sirén J, Eskandar P, Ungaro MT, Hickey G, Eizenga JM, Novak AM, Chang X, Chang PC, Kolmogorov M, Carroll A, et al. 2024. Personalized pangenome references. *Nat Methods* **21:** 2017–2023. doi:10.1038/s41592-024-02407-2
Smith TP, Bickhart DM, Boichard D, Chamberlain AJ, Djikeng A, Jiang Y, Low WY, Pausch H, Demyda-Peyrás S, Prendergast J, et al. 2023. The Bovine Pangenome Consortium: democratizing production and accessibility of genome assemblies for global cattle breeds and other bovine species. *Genome Biol* **24:** 139. doi:10.1186/s13059-023-02975-0
Tavakoli N, Gibney D, Aluru S. 2022. Haplotype-aware variant selection for genome graphs. In *Proceedings of the 13th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics (BCB '22)*, Article 51, pp. 1–9. Association for Computing Machinery, New York. doi:10.1145/3535508.3545556
Vaddadi NSK, Mun T, Langmead B. 2023. Minimizing reference bias: the impute-first approach for personalized genome analysis. In *Proceedings of the 14th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics (BCB '23)*, Article 81, p. 1. Association for Computing Machinery, New York. doi:10.1145/3584371.3613034

Received February 16, 2025; accepted in revised form July 29, 2025.