



Haplotype-aware sequence alignment to pangenome graphs

Ghanshyam Chandra, Daniel Gibney and Chirag Jain

Genome Res. published online July 16, 2024

Access the most recent version at doi:[10.1101/gr.279143.124](https://doi.org/10.1101/gr.279143.124)

P<P Published online July 16, 2024 in advance of the print journal.

Accepted Manuscript Peer-reviewed and accepted for publication but not copyedited or typeset; accepted manuscript is likely to differ from the final, published version.

Creative Commons License This article is distributed exclusively by Cold Spring Harbor Laboratory Press for the first six months after the full-issue publication date (see <https://genome.cshlp.org/site/misc/terms.xhtml>). After six months, it is available under a Creative Commons License (Attribution-NonCommercial 4.0 International), as described at <http://creativecommons.org/licenses/by-nc/4.0/>.

Email Alerting Service Receive free email alerts when new articles cite this article - sign up in the box at the top right corner of the article or [click here](#).

Comprehensive immune receptor profiling.
Discover the **DriverMap™ AIR Assay** difference.



To subscribe to *Genome Research* go to:
<https://genome.cshlp.org/subscriptions>

Published by Cold Spring Harbor Laboratory Press

Haplotype-aware sequence alignment to pangenome graphs

Ghanshyam Chandra¹, Daniel Gibney², Chirag Jain^{1*}

¹Department of Computational and Data Sciences, Indian Institute of Science, Bangalore KA 560012, India

²Department of Computer Science, The University of Texas at Dallas, TX 75080, USA

*corresponding author: chirag@iisc.ac.in

Abstract

Modern pangenome graphs are built using haplotype-resolved genome assemblies. When mapping reads to a pangenome graph, prioritizing alignments that are consistent with the known haplotypes improves genotyping accuracy. However, the existing rigorous formulations for co-linear chaining and alignment problems do not consider the haplotype paths in a pangenome graph. This often leads to spurious read alignments to those paths that are unlikely recombinations of the known haplotypes.

In this paper, we develop novel formulations and algorithms for sequence-to-graph alignment and chaining problems. Inspired by the genotype imputation models, we assume that a query sequence is an imperfect mosaic of reference haplotypes. Accordingly, we introduce a recombination penalty in the scoring functions for each haplotype switch. First, we solve haplotype-aware sequence-to-graph 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. To complement our solution, we prove that an algorithm significantly faster than $O(|Q||E||\mathcal{H}|)$ is impossible under the Strong Exponential Time Hypothesis (SETH). Second, we propose a haplotype-aware chaining algorithm that runs in $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time after graph preprocessing, where N is the count of input anchors. We then establish that a chaining algorithm significantly faster than $O(|\mathcal{H}|N)$ is impossible under SETH. As a proof-of-concept, we implemented our chaining algorithm in the Minichain aligner. By aligning sequences sampled from the human major histocompatibility complex (MHC) to a pangenome graph of 60 MHC haplotypes, we demonstrate that our algorithm achieves better consistency with ground-truth recombinations when compared to a haplotype-agnostic algorithm.

Introduction

Pangenome graphs represent the maximum possible diversity that exists in the population (Singh et al., 2022). A variety of methods have been developed that use pangenome graphs for common applications including

genotyping, variant calling, haplotype reconstruction, metagenomic read classification, etc. (Eizenga et al., 2020). 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 genomes (Cheng et al., 2021; Nurk et al., 2022; Rautiainen et al., 2023; Wenger et al., 2019). Building a pangenome graph directly from phased assemblies instead of variant calls (VCF files) allows a more comprehensive representation of the variation (Wang et al., 2022). Accordingly, the latest methods for pangenome graph construction use phased assemblies to construct a graph (Ebler et al., 2022; Hickey et al., 2023; Garrison et al., 2023; Liao et al., 2023; Li et al., 2020, 2024). The input haplotypes are stored as paths in the graph.

Pangenome graphs raise fundamental questions about the trade-offs 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. This issue has been addressed previously by using different techniques, e.g., by limiting the amount of variation in the graph (Jain et al., 2021; Pritt et al., 2018; Tavakoli et al., 2022), artificially simplifying complex regions (Garrison et al., 2018), or restricting the alignment to either one (Mokveld et al., 2020; Sirén et al., 2021, 2020) or two haplotype paths (Avila Cartes et al., 2022). A more principled approach to tackle this problem may be to leverage linkage disequilibrium, i.e., the correlations between two or more genetic variants, i.e. where individuals tend to have same genotype (Ebler et al., 2022; Grytten et al., 2022). For example, PanGenie is an alignment-free genotyping algorithm that leverages long-range haplotype information inherent in the phased genomes (Ebler et al., 2022).

A pangenome graph is represented either as a directed cyclic graph or a directed acyclic graph (DAG), where each vertex is labeled by a sequence. The primary formulation for sequence-to-graph alignment problem seeks a path in the graph which spells a sequence with minimum edit distance from the query sequence (Amir et al., 2000; Equi et al., 2023b; Jain et al., 2020; Lee et al., 2002; Navarro, 2000; Manber and Wu, 1992; Rautiainen et al., 2019). $O(|V| + |Q||E|)$ -time algorithms exist for both exact and approximate pattern matching problems for graphs, where Q denotes the query sequence, 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 (Pritt et al., 2018). Co-linear chaining is another common technique used in modern aligners. It is used to identify a coherent subset of anchors (short exact matches) that can be joined together to produce an alignment. The existing formulations for chaining on graphs share the same limitation of not considering the associations between genetic variants

(Chandra and Jain, 2023; Ma et al., 2023; Mäkinen et al., 2019; Rajput et al., 2024; Rizzo et al., 2023a). Some of these chaining algorithms run in $O(KN \log KN)$ time after graph preprocessing, where K denotes the minimum number of paths required to cover all the vertices and N denotes the count of input anchors (Chandra and Jain, 2023; Ma et al., 2023).

In this paper, we address the above limitations by introducing ‘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 (Li and Stephens, 2003). The Li-Stephens model is a probabilistic generative model which assumes that a sampled haplotype is an imperfect mosaic of known haplotypes. Similarly, our formulation for haplotype-aware sequence-to-DAG alignment problem optimizes the number of edits and haplotype switches simultaneously (Figure 1). We give $O(|Q||E||\mathcal{H}|)$ time 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 not possible under the strong exponential time hypothesis (SETH) (Vassilevska Williams, 2015). We also formulate the haplotype-aware co-linear chaining problem. We solve it in $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time, assuming that a one-time preprocessing of the DAG is done in $O(|E||\mathcal{H}|)$ time. To provide evidence of the near optimality of this algorithm, we show that it is impossible to solve this problem significantly faster than $O(|\mathcal{H}|N)$ under SETH.

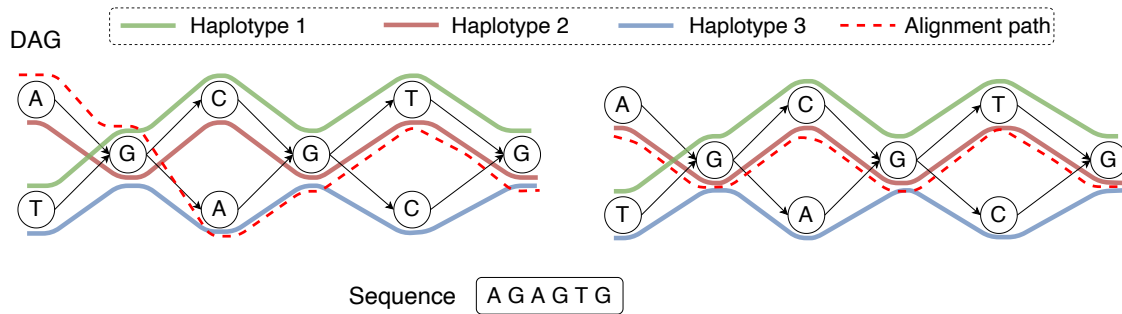


Fig. 1: An example of an acyclic pangenome graph built using three haplotypes. The left figure shows the optimal alignment of the query sequence to the DAG with minimum edit distance. Accordingly, the edit distance is zero and the count of recombinations is two. Next, suppose that we use recombination penalty -5 (formally defined in Section 1). The right figure shows the new optimal alignment, where there is no recombination because the alignment path is consistent with Haplotype 2.

We implemented the proposed chaining algorithm in Minichain and evaluated it using simulated and real sequences. We built a pangenome DAG using 60 publicly-available complete major histocompatibility complex (MHC) sequences (Li, 2022). 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’s correlation up to 0.94 between the observed recombination count and the ground-truth recombination count, whereas the correlation remained below 0.23 if recombinations are not penalized.

We also tested Minichain using publicly-available PacBio and Oxford Nanopore long reads from CHM13 human genome (Nurk et al., 2022; Vollger et al., 2020). We demonstrate a better consistency between the ground-truth haplotype (CHM13) and the selected haplotype in the output. Our experiments suggest that haplotype-aware pattern matching to pangenome graphs can be useful to improve read mapping and variant calling accuracy.

Methods

Haplotype-aware pangenome graphs and sequence alignments

First we provide the background concepts and notations before presenting the problem formulations and algorithms for haplotype-aware sequence alignment to pangenome graphs.

Haplotype-aware pangenome graph. Let $G(V, E, \sigma, \mathcal{H})$ be a DAG representing a pangenome reference. We assume that the DAG is weakly connected and $|E| \geq |V| - 1$. Let Σ denote an alphabet. Function $\sigma : V \mapsto \Sigma$ assigns a character label to each vertex. $\mathcal{H} = \{H_1, H_2, \dots, H_{|\mathcal{H}|}\}$ 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 $\{i \mid H_i \text{ includes vertex } v\}$. We assume that $haps(v) \neq \emptyset$ for all $v \in V$ and $haps(u) \cap haps(v) \neq \emptyset$ 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 sequence $Q \in \Sigma^+$, we will find its exact and approximate matches in the DAG. For brevity, we use $[m]$ to denote the set $\{1, 2, \dots, m\}$, $m \in \mathbb{N}$. Let an *alignment path* of length l in the DAG be denoted as an ordered sequence $(w_1, w_2, \dots, w_l)$, 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 sequence 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 (Backurs and Indyk, 2015). The *Orthogonal Vector (OV) Problem*

asks: given two sets of d -dimensional binary vectors A and B , $|A| = |B| =: n$, decide if there is a pair $a \in A, b \in B$ such that $a \cdot b = 0$. If strong exponential time hypothesis (SETH) (Vassilevska Williams, 2015) holds and $d = \omega(\log n)$, there is no $\epsilon > 0$ for which an $O(n^{2-\epsilon} \text{poly}(d))$ algorithm can solve the OV problem (Williams, 2005). This result is frequently used to study the hardness of other problems (Backurs and Indyk, 2015; Equi et al., 2023a,b; Gibney et al., 2022; Hoppenworth et al., 2020). For any two sequences 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 sequences 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 (Backurs and Indyk, 2015) proved the hardness of computing edit distance by designing two reductions: (i) OV to PATTERN, and (ii) PATTERN to EDIT. We recall the properties of their first reduction gadget.

Lemma 1 (Backurs and Indyk (2015)). *Given an OV instance with sets $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ comprising d -dimensional binary vectors, there exist transformations of sets A and B into two sequences 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 is a pair of 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 sequences 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 sequence S_1 from set A as f_1 . Similarly, injective function f_2 generates sequence S_2 from set B . The exact definitions of f_1 and f_2 are available elsewhere (Backurs and Indyk, 2015).

Problem formulations for haplotype-aware sequence 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 k, c_1, c_2 be user-specified thresholds, where $k \in \mathbb{N}$ and $c_1, c_2 \in \mathbb{R}_{\geq 0}$. Here c_1 indicates cost per edit and c_2 indicates cost per recombination. Determine if there is an alignment path $\mathcal{P}$ in DAG G such that $c_1 \cdot EDIT(s(\mathcal{P}), Q) + c_2 \cdot r(\mathcal{P}) < k$.

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 (Petes, 2001). In the above set of problems, we have omitted the case with exact matching and no recombination. One way to solve this case is by doing exact sequence matching against each haplotype independently.

Proposed algorithms and theoretical analysis

Lemma 2. *Problems 1-3 can be solved in $O(|Q||E||\mathcal{H}|)$ time.*

Proof. We show how to solve Problem 3 using dynamic programming. The same can be extended to Problems 1 and 2 as well. Our approach is similar to the standard sequence-to-DAG alignment algorithm (Lee et al., 2002) 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 that 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:

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

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

$$D''(i, v) = \min_{h' \in \text{haps}(v)} D(i, v, h').$$

The recursion computes the optimal value of $D(i, v, h)$ by considering the possibility of character deletion, insertion, match, and mismatch. The product $c_1 \cdot \text{EDIT}(\sigma(v), Q[i])$ equals zero if there is a match between characters $\sigma(v)$ and $Q[i]$. It equals c_1 in case of a mismatch. The other expressions involve addition of c_1 to consider the possibility of an insertion or a deletion. The expressions involving the addition of c_2 evaluate the possibility of a recombination. After updating $D(i, v, h)$ for all $h \in \text{haps}(v)$, we update $D''(i, v)$ using the above equation. We compute values in table D in the increasing order of i , and then the increasing topological order of $v \in V \cup \{v_e\}$ for a fixed i . The output, i.e., the minimum alignment cost is $\min_{v \in V \cup \{v_e\}} D''(|Q|, v)$. The runtime of the algorithm is dominated by the recursion. Solving the recursion takes $O(|Q| \sum_{v \in V \cup \{v_e\}} (|\text{haps}(v)| \cdot (\delta_{in}(v) + 1)))$ time, where $\delta_{in}(v)$ denotes the in-degree of vertex v . As a result, the time complexity of the algorithm is $O(|Q||E||\mathcal{H}|)$. $\square$

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

The existence of a faster algorithm for Problem 1 remains open.

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

Proof. Problem 3 is 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 $A_1, A_2, \dots, A_{\sqrt{n}}$, each containing $\sqrt{n}$ vectors. Similarly, we partition set B into $\sqrt{n}$ subsets $B_1, B_2, \dots, B_{\sqrt{n}}$, 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 instance (A, B) .

Next, we describe the construction of our gadget comprising query sequences $Q_1, \dots, Q_{\sqrt{n}}$ 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., $|H_1| = |H_2| = \dots = |H_{\sqrt{n}}|$. 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 (see 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

176 path that spells each of the $\sqrt{n}$ haplotype sequences in this DAG. Finally, we discard the vertices and edges
 177 that are not used by any haplotype sequence (Figure 2B).

Next, the above gadget will allow us to solve the OV problem by invoking the haplotype-aware sequence to graph alignment algorithm $\sqrt{n}$ times, once for each of the $\sqrt{n}$ query sequences. An alignment path with no recombination in the DAG spells a substring of a haplotype. Using Lemma 1, we make the following claim. There exist two orthogonal vectors $a \in A$ and $b \in B$ if and only if we have at least one query sequence $q \in \{Q_1, \dots, Q_{\sqrt{n}}\}$ for which an alignment path $\mathcal{P}$ exists such that $r(\mathcal{P}) = 0$ and $EDIT(s(\mathcal{P}), q) < \beta_1 \sqrt{n}$. 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) number of haplotypes is $|\mathcal{H}| = \sqrt{n}$, 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 $O(|Q|^{1-\epsilon}|E||\mathcal{H}| + |Q||E|^{1-\epsilon}|\mathcal{H}| + |Q||E||\mathcal{H}|^{1-\epsilon})$ time for some $\epsilon > 0$, then the OV problem can be solved in $O(n^{2-\epsilon/2} \cdot \text{poly}(d))$ time. $\square$

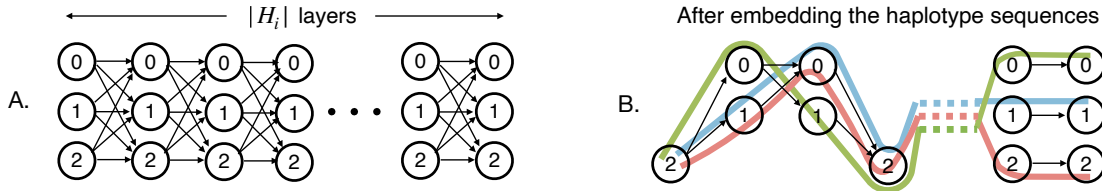


Fig. 2: An illustration of the pangenome DAG used in the reduction proof of Lemma 3.

178 Methods for haplotype-aware chaining on graphs

179 Most alignment tools use seed-chain-extend heuristic to compute the alignments quickly (Sahlin et al.,
 180 2023; Shaw and Yu, 2023). Given a set of seed matches (anchors) as input, co-linear chaining is a rigorous
 181 optimization technique to identify promising alignment regions in a reference. It identifies a coherent subset
 182 of anchors such that their coordinates are ordered on the query and the reference. The selected anchors
 183 are subsequently combined together to form an alignment. Several versions of co-linear chaining problems
 184 have been studied for aligning two sequences (Abouelhoda and Ohlebusch, 2005; Jain et al., 2022; Mäkinen
 185 and Sahlin, 2020; Myers and Miller, 1995; Eppstein et al., 1992a,b; Otto et al., 2011). Recent works have
 186 further studied the extension of chaining on acyclic (Mäkinen et al., 2019; Ma et al., 2023; Chandra and
 187 Jain, 2023; Rizzo et al., 2023a) and cyclic pangenome graphs (Antipov et al., 2016; Rajput et al., 2024) but
 188 these formulations do not consider the haplotype paths.

Concepts and notations for haplotype-aware co-linear chaining

From hereon we assume that vertices of a haplotype-aware pangenome DAG are labeled with sequences. Sequence 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 sequence-labeled form to character-labeled form, and vice versa. Let function $\sigma : V \rightarrow \Sigma^+$ label each vertex $v \in V$ with sequence $\sigma(v)$ in DAG $G(V, E, \sigma, \mathcal{H})$. The input to the chaining problem is a set of anchors $\{M[1], M[2], \dots, M[N]\}$. 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 $(s_1, s_2, \dots, s_p)$, where each s_j is a pair of the form (i, h) such that $i \in [N]$, $h \in \text{haps}(M[i].v)$ 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 (Figure 3). For the sake of efficiency, our precedence condition does not permit overlap between adjacent anchors in a chain. 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 (Mäkinen et al., 2019), 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 (Chandra and Jain, 2023; Ma et al., 2023; Mäkinen et al., 2019), identify a path cover with the minimum number of paths because the runtime of sparse dynamic programming increases with the size of the path cover (Mäkinen et al., 2019). In the haplotype-aware setting, we will directly use the paths corresponding to the haplotype sequences as path cover. Let $P_1, P_2, \dots, P_{|\mathcal{H}|}$ be the paths corresponding to the haplotype sequences $H_1, H_2, \dots, H_{|\mathcal{H}|}$ respectively. $\{P_1, P_2, \dots, P_{|\mathcal{H}|}\}$ is a path cover of the DAG because each vertex in the DAG is included in at least one haplotype (Section 1). Suppose $\text{last2reach}(v, h), v \in V, h \in [|\mathcal{H}|]$ denotes the last vertex in path P_h that belongs to $R^-(v)$. $\text{last2reach}(v, h)$ does not exist if no vertex in path P_h can reach vertex v . We precompute $\text{last2reach}(v, h)$

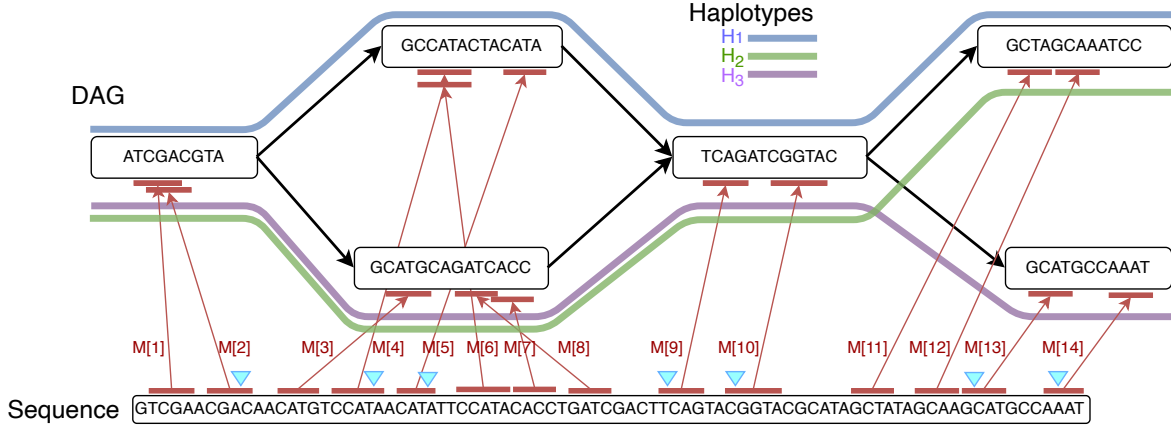


Fig.3: An example that shows anchors (in red) between a query sequence and a pangenome DAG. The DAG comprises three haplotype sequences H_1, H_2, H_3 . In this example, the ordered sequence of pairs $((2, 1), (4, 1), (5, 1), (9, 1), (10, 1), (13, 3), (14, 3))$ forms a chain and the corresponding anchors are highlighted using blue markers. This chain includes a single recombination.

for all $v \in V$ and $h \in [|\mathcal{H}|]$ in $O(|E||\mathcal{H}|)$ time by using the algorithm from Makinen *et al.* (Mäkinen *et al.*, 2019). This is a one-time preprocessing step that will be useful to solve the chaining problems efficiently. Next, we also use the following standard data structure in our chaining algorithm.

Lemma 4 (Mäkinen *et al.* (2015)). Let n be the number of leaves in a tree, each storing a $(key, value)$ pair. The data structure uses a balanced binary search tree. It 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 $\max\{value(w) \mid l < key(w) < r\}$.

Given n $(key, value)$ pairs, the tree can be constructed in $O(n \log n)$ time and $O(n)$ space.

Problem formulations for haplotype-aware co-linear chaining

– *Problem 4 (Limited recombinations)*: Determine an optimal chain $S = s_1, s_2, \dots, s_p$ 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 used in (Chandra and Jain, 2023, Problem 2a).

– *Problem 5 (Limited recombinations, gap-sensitive)*: Determine an optimal chain $S = s_1, s_2, \dots, s_p$ that maximises the chaining score, defined as $\sum_{j=1}^p \text{weight}(M[s_j.i]) - \sum_{j=2}^p \text{gap}(M[s_{j-1}.i], M[s_j.i]) - \gamma \cdot r(S)$.

Proposed chaining algorithms and theoretical analysis

First, it is useful to discuss a simple dynamic programming solution for Problem 4. Let $C(i, h)$ denote the optimal score of a chain ending at pair (i, h) , where $i \in [N], h \in \text{haps}(M[i].v)$. We use $C''(i)$ to store $\max_{h \in \text{haps}(M[i].v)} C(i, h)$. A single anchor is a valid chain of size one; therefore, we initialize $C(i, h)$ to $\text{weight}(M[i])$ for all $i \in [N], h \in \text{haps}(M[i].v)$. Also, initialize $C''(i)$ to $\text{weight}(M[i])$ for all $i \in [N]$. We update $C(i, h)$ to its optimal value using the following recursion:

$$C(i, h) = \max \left(\text{weight}(M[i]), \max_{\substack{M[k] \prec M[i], h' \in \text{haps}(M[k].v), \\ h' = h, k \in [N]}} C(k, h') + \text{weight}(M[i]) \right),$$

$$\max_{M[k] \prec M[i], k \in [N]} C''(k) + \text{weight}(M[i]) - \gamma \Big),$$

$$C''(i) = \max_{h' \in \text{haps}(M[i].v)} C(i, h')$$

If $M[k] \prec M[i]$ and path P_h covers both the vertices $M[k].v$ and $M[i].v$, then the chaining score obtained using $C(k, h)$ is considered without recombination penalty. The third term in the expression considers the score with recombination penalty γ . This is needed when (k, h') , $h' \neq h$ precedes (i, h) . After updating $C(i, h)$ for all $h \in \text{haps}(M[i].v)$, we also update $C''(i)$ using the above equation. Let function $\text{rank}(v)$ 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 $(\text{rank}(M[i].v), M[i].x)$. 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 using a linear scan would require considering all N number of anchors and filtering those which satisfy the precedence criteria. Even if we ignore the time for filtering and checking precedence, 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. To address this, we propose a faster $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time algorithm for solving Problem 4. First, we note the following inequality.

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

Proof. Consider any value of $i \in [N]$ and $h \in \text{haps}(M[i].v)$. Let $h' = \arg\max_{k \in \text{haps}(M[i].v)} C(i, k)$, breaking ties arbitrarily if necessary. 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 $S_1 = s_1, s_2, \dots, s_{p-1}, s_p$ be an optimal chain that ends at pair (i, h') . We consider another chain $S_2 = s_1, s_2, \dots, s_{p-1}, s_p'$ 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 $\geq x$. $\square$

Lemma 6. Assuming DAG $G(V, E, \sigma, \mathcal{H})$ is preprocessed, Problem 4 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 Mäkinen et al. (2019) to our haplotype-aware setting. Our path cover $\{P_1, P_2, \dots, P_{|\mathcal{H}|}\}$ contains $|\mathcal{H}|$ paths corresponding to the $|\mathcal{H}|$ haplotype sequences. We use $|\mathcal{H}|$ search trees $\mathcal{T}_1, \mathcal{T}_2, \dots, \mathcal{T}_{|\mathcal{H}|}$, 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 $C(i, h), i \in [N], h \in \text{haps}(M[i].v)$, the algorithm would have already finished processing $C(k, h')$ for all k, h' such that $M[k] \prec M[i], h' \in \text{haps}(M[k].v)$. The score $C(k, h')$ would have already been recorded in search tree $\mathcal{T}_{h'}$ (Line 26).

All the anchors that lack a preceding anchor are optimally processed at the initialization stage (Line 2). Suppose an optimal chain ending at (i, h) is $s_1, s_2, \dots, s_{p-1}, s_p$ 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 $\mathcal{T}_h$ (Line 19). Since $M[k] \prec M[i]$ and $h' = h$, search tree $\mathcal{T}_h$ 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 $\text{last2reach}(M[i].v, h')$ must exist in this case because $M[k].v$ must reach $M[i].v$. 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 $\mathcal{T}_{h'}$ (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 $\text{last2reach}(M[i].v, h')$ have been processed and search tree $\mathcal{T}_{h'}$ has been updated because of our definition and ordering of the tuples in array Z (Lines 12, 15). Later when we update $C(i, h)$ in Line 19, it gets its optimal value using $C_{l2r}(i)$.

Algorithm 1: $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time chaining algorithm for Problem 4

Input: Weighted anchors $M[1..N]$, haplotype paths $P_1, P_2, \dots, P_{|\mathcal{H}|}$, $last2reach$ 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 $\mathcal{T}_h$, 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 **if** $h \in haps(M[i].v)$ **then**
- 19 $C(i, h) \leftarrow \max(C_{l2r}(i), C(i, h), wt + \mathcal{T}_h.RMQ(0, M[i].c))$
- 20 $C''(i) \leftarrow \max(C''(i), C(i, h))$
- 21 **else**
- 22 $C_{l2r}(i) \leftarrow \max(C_{l2r}(i), wt + \mathcal{T}_h.RMQ(0, M[i].c) - \gamma)$
- 23 **else if** $z.task = 1$ **then**
- 24 $C(i, h) \leftarrow \max(C''(i) - \gamma, C(i, h))$
- 25 **else if** $z.task = 2$ **then**
- 26 $\mathcal{T}_h.update(M[i].d, C(i, h))$
- 27 **end**

283 – (Case 3) $h \neq h'$ and path $P_{h'}$ covers $M[i].v$: If $(s_1, s_2, \dots, (k, h'), (i, h))$ is an optimal chain ending at
284 (i, h) , then $(s_1, s_2, \dots, (k, h'), (i, h'))$ must be an optimal chain that ends at (i, h') . Accordingly, $C(i, h') =$
285 $C(i, h) + \gamma$. It implies $C''(i) \geq C(i, h) + \gamma$. Using the inequality in Lemma 5, we get $C(i, h) = C''(i) - \gamma$.
286 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 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 (Lemma 4). $\square$

287 In the above algorithm, the optimal chain can be obtained easily by adding backtracking pointers. We
288 next show that any algorithm to solve Problem 4 has a time complexity that is not polynomially faster than
289 $O(|\mathcal{H}|N)$ under SETH.

Lemma 7. For any constant $\epsilon > 0$, there is no algorithm that solves Problem 4 in $O(|\mathcal{H}|^{1-\epsilon}N + |\mathcal{H}|N^{1-\epsilon})$ time, unless *SETH* is false.

Proof. The reduction from the OV problem is as follows. Suppose an OV instance is given as $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$ and each set is comprised of d -dimensional binary vectors.

We perform this reduction by constructing a haplotype-aware graph based on A and a sequence based on B with anchors such that there is an anchor chain of some desired value if and only if there is an orthogonal pair of vectors. The main intuition is that an orthogonal pair of vectors should allow a high-value portion of the chain between the corresponding vector gadgets, whereas the remaining vector gadgets can be used in some portion of the chain and give a predictable value.

To this end, we first define component gadgets:

$$CG(0) = 000 \quad CG(1) = 001$$

We use $\bigcirc_{i=1}^d s_i$ to denote the concatenation of sequences $s_1, \dots, s_d$ and $s_1 \circ s_2$ to denote the concatenation of two sequence s_1 and s_2 . For a symbol c and an integer i , we use c^i to denote symbol c repeated i times. Using the above component gadgets, we define our vector gadgets:

$$VG(a_i) = \bigcirc_{x=1}^d CG(a_i[x]) \quad VG(b_i) = \bigcirc_{x=1}^d CG(b_i[x]).$$

The query sequence is defined as $Q = \bigcirc_{i=1}^n VG(b_i)$. For $1 \leq i \leq n$, we construct haplotype sequence H_i from vector a_i :

$$H_i = 0^{nd} \circ VG(a_i) \circ 0^{nd}.$$

To create the vertices of graph G we start with: a vertex denoted v_{left} with label 0^{nd} ; $2d$ vertices denoted $v_{x,0}, v_{x,1}$ for $1 \leq x \leq d$ where each $v_{x,0}$ has label $CG(0)$ and each $v_{x,1}$ has label $CG(1)$; and a vertex denoted v_{right} with label 0^{nd} . For edges, we add edges from v_{left} to $v_{1,0}$ and from v_{left} to $v_{1,1}$; for $1 \leq x < d$, we add edges from $v_{x,0}$ to $v_{x+1,0}$ and $v_{x+1,1}$, and from $v_{x,1}$ to $v_{x+1,0}$ and $v_{x+1,1}$; we then add edges from $v_{d,0}$ to v_{right} and from $v_{d,1}$ to v_{right} . Next, we embed each haplotype in G by following its only possible path from v_{left} to v_{right} and deleting any vertices and edges not supported by some haplotype. See Figure 4. We call the subgraph of G excluding only v_{left} and v_{right} the *center* of G .

The anchors are described next. For all $i \in [n]$ and for all $x \in [d]$, we define the following weight 1 anchors:

$$M_{i,x}^{left} = (v_{left}, [(i-1)d+x \dots (i-1)d+x], [3(i-1)d+3(x-1)+1 \dots 3(i-1)d+3(x-1)+1])$$

We similarly define weight 1 anchors:

$$M_{i,x}^{right} = (v_{right}, [(i-1)d+x \dots (i-1)d+x], [3(i-1)d+3(x-1)+1 \dots 3(i-1)d+3(x-1)+1])$$

For all $i \in [n]$ and for all $x \in [d]$, if $b_i[x] = 0$ and if $v_{x,0}$ and $v_{x,1}$ exist, we define the following weight 2 anchors

$$M_{i,x,0}^{center} = (v_{x,0}, [1..2], [3(i-1)d+3(x-1)+1 \dots 3(i-1)d+3(x-1)+2])$$

$$M_{i,x,1}^{center} = (v_{x,1}, [1..2], [3(i-1)d+3(x-1)+1 \dots 3(i-1)d+3(x-1)+2])$$

If either $v_{x,0}$ or $v_{x,1}$ do not exist, we omit the corresponding anchor. If $b_i[x] = 1$ and $v_{x,0}$ exists, we define the following weight 2 anchor

$$M_{i,x,0}^{center} = (v_{x,0}, [1..2], [3(i-1)d+3(x-1)+1 \dots 3(i-1)d+3(x-1)+2])$$

We set the recombination penalty $\gamma = 1$. We show the correctness of the above reduction in Lemmas 8-11. $\square$

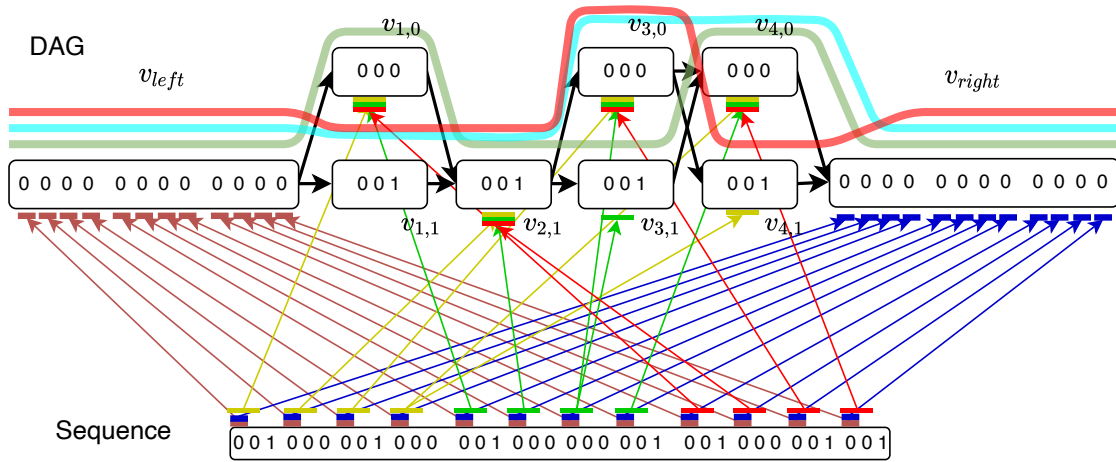


Fig. 4: Reduction from OV problem with sets $A = \{0110, 1100, 1101\}$ and $B = \{1010, 1001, 1011\}$.

Lemma 8. *If there exists an orthogonal pair $a_i \in A$ and $b_j \in B$, then there exists an anchor chain with cost $2d + (n - 1)d$.*

Proof. First take anchors $M_{j',x}^{left}$ for $1 \leq j' < j$, $1 \leq x \leq d$. This part of the chain has a value of $(j - 1)d$. Next, for $1 \leq x \leq d$, if $a_i[x] = 0$ and $b_j[x] = 0$ or $a_i[x] = 0$ and $b_j[x] = 1$ take $M_{j,x,0}^{center}$; if $a_i[x] = 1$ and $b_j[x] = 0$ take $M_{j,x,1}^{center}$. This part of the chain has a value of $2d$. Lastly, for $j' > j$, $1 \leq x \leq d$, take anchors $M_{j',x}^{right}$, this part of the chain has value $(n - j)d$. This forms a valid chain and requires no recombinations. Thus, the total score is $(j - 1)d + 2d + (n - j)d = 2d + (n - 1)d$. $\square$

Lemma 9. *If a chain exists with score $2d + (n - 1)d$, then in this chain, no recombinations are used, and anchors starting at all the anchor starting positions in Q are used.*

Proof. For a given x , only one anchor to either $v_{x,0}$ or $v_{x,1}$ can be used while maintaining the precedence condition. Since each anchor to the center of G has weight 2, the total contribution of the center of G to the score is at most $2d$. The remaining anchors in the chain have a weight of 1, and there are at most $(n - 1)d$ of them after d anchors are used in the center. Hence, the maximum possible score is $2d + (n - 1)d$. Notice also that taking d anchors in the center is the only way to achieve such a score.

The claim that there are no recombinations follows, as the maximum possible score using recombination becomes $2d + (n - 1)d - 1$. The second claim follows since any chain using fewer than dn anchors can not achieve the maximum score of $2d + (n - 1)d$. $\square$

Lemma 10. *If a chain exists with the score $2d + (n - 1)d$, then in this chain, all anchors for some vector gadget in Q are to the center of G .*

Proof. If anchors from two different vector gadgets in Q are to the center of G , then since the center is contributing $2d$ to the score (the only way a score of $2d + (n - 1)d$ is achieved), there must be anchors M and M' from two different vector gadgets that occur in adjacent vertices in the center, say with $M \prec M'$. However, this implies that there are at least $d - 1$ anchor positions in Q between M and M' are not used in the solution, contradicting Lemma 9. $\square$

Lemma 11. *If there exists a chain with score $2d + (n - 1)d$, then there exists orthogonal vectors a_i and b_j .*

Proof. By Lemma 10, some vector gadget $VG(b_j)$ has all of its anchors going to the center of G . By Lemma 9, no recombinations are used, and these must lie on some path for a haplotype constructed from a vector a_i . By design of the component gadgets, for $1 \leq x \leq d$ if $a_i[x] = 1$ we get anchor between $v_{x,1}$ on the haplotype path for a_i and $VG(b_j)$ if and only if $b_j[x] = 0$. We conclude that a_i and b_j are orthogonal. $\square$

Observe that the total number of M^{left} anchors is dn , the total number of M^{right} anchors is dn , and the total number of M^{center} anchors is at most $2dn$, thus $N = O(dn)$. The number of haplotypes is $|\mathcal{H}| = n$. Furthermore, $|Q| = O(dn)$ and the graph G has $O(d)$ vertices and edges, and the sum of vertex label lengths is $O(dn)$. Hence, the reduction takes $O(dn)$ time. Combined with Lemmas 8 and 11, it follows that an algorithm for Problem 4 running in time $O(|\mathcal{H}|^{1-\epsilon}N + |\mathcal{H}|N^{1-\epsilon})$, solves the OV problem in time $O(n^{2-\epsilon} \cdot \text{poly}(d))$. This completes the proof of Lemma 7.

Problem 5 can be solved by extending Algorithm 1 (Supplemental Note S1). The extension requires a few additional terms corresponding to gap cost between adjacent anchors. This algorithm also runs in $O(|\mathcal{H}|N \log |\mathcal{H}|N)$ time. The reduction presented in Lemma 7 can be easily extended for Problem 5 as well.

Implementation and benchmarking details

Software implementation. We implemented Algorithm 1 and the modifications to include gap penalty (described in Supplemental Note S1). We added our implementation in the Minichain code (Chandra and Jain, 2023). Our new chaining implementation replaces the haplotype-agnostic chaining implementation proposed in our previous work. Minichain supports an end-to-end seed-chain-extend workflow to align long reads and contigs to acyclic pangenome graphs. Similar to the previous version of the software, we use the seeding and base-to-base alignment routines from Minigraph (Li, 2018). We leave the implementation of our $O(|Q||E||\mathcal{H}|)$ time base-to-base alignment algorithm (Section 1) as future work because it requires further practical optimizations to reduce the runtime, e.g., using banded alignment or wavefront heuristics (Marco-Sola et al., 2021; Rautiainen and Marschall, 2020; Zhang et al., 2022).

In Minichain, we parse the set of vertices, edges, and haplotype paths from a graphical fragment assembly (GFA) file (Jackman et al., 2015). 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 graph, 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 (Roberts et al., 2004) 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 based on our experimental observations. Thus for $k = 17$, each anchor in a chain contributes 3400 to the total chain score. The multiplicative factor of 200 prevents gap penalties to dominate the overall chain score. Minichain can report multiple best scoring chains and assign a mapping quality (Li et al., 2008) to each chain; the criteria

for this remains same as before (Chandra and Jain, 2023). We evaluate the chains produced by our algorithm without subjecting it to further Minigraph’s heuristics.

Pangenome graph construction. We used 60 publicly-available complete major histocompatibility complex (MHC) haplotype sequences (Li, 2022). This set of 60 MHC haplotype sequences comprises the CHM13 MHC haplotype (Nurk et al., 2022) and 59 other haplotypes from the Human Pangenome Reference Consortium (Liao et al., 2023). 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 (Dilthey, 2021). We used Minigraph (v0.20-r559) (Li et al., 2020) to build an acyclic pangenome graph using the 60 MHC haplotypes. The graph construction algorithm in Minigraph augments structural variants (SVs) of size ≥ 50 bp in the graph. Our MHC pangenome graph contains 984 vertices, 1387 edges, and 210 SVs. Since Minigraph does not output haplotype paths in the graph, we realigned each haplotype to the graph to obtain the corresponding paths.

Simulation of query sequences. We simulated 135 MHC haplotype sequences. Each sequence was simulated as an imperfect mosaic of the 60 reference haplotypes (Figure 5). We used the following procedure for simulation: (i) Select a random haplotype path in the graph, (ii) Read the first l bases from the chosen haplotype path in the graph, (iii) If the $(l+1)^{th}$ base is in vertex v and $|haps(v)| \geq 2$, 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. Our pangenome graph and the set of query sequences are available on Zenodo (DOI: 10.5281/zenodo.10663934).

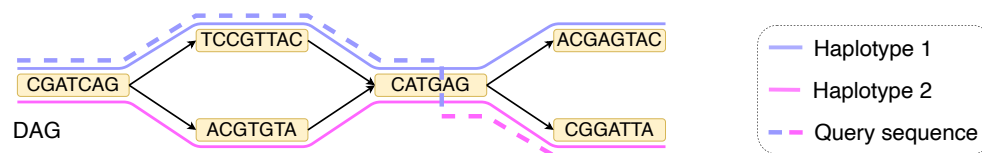


Fig. 5: An example to illustrate simulation of a query sequence as a mosaic of reference haplotypes. In this example, $l = 19$.

Results

Comparison with haplotype-agnostic chaining using simulated data

We demonstrate that integrating recombination penalty in the sequence-to-graph chaining 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 (Chandra and Jain, 2023). Recombination penalty $\gamma = \infty$ corresponds to haplotype-restricted chaining, where a chain can use only one of the reference haplotypes. We evaluated Minichain by using simulated MHC haplotype sequences and 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 haplotype indices. Thus, we know the sequence of haplotype indices and the number of recombinations in each chain.

We show the Pearson’s correlation coefficients between the observed number of recombinations and the true number of recombinations using different values of substitution rates and recombination penalties (Figure 6A-C). 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. In the haplotype-agnostic setting ($\gamma = 0$), a highest-scoring chain of minimizers can contain incorrect haplotype switches (see Supplemental Figure S1 for an example). $\gamma = 10^3$ is also too low for setting recombination penalty because a single anchor in a chain contributes 3400 to the chaining score (Methods), and the algorithm may still favor an additional anchor even if it involves an incorrect haplotype switch. Using $\gamma \geq 10^4$, the recombination penalty equals the score contributed by three or more anchors.

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 $H_{i_1}, H_{i_2}, \dots, H_{i_n}$, then we recorded the “true haplotype recombination pairs” as a set of pairs $\{(start, H_{i_1}), (H_{i_1}, H_{i_2}), \dots, (H_{i_{n-1}}, H_{i_n}), (H_{i_n}, end)\}$. 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 7A-C, Supplemental Figure S4, Supplemental Table S1). In this experiment, precision was determined by the fraction of observed recombination pairs that were in the true set. Similarly, recall was determined by the fraction of true recombination pairs that were observed in the output. We achieved higher F1-scores with

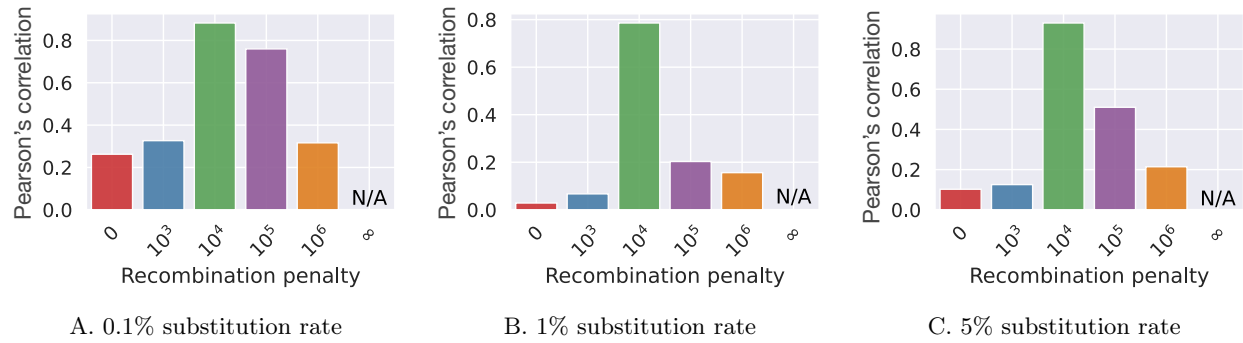


Fig. 6: Pearson's correlation between the number of recombinations in Minichain's output chain and the true count. We evaluated the performance by using different substitution rates and recombination penalties.

414 $\gamma = 10^4, 10^5$ compared to the haplotype-agnostic ($\gamma = 0$) and haplotype-restricted ($\gamma = \infty$) modes. These
 415 results suggest that haplotype-aware chaining and alignment algorithms can be useful to prevent spurious
 416 recombinations.

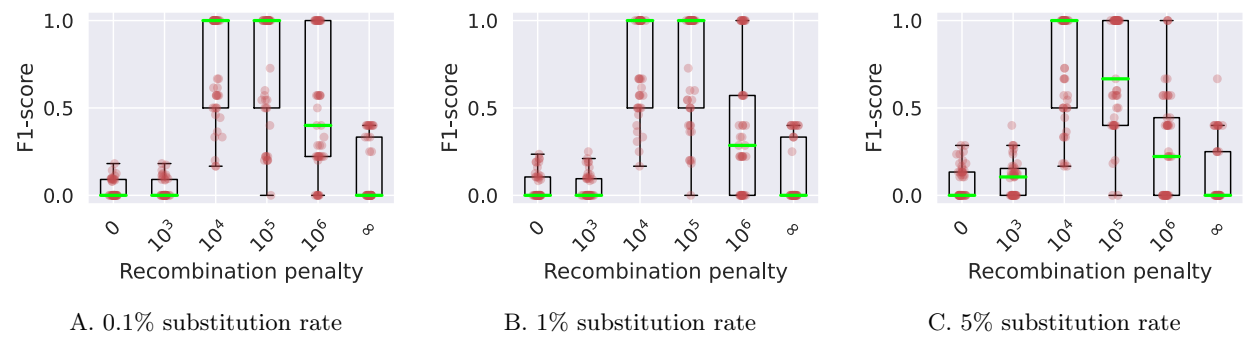


Fig. 7: Box plots show the levels of consistency between the haplotype recombination pairs in Minichain's output chain and the ground-truth using three different sets of simulated MHC sequences with substitution rates (a) 0.1%, (b) 1%, and (c) 5%. We tested using different recombination penalties. Each red dot in the plots corresponds to a query sequence. The median values are highlighted in green.

417 We restricted the maximum count of recombinations in the simulated MHC haplotypes to 4 in the above
 418 experiments because MHC sequences are known to have long-range haplotype structures (Dilthey, 2021). We
 419 performed additional experiments with higher number of recombination events (range: 3 to 9) and reached
 420 similar conclusions (Supplemental Figures S2 and S3).

421 Our MHC pangenome graph comprises SVs only. We expect further improvements in the accuracy of
 422 our algorithm if substitution and short indel variation are also augmented in a pangenome graph (Hickey
 423 et al., 2023). This is because more variants will help to distinguish between near-identical or closely-related
 424 haplotypes. Doing this will also require improvements in Minichain's seeding and chaining implementation

to allow the use of anchors that cover small bubbles in a graph. This is possible by considering a more flexible definition of anchors that can span multiple vertices (Rizzo et al., 2023b,a; Rossi et al., 2022). Our implementation uses minimizers which also have a known weakness of being context-dependent, i.e., the decision to sample a k -mer is made based on its neighboring k -mers (Edgar, 2021).

The runtime of Minichain ranged from 5 to 11 minutes for aligning a simulated MHC sequence to the pangenome graph. 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, whereas the haplotype-agnostic chaining algorithms use a path cover of minimum size (Chandra and Jain, 2023; Ma et al., 2023; Mäkinen et al., 2019). For example, our graph with 60 haplotype paths has a minimum path cover of size 5. Thus, we observed that our haplotype-aware chaining algorithm in Minichain was about $10\times$ slower compared to the haplotype-agnostic chaining algorithm (Chandra and Jain, 2023). Space requirements of our algorithm are also proportional to the count of haplotype paths (e.g., to store the score table C). As a result, we observed that the haplotype-aware algorithm required about $15\times$ more memory compared to the haplotype-agnostic algorithm (Chandra and Jain, 2023).

Evaluation using real data

We evaluated Minichain by using real long-read sequencing datasets from CHM13 human genome. Our MHC pangenome graph includes CHM13 as one of the 60 haplotypes. Since the CHM13 genome is effectively haploid, we expect all chains to be consistent with the CHM13 haplotype path in the graph. We used two PacBio HiFi datasets (SRX5633451, SRR11292121) and one Oxford Nanopore dataset (SRR23365080). We filtered the reads sampled from MHC region by mapping the reads to T2T-CHM13v2.0 genome reference (GCF_009914755.1) using minimap2 v2.26 (Li, 2018). Next, we discarded reads of length ≤ 1 kbp. After these filtering steps, the N50 read lengths of the three datasets were 11 kbp, 20 kbp, and 83 kbp, respectively. We considered the chaining output of a read as correct if all the anchors in the highest scoring chain overlapped with the CHM13 haplotype path in the pangenome graph.

The absolute fractions of correctly chained reads using the haplotype-agnostic objective ($\gamma = 0$) on the three datasets were 97.3%, 96.1%, and 91.1%, respectively. Using recombination penalty $\gamma = 10^5$, the fraction improved to 97.8%, 97.2%, and 94.2% respectively. Relatively speaking, the fraction of incorrect chains reduced by 19%, 28%, and 35%, respectively.

These results, although preliminary, suggest that the proposed haplotype-aware sequence-to-graph alignment algorithms can be useful for genotyping and variant calling using pangenome references.

Discussion

Pangenome graph has emerged as a powerful alternative representation as a reference during resequencing of a genome. These graphs can be represented in either a haplotype-agnostic or haplotype-aware manner. The former excludes the information of the reference haplotypes, whereas the latter incorporates it. Leveraging the haplotype path information during read-to-graph or contig-to-graph mapping can be useful to compute accurate alignments and avoid unlikely haplotype recombinations. This will become even more important in near future as more reference-quality human genomes are produced and included in a human pangenome graph. Prior work on haplotype-aware mapping has focused on restricting the recombination count to either zero or one (Mokveld et al., 2020; Sirén et al., 2021, 2020).

In this paper, we showed how the routine chaining and alignment tasks can be rigorously formulated and solved in a haplotype-aware manner. Our formulations focus on optimizing the standard metrics, e.g., edit distance for alignment and chaining score for co-linear chaining, while considering the switches used between the reference haplotypes. Thus, the proposed algorithms will help address the combinatorial challenges as more variants are included in a graph. The other key theoretical contribution is that we proved that our alignment and chaining algorithms are near-optimal assuming SETH. Subsequently, we demonstrated the practical value of our haplotype-aware chaining algorithm. We developed a proof-of-concept implementation in Minichain (v1.3). Our experiments show that the algorithm produces alignment with fewer false recombinations and better selection of the correct haplotypes.

More work is needed, especially to establish that the proposed haplotype-aware mapping algorithms can lead to improvements in routine genomic applications. This requires a more careful engineering of Minichain to make it useful for graphs that comprise SNPs and indels. The current implementation of Minichain is restricted to acyclic pangenome graphs that contain structural variation only. Lastly, it will also be interesting to investigate whether the genotype imputation accuracy can be improved using the proposed algorithms on low-pass whole-genome sequencing data.

Software availability

The Minichain aligner is available at <https://github.com/at-cg/minichain>. The scripts and datasets to reproduce the results are available at <https://github.com/at-cg/minichain/tree/master/data/v1.3> and Zenodo (DOI: 10.5281/zenodo.10665350) respectively.

Competing interest statement

The authors report no competing interests.

Acknowledgements

This work is supported by funding from the National Supercomputing Mission India, DBT/Wellcome Trust India Alliance Fellowship (grant number IA/I/23/2/506979), and the Intel India Research Fellowship. We used computing resources at the C-DAC National PARAM Supercomputing Facility 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 performed the experiments.

References

- Abouelhoda M and Ohlebusch E. 2005. Chaining algorithms for multiple genome comparison. *Journal of Discrete Algorithms* **3**: 321–341.
- Amir A, Lewenstein M, and Lewenstein N. 2000. Pattern matching in hypertext. *Journal of Algorithms* **35**: 82–99.
- Antipov D, Korobeynikov A, McLean JS, and Pevzner PA. 2016. hybridspades: an algorithm for hybrid assembly of short and long reads. *Bioinformatics* **32**: 1009–1015.
- Avila Cartes J, Bonizzoni P, Ciccolella S, Della Vedova G, Denti L, Monti D, Pirola Y, and Porto F. 2022. Recgraph: adding recombinations to sequence-to-graph alignments. *bioRxiv* pp. 2022–10.
- Backurs A and Indyk P. 2015. 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.
- Chandra G and Jain C. 2023. Gap-sensitive colinear chaining algorithms for acyclic pangenome graphs. *Journal of Computational Biology* **30**: 1182–1197.
- Cheng H, Concepcion GT, Feng X, Zhang H, and Li H. 2021. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nature methods* **18**: 170–175.
- Dilthey AT. 2021. State-of-the-art genome inference in the human MHC. *The International Journal of Biochemistry & Cell Biology* **131**: 105882.
- 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. *Nature genetics* **54**: 518–525.
- Edgar R. 2021. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ* **9**: e10805.
- Eizenga JM, Novak AM, Sibbesen JA, Heumos S, Ghaffaari A, Hickey G, Chang X, Seaman JD, Rounthwaite R, Ebler J, et al.. 2020. Pangenome graphs. *Annual review of genomics and human genetics* **21**: 139.
- Eppstein D, Galil Z, Giancarlo R, and Italiano GF. 1992a. Sparse dynamic programming i: linear cost functions. *Journal of the ACM* **39**: 519–545.
- Eppstein D, Galil Z, Giancarlo R, and Italiano GF. 1992b. Sparse dynamic programming ii: convex and concave cost functions. *Journal of the ACM* **39**: 546–567.
- Equi M, Mäkinen V, and Tomescu AI. 2023a. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless seth fails. *Theoretical Computer Science* **975**: 114128.

- Equi M, Mäkinen V, Tomescu AI, and Grossi R. 2023b. On the complexity of string matching for graphs. *ACM Transactions on Algorithms* **19**: 1–25.
- Garrison E, Guarracino A, Heumos S, Villani F, Bao Z, Tattini L, Hagmann J, Vorbrugg S, Marco-Sola S, Kubica C, et al.. 2023. Building pangenome graphs. *bioRxiv* pp. 2023–04.
- Garrison E, Sirén J, Novak AM, Hickey G, Eizenga JM, Dawson ET, Jones W, Garg S, Markello C, Lin MF, et al.. 2018. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature biotechnology* **36**: 875–879.
- Gibney D, Thankachan SV, and Aluru S. 2022. On the hardness of sequence alignment on de bruijn graphs. *Journal of Computational Biology* **29**: 1377–1396.
- Grytten I, Dagestad Rand K, and Sandve GK. 2022. Kage: Fast alignment-free graph-based genotyping of snps and short indels. *Genome Biology* **23**: 209.
- Hickey G, Monlong J, Ebler J, Novak AM, Eizenga JM, Gao Y, Marschall T, Li H, and Paten B. 2023. Pangenome graph construction from genome alignments with minigraph-cactus. *Nature Biotechnology* pp. 1–11.
- Hoppenworth G, Bentley JW, Gibney D, and V Thankachan S. 2020. The fine-grained complexity of median and center string problems under edit distance. In *28th Annual European Symposium on Algorithms, ESA 2020*.
- Jackman S et al.. 2015. Graphical Fragment Assembly (GFA) Format Specification. <https://github.com/GFA-spec/GFA-spec>. [Online; accessed 7-Feb-2024].
- Jain C, Gibney D, and Thankachan SV. 2022. Co-linear chaining with overlaps and gap costs. In *International Conference on Research in Computational Molecular Biology (RECOMB)*, pp. 246–262. Springer.
- Jain C, Tavakoli N, and Aluru S. 2021. A variant selection framework for genome graphs. *Bioinformatics* **37**: i460–i467.
- Jain C, Zhang H, Gao Y, and Aluru S. 2020. On the complexity of sequence-to-graph alignment. *Journal of Computational Biology* **27**: 640–654.
- Lee C, Grasso C, and Sharlow MF. 2002. Multiple sequence alignment using partial order graphs. *Bioinformatics* **18**: 452–464.
- Li H. 2018. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* **34**: 3094–3100.
- Li H. 2022. Sample graphs and sequences for testing sequence-to-graph alignment.
- Li H, Feng X, and Chu C. 2020. The design and construction of reference pangenome graphs with minigraph. *Genome Biology* **21**.

- Li H, Marin M, and Farhat MR. 2024. Exploring gene content with pangenome gene graphs. *arXiv preprint arXiv:2402.16185* .
- Li H, Ruan J, and Durbin R. 2008. Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome research* **18**: 1851–1858.
- Li N and Stephens M. 2003. Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics* **165**: 2213–2233.
- 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.
- Ma J, Cáceres M, Salmela L, Mäkinen V, and Tomescu AI. 2023. Chaining for accurate alignment of erroneous long reads to acyclic variation graphs. *Bioinformatics* **39**: btad460.
- Mäkinen V, Belazzougui D, Cunial F, and Tomescu AI. 2015. *Genome-scale algorithm design*. Cambridge University Press.
- Mäkinen V and Sahlin K. 2020. Chaining with overlaps revisited. In *31st Annual Symposium on Combinatorial Pattern Matching (CPM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.
- Mäkinen V, Tomescu AI, Kuosmanen A, Paavilainen T, Gagie T, and Chikhi R. 2019. Sparse dynamic programming on DAGs with small width. *ACM Transactions on Algorithms* **15**: 1–21.
- Manber U and Wu S. 1992. Approximate string matching with arbitrary costs for text and hypertext. In *Advances In Structural And Syntactic Pattern Recognition*, pp. 22–33. World Scientific.
- Marco-Sola S, Moure JC, Moreto M, and Espinosa A. 2021. Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics* **37**: 456–463.
- Mokveld T, Linthorst J, Al-Ars Z, Holstege H, and Reinders M. 2020. Chop: haplotype-aware path indexing in population graphs. *Genome biology* **21**: 1–16.
- Myers G and Miller W. 1995. Chaining multiple-alignment fragments in sub-quadratic time. In *SODA*, volume 95, pp. 38–47.
- Navarro G. 2000. Improved approximate pattern matching on hypertext. *Theoretical Computer Science* **237**: 455–463.
- Nurk S, Koren S, Rhie A, Rautiainen M, et al.. 2022. The complete sequence of a human genome. *Science* **376**: 44–53.
- Otto C, Hoffmann S, Gorodkin J, and Stadler PF. 2011. Fast local fragment chaining using sum-of-pair gap costs. *Algorithms for Molecular Biology* **6**: 4.
- Petes TD. 2001. Meiotic recombination hot spots and cold spots. *Nature Reviews Genetics* **2**: 360–369.

- Pritt J, Chen NC, and Langmead B. 2018. Forge: prioritizing variants for graph genomes. *Genome biology* **19**: 1–16.
- Rajput J, Chandra G, and Jain C. 2024. Co-linear chaining on pangenome graphs. *Algorithms for Molecular Biology* **19**: 4.
- Rautiainen M, Mäkinen V, and Marschall T. 2019. Bit-parallel sequence-to-graph alignment. *Bioinformatics* **35**: 3599–3607.
- Rautiainen M and Marschall T. 2020. Graphaligner: rapid and versatile sequence-to-graph alignment. *Genome biology* **21**: 1–28.
- Rautiainen M, Nurk S, Walenz BP, Logsdon GA, Porubsky D, Rhie A, Eichler EE, Phillippy AM, and Koren S. 2023. Telomere-to-telomere assembly of diploid chromosomes with verkko. *Nature Biotechnology* pp. 1–9.
- Rizzo N, Cáceres M, and Mäkinen V. 2023a. 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.
- Rizzo N, Cáceres M, and Mäkinen V. 2023b. Finding Maximal Exact Matches in Graphs. In *23rd International Workshop on Algorithms in Bioinformatics (WABI 2023)* (eds. D Belazzougui and A Ouangraoua), volume 273 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 10:1–10:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- Roberts M, Hayes W, Hunt BR, Mount SM, and Yorke JA. 2004. Reducing storage requirements for biological sequence comparison. *Bioinformatics* **20**: 3363–3369.
- Rossi M, Oliva M, Langmead B, Gagie T, and Boucher C. 2022. Moni: a pangenomic index for finding maximal exact matches. *Journal of Computational Biology* **29**: 169–187.
- Sahlin K, Baudeau T, Cazaux B, and Marchet C. 2023. A survey of mapping algorithms in the long-reads era. *Genome Biology* **24**: 1–23.
- Shaw J and Yu YW. 2023. 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**: 1175–1187.
- Singh V, Pandey S, and Bhardwaj A. 2022. From the reference human genome to human pangenome: Premise, promise and challenge. *Frontiers in Genetics* **13**: 1042550.
- Sirén J, Garrison E, Novak AM, Paten B, and Durbin R. 2020. Haplotype-aware graph indexes. *Bioinformatics* **36**: 400–407.

- 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.
- Tavakoli N, Gibney D, and 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*, pp. 1–9.
- Vassilevska Williams V. 2015. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
- Vollger MR, Logsdon GA, Audano PA, Sulovari A, Porubsky D, Peluso P, Wenger AM, Concepcion GT, Kronenberg ZN, Munson KM, et al.. 2020. Improved assembly and variant detection of a haploid human genome using single-molecule, high-fidelity long reads. *Annals of human genetics* **84**: 125–140.
- Wang T, Antonacci-Fulton L, Howe K, et al.. 2022. The human pangenome project: a global resource to map genomic diversity. *Nature* **604**: 437–446.
- Wenger AM, Peluso P, Rowell WJ, Chang PC, Hall RJ, Concepcion GT, Ebler J, Functammasan A, Kolesnikov A, Olson ND, et al.. 2019. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature biotechnology* **37**: 1155–1162.
- Williams R. 2005. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science* **348**: 357–365.
- Zhang H, Wu S, Aluru S, and Li H. 2022. Fast sequence to graph alignment using the graph wavefront algorithm. *arXiv preprint arXiv:2206.13574* .