Haplotype-aware Sequence-to-Graph Alignment

Ghanshyam Chandra and Chirag Jain

Department of Computational and Data Sciences, Indian Institute of Science, Bangalore KA 560012, India
{ghanshyamc,chirag}@iisc.ac.in

Abstract. Modern pangenome graphs are built using high-quality phased haplotype sequences such that each haplotype sequence corresponds to a path in the graph. Prioritizing the alignment of reads to these paths improves genotyping accuracy (Siren *et al.*, Science 2021). However, rigorous formulations for sequence-to-graph chaining and alignment do not consider the haplotype paths. As a result, the search space increases combinatorially as more variants are augmented in the graph. This limitation affects the effectiveness of the algorithms. In this paper, we propose novel formulations and provably good algorithms for haplotype-aware pattern matching of sequences to directed acyclic graphs (DAGs). Our work considers both sequence-to-DAG chaining and sequence-to-DAG alignment problems. Drawing inspiration from the commonly used models for genotype imputation, we assume that a query sequence is an imperfect mosaic of the reference haplotypes. Accordingly, our formulations extend previous chaining and alignment formulations by introducing a recombination penalty for a haplotype switch. First, we solve the haplotype-aware sequence-to-DAG alignment in $O(|Q||E||\mathcal{H}|)$ time where Q is the query sequence, E is the set of edges, and $\mathcal{H}$ is the set of haplotypes represented in the graph. Second, we prove that an algorithm significantly faster than $O(|Q||E||\mathcal{H}|)$ is unlikely. Third, we propose a haplotype-aware chaining algorithm that uses $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time, where N is the count of exact matches. As a proof-of-concept, we implemented the chaining algorithm in the Minichain aligner (https://github.com/at-cg/minichain). Using simulated human major histocompatibility complex (MHC) query sequences and a pangenome graph of 60 publicly available MHC haplotypes, we show that the proposed algorithm offers a much better consistency between the ground-truth recombinations and the recombinations in the output chains when compared to a haplotype-agnostic algorithm.

1 Introduction

Pangenome graphs represent the maximum possible diversity that exists in the population [52]. Efforts toward using pangenome graphs have been further catalyzed by the advent of long and accurate reads. The recent progress in long-read technologies and assembly algorithms enables high-quality haplotype-phased assembly of human diploid genomes [7,39,45,56]. Building a pangenome graph directly from phased assemblies instead of variant calls allows a more comprehensive representation of the variation [55]. Accordingly, the recent pangenome graph construction methods use phased assemblies to construct a graph [9,18,14,29,26]. The input haplotypes are represented as paths in the graph.

Pangenome graphs also raise fundamental questions about the tradeoffs between complexity and usability. An arbitrary path in a pangenome graph corresponds to either the original haplotypes or their recombinations. The number of sequences spelled by a graph increases combinatorially with the number of variants. Accordingly, the possibility of computing an incorrect alignment of a pattern increases with the pattern length and the number of variants. This issue has been addressed using different techniques, e.g., by limiting the amount of variation in the graph [21,42], artificially simplifying complex regions [15], or restricting the alignment to either one [36,54,53] or two haplotype paths [4]. A more principled approach to tackle this problem may be to leverage the correlations between two or more genetic variants, i.e. where individuals tend to have same genotype [9,17]. For example, PanGenie is an alignment-free genotyping algorithm that leverages long-range haplotype information inherent in the phased genomes [9].

A pangenome graph is represented either as a directed graph or a directed acyclic graph (DAG), where each vertex is labeled by a sequence. The fundamental problem formulation corresponding to exact(approximate) matching of strings to a graph seeks a path in the graph which spells a string that matches exactly(approximately) with the query string [2,13,22,23,38,34,44]. $O(|V| + |Q||E|)$-time algorithms are known for both exact and approximate matching problems, where Q denotes the query, E denotes the set of edges, and V denotes the

set of vertices. These formulations do not consider the associations between genetic variants and may lead to alignments with spurious recombinations in variant-dense regions of the graph [42]. The existing formulations for co-linear chaining on graphs share the same limitation [6,33,46,30,43]. Chaining on DAGs can be solved in $O(KN \log KN)$ time, where K is the minimum number of paths covering all the vertices and N is the number of exact matches between the query and the DAG [6,30].

In this paper, we propose 'haplotype-aware' formulations for (i) sequence-to-DAG alignment and (ii) sequence-to-DAG chaining problems. Our formulations use the haplotype path information available in modern pangenome graphs. The formulations are inspired from the classic Li-Stephens haplotype copying model [28]. The Li-Stephens model is a probabilistic generative model which assumes that a sampled haplotype is an imperfect mosaic of known haplotypes. Similarly, our haplotype-aware sequence-to-DAG alignment formulation optimizes the number of edits and haplotype switches simultaneously (Figure 1). We give $O(|Q||E||\mathcal{H}|)$ algorithm for solving the problem, where $\mathcal{H}$ denotes the set of haplotypes represented in the pangenome DAG. We further prove that the existence of a significantly faster algorithm than $O(|Q||E||\mathcal{H}|)$ is unlikely under the strong exponential time hypothesis (SETH). We also formulate haplotype-aware co-linear chaining problem. We solve it in $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time, assuming a one-time $O(|E||\mathcal{H}|)$ indexing of the DAG.

We implemented the proposed chaining algorithm in Minichain. We tested Minichain using a pangenome DAG constructed using 60 complete major histocompatibility complex (MHC) sequences [25]. We simulated query MHC haplotypes as mosaics of the 60 haplotypes. We demonstrate that introducing recombination penalty in the formulation leads to a much better consistency between the observed recombination events and the ground truth. We achieved Pearson correlation up to 0.94 between the observed recombination count and the ground-truth recombination count, where as the correlation remained below 0.32 if recombinations are not penalized. Our results suggest that haplotype-aware pattern matching to pangenome graphs can be useful to improve read mapping and variant calling accuracy.

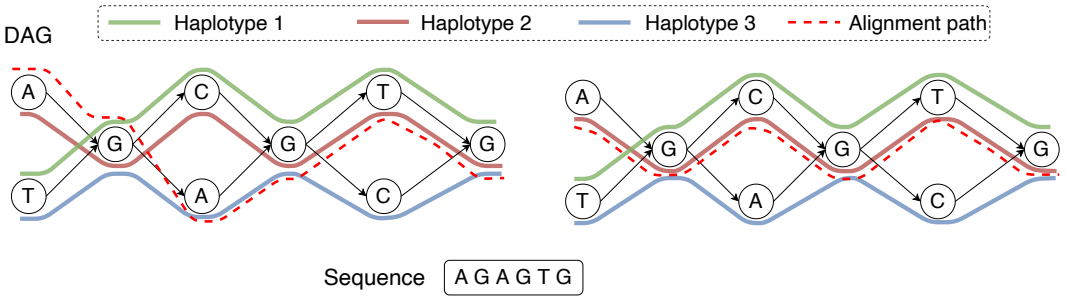


Fig. 1: An example of a pangenome DAG built using three haplotypes. The left figure shows the optimal alignment of the query sequence to the DAG with minimum edit distance. Accordingly, the edit distance is zero and the count of recombinations is two. Next, suppose we use recombination penalty 5 to control recombinations. The right figure shows the new optimal alignment, where there is no recombination because the alignment path is consistent with haplotype 2.

2 Haplotype-aware Sequence-to-DAG Alignment

2.1 Concepts and Notations

Haplotype-aware pangenome DAG. Let $G(V, E, \sigma, \mathcal{H})$ be a DAG representing a pangenome reference. We assume $|E| \geq |V| - 1$. Let Σ denote an alphabet. Function $\sigma : V \mapsto \Sigma$ assigns a character label to each vertex. $\mathcal{H} = \{h_1, h_2, \ldots, h_{|\mathcal{H}|}\}$ denotes the set of haplotype sequences represented in the DAG. Each haplotype sequence can be identified using a path in the DAG. We define $haps(v)$ as $\{i \mid h_i$ includes vertex $v\}$. We assume that $haps(v) \neq \phi$ for all $v \in V$ and $haps(u) \cap haps(v) \neq \phi$ for all edges $(u, v) \in E$. In other words, each vertex and edge must be supported by at least one haplotype. Given a query string $Q \in \Sigma^+$, we will find its exact and approximate matches in the DAG. Let an *alignment path* of length l in the DAG

be denoted as an ordered sequence $(w_1, w_2, \ldots, w_l)$, where each w_i is a pair of the form (v, h) such that $v \in V$, $h \in haps(v)$, and $(w_i.v, w_{i+1}.v) \in E$ for all $i \in [l-1]$. Thus, in our haplotype-aware setting, an alignment path specifies a path in the DAG and the indices of the selected haplotypes along the path. We say that a *recombination* has occurred in between w_i and w_{i+1} if $w_i.h \neq w_{i+1}.h$. Given an alignment path $\mathcal{P}$ in the DAG, we use functions $s(\mathcal{P})$ and $r(\mathcal{P})$ to denote the string spelled by the path and the count of recombinations in the path, respectively.

Hardness result for the Edit Distance problem. Our hardness result (Lemma 3) will build on top of the seminal work of Backurs and Indyk [5] . The Orthogonal Vector (OV) problem asks: given two sets of binary vectors $A \subseteq \{0,1\}^d$ and $B \subseteq \{0,1\}^d$, $|A| = |B| = n$, decide if there is a pair $a \in A$, $b \in B$ such that $a \cdot b = 0$. Assuming SETH holds and $d = \omega(n)$, there is no $\epsilon > 0$ for which an $O(n^{2-\epsilon}\text{poly}(d))$ algorithm can solve the OV problem [57]. This result is frequently used to study the hardness of other problems [13,12,16,19,5]. For any two strings P_1 and P_2, let $\text{EDIT}(P_1, P_2)$ be the edit distance between them. Let us define another distance measure between two strings as following:

$$\text{PATTERN}(P_1, P_2) = \min_{x \text{ is a substring of } P_2} \text{EDIT}(P_1, x)$$

Backurs and Indyk [5] proved hardness of the Edit Distance problem by designing two reductions: (i) OV to PATTERN, and (ii) PATTERN to EDIT. We recall the properties of their first reduction gadget.

Lemma 1 (ref. [5]). *Given an OV instance with sets $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ comprising d-dimensional binary vectors, there exist transformations of sets A and B into two strings S_1 and S_2 respectively over the alphabet $\{0, 1, 2\}$ such that the following properties hold:*

- *Computing* $\text{PATTERN}(S_1, S_2)$ *determines the existence of orthogonal vectors. There exist constants $\beta_1, \beta_2 \in \mathbb{N}$ such that if there are two orthogonal vectors $a \in A, b \in B$, then* $\text{PATTERN}(S_1, S_2) \leq \beta_1 n - \beta_2$*; otherwise,* $\text{PATTERN}(S_1, S_2) = \beta_1 n$.
- *$|S_1|$ is a function of n and d. Similarly, $|S_2|$ is a function of n and d. Both $|S_1|$ and $|S_2|$ are in $O(n \cdot \text{poly}(d))$.*
- *Time to compute strings S_1 and S_2 from sets A and B is in $O(n \cdot \text{poly}(d))$.*

We denote the injective function that generates the gadget string S_1 from set A as f_1. Similarly, injective function f_2 generates the gadget string S_2 from set B. The exact definitions of f_1 and f_2 are available in [5].

2.2 Problem Definitions for Haplotype-Aware Alignment

The input to each of the following problem is a DAG $G(V, E, \sigma, \mathcal{H})$, a query $Q \in \Sigma^+$, and a user-specified threshold $k \in \mathbb{N}$. In each problem, we seek a match of the query in the DAG while controlling the number of recombinations.

– *Problem 1 (Exact matching, limited recombinations):* Determine if there is an alignment path $\mathcal{P}$ in DAG G such that $EDIT(s(\mathcal{P}), Q) = 0$ and $r(\mathcal{P}) < k$.
– *Problem 2 (Approximate matching, no recombination):* Determine if there is an alignment path $\mathcal{P}$ in DAG G such that $EDIT(s(\mathcal{P}), Q) < k$ and $r(\mathcal{P}) = 0$.
– *Problem 3 (Approximate matching, limited recombinations):* Let $c \in \mathbb{R}_{\geq 0}$ be a user-specified parameter. Determine if there is an alignment path $\mathcal{P}$ in DAG G such that $EDIT(s(\mathcal{P}), Q) + c \cdot r(\mathcal{P}) < k$.

In the above set of problems, we have omitted the case with exact matching and no recombination because it is trivially solvable using the standard string to string matching algorithms. In Problem 3, we assume a constant cost per recombination but one can also consider using different cost per locus based on known rates of recombination [41]. Next, we state a generalized version of Problems 1-3.

– *Problem 4 (Generalized version):* Let k, c_1, c_2 be user-specified thresholds, where $k \in \mathbb{N}$ and $c_1, c_2 \in \mathbb{R}_{\geq 0}$. Determine if there is an alignment path $\mathcal{P}$ in DAG G such that $c_1 \cdot EDIT(s(\mathcal{P}), Q) + c_2 \cdot r(\mathcal{P}) < k$.

2.3 Proposed Algorithms and Complexity Analysis

Lemma 2. *Problem 4 can be solved in $O(|Q||E||\mathcal{H}|)$ time.*

Proof. We use dynamic programming. Our approach is similar to the standard sequence-to-DAG alignment algorithm [23] except that we also track recombinations. We add a dummy source vertex v_ϵ with an empty label. It has zero incoming edges and $|V|$ outgoing edges to all $v \in V$. Assume $haps(v_\epsilon) = [|\mathcal{H}|]$. Let $D(i, v, h)$ denote the minimum cost for aligning the (possibly empty) prefix of sequence Q ending at index i against an alignment path which ends at pair (v, h), $v \in V \cup \{v_\epsilon\}, h \in haps(v)$. We also maintain a second table D'' such that $D''(i, v)$ will store $\min_{h \in haps(v)} D(i, v, h)$. Initialize $D(0, v, h) = D''(0, v) = 0$ for all $v \in V \cup \{v_\epsilon\}, h \in haps(v)$. Initialize the remaining cells in table D'' to ∞. For $i \geq 1, v \in V \cup \{v_\epsilon\}, h \in haps(v)$, update $D(i, v, h)$ and $D''(i, v)$ using the following recursion:

$$D(i,v,h) = \min\Bigg(D(i-1,v,h) + c_1, \min_{(u,v)\in E, h\in haps(u)} \begin{cases} D(i,u,h) + c_1 \\ D(i-1,u,h) + c_1 \cdot EDIT(\sigma(v), Q[i]) \end{cases}, \min_{(u,v)\in E} \begin{cases} D''(i,u) + c_1 + c_2 \\ D''(i-1,u) + c_1 \cdot EDIT(\sigma(v), Q[i]) + c_2 \end{cases} \Bigg),$$

$$D''(i,v) = \min(D''(i,v), D(i,v,h))$$

The recursion computes the optimal value of $D(i, v, h)$ by considering the possibility of character deletion, insertion, match, and mismatch, both within and across haplotypes. Immediately after computing each $D(i, v, h)$, we also update $D''(i, v)$. We compute values in table D in the increasing order of i, and then the increasing topological order of $v \in V \cup \{v_\epsilon\}$ for a fixed i. The output, i.e., the minimum alignment cost is $\min_{v \in V \cup \{v_\epsilon\}} D''(|Q|, v)$. The runtime of the algorithm is dominated by the recursion. Solving the recursion takes $O\big(|Q| \Sigma_{v \in V \cup \{v_\epsilon\}} (|haps(v)| \cdot (\delta_{in}(v) + 1))\big)$ time, where $\delta_{in}(v)$ denotes the in-degree of vertex v. As a result, the time complexity of the algorithm is $O(|Q||E||\mathcal{H}|)$. □

Next, we prove that a significantly faster algorithm than $O(|Q||E||\mathcal{H}|)$ for solving Problems 2 or 3 is unlikely. The existence of a faster algorithm for Problem 1 remains open.

Lemma 3. *For any $\epsilon > 0$, there is no algorithm that solves Problem 2 or Problem 3 in either $O(|Q|^{1-\epsilon}|E||\mathcal{H}|)$ or $O(|Q||E|^{1-\epsilon}|\mathcal{H}|)$ or $O(|Q||E||\mathcal{H}|^{1-\epsilon})$ time, unless SETH is false.*

Proof. Problem 3 is a generalized version of Problem 2. Therefore, it suffices to prove the hardness of Problem 2. We derive the reduction from OV problem. Suppose we have an OV instance with equal sized sets A and B comprising d-dimensional binary vectors. Define $n = |A| = |B|$. We assume w.l.o.g. that n is a perfect square. We partition set A into $\sqrt{n}$ subsets $A_1, A_2, \ldots, A_{\sqrt{n}}$, each containing $\sqrt{n}$ vectors. Similarly, we partition set B into $\sqrt{n}$ subsets $B_1, B_2, \ldots, B_{\sqrt{n}}$, each containing $\sqrt{n}$ vectors. Observe that solving the OV problem for all (A_i, B_j) instances, $i, j \in [\sqrt{n}]$, solves the OV problem for (A, B) instance.

Next, we describe the construction of our gadget comprising query sequences $Q_1, \ldots, Q_{\sqrt{n}}$ and haplotype-aware pangenome DAG $G(V, E, \sigma, \mathcal{H})$. Assume alphabet $\{0, 1, 2\}$ for defining the query sequences and the vertex labels in the DAG. Define $Q_i = f_1(A_i)$ for all $i \in [\sqrt{n}]$ (ref. Lemma 1). Next, we will construct $\sqrt{n}$ haplotype sequences. Define $h_i = f_2(B_i)$ for all $i \in [\sqrt{n}]$. Using Lemma 1, we know that all our haplotype sequences have uniform length, i.e., $|h_1| = |h_2| = \ldots = |h_{\sqrt{n}}|$. Next, we construct a DAG where these haplotype sequences can be represented. We build a DAG with $|h_i|$ 'layers'. Each layer comprises three vertices labelled with characters '0', '1' and '2' respectively (Figure 2a). Each layer (except the last) is *fully connected* to its next adjacent layer using $3^2 = 9$ directed edges. Subsequently, we identify the unique path which spells each haplotype sequence in this DAG. Finally, we discard the vertices and edges which are not used by any haplotype sequence (Figure 2b).

The above gadget allows us to solve the OV problem by invoking the haplotype-aware sequence to graph alignment algorithm $\sqrt{n}$ times, once for each query sequence. There exist two orthogonal vectors $a \in A$ and

$b \in B$ if and only if we have at least one query sequence $q \in \{Q_1, \ldots, Q_{\sqrt{n}}\}$ for which an alignment path $\mathcal{P}$ exists such that $r(\mathcal{P}) = 0$ and $EDIT(s(\mathcal{P}), q) < \beta_1\sqrt{n}$. Having $r(\mathcal{P}) = 0$ restricts the alignment to one of the haplotype sequences. The cutoff $\beta_1\sqrt{n}$ is motivated from Lemma 1. It is only remaining to show that a faster algorithm for Problem 2 will contradict SETH by enabling a faster algorithm for the OV problem. Observe that (i) each $|Q_i|$ and $|h_i|$ is in $O(\sqrt{n} \cdot \text{poly}(d))$, (ii) $|V|$ and $|E|$ are asymptotically equal to $|h_i|$, (iii) $|\mathcal{H}| = \sqrt{n}$, and (iv) we invoke the alignment algorithm $\sqrt{n}$ times. Construction of the gadget requires $O(n \cdot \text{poly}(d))$ time. Therefore, if Problem 2 can be solved in either $O(|Q|^{1-\epsilon}|E||\mathcal{H}|)$ or $O(|Q||E|^{1-\epsilon}|\mathcal{H}|)$ or $O(|Q||E||\mathcal{H}|^{1-\epsilon})$ time, then the OV problem can be solved in $O(n^{2-\epsilon/2} \cdot \text{poly}(d))$ time. □

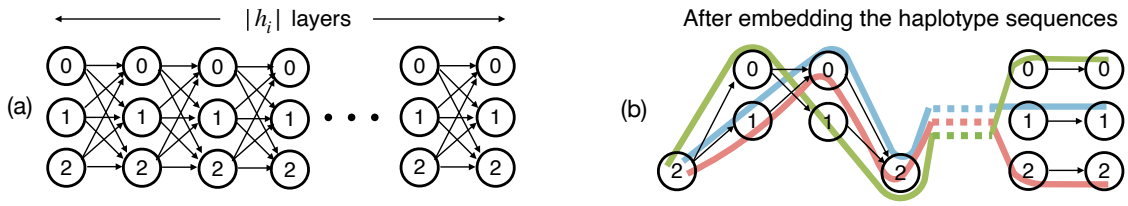


Fig. 2: An illustration of the pangenome DAG used in the reduction proof of Lemma 3.

3 Haplotype-aware Sequence-to-DAG Co-linear Chaining

Most alignment tools use seed-chain-extend heuristic to compute the alignments quickly [50,51]. Given a set of seed matches (anchors) as input, co-linear chaining is a rigorous optimization technique to identify promising alignment regions in a reference. It identifies a coherent subset of anchors such that their coordinates are ordered on the query and the reference. The selected anchors are subsequently combined together to form an alignment. Several versions of co-linear chaining problems have been studied for aligning two sequences [1,20,32,37,10,11,40]. Recent works have further studied the extension of chaining on acyclic [33,30,6,46] and cyclic pangenome graphs [3,43] but these formulations do not consider the haplotype paths.

Concepts and Notations. From hereon we assume that vertices of a haplotype-aware pangenome DAG are labeled with strings. String labeled vertices permit a more compact graph representation which will be useful in the context of chaining. It is trivial to transform a graph from string-labeled form to character-labeled form, and vice versa. Let function $\sigma : V \rightarrow \Sigma^+$ label each vertex $v \in V$ with string $\sigma(v)$ in DAG $G(V, E, \sigma, \mathcal{H})$. The input to the chaining problem is a set of anchors $\{M[1], M[2], \ldots, M[N]\}$. An anchor is represented by a 3-tuple $(v, [x..y], [c..d])$, which signifies a match between query substring $Q[c..d]$ and substring $\sigma(v)[x..y]$ in the DAG (Figure 3). The *weight* function assigns a user-specified weight to each anchor. Anchor weights are typically set proportional to the length of the matching substrings in practice. We use $R^-(v) \subseteq V$ to denote the subset of vertices that can reach v using a path in the DAG. Vertex v is also included in $R^-(v)$. Next, we define the precedence relationship between a pair of anchors.

Definition 1 (Precedence). *Given two anchors $M[i]$ and $M[j]$, $M[i]$ precedes ($\prec$) $M[j]$ if (i) $M[i].d < M[j].c$, (ii) $M[i].v \in R^-(M[j].v)$, and (iii) $M[i].y < M[j].x$ if $M[i].v = M[j].v$.*

Definition 2 (Chain). *A chain is an ordered sequence $(s_1, s_2, \ldots, s_p)$, where each s_j is a pair of the form (i, h) such that $i \in [N]$, $h \in haps(M[i].v)$ and $M[s_j.i] \prec M[s_{j+1}.i]$ for all $j \in [p-1]$.*

In our definition, a chain specifies the indices of the selected haplotypes alongside the anchors. We say that a recombination has occurred in between s_j and s_{j+1} if $s_j.h \neq s_{j+1}.h$. Given a chain S, we use function $r(S)$ to denote the count of recombinations in S. Let $\gamma \in \mathbb{R}_{\geq 0}$ be a user-specified parameter that specifies cost per recombination.

Similar to [33], we will use *path cover* to facilitate efficient sparse dynamic programming on the DAG. A path cover is a set of paths in the DAG such that every vertex in V is included in at least one path. Previous chaining algorithms, e.g., in [6,30,33], identify a path cover with the minimum number of paths because the

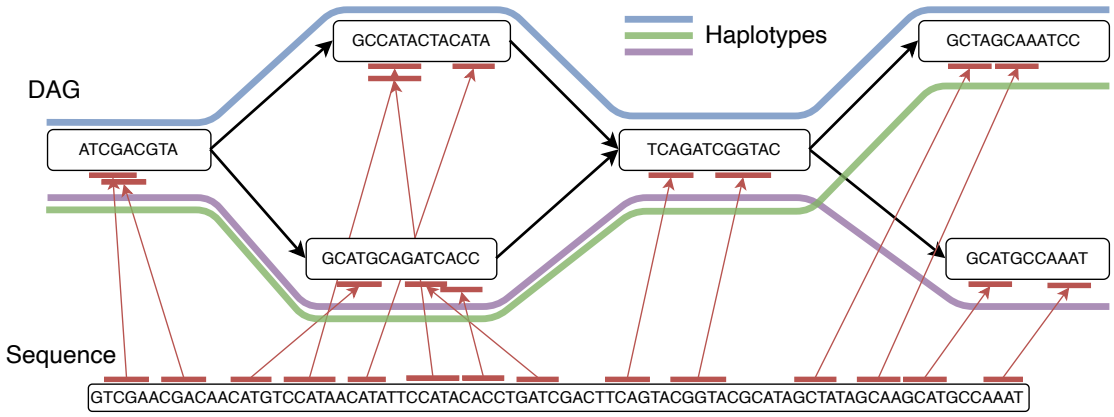


Fig. 3: An example showing anchors between the query sequence and a pangenome DAG.

runtime of sparse dynamic programming increases with the size of the path cover [33]. In the haplotype-aware setting, we will directly use the paths corresponding to the haplotype sequences as path cover. Let $P_1, P_2, \ldots, P_{|\mathcal{H}|}$ be the paths corresponding to the haplotype sequences $h_1, h_2, \ldots, h_{|\mathcal{H}|}$ respectively. Suppose $last2reach(v, h), v \in V, h \in [|\mathcal{H}|]$ denotes the last vertex in path P_h that belongs to $R^-(v)$. $last2reach(v, h)$ does not exist if no vertex in path P_h can reach vertex v. Computing $last2reach(v, h)$ for all $v \in V$ and $h \in [|\mathcal{H}|]$ requires $O(|E||\mathcal{H}|)$ time [33]. We precompute $last2reach$ information; this is a one-time preprocessing step before solving the chaining problem. Next, it is common to use the following balanced binary search tree data structure in chaining algorithms for executing range queries efficiently.

Lemma 4 (ref. [31]). *Let n be the number of leaves in a tree, each storing a $(key, value)$ pair. The data structure supports* update *and* RMQ *(range maximum query) operations, defined below, in $O(\log n)$ time:*
– *update(k, val): For the leaf w with $key = k$, $value(w) \leftarrow \max(value(w), val)$.*
– *RMQ(l, r): Return $\max\{value(w) \mid l < key(w) < r\}$ such that w is a leaf.*
Given n $(key, value)$ pairs, the tree can be constructed in $O(n \log n)$ time and $O(n)$ space.

3.1 Problem Definitions for Haplotype-aware Chaining

– *Problem 5 (Limited recombinations):* Determine an optimal chain $S = s_1, s_2, \ldots, s_p$ that maximises the chaining score, defined as $\sum_{j=1}^{p} weight(M[s_j.i]) \; - \gamma \cdot r(S)$.

The second term $\gamma \cdot r(S)$ in the above scoring function corresponds to the penalty for haplotype switches in a chain. While scoring a chain, it is also essential to penalise the gap corresponding to the distance (in basepairs) between adjacent pair of anchors. Accordingly, we formulate a gap-sensitive version of the above problem. We assume the same definition of function $gap(M[i], M[j]), i, j \in [N]$ as provided in [6].

– *Problem 6 (Limited recombinations, gap-sensitive):* Determine an optimal chain $S = s_1, s_2, \ldots, s_p$ that maximises the chaining score, defined as $\sum_{j=1}^{p} weight(M[s_j.i]) - \sum_{j=2}^{p} gap(M[s_{j-1}.i], M[s_j.i]) - \gamma \cdot r(S)$.

3.2 Proposed Algorithms and Complexity Analysis

First, it is useful to see a naive dynamic programming solution for Problem 5. Let $C(i, h)$ denote the optimal score of a chain ending at pair (i, h), where $i \in [N], h \in haps(M[i].v)$. We use $C''(i)$ to store $min_{h \in haps(M[i].v)} C(i, h)$. A single anchor is a valid chain of size one; therefore, we initialize $C(i, h)$ to $weight(M[i])$ for all $i \in [N], h \in haps(M[i].v)$. Also, initialize $C''(i)$ to $weight(M[i])$ for all $i \in [N]$. We update $C(i, h)$ to its optimal value using the following recursion:

$$C(i,h) = \max\Bigg(weight(M[i]), \max_{M[k]\prec M[i], h\in haps(M[k].v), k\in[N]} C(k,h) + weight(M[i]) \ , \max_{M[k]\prec M[i], k\in[N]} C''(k) + weight(M[i]) - \gamma\Bigg),$$

$$C''(i) = \max(C''(i), C(i,h))$$

If $M[k] \prec M[i]$ and path P_h covers vertex $M[k].v$, then the chaining score obtained using $C(k,h)$ is considered without recombination penalty. This is when (i,h) is next to (k,h) in an optimal chain ending at (i,h). The third term in the expression considers the score with recombination penalty γ. This is needed when (i,h) is next to (k,h'), $h' \neq h$. Immediately after updating each $C(i,h)$, we also update $C''(i)$ in the above recursion. Let function $rank(v)$ specify the topological ordering of vertex v in the DAG. The $C(i,h)$ values should be computed in lexicographically ascending order based on the key $(rank(M[i].v), M[i].x)$. For a fixed i, $C(i,h)$ values may be computed in any order. The above algorithm fills up to $|\mathcal{H}|N$ values in table C. Computing each value requires considering all N number of anchors and filtering those which satisfy the precedence criteria. Even if we ignore the time for filtering, the worst-case asymptotic runtime of this algorithm grows at least as fast as $|\mathcal{H}|N^2$. This makes it too slow when N is in millions, which typically happens when aligning megabase-long *de novo* assembled sequences to human pangenome DAGs. Next, we propose a faster $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time algorithm for solving Problem 5. First, we note the following inequality.

Lemma 5. *$C(i,h) \geq C''(i) - \gamma$ for all $i \in [N], h \in haps(M[i].v)$.*

Proof. Consider any value of $i \in [N]$ and $h \in haps(M[i].v)$. Let $h' = \text{argmax}_{k\in haps(M[i].v)} C(i,k)$, then $C''(i)$ equals $C(i,h')$. If $h' = h$, then $C(i,h) = C''(i)$, thus the inequality holds. Next, consider the case when $h' \neq h$. Let $S_1 = s_1, s_2, \ldots, s_{p-1}, s_p$ be an optimal chain ending at pair (i,h'). We consider another chain $S_2 = s_1, s_2, \ldots, s_{p-1}, s_p{}'$ such that $s_p{}' = (i,h)$. Suppose the score of chain S_2 is x. Both chains S_1 and S_2 differ only by the haplotype used in their last pairs. Therefore, the difference in the chaining scores of S_1 and S_2 is at most γ (one recombination). Thus, $x \geq C''(i) - \gamma$. Also, $C(i,h)$ must be $\geq x$. □

Lemma 6. *Problem 5 can be solved in $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time.*

Proof. See Algorithm 1 for an outline of our approach. We extend the sparse dynamic programming framework of Makinen *et al.* [33] to our haplotype-aware setting. Our path cover $\{P_1, P_2, \ldots, P_{|\mathcal{H}|}\}$ contains $|\mathcal{H}|$ paths corresponding to the $|\mathcal{H}|$ haplotype sequences. We use $|\mathcal{H}|$ search trees $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{|\mathcal{H}|}$, one per path. Each search tree $\mathcal{T}_h$ will record $C(i,h)$ scores specific to that haplotype.

We initialise each search tree with keys $\{M[i].d \mid 1 \leq i \leq N\}$ and values $-\infty$ (Line 1). In Lines 4-13, we create an array Z with $O(|\mathcal{H}|N)$ tuples. After sorting of Z (Line 15), these tuples ensure that the search trees are updated and queried in a proper order. At the time of processing $C(i,h), i \in [N], h \in haps(M[i].v)$, the algorithm would have already finished processing $C(k,h')$ for all k, h' such that $M[k] \prec M[i], h' \in haps(M[k].v)$. The score $C(k,h')$ would have already been recorded in search tree $\mathcal{T}_{h'}$ (Line 26).

All the anchors which lack a preceding anchor are optimally processed at the initialization stage (Line 2). Suppose an optimal chain ending at (i,h) is $s_1, s_2, \ldots, s_{p-1}, s_p$ such that $s_{p-1} = (k,h')$ and $s_p = (i,h)$. Assume that $C(k,h')$ is already computed optimally. While calculating $C(i,h)$, the algorithm handles the following three cases:

- (Case 1) $h = h'$: In this case, we optimally compute $C(i,h)$ by executing a range query in search tree T_h (Line 19). Since $M[k] \prec M[i]$ and $h' = h$, search tree T_h returns the value $C(k,h')$ stored in key $M[k].d$.
- (Case 2) $h \neq h'$ and path $P_{h'}$ does not cover $M[i].v$: Observe that $last2reach(M[i].v, h')$ must exist in this case because $M[k].v$ must be reachable to $M[i].v$. In Lines 10-12, we add tuples in array Z to ensure that $C_{l2r}(i)$ is updated by using a range query in search tree $T_{h'}$ (Line 22). We also put a penalty γ due to recombination in this case. This update to $C_{l2r}(i)$ happens after all anchors in vertex $last2reach(M[i].v, h')$ have been processed and search tree $T_{h'}$ has been updated. Later when we update $C(i,h)$ in Line 19, it gets its optimal value using $C_{l2r}(i)$.

– (Case 3) $h \neq h'$ and path $P_{h'}$ covers $M[i].v$: If $\big(s_1, s_2, \ldots, (k, h'), (i, h)\big)$ is an optimal chain ending at (i, h), then $\big(s_1, s_2, \ldots, (k, h'), (i, h')\big)$ must be an optimal chain which ends at (i, h'). Accordingly, $C(i, h') = C(i, h) + \gamma$. It implies $C''(i) \geq C(i, h) + \gamma$. Using the inequality in Lemma 5, we get $C(i, h) = C''(i) - \gamma$. Accordingly, $C(i, h)$ is updated to its optimal value in Line 24.

The total runtime of the algorithm is dominated by sorting of array Z. Sorting an array of size $O(|\mathcal{H}|N)$ takes $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time. Initializing $|\mathcal{H}|$ search trees at the beginning takes $O(|\mathcal{H}|N \log N)$ time. The algorithm uses $O(|\mathcal{H}|N)$ number of $O(\log N)$-time RMQ and update operations on the search trees. □

Algorithm 1: $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time chaining algorithm for Problem 5

Input: Weighted anchors $M[1..N]$, haplotype paths $P_1, P_2, \ldots, P_{|\mathcal{H}|}$, *last2reach* values, parameter γ
Output: Table C such that $C(i, h)$ = optimal score of a chain that ends at (i, h), for all $i \in [N], h \in [|\mathcal{H}|]$

```
1  Initialize search trees T_h, for all h ∈ [|H|], using keys {M[i].d | 1 ≤ i ≤ N} and values −∞
2  Initialize C(i,h), C''(i) and C_l2r(i) as weight(M[i]), for all i ∈ [N], h ∈ [|H|]
3  /* Create array Z that stores tuples of the form (v, pos, task, anchor, haplotype), where v ∈ V, pos ∈ ℕ,
     task ∈ {0,1,2}, anchor ∈ [N] and haplotype ∈ [|H|].*/
4  for i ← 1 to N do
5      for h ← 1 to |H| do
6          if h ∈ haps(M[i].v) then
7              Z.push(M[i].v, M[i].x, 0, i, h)
8              Z.push(M[i].v, M[i].x, 1, i, h)
9              Z.push(M[i].v, M[i].y, 2, i, h)
10         else if last2reach(M[i].v, h) exists then
11             v_l2r ← last2reach(M[i].v, h)
12             Z.push(v_l2r, |σ(v_l2r)| + 1, 0, i, h)
13     end
14 end
15 for z ∈ Z in lexicographically ascending order based on the key (rank(v), pos, task) do
16     i ← z.anchor, h ← z.haplotype, v ← z.v, wt ← weight(M[i])
17     if z.task = 0 then
18         if h ∈ haps(M[i].v) then
19             C(i,h) ← max(C_l2r(i), C(i,h), wt + T_h.RMQ(0, M[i].c))
20             C''(i) ← max(C''(i), C(i,h))
21         else
22             C_l2r(i) ← max(C_l2r(i), wt + T_h.RMQ(0, M[i].c) − γ)
23     else if z.task = 1 then
24         C(i,h) ← max(C''(i) − γ, C(i,h))
25     else if z.task = 2 then
26         T_h.update(M[i].d, C(i,h))
27 end
```

The optimal chain can be obtained by adding backtracking pointers in the algorithm. Next, we solve Problem 6 by extending Algorithm 1 and including a few additional terms corresponding to gap costs, as described in [6]. An $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time algorithm is provided in Appendix Algorithm 2.

4 Experimental Results

Implementation details. We implemented Algorithm 2 for solving haplotype-aware co-linear chaining in Minichain (v1.3). Our new chaining implementation replaces the haplotype-agnostic chaining implementation proposed in our previous work [6]. Minichain supports an end-to-end seed-chain-extend workflow to align long reads and contigs to pangenome DAGs. Similar to the previous version of the software, we use the seeding and base-to-base alignment routines from Minigraph [24]. We leave the implementation of our $O(|Q||E||\mathcal{H}|)$

time base-to-base alignment algorithm (Section 2.3) as future work because it requires further practical optimizations to reduce the runtime, e.g., using banded alignment or wavefront heuristics [35,58].

In Minichain, we parse the set of vertices, edges, and haplotype paths from a graphical fragment assembly (GFA) file. A read may be sampled from the same or the opposite orientation with respect to the graph. Accordingly, for each connected component in the input DAG, we create another component of the same size with reverse-complemented vertex labels and reversed edges. We use the same path cover for the second component as the original component but the direction of each path is reversed. We compute anchors by using Minigraph's seeding method, which finds (w, k)-minimizer matches [48] between the query sequence Q and the vertex string labels $\sigma(v)$ for all $v \in V$. The default values of w and k are 11 and 17 respectively. We set the weight of each anchor as $200 \cdot k$ by default. For $k = 17$, each anchor in a chain contributes 3400 to the total chain score. Minichain can report multiple best scoring chains and assign a mapping quality [27] to each chain; the criteria for this remains same as before [6].

Pangenome DAG construction. We used 60 publicly-available complete major histocompatibility complex (MHC) haplotype sequences. This set is obtained using complete assembly of CHM13 human cell line [39] and 59 other haplotype-resolved assemblies from the Human Pangenome Research Consortium [25,29]. The length of these sequences varies from 4.8 Mbp to 5.1 Mbp. The MHC region is known to have significant polymorphism, inter-gene sequence similarity and long-range haplotype structures [8]. We used Minigraph (v0.20-r559) [26] to build a DAG using the 60 MHC haplotypes. The graph construction algorithm in Minigraph augments structural variants of size ≥ 50 bp in the graph. After computing the DAG, we realigned each haplotype to the DAG to obtain the corresponding haplotype paths in the DAG.

Simulation of query sequences. We simulated 135 MHC haplotype sequences. Each sequence was simulated as an imperfect mosaic of the 60 reference haplotypes (Figure 4). We used the following procedure for simulation: (i) Select a random haplotype path in the DAG, (ii) Read the first l bases from the chosen haplotype path in the DAG, (iii) If the $(l+1)^{th}$ base is in vertex v, then switch to another randomly chosen haplotype path at vertex v, (iv) Read the next l bases, and repeat the procedure until we hit the last base of a haplotype path. After generating a sequence, we added substitutions at randomly chosen loci in the sequence. These substitutions approximately mimic true mutations and sequencing errors in real data. For each substitution rate of 0.1%, 1% and 5%, we simulated 45 sequences. In each case, three sets of 15 sequences were simulated using $l = 1$ Mbp, $l = 2$ Mbp and $l = 3$ Mbp respectively. Using this procedure, the number of recombination events in all the simulated sequences ranged from 1 to 4.

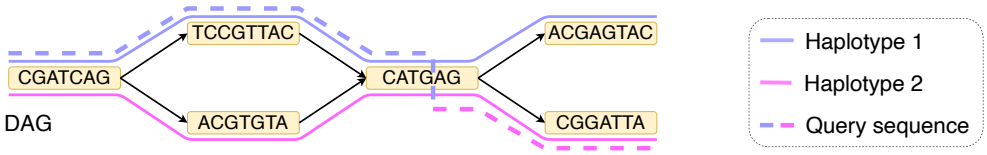
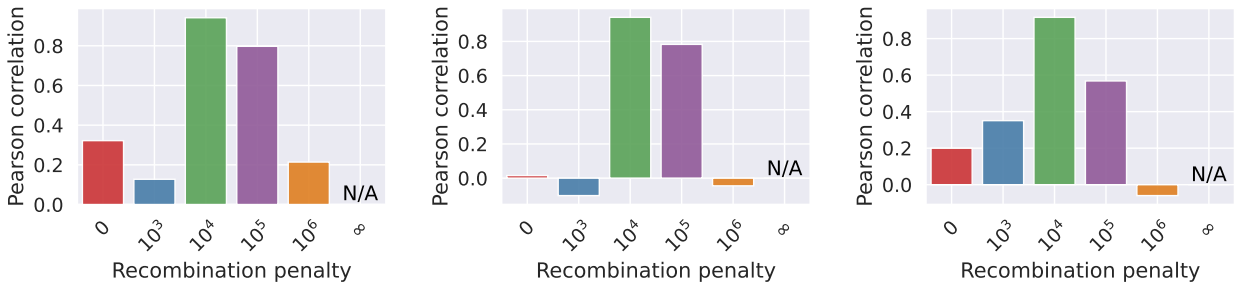


Fig. 4: An example to illustrate simulation of a query sequence as a mosaic of reference haplotypes.

Comparison with haplotype-agnostic and haplotype-restricted chaining. We demonstrate that integrating recombination penalty in the problem formulation leads to an improved concordance of the haplotypes used in our chains and the ground-truth. A positive finite recombination penalty in our problem formulation allows the user to limit the recombinations in an optimal chain. If we set recombination penalty $\gamma = 0$, then our algorithm is equivalent to the haplotype-agnostic chaining algorithm in [6]. Recombination penalty $\gamma = \infty$ corresponds to haplotype-restricted chaining, where a chain can use only one of the reference haplotypes. We evaluated Minichain with different values of recombination penalty $\gamma = 0, 10^3, 10^4, 10^5, 10^6, \infty$. Minichain computes an optimal chain as a sequence of pairs of the selected anchors and the haplotypes. Thus, we can know the sequence of haplotypes and the number of recombinations in each chain.

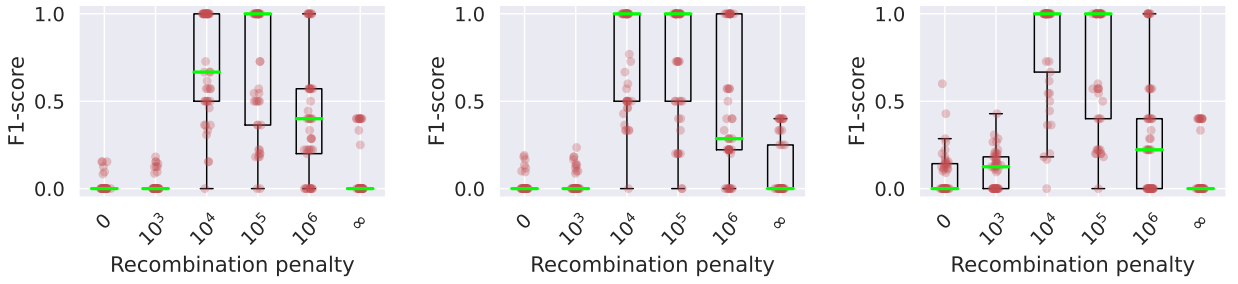
We show the Pearson correlation coefficients between the observed number of recombinations and the true number of recombinations using different values of substitution rates and recombination penalties (Figure 5). The results suggest that $\gamma = 10^4$ gives the best correlation across different substitution rates. The correlation is weak when $\gamma = 0$ or $\gamma = 10^6$. The correlation is not defined when $\gamma = \infty$ because the observed number of recombinations remained zero; thus the standard deviation of the observed values remained zero.



(a) 0.1% substitution rate (b) 1% substitution rate (c) 5% substitution rate

Fig. 5: Pearson correlation between the number of recombinations in Minichain's output chain and the true count. We evaluated the performance using different substitution rates and recombination penalties.

Next, we compared the list of query sequences' source haplotypes to the selected haplotypes in the output chains. Suppose a query sequence was simulated using haplotypes in the order $H_1, H_2, \ldots, H_n$, then we recorded the "true haplotype recombination pairs" as a set of pairs $\{(start, H_1), (H_1, H_2), \ldots, (H_{n-1}, H_n), (H_n, end)\}$. Similarly, we recorded the sets corresponding to the "observed haplotype recombination pairs" from Minichain's output chains. We compared these two sets for each query and calculated the F1-scores (Figure 6, Appendix Table 1). We achieved higher F1-scores with $\gamma = 10^4, 10^5$ compared to the haplotype-agnostic ($\gamma = 0$) and haplotype-restricted ($\gamma = \infty$) modes. These results suggest that haplotype-aware chaining and alignment algorithms can be useful to prevent spurious recombinations.



(a) 0.1% substitution rate (b) 1% substitution rate (c) 5% substitution rate

Fig. 6: Box plots showing the levels of consistency between the haplotype recombination pairs in Minichain's output chain and the ground-truth using different substitution rates and recombination penalties. Each red dot in the plot corresponds to a query sequence. The median values are highlighted using green lines.

We expect further improvements in the accuracy of our algorithm if SNPs and short indels are also augmented in a pangenome DAG [18]. More variants will help to distinguish between closely-related haplotypes which may not be possible using structural variants. Doing this will also require improvements in Minichain's seeding and chaining implementation to allow the use of anchors which cover small bubbles in a DAG. This should be possible by considering a more flexible definition of anchors which can span multiple vertices [47,46,49]. The runtime of Minichain ranged from 5 to 11 minutes for aligning an MHC sequence to the DAG. The worst-case time complexity of the proposed haplotype-aware chaining algorithm is higher compared to the existing haplotype-agnostic chaining algorithms. This is because we require haplotype paths as the path cover where as haplotype-agnostic chaining algorithms use a path cover of minimum size [6,30,33]. For example, our DAG with 60 haplotype paths has a minimum path cover of size 5. Thus, we observed that our haplotype-aware chaining algorithm in Minichain is about 10× slower compared to the haplotype-agnostic chaining algorithm [6].

Acknowledgements

This work is supported by funding from the National Supercomputing Mission, India under DST/NSM/R&D_HPC_Applications, the Science and Engineering Research Board (SERB) under SRG/2021/000044, and the Kotak AI-ML Center at the Indian Institute of Science. We used computing resources provided by the C-DAC National PARAM Supercomputing Facility, India and the National Energy Research Scientific Computing Center, USA.

References

1. Abouelhoda, M., Ohlebusch, E.: Chaining algorithms for multiple genome comparison. Journal of Discrete Algorithms **3**(2-4), 321–341 (2005)
2. Amir, A., Lewenstein, M., Lewenstein, N.: Pattern matching in hypertext. Journal of Algorithms **35**(1), 82–99 (2000)
3. Antipov, D., Korobeynikov, A., McLean, J.S., Pevzner, P.A.: hybridspades: an algorithm for hybrid assembly of short and long reads. Bioinformatics **32**(7), 1009–1015 (2016)
4. Avila Cartes, J., Bonizzoni, P., Ciccolella, S., Della Vedova, G., Denti, L., Monti, D., Pirola, Y., Porto, F.: Recgraph: adding recombinations to sequence-to-graph alignments. bioRxiv pp. 2022–10 (2022)
5. Backurs, A., Indyk, P.: Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In: Proceedings of the forty-seventh annual ACM symposium on Theory of computing. pp. 51–58 (2015)
6. Chandra, G., Jain, C.: Sequence to graph alignment using gap-sensitive co-linear chaining. In: Research in Computational Molecular Biology: 27th Annual International Conference, RECOMB 2023, Istanbul, Turkey, April 16–19, 2023, Proceedings. pp. 58–73. Springer (2023)
7. Cheng, H., Concepcion, G.T., Feng, X., Zhang, H., Li, H.: Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. Nature methods **18**(2), 170–175 (2021)
8. Dilthey, A.T.: State-of-the-art genome inference in the human MHC. The International Journal of Biochemistry & Cell Biology **131**, 105882 (2021)
9. Ebler, J., Ebert, P., Clarke, W.E., Rausch, T., Audano, P.A., Houwaart, T., Mao, Y., Korbel, J.O., Eichler, E.E., Zody, M.C., et al.: Pangenome-based genome inference allows efficient and accurate genotyping across a wide spectrum of variant classes. Nature genetics **54**(4), 518–525 (2022)
10. Eppstein, D., Galil, Z., Giancarlo, R., Italiano, G.F.: Sparse dynamic programming i: linear cost functions. Journal of the ACM **39**(3), 519–545 (1992)
11. Eppstein, D., Galil, Z., Giancarlo, R., Italiano, G.F.: Sparse dynamic programming ii: convex and concave cost functions. Journal of the ACM **39**(3), 546–567 (1992)
12. Equi, M., Mäkinen, V., Tomescu, A.I.: Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless seth fails. Theoretical Computer Science **975**, 114128 (2023)
13. Equi, M., Mäkinen, V., Tomescu, A.I., Grossi, R.: On the complexity of string matching for graphs. ACM Transactions on Algorithms **19**(3), 1–25 (2023). https://doi.org/10.1145/3588334
14. Garrison, E., Guarracino, A., Heumos, S., Villani, F., Bao, Z., Tattini, L., Hagmann, J., Vorbrugg, S., Marco-Sola, S., Kubica, C., et al.: Building pangenome graphs. bioRxiv pp. 2023–04 (2023)
15. Garrison, E., Sirén, J., Novak, A.M., Hickey, G., Eizenga, J.M., Dawson, E.T., Jones, W., Garg, S., Markello, C., Lin, M.F., et al.: Variation graph toolkit improves read mapping by representing genetic variation in the reference. Nature biotechnology **36**(9), 875–879 (2018)
16. Gibney, D., Thankachan, S.V., Aluru, S.: On the hardness of sequence alignment on de bruijn graphs. Journal of Computational Biology **29**(12), 1377–1396 (2022)
17. Grytten, I., Dagestad Rand, K., Sandve, G.K.: Kage: Fast alignment-free graph-based genotyping of snps and short indels. Genome Biology **23**(1), 209 (2022)
18. Hickey, G., Monlong, J., Ebler, J., Novak, A.M., Eizenga, J.M., Gao, Y., Marschall, T., Li, H., Paten, B.: Pangenome graph construction from genome alignments with minigraph-cactus. Nature Biotechnology pp. 1–11 (2023)
19. Hoppenworth, G., Bentley, J.W., Gibney, D., V Thankachan, S.: The fine-grained complexity of median and center string problems under edit distance. In: 28th Annual European Symposium on Algorithms, ESA 2020 (2020)
20. Jain, C., Gibney, D., Thankachan, S.V.: Co-linear chaining with overlaps and gap costs. In: International Conference on Research in Computational Molecular Biology (RECOMB). pp. 246–262. Springer (2022)

21. Jain, C., Tavakoli, N., Aluru, S.: A variant selection framework for genome graphs. Bioinformatics **37**(Supplement_1), i460–i467 (2021)
22. Jain, C., Zhang, H., Gao, Y., Aluru, S.: On the complexity of sequence-to-graph alignment. Journal of Computational Biology **27**(4), 640–654 (2020)
23. Lee, C., Grasso, C., Sharlow, M.F.: Multiple sequence alignment using partial order graphs. Bioinformatics **18**(3), 452–464 (2002)
24. Li, H.: Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics **34**(18), 3094–3100 (may 2018). https://doi.org/10.1093/bioinformatics/bty191
25. Li, H.: Sample graphs and sequences for testing sequence-to-graph alignment (2022). https://doi.org/10.5281/zenodo.6617246
26. Li, H., Feng, X., Chu, C.: The design and construction of reference pangenome graphs with minigraph. Genome Biology **21**(1) (oct 2020)
27. Li, H., Ruan, J., Durbin, R.: Mapping short DNA sequencing reads and calling variants using mapping quality scores. Genome research **18**(11), 1851–1858 (2008)
28. Li, N., Stephens, M.: Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. Genetics **165**(4), 2213–2233 (2003)
29. Liao, W.W., Asri, M., Ebler, J., Doerr, D., Haukness, M., Hickey, G., Lu, S., Lucas, J.K., Monlong, J., Abel, H.J., et al.: A draft human pangenome reference. Nature **617**(7960), 312–324 (2023)
30. Ma, J., Cáceres, M., Salmela, L., Mäkinen, V., Tomescu, A.I.: Chaining for accurate alignment of erroneous long reads to acyclic variation graphs. Bioinformatics **39**(8), btad460 (2023)
31. Mäkinen, V., Belazzougui, D., Cunial, F., Tomescu, A.I.: Genome-scale algorithm design. Cambridge University Press (2015)
32. Mäkinen, V., Sahlin, K.: Chaining with overlaps revisited. In: 31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
33. Mäkinen, V., Tomescu, A.I., Kuosmanen, A., Paavilainen, T., Gagie, T., Chikhi, R.: Sparse dynamic programming on DAGs with small width. ACM Transactions on Algorithms **15**(2), 1–21 (Apr 2019)
34. Manber, U., Wu, S.: Approximate string matching with arbitrary costs for text and hypertext. In: Advances In Structural And Syntactic Pattern Recognition, pp. 22–33. World Scientific (1992)
35. Marco-Sola, S., Moure, J.C., Moreto, M., Espinosa, A.: Fast gap-affine pairwise alignment using the wavefront algorithm. Bioinformatics **37**(4), 456–463 (2021)
36. Mokveld, T., Linthorst, J., Al-Ars, Z., Holstege, H., Reinders, M.: Chop: haplotype-aware path indexing in population graphs. Genome biology **21**, 1–16 (2020)
37. Myers, G., Miller, W.: Chaining multiple-alignment fragments in sub-quadratic time. In: SODA. vol. 95, pp. 38–47 (1995)
38. Navarro, G.: Improved approximate pattern matching on hypertext. Theoretical Computer Science **237**(1-2), 455–463 (2000)
39. Nurk, S., Koren, S., Rhie, A., Rautiainen, M., et al.: The complete sequence of a human genome. Science **376**(6588), 44–53 (apr 2022). https://doi.org/10.1126/science.abj6987
40. Otto, C., Hoffmann, S., Gorodkin, J., Stadler, P.F.: Fast local fragment chaining using sum-of-pair gap costs. Algorithms for Molecular Biology **6**(1), 4 (2011)
41. Petes, T.D.: Meiotic recombination hot spots and cold spots. Nature Reviews Genetics **2**(5), 360–369 (2001)
42. Pritt, J., Chen, N.C., Langmead, B.: Forge: prioritizing variants for graph genomes. Genome biology **19**(1), 1–16 (2018)
43. Rajput, J., Chandra, G., Jain, C.: Co-Linear Chaining on Pangenome Graphs. In: 23rd International Workshop on Algorithms in Bioinformatics (WABI 2023). vol. 273, pp. 12:1–12:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2023)
44. Rautiainen, M., Mäkinen, V., Marschall, T.: Bit-parallel sequence-to-graph alignment. Bioinformatics **35**(19), 3599–3607 (2019)
45. Rautiainen, M., Nurk, S., Walenz, B.P., Logsdon, G.A., Porubsky, D., Rhie, A., Eichler, E.E., Phillippy, A.M., Koren, S.: Telomere-to-telomere assembly of diploid chromosomes with verkko. Nature Biotechnology pp. 1–9 (2023)
46. Rizzo, N., Cáceres, M., Mäkinen, V.: Chaining of maximal exact matches in graphs. In: String Processing and Information Retrieval: 30th International Symposium, SPIRE 2023, Pisa, Italy, September 26–28, 2023, Proceedings. p. 353–366. Springer-Verlag, Berlin, Heidelberg (2023)
47. Rizzo, N., Cáceres, M., Mäkinen, V.: Finding Maximal Exact Matches in Graphs. In: Belazzougui, D., Ouangraoua, A. (eds.) 23rd International Workshop on Algorithms in Bioinformatics (WABI 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 273, pp. 10:1–10:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2023)

48. Roberts, M., Hayes, W., Hunt, B.R., Mount, S.M., Yorke, J.A.: Reducing storage requirements for biological sequence comparison. Bioinformatics **20**(18), 3363–3369 (jul 2004). https://doi.org/10.1093/bioinformatics/bth408
49. Rossi, M., Oliva, M., Langmead, B., Gagie, T., Boucher, C.: Moni: a pangenomic index for finding maximal exact matches. Journal of Computational Biology **29**(2), 169–187 (2022)
50. Sahlin, K., Baudeau, T., Cazaux, B., Marchet, C.: A survey of mapping algorithms in the long-reads era. Genome Biology **24**(1), 1–23 (2023)
51. Shaw, J., Yu, Y.W.: Proving sequence aligners can guarantee accuracy in almost O(m log n) time through an average-case analysis of the seed-chain-extend heuristic. Genome Research **33**(7), 1175–1187 (2023)
52. Singh, V., Pandey, S., Bhardwaj, A.: From the reference human genome to human pangenome: Premise, promise and challenge. Frontiers in Genetics **13**, 1042550 (2022)
53. Sirén, J., Garrison, E., Novak, A.M., Paten, B., Durbin, R.: Haplotype-aware graph indexes. Bioinformatics **36**(2), 400–407 (2020)
54. Sirén, J., Monlong, J., Chang, X., Novak, A.M., Eizenga, J.M., Markello, C., Sibbesen, J.A., Hickey, G., Chang, P.C., Carroll, A., et al.: Pangenomics enables genotyping of known structural variants in 5202 diverse genomes. Science **374**(6574), abg8871 (2021)
55. Wang, T., Antonacci-Fulton, L., Howe, K., et al.: The human pangenome project: a global resource to map genomic diversity. Nature **604**(7906), 437–446 (apr 2022)
56. Wenger, A.M., Peluso, P., Rowell, W.J., Chang, P.C., Hall, R.J., Concepcion, G.T., Ebler, J., Fungtammasan, A., Kolesnikov, A., Olson, N.D., et al.: Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. Nature biotechnology **37**(10), 1155–1162 (2019)
57. Williams, R.: A new algorithm for optimal 2-constraint satisfaction and its implications. Theoretical Computer Science **348**(2-3), 357–365 (2005)
58. Zhang, H., Wu, S., Aluru, S., Li, H.: Fast sequence to graph alignment using the graph wavefront algorithm. arXiv preprint arXiv:2206.13574 (2022)

Appendix

1 Algorithm for solving Problem 6

To efficiently compute gap costs between anchors during co-linear chaining, we follow the idea as described in [6]. First, we precompute an index of the DAG to facilitate efficient calculation of gaps during chaining. Let $D(v_1, v_2)$ denote the shortest path measured in terms of the number of characters from v_1 to v_2. If path P_h covers vertex v, then suppose $dist2begin(v, h)$ is the number of characters in haplotype h before vertex v. Accordingly, our precomputed index contains $last2reach(v, h)$, $D(last2reach(v, h), v)$ and $dist2begin(v, h)$ for all $v \in V$ and $h \in [|\mathcal{H}|]$. All these quantities can be computed in $O(|E||\mathcal{H}|)$ time [6] during preprocessing of the DAG. See Algorithm 2 for an outline of the gap-sensitive haplotype-aware chaining procedure. This is similar to Algorithm 1 except that we introduce two variables g_1 and g_2 (Lines 18, 27) to consider gaps between anchors.

Algorithm 2: $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time chaining algorithm for Problem 6

Input: Weighted anchors $M[1..N]$, haplotype paths $P_1, \ldots, P_{|\mathcal{H}|}$, precomputed index of DAG, parameter γ
Output: Table C such that $C(i, h)$ = optimal score of a chain that ends at (i, h), for all $i \in [N], h \in [|\mathcal{H}|]$

1 Initialize search trees $\mathcal{T}_h$, for all $h \in [|\mathcal{H}|]$, using keys $\{M[i].d \mid 1 \leq i \leq N\}$ and values $-\infty$
2 Initialize $C(i, h), C''(i)$ and $C_{l2r}(i)$ as $weight(M[i])$, for all $i \in [N], h \in [|\mathcal{H}|]$
3 /* Create array Z that stores tuples of the form $(v, pos, task, anchor, haplotype)$, where $v \in V$, $pos \in \mathbb{N}$, $task \in \{0, 1, 2\}$, $anchor \in [N]$ and $haplotype \in [|\mathcal{H}|]$.*/
4 **for** $i \leftarrow 1$ ***to*** N **do**
5   **for** $h \leftarrow 1$ ***to*** $|\mathcal{H}|$ **do**
6   **if** $h \in haps(M[i].v)$ **then**
7             $Z.push(M[i].v, M[i].x, 0, i, h)$
8             $Z.push(M[i].v, M[i].x, 1, i, h)$
9             $Z.push(M[i].v, M[i].y, 2, i, h)$
10   **else if** $last2reach(M[i].v, h)$ *exists* **then**
11             $v_{l2r} \leftarrow last2reach(M[i].v, h)$
12             $Z.push(v_{l2r}, |\sigma(v_{l2r})| + 1, 0, i, h)$
13   **end**
14 **end**
15 **for** $z \in Z$ *in lexicographically ascending order based on the key* $(rank(v), pos, task)$ **do**
16     $i \leftarrow z.anchor, h \leftarrow z.haplotype, v \leftarrow z.v, wt \leftarrow weight(M[i])$
17   **if** $z.task = 0$ **then**
18         $g_1 \leftarrow M[i].x + dist2begin(v, h) + D(v, M[i].v) + M[i].c - 2$
19   **if** $h \in haps(M[i].v)$ **then**
20             $C(i, h) \leftarrow \max(C_{l2r}(i),\ C(i, h),\ wt + \mathcal{T}_h.\mathsf{RMQ}(0, M[i].c) - g_1)$
21             $C''(i) \leftarrow \max(C''(i), C(i, h))$
22   **else**
23             $C_{l2r}(i) \leftarrow \max(C_{l2r}(i),\ wt + \mathcal{T}_h.\mathsf{RMQ}(0, M[i].c) - \gamma)$
24   **else if** $z.task = 1$ **then**
25         $C(i, h) \leftarrow \max(C''(i) - \gamma,\ C(i, h))$
26   **else if** $z.task = 2$ **then**
27         $g_2 \leftarrow M[i].y + dist2begin(v, h) + M[i].d$
28         $\mathcal{T}_h.\mathsf{update}(M[i].d,\ C(i, h) + g_2)$
29 **end**

Table 1: We show the true recombinations between haplotypes and the recombinations observed in co-linear chains using recombination penalty $\gamma = 10^4$. We show this data for 45 query sequences that were generated using 0.1% substitution rate. 'S' indicates the start and 'E' indicates the end.

Query	True haplotype switches	Haplotype switches in chains	F1-score
1	>S>HG01123#1>HG02717#1>HG005#1 >HG01358#1>HG01978#2>E	>S>HG02080#2>HG00621#2>HG01978#2>E	0.200000
2	>S>HG002#1>HG01358#2>HG00733#1 >HG01258#1>HG01361#2>E	>S>HG002#2>HG01258#1>HG01361#2>E >S>HG002#2>HG01258#1>HG01361#2>E	0.400000
3	>S>HG00438#1>HG01258#2>E	>S>HG00438#1>HG01258#2>E	1.000000
4	>S>HG02886#1>HG02717#1>HG01928#1 >HG01106#1>HG01071#2>E	>S>HG02886#1>HG002#1>HG01928#2 >HG01106#1>CHM13#0>E	0.166667
5	>S>HG02148#1>HG02717#2>HG01891#2 >HG01123#1>HG01952#1>E	>S>HG02148#1>HG01071#2>HG01891#2 >HG01952#1>E	0.333333
6	>S>HG02622#2>HG01891#1>NA18906#1>E	>S>HG02622#2>HG01891#1>NA18906#1>E	1.000000
7	>S>NA18906#2>HG01358#1>HG02559#2 >HG01109#1>HG005#2>E	>S>HG01358#1>HG02559#2>HG01109#1 >HG005#2>E	0.727273
8	>S>HG01071#2>HG00438#2>HG01891#2>E	>S>HG01071#2>HG00438#2>HG01891#2>E	1.000000
9	>S>HG01175#2>HG01258#1>HG01891#2>E	>S>HG01175#2>HG01258#1>HG01891#2>E	1.000000
10	>S>HG02622#1>HG01952#2>HG01106#1 >HG01358#2>HG00438#1>E	>S>HG02622#1>HG01952#2>HG01106#1 >HG01358#2>HG00438#1>E	1.000000
11	>S>HG00438#1>HG01891#2>E	>S>HG00438#1>HG01891#2>E	1.000000
12	>S>HG02717#2>HG00621#2>HG02723#1>E	>S>HG02717#2>HG00621#2>HG02723#1>E	1.000000
13	>S>HG03098#2>HG02257#2>HG03540#1>E	>S>HG03098#2>HG02257#2>HG03540#1>E	1.000000
14	>S>HG02622#2>HG02559#2>HG00741#1>E	>S>HG02622#2>HG02559#2>HG00741#1>E	1.000000
15	>S>HG01175#2>HG03540#1>E	>S>HG01175#2>HG00735#1>HG03540#1>E	0.571429
16	>S>HG01071#2>HG00673#1>E	>S>HG01071#2>HG00673#1>E	1.000000
17	>S>HG01109#2>HG03516#1>HG02886#1>E	>S>HG01109#2>HG03516#1>HG02886#1>E	1.000000
18	>S>HG03098#2>HG01891#1>E	>S>HG03098#2>HG01891#1>E	1.000000
19	>S>HG01952#1>HG01978#2>HG02886#1 >HG002#1>HG01071#2>E	>S>HG01952#1>HG02886#1>HG005#2 >HG01071#2>E	0.363636
20	>S>HG01952#1>HG002#1>HG002#2 >HG00438#1>HG00733#1>E	>S>HG01952#1>HG002#1>HG002#2 >HG00438#1>CHM13#0>E	0.666667
21	>S>HG02717#2>HG02622#2>HG01952#2>E	>S>HG02717#2>HG02622#2>HG01952#2>E	1.000000
22	>S>HG01978#1>HG03098#2>HG02486#2>E	>S>HG01978#1>HG03098#2>HG02486#2>E	1.000000
23	>S>HG01106#2>HG01258#2>E	>S>HG01106#2>HG01258#2>E	1.000000
24	>S>HG00741#1>HG02886#1>HG01361#1>E	>S>HG00741#1>HG02886#1>HG01361#1>E	1.000000
25	>S>HG02080#1>HG03540#2>HG01106#2 >HG02559#1>HG01358#2>E	>S>HG03540#2>HG01106#2>HG00735#1 >HG005#1>HG01358#2>E	0.333333
26	>S>HG01928#1>HG00735#1>E	>S>HG01928#1>HG00735#1>E	1.000000
27	>S>HG01106#2>HG02717#1>E	>S>HG01106#2>HG00621#2>HG02717#1>E	0.571429
28	>S>HG00735#1>HG01258#1>HG02257#2 >HG00673#1>HG01928#1>E	>S>HG00438#2>HG02257#2>HG00621#2 >CHM13#0>HG01928#1>E	0.166667
29	>S>HG01361#1>HG02723#1>CHM13#0>E	>S>HG01361#1>HG02723#1>CHM13#0>E	1.000000
30	>S>HG02886#2>HG01106#2>HG01978#1 >HG02109#2>HG02148#1>E	>S>HG02886#2>HG01978#1>HG00735#1E >HG02148#1>E	0.363636
31	>S>HG01361#2>HG01109#2>HG01891#1>E	>S>HG01361#2>HG01109#2>HG01891#1>E	1.000000
32	>S>HG02148#1>HG02559#1>E	>S>HG02148#1>HG02559#1>E	1.000000
33	>S>HG00733#1>HG02723#1>HG01175#2 >HG02148#1>HG01123#1>E	>S>HG02109#2>HG02723#1>HG01175#2 >HG02148#1>HG01123#1>E	0.666667
34	>S>HG00673#1>NA18906#1>HG002#1>E	>S>HG00673#1>NA18906#1>HG002#1>E	1.000000
35	>S>HG00621#2>HG005#1>E	>S>HG00621#2>HG005#1>E	1.000000
36	>S>HG00621#2>HG02080#1>HG01358#2>HG03540#1 >HG01361#2>E	>S>HG02109#2>HG01358#2>HG01258#1 >HG01071#2>HG01361#2>E	0.166667
37	>S>HG02257#2>HG005#2>E	>S>HG02257#2>HG005#2>E	1.000000
38	>S>HG03516#1>HG01928#1>HG01952#2 >HG03098#2>HG02559#1>E	>S>HG03098#2>HG01928#1>HG01952#2 >CHM13#0>E	0.181818
39	>S>HG01361#2>HG00438#2>E	>S>HG01361#2>HG00438#2>E	1.000000
40	>S>HG02559#2>HG03098#2>E	>S>HG02559#2>HG03098#2>E	1.000000
41	>S>HG01123#1>HG02257#1>HG01928#2>E	>S>HG01123#1>HG02257#1>HG01928#2>E	1.000000
42	>S>HG01123#2>HG01123#1>E	>S>HG01123#2>HG00735#1>HG01123#1>E	0.571429
43	>S>HG01071#2>HG01928#2>E	>S>HG01071#2>HG01928#2>E	1.000000
44	>S>HG00438#1>HG03540#1>HG02717#2 >HG00735#1>HG02622#2>E	>S>HG02109#2>HG00621#1>HG02622#1 >HG02080#2>HG00735#1>HG02622#2>E	0.307692
45	>S>HG03098#2>HG00621#1>HG01358#1>E	>S>HG03098#2>HG00621#1>HG01358#1>E	1.000000