Revisiting $O(n \log \log n)$ Chaining for Anchored Edit Distance

Nicola Rizzo
University of Helsinki, Finland

Ragnar Groot Koerkamp
Karlsruhe Institute of Technology, Germany

Abstract

Colinear chaining is a classical heuristic for sequence alignment: it enables scalable genome comparison and is a main component of many state-of-the-art read mappers based on seed-chain-extend. The earliest $O(n \log \log n)$ time algorithms by Eppstein et al. (J. ACM, 1992) chained n fragments between two sequences T and Q while minimizing a gap cost based on the diagonal distance Δ_{diag} between consecutive fragments. They also forbid fragment overlaps, which are essential in current chaining formulations: in long-read mapping, overlaps improve sensitivity and avoid restrictions on the fragment class considered. Jain, Gibney, and Thankachan (J. Comput. Biol. 2022) recently combined a $\Delta_{\text{diag}} = |\Delta_T - \Delta_Q|$ overlap cost with the classic $L_\infty = \max(\Delta_T, \Delta_Q)$ gap cost that takes the maximum between the horizontal and vertical gap between the fragments and they proved that chaining under this cost model is equivalent to the *anchored edit distance*.

We improve the existing $O(n \log^3 n)$-time algorithm for anchored edit distance to $O(n \log \log n)$ time in $O(n)$ space, by combining the gap-cost computation of Chao and Miller (Algorithmica, 1995) with the overlap-cost computation of Baker and Giancarlo (ESA, 1998). By developing `llchain`, a simpler $O(n \log n)$-time implementation of our method, we show how chaining algorithms that might have been recently overlooked by the bioinformatics community scale competitively to millions of fragments and large genomes. On average, `llchain` is 10× faster than other methods on instances with 3 000 000 anchors, and over 3× faster on MEMs between HiFi reads and a reference human genome.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Theory of computation → Sorting and searching; Applied computing → Genomics

Keywords and phrases Colinear chaining, Anchored edit distance, Sequence alignment, Predecessor structure

Supplementary Material *Software*: `https://github.com/nrizzo/llchain`

Acknowledgements The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI) for supporting this project with computational and data storage resources.

1 Introduction

Colinear chaining, known throughout the decades as *Wilbur–Lipman sequence comparison*, *global fragment chaining*, *sparse dynamic programming*, and just *chaining* is a fundamental technique for approximate alignment that dates back to the 1980s [42, 43, 17, 30]. Chaining techniques have been widely adopted in different sub-fields of computational genomics: whole-genome alignment tools [31]; sequence aligners [22, 23, 15]; read mappers employing the seed-chain-extend strategy [39]; and even assembly evaluation [7, 26]. While the general chaining problem statement is consistent in the literature—to find the highest-scoring ordered subset of the input fragments that is consistent with an alignment—the exact formulation and algorithms change depending on the application, and it can be difficult to compare the different results. In particular, the behaviour of chaining is mainly affected by fragment score, gap cost, and overlap cost. Popular aligners use a heuristic chaining cost based on the diagonal distance and allow overlaps [23, 35, 19], whereas some methods chain without

overlaps and consider a function of the gaps in the two sequences [1, 32], in one case obtaining a *lower bound* on the cost of the optimal alignment [15]. Instead, we consider the *anchored edit distance* [3, 24, 20], which gives an *upper* bound on the true edit distance. In this model, a number of *anchors* or *fragments* are given, which are exact matches between the two input sequences. Then, the anchored edit distance is the minimum cost of a global alignment where only the matches supported by an anchor are free.

▶ **Definition 1** (Anchored edit distance). *Given two strings T and Q and a set of n exactly matching* anchors *(or* fragments*) $a_i = (t_s, t_e, q_s, q_e)$ with $T[t_s, t_e) = Q[q_s, q_e)$, the* anchored edit distance *is the cost of a global alignment of T and Q that minimizes the total number of insertions, deletions, substitutions, and* matches not supported by an anchor.

Jain et al. [20] show that the anchored edit distance can be found by computing a minimum-cost chain using an $O(n \log^3 n)$ algorithm, and Rizzo et al. [37] simplified the chaining cost as follows.

▶ **Theorem 2** (Equivalence to chaining). *The anchored edit distance equals the minimum-cost chain of fragments under the following colinear chaining cost between the end $E = (t_e, q_e)$ of one anchor a and the start $S' = (t'_s, q'_s)$ of the next anchor a':*

$$\mathsf{connect}(a, a') := \max(L_\infty(E, S'), \Delta_{diag}(E, S')),$$

where

$$L_\infty((t,q),(t',q')) := \max(\Delta_T, \Delta_Q) = \max(t'-t, q'-q) \quad \textit{is the maximum gap, and}$$
$$\Delta_{diag}((t,q),(t',q')) := |\Delta_T - \Delta_Q| = |\operatorname{diag}(t,q) - \operatorname{diag}(t',q')| \quad \textit{is the diagonal distance.}$$

Contributions. Our main contribution is to connect the anchored edit distance to previously studied chaining problems, resulting in an $O(n \log \log n)$ theoretical algorithm working in $O(n)$ space. Our method works by combining the L_∞ gap-cost computation in $O(n \log \log n)$ time of Eppstein et al. [13] and Chao and Miller [9] with the Δ_{diag} overlap cost computation of Baker and Giancarlo [3]. The $O(n \log \log n)$ time complexity is obtained by using a fast integer sorting algorithm as well as predecessor structures. A simplified $O(n \log n)$ time implementation, `llchain`, performs up to 10× faster (3× on average) than the existing heuristics for anchored edit distance, when chaining the MEMs between HiFi reads and a human genome.

2 Related work

To provide the needed historical context to our work and to help the reader navigate the literature, we start with a review of the most important results on pairwise chaining focusing on the time complexity in the number of input anchors, summarized in Table 1.

The original chaining formulations by Wilbur and Lipman [43] considered as input exact-match fragments between two sequences T and Q, to be selected and connected together in an order that is consistent with an alignment (i.e. a *chain*) while maximizing the number of matched bases minus a constant cost for each gap [42], or maximizing *weighted* matches minus an arbitrary gap cost $g(\Delta_T, \Delta_Q)$ [43], where Δ_Q and Δ_T are the length of the gap in T and Q (see Figure 1). They provided an $O(n^2)$ time algorithm. In 1992, the first efficient sub-quadratic-time algorithms were developed by Eppstein et al. [13, 14] to maximize the number of matched bases minus gap costs computed as $f(\Delta_{\text{diag}})$, a one-dimensional function in the diagonal distance between consecutive fragments, and they run

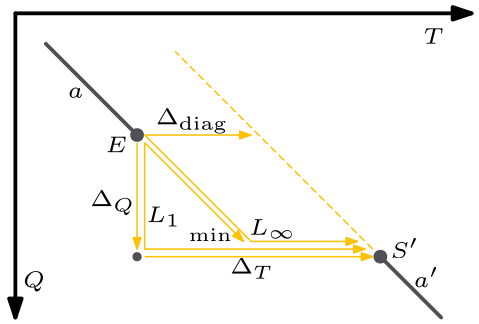

Figure 1 Different possible chaining costs between two anchors a and a': The distance is between the end E of a and start S' of a', and can be the distance in Q (Δ_Q), the distance in T (Δ_T), or the sum (L_1), minimum (min), maximum (L_∞), or absolute difference ($\Delta_{\text{diag}} = |\Delta_T - \Delta_Q|$) of these.

in $O(n \log \log n)$ or $O(n \log n)$ time, depending on whether f is linear or concave. Gap costs of the form $f(\Delta_{\text{diag}})$ are still widely used today [23, 35] but they do not explicitly penalize large gaps. This was addressed in efficient algorithms in 1995[1] by Myers and Miller [28] when generalizing chaining to two or more sequences, and by Chao and Miller [9] when extending the approach by Eppstein et al. to find m best-scoring local alignments. Myers and Miller's gap cost $\varepsilon \cdot \min(\Delta_T, \Delta_Q) + \lambda \cdot \Delta_{\text{diag}}$ found a lot of attention in the alignment of whole genomes [31], whereas pairwise chaining implementations like `CHAINER` [1] concentrated on the $L_1 = \Delta_T + \Delta_Q$ gap cost (i.e. $(\varepsilon, \lambda) = (2, 1)$). To the best of our knowledge, the first chaining implementation to support generic ε and λ—and thus $L_\infty = \max(\Delta_T, \Delta_Q)$ for $(\varepsilon, \lambda) = (1, 1)$—is `clasp` [32]. All chaining formulations before 1998, and many formulations since, do not allow anchors to overlap (see e.g. Figure 2): historically overlaps have been ignored, removed in a pre-processing step, or recovered in downstream refinement phases [10, 8]. However, overlaps improve sensitivity and are widely employed in long-read alignment, where the chaining formulation of `minimap2` [23] is the *de facto* standard way of chaining minimizer seeds, a popular class of sampled fixed-length seeds [39]. Other notable applications of chaining include the evaluation of genome assemblies [7, 26], chaining with inversions [35], and computing an admissible lower bound for the A*PA heuristic to sequence alignment [15].

A complementary problem is that of *anchored longest common subsequence* (ancored LCS), where one searches an alignment that maximizes the number of matches supported by an anchor. Baker and Giancarlo [3] showed in 1998 that chaining to minimize an $L_1 = \Delta_T + \Delta_Q$ gap cost and a Δ_{diag} overlap cost actually solves the anchored LCS, and solved the problem in $O(n \log \log n)$ time. The connection to LCS was rediscovered in 2021 by Mäkinen and Sahlin [24] with an alternative but equivalent chaining formulation. Further work in this direction includes LCSk [6, 5], where anchors are the set of all k-mer matches and may not overlap, and LCSk+ [34], where anchors are all matches of length at least k. In [33], an $O(n \lg \ell)$ algorithm is given, where $\ell \leq n$ is the number of anchors in the optimal chain.

[1] Zhang et al. [45] define an affine gap score for chaining fragments between $k \geq 2$ sequences already in 1994, but they address the problem with k-dimensional trees and do not provide a worst-case complexity analysis.

chaining formulation	fragment type	fragment score	gap cost	overlap cost	time complexity
Wilbur and Lipman, 1983 [42]	exact	length	constant	–	$O(n^2)$
Wilbur and Lipman, 1984 [43]	square	context-dependent	$f(\Delta_T, \Delta_Q) \geq 0$	–	$O(n^2)$
Eppstein et al., 1992 [13]	exact	matches	$\ell(\Delta_{\text{diag}})$	$\Delta_{\text{diag}} = 0$ penalty	$O(n \log\log n)$
Eppstein et al., 1992 [14]	exact	matches	$f_\cup(\Delta_{\text{diag}})$	$\Delta_{\text{diag}} = 0$ penalty	$O(n \log n)$
Myers and Miller, 1995 [28]	inexact	arbitrary	$\varepsilon \cdot \min(\Delta_T, \Delta_Q) + +\lambda \cdot \lvert\Delta_T - \Delta_Q\rvert$	–	$O(n \log^2 n)$
Chao and Miller, 1995 [9]	exact	length	affine	–	$O(n \log\log n)$
Baker and Giancarlo, 1998 [3]	exact	–	$L_1 = \Delta_T + \Delta_Q$	Δ_{diag}	$O(n \log\log n)$
Abouelhoda and Ohlebusch, 2004 [1, 2] `CHAINER`	inexact	arbitrary	L_1	–	$O(n \log n)$
Abouelhoda and Ohlebusch, 2005 [2]	inexact	arbitrary	$\varepsilon \cdot \min(\Delta_T, \Delta_Q) + +\lambda \cdot \lvert\Delta_T - \Delta_Q\rvert$	–	$O(n \log n \log\log n)$
Otto et al., 2011 [32] `clasp`	inexact	arbitrary	$\varepsilon \cdot \min(\Delta_T, \Delta_Q) + +\lambda \cdot \lvert\Delta_T - \Delta_Q\rvert$	–	$O(n \log^2 n)$
Bushmanova et al., 2016 [7] `rnaQUAST`	inexact	arbitrary	–	ov_T	$O(n^2)$
Li, 2018 [23] `minimap2`	fixed-length, exact	matches	$p_\cup(\Delta_{\text{diag}})$	$p_\cup(\Delta_{\text{diag}})$	$O(n^2)$
Ren and Chaisson, 2021 [35] `lra`	inexact	arbitrary	$f_\cup(\Delta_{\text{diag}})$	–	$O(n \log^2 n)$
Sahlin and Mäkinen, 2020 [24]	exact	matches	0	0	$O(n \log n)$
Sahlin and Mäkinen, 2021 [24, 40] `uLTRA`	inexact	mutual, weighted coverage	Δ_Q	$\varepsilon \cdot \max(\text{ov}_T, \text{ov}_Q)$	$O(n^2)$
Jain et al., 2022 [20, 37]	exact	–	L_∞	Δ_{diag}	$O(n \log^3 n)$
Groot Koerkamp and Ivanov, 2024 [15] `A*PA`	non-overlapping, inexact	edit distance	$\max(\Delta_{\text{diag}}, \lfloor \Delta_T / k \rfloor)$	–	$O(n \log^2 n)$
this work, Alg 1	exact	–	L_∞	Δ_{diag}	$O(n \log\log n)$
this work, `llchain`	exact	–	L_∞	Δ_{diag}	$O(n \log n)$

Table 1 Main algorithms and implementations (in `typewriter font`) for the global or semi-global chaining of n fragments between two sequences T and Q, with Q the smaller string. For fragment types, 'exact' refers to exact matches, 'square' to matches of the same length, and 'inexact' to approximate matches. For 'arbitrary' fragment scores, each fragment has a score in input, whereas 'matches' is equal to the number of matching bases in the corresponding alignment (also known as mutual coverage [24]). For the gap costs, $f(\Delta_T, \Delta_Q)$ is an arbitrary 2D function, $\ell(\Delta_{\text{diag}})$ is a linear 1D, $f_\cup$ is a concave function, $p_\cup$ is a piece-wise function with a concave part flanked by constant values, ε and λ are linear substitution and indel costs, and $L_\infty = \max(\Delta_T, \Delta_Q)$. Gap costs that have L_∞ as a special case are marked in yellow, and Δ_{diag} overlap costs in blue.

3 Preliminaries on chaining, range minimum queries, priority queues

We adapt the notation from Rizzo, Cáceres, and Mäkinen [37] to right-open intervals and 0-indexed strings. We denote integer interval $\{x, x+1, \ldots, y\}$ as $[x..y]$, or $[x..y+1)$. Given 0-indexed strings T and Q over finite alphabet Σ, interval pair $a = ([t_s..t_e), [q_s..q_e))$ is an *exact match anchor* (or fragment, or just anchor) between T and Q if $T[t_s..t_e) = Q[q_s..q_e)$, where $0 \leq t_s \leq t_e \leq |T|$ and $0 \leq q_s \leq q_e \leq |Q|$. The main invariant of exact match anchors that we will use throughout the paper is $t_e - t_s = q_e - q_s \iff t_e - q_e = t_s - q_s$.

A colinear chaining formulation is first defined via *anchor precedence*, a partial order $\prec$ specifying which anchors can follow one another to compose a chain $a_1 \prec \cdots \prec a_c$. One then maximizes the *score* of a chain. Indeed, all chaining formulations in Table 1 maximize a formula of the type

$$\mathsf{score}(A) = \sum_{i=1}^{c} \mathsf{fscore}(a_i) - \sum_{i=1}^{c-1} \mathsf{connect}(a_i, a_{i+1})$$

where $\mathsf{fscore}(a)$ is the non-negative fragment score obtained by picking anchor a, and

$\mathsf{connect}(a_i, a_{i+1})$ is the non-negative penalty (cost) of choosing a_{i+1} after a_i. Since our chaining formulation will have no fragment score, our goal will be to minimize a function very similar to $\mathsf{cost}(A) = \sum_{i=1}^{c-1} \mathsf{connect}(a_i, a_{i+1})$.

3.1 Full problem statement

Consider the following definitions alongside Figure 2.

▶ **Definition 3** (Anchor precedence and colinear chain [3, 20]). *Given anchor $a = ([t_s..t_e), [q_s..q_e))$ between T and Q, let $\mathsf{start}(a) = (t_s, q_s)$ be the start-point of a. Then a* precedes *$a' = ([t'_s..t'_e), [q'_s..q'_e))$, in symbols $a \prec a'$, if $\mathsf{start}(a)$ is distinct from and reaches $\mathsf{start}(a')$ in a possible alignment, that is,*

$$a \prec a' \quad \textit{iff} \quad t_s \le t'_s \textit{ and } q_s \le q'_s \textit{ and } (t_s, q_s) \neq (t'_s, q'_s).$$

Then, a sequence of anchors $a_1, \ldots, a_c$ is called a colinear chain *or just $\prec$-chain if $a_i \prec a_{i+1}$ for all $i \in [1..c)$. Note that relation $\prec$ is referred to as* weak precedence *in [20].*

▶ **Definition 4** (Connect function [20], [37, Lemma 1]). *Given anchors $a \prec a'$, we define the following quantities (some shown in Figure 1):*

$$\begin{aligned}
\Delta_T(a, a') &:= \max(0, t'_s - t_e) && T\textit{-gap}\\
\Delta_Q(a, a') &:= \max(0, q'_s - q_e) && Q\textit{-gap}\\
L_\infty(a, a') &:= \max(\Delta_T(a, a'), \Delta_Q(a, a')) && \textit{max-gap}\\
\mathsf{ov}_T(a, a') &:= \max(0, t_e - t'_s) && T\textit{-overlap}\\
\mathsf{ov}_Q(a, a') &:= \max(0, q_e - q'_s) && Q\textit{-overlap}\\
\Delta_{diag}(a, a') &:= |\Delta_T(a, a') - \Delta_Q(a, a')| = |\operatorname{diag}(a) - \operatorname{diag}(a')| && \textit{diagonal distance}
\end{aligned}$$

where $\operatorname{diag}(a) = t_s - q_s$ is the diagonal of a. Then the cost of connecting two anchors is obtained as the maximum of the L_∞ gap cost and the Δ_{diag} overlap cost:

$$\mathsf{connect}(a, a') = \max(L_\infty(a, a'), \Delta_{diag}(a, a')) = \begin{cases} L_\infty(a, a') & \textit{if } (t_e, q_e) \preceq (t'_s, q'_s) \quad \textit{(gap-gap)},\\ \Delta_{diag}(a, a') & \textit{otherwise} \quad \textit{(overlap case)}. \end{cases}$$

▶ **Definition 5** (Global and semi-global cost [20]). *Let $A = a_1, \ldots, a_c$ be a $\prec$-chain of anchors between T and Q. We define the* global (alignment) cost *of A as $\mathsf{gcost}(A) = \sum_{i=0}^{c} \mathsf{connect}(a_i, a_{i+1})$, where dummy anchors $a_0 = a_{\text{start}} = ([0, 0), [0, 0))$ and $a_{c+1} = a_{\text{end}} = ([|T|, |T|), [|Q|, |Q|))$ indicate the start and end of the alignment. The* semi-global cost $\mathsf{sgcost}(A)$ *ignores the initial and final gap in T, that is,*

$$\mathsf{sgcost}(A) = \Delta_Q(a_{\text{start}}, a_1) + \left(\sum_{i=1}^{c-1} \mathsf{connect}(a_i, a_{i+1})\right) + \Delta_Q(a_c, a_{\text{end}}).$$

▶ **Problem 6** (Colinear chaining [20]). *Given a set $\mathcal{A}$ of n exact match anchors, the* colinear chaining problem with L_∞ gap costs and Δ_{diag} overlap costs *consists of finding an ordered subset $A = a_1, \ldots, a_c$ of $\mathcal{A}$ that is a $\prec$-chain and has minimum global cost $\mathsf{gcost}(A)$. The* semi-global *variant of the problem minimizes $\mathsf{sgcost}(A)$ instead.*

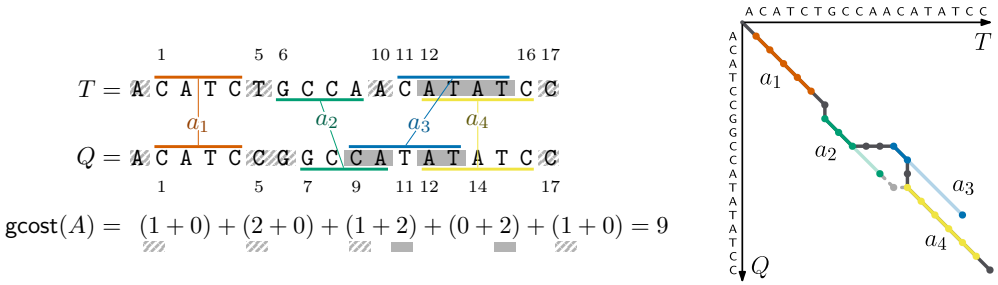

Figure 2 Example of a chain $A = a_1, a_2, a_3, a_4$ (under all precedence notions introduced) and of cost $\mathsf{gcost}(A) = 9$, where on the left the gaps and overlaps between consecutive anchors are marked with striped and solid background, respectively. On the right, a possible alignment in the corresponding edit graph is shown that uses a prefix of each anchor. Skipping a_3 would reduce the cost from 9 to 6.

3.2 Variants, algorithms, and anchored edit distance

Jain, Gibney, and Thankachan [20] introduced and studied efficient solutions to Problem 6, as well as variants using different precedence formulations, which we will briefly discuss now. While the definition of non-overlapping anchors has been consistent in the literature, overlaps can be handled in multiple ways: for example, $([1..3), [1..3)), ([2..3), [5..6))$ is a $\prec$-chain but it aligns $T[2..3)$ to two different regions of Q, raising doubts about the usefulness of allowing this case. Luckily, the colinear chaining problems under $\prec$ and under the following formulations are equivalent,[2] and in turn they all compute the anchored edit distance.

▶ **Definition 7** (Alternative precedence formulations [20, 37]). *Anchor* $a = ([t_s..t_e), [q_s..q_e))$ strictly precedes $a' = ([t'_s..t'_e), [q'_s..q'_e))$, *in symbols* $a \prec_{\mathrm{s}} a'$, *if* $a \neq a'$ *and* $t_s \leq t'_s$, $t_e \leq t'_e$, $q_s \leq q'_s$, $q_e \leq q'_e$. *Also, we say that* a strongly precedes a', *in symbols* $a \prec_{\texttt{ChainX}} a'$, *if* $t_s < t'_s$, $t_e < t'_e$, $q_s < q'_s$, *and* $q_e < q'_e$.

▶ **Theorem 8** ([20, Theorem 2] and [38, Theorem 3]). *For a fixed set of anchors* $\mathcal{A}$, *the following quantities are equal: the anchored edit distance (Definition 1) and the optimal* $\prec$*-chain,* $\prec_{\mathrm{s}}$*-chain, and* $\prec_{\texttt{ChainX}}$*-chain global cost (*gcost, *Definition 5). Similarly, the semi-global variant of anchored edit distance (i.e. initial and final* T*-gaps cost 0) is equal to the minimum semi-global cost* $\mathsf{sgcost}(A)$ *of an optimal* $\prec$*-,* $\prec_{\mathrm{s}}$*-, or* $\prec_{\texttt{ChainX}}$*-chain* A.

There are already several dynamic-programming solutions to Problem 6 and its variants. Note how each precedence formulation (Definitions 3 and 7) defines a partial order on the input anchor set $\mathcal{A}$: consistently with the literature [43], the chaining algorithms consider $\mathcal{A}$ to be in a topological order. All solutions, including ours, compute values $C[j]$, the optimal partial costs $\mathsf{pgcost}(A) = \sum_{i=0}^{c-1} \mathsf{connect}(a_i, a_{i+1})$ over all chains $A = (a_1, \ldots, a_c)$ starting in a_{start} and ending in anchor j (and thus not yet connecting to the final anchor a_{end}). Due to the additivity of the gcost function, values $C[j]$ can be computed in $O(n^2)$ time via dynamic programming (DP) as $C[0] = 0$ and

$$C[j] = \min_{i \in [0..j): a_i \prec a_j} C[i] + \mathsf{connect}(a_i, a_j) \qquad \forall j \in [1..n+1], \tag{1}$$

and it is easy to see that $C[n+1]$ is the minimum gcost of a $\prec$-chain. By changing the initial condition, a variant of Equation (1) also solves semi-global chaining.

Jain et al. also introduced sub-quadratic time solutions and a practical heuristic.

[2] We choose then to concentrate on $\prec$ and not one of its variants because the algorithmic results will be easier to prove under this precedence.

▶ **Theorem 9** ([20, Theorem 1]). *The colinear chaining problem with overlap and gap costs (Problem 6) can be solved in time:*

- $O(n \log^2 n)$ *for* $\prec_{\mathrm{s}}$*-chains with fixed-length anchors;*
- $O(n \log^3 n)$ *for* $\prec$*-chains; and*
- $O(n \log^4 n)$ *for* $\prec_{\mathrm{s}}$*-chains (which are now known to be equivalent to* $\prec$*-chains).*

Values $C[i]$ can be computed using dynamic multi-dimensional range minimum query data structures (a case of orthogonal range search in computational geometry [11]), but this was not implemented. Instead, [20] implements a practical algorithm that *approximates* the optimum $\prec_{\texttt{ChainX}}$-chain cost in $O(\mathrm{SOL} \cdot n + n \log n)$ expected time, where SOL is the cost of the output chain and we assume a uniform distribution of start positions of the anchors in Q or T [20, Algorithm 2]. This solution, called `ChainX`, was tested on sequences of bacterial-like size and features: experimentally, `ChainX`'s output on maximal unique match (MUM) anchors correlates with edit distance and is faster than programs directly computing the edit distance, but the performance degrades significantly when the number of anchors increases [20, Section 6]. The approximate method `ChainX` was completed by Rizzo et al. into an exact algorithm `ChainX-opt` [37, Algorithm 1], shown to find a better score for a small number of cases with minimal practical slowdown compared to `ChainX`.

▶ **Theorem 10** ([37, Theorem 2] and [38, Corollary 2]). *The colinear chaining problems with overlap and gap costs under* $\prec$*,* $\prec_{\mathrm{w}}$*, and* $\prec_{\texttt{ChainX}}$ *precedence can be solved in time* $O(n \log n + \mathrm{OPT} \cdot n)$*, where* OPT *is the optimum chain cost, assuming that* $n \leq \min\{|Q|, |T|\}$ *and the anchors are uniformly distributed in* Q*.*

Properties of anchored edit distance. We now list a few additional properties of the anchored edit distance. As implied by the name, the anchored edit distance is a variant on the classical edit distance, and admits a corresponding *edit graph* [27] that contains edges corresponding to insertions, deletions, and substitutions (all cost 1) and matches as indicated by the anchors (of cost 0), see Figure 2. We define d_{ij} as the distance (length of the shortest path) from $(0, 0)$ to (i, j). The proof of Theorem 8 gives the following fact.

▶ **Corollary 11.** *Let* $\mathcal{A} = a_1, \ldots, a_n$ *be sorted according to any* $\prec$*-topological order. The optimal partial cost* $C[j]$ *to the start of an anchor* a_j *then equals* $d_{\mathsf{start}(a_j)}$*, the smallest distance from* $(0, 0)$ *to* $\mathsf{start}(a_j)$ *in the edit graph defined by* $\mathcal{A}$*.*

From this correspondence, it is clear [37, Corollary 1] that overlapping anchors on the same diagonal can be merged, and similarly, that an anchor of length ℓ can be split into ℓ anchors of length 1, or into two anchors of lengths $\ell_1 + \ell_2 = \ell$, while keeping all partial costs the same.

3.3 Dynamic predecessor data structures

A data structure for dynamic predecessor queries supports the following operations on a set S of totally ordered keys:

- $\mathsf{insert}(x)$: Add x to S.
- $\mathsf{remove}(x)$: Remove x from S.
- $\mathsf{succ}(x)$: Return $\min\{y \in S \mid y > x\}$, the smallest element larger than x.
- $\mathsf{pred}(x)$: Return $\max\{y \in S \mid y < x\}$, the largest element less than x.

In practice, keys can also have associated values, in which case $\mathsf{get}(x)$ can return this value in case $x \in S$.

Common implementations use balanced binary trees (red-black trees [16]) or B-trees [4], taking $O(n)$ words of space and $O(\log n)$ time for a set of n elements. When the keys are integers up to U, Van Emde Boas trees [41] use $O(U)$ space and allow all operations in $O(\log\log U)$ time (in the word RAM model). Y-fast tries [44] improve this to $O(\log\log U)$ time in $O(n)$ space. Johnson's priority queue [21] is another variant often used in chaining methods that uses $O(U^{1/p})$ space for $O(p \log\log D)$ queries, where D is the distance between the two elements surrounding the queried element x and p is a small positive integer parameter that can be chosen freely.

With known keys. In case the set of all $O(n)$ inserted keys x is known in advance, they can be sorted in $O(n \log\log n)$ time and $O(n)$ space [18]. Then, each key can be assigned its *rank* from 1 to n in the sorted list (e.g. via a lookup table), and the predecessor structure can use the ranks instead. Thus, this case allows for $O(\log\log n)$ rather than $O(\log\log U)$ queries while using $O(n)$ space.

Incremental suffix minimum queries. We can also use predecessor structures for *incremental suffix minimum queries*, where we insert keys x with an associated value $f(x)$ (but never remove them) and get queries that ask for $\mathsf{smq}(x) = \min\{f(y) \mid y \in S : y \geq x\}$. A standard way to support this is the following: if we try to insert $(x, f(x))$ but $f(x) \geq \mathsf{smq}(x)$, do nothing; otherwise, insert $(x, f(x))$ (or update the value associated with x), and repeatedly find $y = \mathsf{pred}(x)$, then remove y as long as $f(x) \leq f(y)$. If S is known in advance, insertions and smq queries can both be supported in $O(\log\log n)$ amortized time.

4 Chaining in $O(n \log\log n)$ time

We now give an algorithm for colinear chaining with L_∞ gap costs and Δ_{diag} overlap costs on the input set $\mathcal{A} = \{a_1, \ldots, a_n\}$ (Problem 6) in $O(n \log\log n)$ time and $O(n)$ space. We assume that $\mathcal{A}$ is sorted in lexicographical order of 2D points $\mathsf{start}(a)$. We do not worry about ties, because we also assume that $\mathcal{A}$ does not contain perfectly-overlapping or touching anchors (on the same diagonal), as these can be merged into longer anchors without changing the anchor-supported character matches and thus the optimal gcost chain (Theorem 8). These two assumptions can be enforced in $O(n \log\log n)$ time and $O(n)$ space by sorting [18].

We follow the same general strategy as all sub-quadratic-time chaining algorithms (see e.g. [13, 3, 20]): we partition the minimum of Equation (1) into four disjoint cases described by the $\mathsf{connect}$ function and use a line sweep of the 2D plane $[0..|T|) \times [0..|Q|)$. Consider Figure 3 and the following equation:

$$C[j] = \min\left(C^{1a}[j], C^{1b}[j], C^{2}[j], C^{3}[j]\right), \qquad \text{where} \tag{2}$$

$$C^{1a}[j] := \min_{\substack{i\in[0..j):\\ a_i \prec a_j,\ \mathsf{ov}_T(a_i,a_j)>0,\\ \mathrm{diag}(a_i)\geq\mathrm{diag}(a_j)}} C[i] + \mathrm{diag}(a_i) - \mathrm{diag}(a_j) \quad \text{(overlap, larger } T\text{-overlap)} \tag{3}$$

$$C^{1b}[j] := \min_{\substack{i\in[0..j):\\ a_i \prec a_j,\ \mathsf{ov}_Q(a_i,a_j)>0,\\ \mathrm{diag}(a_i)<\mathrm{diag}(a_j)}} C[i] + \mathrm{diag}(a_j) - \mathrm{diag}(a_i) \quad \text{(overlap, larger } Q\text{-overlap)} \tag{4}$$

$$C^{2}[j] := \min_{\substack{i\in[0..j):\\ \mathsf{end}(a_i)\preceq\mathsf{start}(a_j),\\ \mathrm{diag}(a_i)>\mathrm{diag}(a_j)}} C[i] + \Delta_Q(a_i, a_j) \quad \text{(gap-gap, larger } Q\text{-gap)} \tag{5}$$

$$C^{3}[j] := \min_{\substack{i\in[0..j):\\ \mathsf{end}(a_i)\preceq\mathsf{start}(a_j),\\ \mathrm{diag}(a_i)\leq\mathrm{diag}(a_j)}} C[i] + \Delta_T(a_i, a_j) \quad \text{(gap-gap, larger } T\text{-gap)} \tag{6}$$

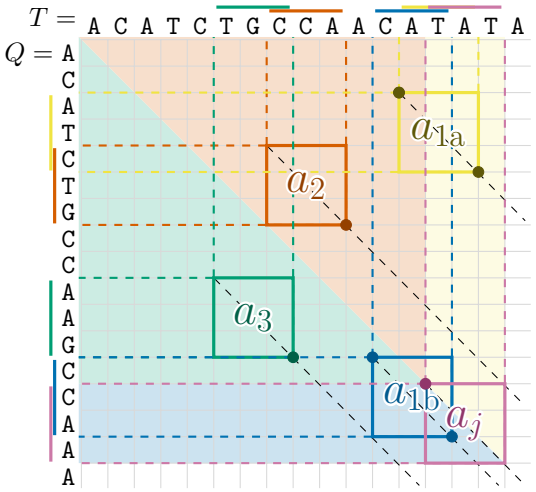

Figure 3 Anchors a_{1a}, a_{1b} a_2, a_3 showing the four mutually exclusive cases for the computation of $C[j]$ (Equation (2)): overlap case with bigger T-overlap (1a) or bigger Q-overlap (1b), gap-gap with bigger Q-gap (2), and gap-gap with bigger T-gap (3). The areas containing the endpoint for each recursive anchor a_i are highlighted with different colors, and the startpoints of a_{1a}, a_{1b} are also highlighted to indicate that in Equations (3) and (4) they need to be colinear to a_j.

where $\min(\emptyset) = +\infty$. Then, our algorithm can be divided into three main stages, as partially described in Algorithm 1:

Pre-processing First, we compute the sorted orders for the "left-to-right" (i.e. t_s and t_e) and "top-to-bottom" (i.e. q_s and q_e) line sweeps of the set of anchor startpoints and endpoints, and we additionally sort the anchors by their diagonal value $\text{diag}(a)$ to compute $\mathsf{rank}_{\text{diag}(a)}$, where $\mathsf{rank}_{\text{diag}(a)} = d$ if $\text{diag}(a)$ is the d-th hit diagonal. As explained in Section 3.3, this can be done in $O(n \log \log n)$ time and $O(n)$ space, since sorting considers at most $2n$ points. Following Baker and Giancarlo [3], for each $j \in [1..n]$ we compute anchors $\hat{a}_j$ and $\overleftarrow{a_j}$, respectively the closest anchor overlapping $\mathsf{start}(a_j)$ in a bigger diagonal (i.e. "above") and a smaller diagonal (i.e. "left"), with two line sweeps as detailed in Section 4.1 ($\hat{a}_j$ and $\overleftarrow{a_j}$ are called $visl(a_j)$ and $visa(a_j)$ in [3]). This will allow handling both overlap cases in constant time during the main line sweep.

Main line sweep We compute values $C[j]$ while performing a line sweep of the anchor startpoints and endpoints, from smaller to larger T-coordinate value. Following Eppstein et al. [13] (see also [9, 3]), when a startpoint $\mathsf{start}(a_j)$ is processed, $C[j]$ is calculated as per Equation (2) using $\hat{a}_j$, $\overleftarrow{a_j}$, and the data structures for the two gap-gap cases 2 and 3 as we will describe in Sections 4.2 and 4.3. When a startpoint or endpoint is reached (endpoints are processed before startpoints if they fall on the same T-position), these gap-gap data structures are updated.

Post-processing Value $C[n+1]$ is equal to the anchored edit distance. Instead of augmenting our solution to store backtracking information (see e.g. [37]), we can optionally recover an optimal $\prec$-chain from values $C[j]$ in $O(n)$ time: we start from $j = n+1$ and we iteratively perform a linear search to find a predecessor $i < j$ such that $C[i] + \mathsf{connect}(a_i, a_j) = C[j]$ and we set $j = i$, until $j = 0$.

The rest of this section is dedicated to showing how to solve the recursive cases individually, in order of hardness: the overlap cases (1a and 1b), the bigger Q-gap case (2), and the bigger

Algorithm 1 High-level description of chaining with L_∞ gap costs and Δ_{diag} overlap costs.

Input : Anchors $\mathcal{A} = \{a_1, \ldots, a_n\}$ between T and Q such that no two anchors on the same diagonal overlap or have an empty gap (i.e. $\Delta_T > 0$ or $\Delta_Q > 0$)
Output : The minimum global chaining cost (gcost) over all $\prec$-chains

Sort anchors $a \in \mathcal{A}$ by diagonal value $\text{diag}(a)$, to compute $\mathcal{D} := \{\text{diag}(a) : a \in \mathcal{A}\}$;
`preprocess_1a`$(\mathcal{A})$; ▷ **Computes anchors** $\hat{a}$
`preprocess_1b`$(\mathcal{A})$; ▷ **Computes anchors** $\overleftarrow{a}$
Sort startpoints and endpoints
$P = \{(a_j.\text{start}, j) : a_j \in \mathcal{A}\} \cup \{(a_j.\text{end}, j) : a_j \in \mathcal{A}\} \cup \{((0,0),0), ((|T|,|Q|), n+1)\}$
in increasing T-coordinate order (in breaking ties, endpoints are smaller);
`init_case_2`$(\mathcal{A})$;
`init_case_3`$(\mathcal{A})$;
$C[0] \leftarrow 0$;
for $(p, j) \in P$ *in sorted order* **do**
  if $p = a_j.\text{start}$ **then**
    $C^{1a}[j] \leftarrow C[\hat{a}_j] + \Delta_{\text{diag}}(\hat{a}_j, a_j)$; ▷ **Lemma 13**
    $C^{1b}[j] \leftarrow C[\overleftarrow{a_j}] + \Delta_{\text{diag}}(\overleftarrow{a_j}, a_j)$; ▷ **Lemma 13**
    $C^{2}[j] \leftarrow$ `process_startpoint_case_2`(a_j);
    $C^{3}[j] \leftarrow$ `process_startpoint_case_3`(a_j);
    $C[j] = \min\left(C^{1a}[j], C^{1b}[j], C^{2}[j], C^{3}[j]\right)$; ▷ **Equation (2)**
  else ▷ $p = a_j.\text{end}$
    `process_endpoint_case_2`(a_j);
    `process_endpoint_case_3`(a_j);
return $C[n+1]$; ▷ **Best** pgcost **to final dummy anchor** $a_{n+1} = a_{\text{end}}$

T-gap case (3). The correctness will follow from Equation (2) by induction on $j = 0, \ldots, n+1$, obtaining our main theoretical result thanks to Theorem 8.

▶ **Theorem 12.** *The anchored edit distance (Definition 1) can be computed in $O(n \log\log n)$ time and $O(n)$ space, by solving the colinear chaining problem with L_∞ gap costs and Δ_{diag} overlap costs (Problem 6) in the same time and space.*

4.1 Overlap cases 1a and 1b

We study the computation of values $C^{1a}[j]$ (Equation (3)), as case C^{1b} is solved in a symmetrical way. We are in this case when $a_i \prec a_j$, $\text{ov}_T(a_i, a_j) > 0$, and $\text{diag}(a_i) \geq \text{diag}(a_j)$, meaning that the anchors from $C^{1a}[j]$ are exactly anchors a_i such that: (i) their T-interval $[t^i_s..t^i_e)$ contains t^j_s; and (ii) $\text{diag}(a_i) \geq \text{diag}(a_j)$. Note that conditions i and ii are sufficient for the colinearity $a_i \prec a_j$. By adapting an argument of Baker and Giancarlo [3, Section 3.1], we can prove that we only need to check the closest anchor above a_j, as all other anchors cannot be a better case-1a predecessor.

▶ **Lemma 13.** *In the overlap case with bigger T-overlap (Equation (3)) and under the assumption that no two anchors overlap on the same diagonal, the optimal partial cost $C^{1a}[j]$ to anchor a_j is equal to $C[\hat{a}] + \Delta_{diag}(\hat{a}, a_j)$, where $\hat{a}$ is the anchor with the smallest $\text{diag}(\hat{a})$ bigger than $\text{diag}(a_j)$ such that $t^j_s \in [\hat{t_s}..\hat{t_e})$. If $\hat{a}$ is not defined, then $C^{1a}[j] = +\infty$.*

Proof. Assume by contradiction that $a' = ([t'_s..t'_e), [q'_s..q'_e))$ is a strictly better overlapping anchor considered by case 1a which is farther away, that is, $t^j_s \in [t'_s..t'_e)$, $\text{diag}(a_j) \leq \text{diag}(\hat{a}) <$

Algorithm 2 Procedure handling the computation of anchors $\hat{a}_j$ for $j \in [1..n]$, where $\hat{a}_j$ is defined as the anchor overlapping with a_j with the smallest diagonal greater than $\text{diag}(a_j)$, if it exists, otherwise $\hat{a} = \perp$, the dummy anchor. Note that we assume anchor set $\mathcal{A}$ to not have any anchors overlapping or touching along the same diagonal, so no two startpoints or endpoints can be equal.

```
1  Procedure preprocess_1a(A)
2      Sort startpoints and endpoints P = {(a.start, a) : a ∈ A} ∪ {(a.end, a) : a ∈ A} in
         increasing T-coordinate order (in breaking ties, endpoints are smaller);
3      â_0 ← ⊥;
4      â_{n+1} ← ⊥;
5      S ← ∅;                  ▷ predecessor data structure over D = {diag(a) : a ∈ A}
6      for (p, a) ∈ P in sorted order do
7          if p = a.start then
8              â ← S.succ(diag(a)).value ;                ▷ if it exists, otherwise ⊥
9              S.insert(diag(a) ↦ a);
10         else                                                          ▷ p = a.end
11             S.remove(diag(a));
```

$\text{diag}(a')$ (no two anchors overlap in the same diagonal by assumption), and $C[a'] + \text{diag}(a') - \text{diag}(a_j) < C[\hat{a}] + \text{diag}(\hat{a}) - \text{diag}(a_j)$. If $a' \prec \hat{a}$, we reach a contradiction, as $C[a'] + \mathsf{connect}(a', \hat{a}) < C[\hat{a}]$ (case 1a) violates the definition of $C[\hat{a}]$ (Equation (1)). Otherwise $t'_s \in (\hat{t_s}..\hat{t_e})$ and we can split anchor $\hat{a}$ into two anchors $\hat{a}^1$ and $\hat{a}^2$, the first covering $[\hat{t_s}..t'_s)$ and the second covering $[t'_s..\hat{t_e})$. Due to Corollary 11, the partial optimal costs C' of the modified instance $(\mathcal{A} \setminus \{\hat{a}\}) \cup \{\hat{a}^1, \hat{a}^2\}$ are identical for all anchors but $C'[\hat{a}^1] = C'[\hat{a}^2] = C[\hat{a}]$. But then $a' \prec \hat{a}^2$ by construction and $C'[a'] + \mathsf{connect}(a', \hat{a}^2) < C'[\hat{a}^2]$, reaching a contradiction. ◀

Then, as in [3, Section 3.3], we can compute anchors $\hat{a}_j$ with a T-coordinate sweep line and a successor data structure on the set $S \subseteq \mathcal{D} := \{\text{diag}(a) : a \in \mathcal{A}\}$ of active anchors overlapping with the sweep line, as implemented in Algorithm 2. Since $|\mathcal{D}| \leq n$ and $\mathcal{D}$ can be precomputed, the computation takes $O(n \log \log n)$ time and $O(n)$ space. The overlap case 1b is symmetrical and can be obtained by sorting the points by Q-coordinate and using predecessor queries (see line 8 of Algorithm 2).

4.2 Gap-gap case, bigger gap in Q

If we are in this case, that is, if $\mathsf{end}(a_i) \preceq \mathsf{start}(a_j)$ and $\text{diag}(a_i) > \text{diag}(a_j)$, then $a_i \prec a_j$ and

$$C^2[j] = \min_{\substack{i \in [0..j): \\ \mathsf{end}(a_i) \preceq \mathsf{start}(a_j), \\ \text{diag}(a_i) > \text{diag}(a_j)}} \left(C[i] + q_s^j - q_e^i\right) = q_s^j + \min_{\substack{i \in [0..j): \\ \mathsf{end}(a_i) \preceq \mathsf{start}(a_j), \\ \text{diag}(a_i) > \text{diag}(a_j)}} \left(C[i] - q_e^i\right). \tag{7}$$

In geometric terms, we need to consider each anchor whose endpoint is in the first octant from $\mathsf{start}(a_j)$, as shown in Figure 3. This octant is characterized by the two equations $t_e^i \leq t_s^j$ (i) and $\text{diag}(a_i) > \text{diag}(a_j)$ (ii): the former is guaranteed by processing the points from left to right, and the diagonal constraint is handled via a suffix minimum query over diagonal keys, as shown in Algorithm 3. Indeed, from (i) and (ii) we can derive that $q_e^j \leq q_s^i$

Algorithm 3 Procedure used in Algorithm 1 to handle the computation of $C^2[j]$, the optimal partial cost $\mathsf{pgcost}(A)$ of any $\prec$-chain A ending at anchor a_j with a predecessor a_i in the gap-gap, bigger Q-gap case (see Figure 3 and Equation (2)).

```
Procedure init_case_2(𝒜)
    S ← ∅;                          ▷ suffix min query structure over 𝒟 = {diag(a) : a ∈ 𝒜}
    S.insert(diag(a_0) ↦ 0);                                  ▷ base case for a_0 = a_start
Procedure process_startpoint_case_2(a_j)
    return q_s^j + S.smq(diag(a_j) + 1);
Procedure process_endpoint_case_2(a_i)
    S.insert(diag(a_i) ↦ C[i] − q_e^i);          ▷ insert or update the recursive value
```

and thus $\mathsf{end}(a_i) \preceq \mathsf{start}(a_j)$ as follows:

$$\begin{aligned} \mathrm{diag}(a_i) > \mathrm{diag}(a_j) &\iff t^i_{\mathrm{e}} - q^i_{\mathrm{e}} > t^j_{\mathrm{s}} - q^j_{\mathrm{s}} \\ &\iff q^j_{\mathrm{s}} - q^i_{\mathrm{e}} > t^j_{\mathrm{s}} - t^i_{\mathrm{e}} \geq 0 \qquad (t^i_{\mathrm{e}} \leq t^j_{\mathrm{s}}) \\ &\implies q^j_{\mathrm{e}} \leq q^i_{\mathrm{s}}. \end{aligned}$$

We briefly discuss Algorithm 3. As originally presented in [13, Sections 2 and 4] for similar recursions, the recursive values $C[i] - q^i_{\mathrm{e}}$ are stored in a one-dimensional data structure, as our line sweep conveniently gives us one half-plane for free. Since the set of $\mathrm{diag}(a)$ keys is known in advance and the number of startpoints and endpoints is $2n$, we can handle case 2 in $O(n \log \log n)$ time and $O(n)$ space.

4.3 Gap-gap case, bigger gap in T

This is the final and hardest case to handle for the computation of Equation (2). Even though it is symmetrical to case 2, we do not get a half-plane filter for free as our sweeping of the plane goes from left to right (see Figure 3) while the constraint is on q, and this cannot be easily handled with a one-dimensional data structure or by changing the sweep line. Indeed, in recent chaining results this type of recursive case has been handled with 2D search data structures, resulting in final time complexities of $O(n \log^2 n)$ or $O(n \log n \log \log n)$ [32, 35, 20]. Instead, using the space-partitioning technique of Eppstein et al. [13], we can efficiently represent the optimal recursive cases projected onto the sweep line as dynamic one-dimensional segments, as described in the rest of this section. Our goal is to present a sufficiently detailed variant of the $O(n \log \log n)$-time technique that runs in $O(n)$ space, which is not a widely known result and might be of independent interest.[3]

Symmetrically to Equation (7), we can rewrite $C^3[j]$ as

$$C^3[j] = t^j_{\mathrm{s}} + \min_{\substack{i \in [0..j): \\ \mathsf{end}(a_i) \preceq \mathsf{start}(a_j), \\ \mathrm{diag}(a_i) \leq \mathrm{diag}(a_j)}} \big(C[i] - t^i_{\mathrm{e}}\big). \tag{8}$$

From this we can derive the following first main insight: each anchor a_i defines a *forward region of influence* where it can precede a following anchor a_j in the sense of Equation (8).

[3] The original efficient chaining solutions by Eppstein et al. [13] and Chao and Miller [9] used $O(|T| + |Q|)$ space. Baker and Giancarlo do claim to use $O(n)$ space [3, Theorem 4.1] but do not provide an explicit space-complexity proof for handling the intersection points [3, p. 251] as we will later do in Lemma 16. Other methods use $O(n)$ space but require $O(n \log \log(|T| + |Q|))$ time.

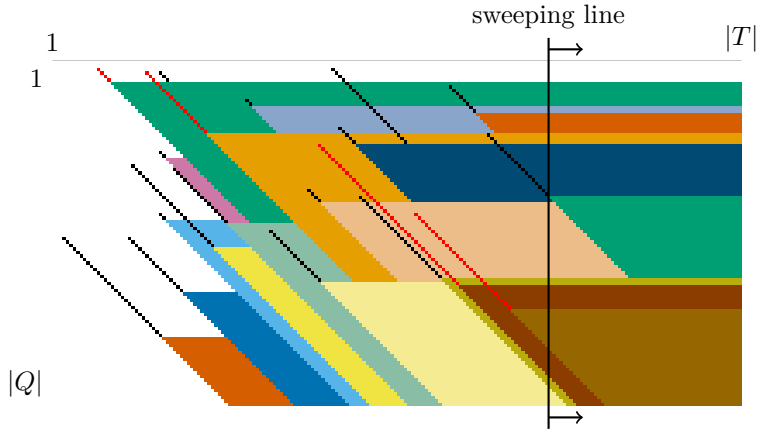

Figure 4 Optimal recursive values for case 3 (Equation (2)) handled in the main left-to-right sweep line. The anchors of an optimal $\prec$-chain are shown in red, whereas the other anchors of $\mathcal{A}$ are black. Each color corresponds to a region "owned" by a different anchor a_i, so values $R_t[q]$ of the owner array (Equation (9)) can be deduced by the color at the sweep line.

This region (also called "area of influence") is triangular and delimited by the horizontal and diagonal lines starting from endpoint $\mathsf{end}(a_i)$ (see Figure 4): for any successor a_j with $\mathsf{start}(a_j) = (t, q)$, we consider the intersection of the half-plane $q^i_{\mathrm{e}} \leq q$ bounded by the horizontal line through $\mathsf{end}(a_i)$ and the half-plane $\mathrm{diag}(t^j_{\mathrm{s}}, q^j_{\mathrm{s}}) \geq \mathrm{diag}(a_i)$ bounded by the diagonal line through $\mathsf{end}(a_i)$. When the areas of multiple anchors intersect, those with the smaller value $C[i] - t^i_{\mathrm{e}}$ take precedence, as remarked by the following observation.

▶ **Observation 14** (Chao and Miller [9, Lemma 1]). *Consider any two anchors* $a_i = ([t^i_s..t^i_e), [q^i_s..q^i_e))$ *and* $a_{i'} = ([t^{i'}_s..t^{i'}_e), [q^{i'}_s..q^{i'}_e))$. *For all anchors* a_j *such that* a_j *is case-*3 *successor of both* a_i *and* $a_{i'}$ *(as in Equation* (8)*), in symbols* $\mathsf{end}(a_i) \preceq \mathsf{start}(a_j)$, $\mathsf{end}(a_{i'}) \preceq \mathsf{start}(a_j)$, $\mathrm{diag}(a_i) \leq \mathrm{diag}(a_j)$, *and* $\mathrm{diag}(a_{i'}) \leq \mathrm{diag}(a_j)$, *we have that the value* $\min(C[i] - t^i_e, C[i'] - t^{i'}_e)$ *does not depend on the exact position of* a_j.

Let $a_j = ([t^j_{\mathrm{s}}..t^j_{\mathrm{e}}), [q^j_{\mathrm{s}}..q^j_{\mathrm{e}}))$ be an arbitrary anchor starting at the current sweep-line coordinate t, that is, $t^j_{\mathrm{s}} = t$, and consider Figure 4. We now define how the optimal recursive anchors for a_j change based on where the startpoint of a_j is along the sweep line (i.e. its Q-coordinate q^j_{s}) as follows, with ties in the arg min broken to larger i:

$$R_t[q] = \operatorname*{arg\,min}_{\substack{i \in [0..n): \\ t^i_{\mathrm{e}} \leq t,\, \mathrm{diag}(a_i) \leq t-q}} \left(C[i] - t^i_{\mathrm{e}}\right), \qquad \text{where } \operatorname{arg\,min}(\emptyset) = \perp \quad \text{and} \quad q \in [0..|Q|]. \tag{9}$$

We call R_t the *owner array*, because value $R_t[q]$ is equal to the anchor index the area belongs to: by construction we have $C^3[j] = t^j_{\mathrm{s}} + C[i] - t^i_{\mathrm{e}}$ with $i = R_{t^j_{\mathrm{s}}}[q^j_{\mathrm{s}}]$. Note that when there is no anchor a_i such that $\mathsf{end}(a_i) \preceq (t, q)$, then $R_t[q] = \perp$, where $\perp$ is the dummy anchor, and $C^3[j] = +\infty$.

The owner array has size $O(|Q|)$, so it would be too expensive to maintain it as a flat array. Instead, Eppstein et al. [13] employ a compact $O(n)$-space version of R_t as a doubly-linked list (called OWNER list) using the following second main insight: while the areas of influence can have complex interactions and shapes as seen in Figure 4, they are always delimited by some horizontal or diagonal line starting in an anchor endpoint.

▶ **Observation 15** ([13]). *For all changes in anchor index $R_t[q] \neq R_t[q+1]$ with $q \in [1..|Q|]$, there is a unique* active dividing line *such that the following holds for some unique anchor a_i, $i \in [0..n]$:*

- *(horizontal dividing line) the active line is $q+1 = q_e^i$ and $R_t[q+1] = i$, that is, a_i owns the region of influence* starting *at $q+1$; or*
- *(diagonal dividing line) the active line is* $\mathrm{diag}(t,q) = \mathrm{diag}(a_i)$ *and $R_t[q] = i$, that is, a_i owns the area of influence* ending *at q.*

Then, we can represent R_t in a compressed representation as shown in Algorithm 4. The active dividing lines from Observation 15, which are separately indexable via two dynamic predecessor data structures (Section 3.3), are used to partition the sweeping line regions with different $R_t[q]$ values. Note that Observation 15 does not indicate *which* anchor is the best recursive anchor between a diagonal and a horizontal line, hence we need to also store the different values of $R_t[q]$. We obtain the following result and thus complete the proof of Theorem 12.

▶ **Lemma 16.** *We can compute values $C^3[j]$ (Equation (8)) as part of the main left-to-right line sweep (Algorithm 1) in $O(n \log\log n)$ time and $O(n)$ space.*

Proof sketch. The correctness follows from Equation (8), Observation 14, and Observation 15, and from the correct handling of the compressed representation of R_t as partially shown in Algorithm 4 and fully shown in [13, Section 4]. As discussed in [13], the total number of diagonal or horizontal lines is $2n$, and the cost of handling intersections can be charged to the diagonal and horizontal lines of origin. The only modification to the solution by Eppstein et al. is that to reach both $O(n \log\log n)$ time and $O(n)$ space, we delay the intersection updates until we reach the closest column to the right of it that contains an anchor start or end, and we execute these updates in an unsorted, first-in-first-out order. We argue that this modification maintains the correctness of the algorithm, since a batch of intersection updates some areas of influence without changing the order (i.e. we get a subsequence of R_t and of the active dividing lines). We can check for new intersection points at any modification of R_t: if they occur before the current t they are handled directly; otherwise they are queued accordingly in $O(\log\log n)$ time per point using a dynamic predecessor query structure on the T-positions hit by some anchor startpoint or endpoint. We leave the full pseudocode implementation and a more detailed proof for the extended version of this work. ◀

5 Experiments

We implemented our algorithm in `llchain` and compare it to `ChainX` and `ChainX-opt`. Our implementation uses a standard C++ `std::map` as the predecessor structure, with $O(\log n)$ time operations. Experiments are run on an Intel Xeon Gold 6148 2.40GHz machine with 376 GB of RAM in the University of Helsinki Kale cluster (the Slurm job is run with options `--exclusive` and `--mem=60G`).

Similar to [37], we consider 100 000 PacBio HiFi reads of a human genome[4] as generated by the T2T project. We then use MUMmer4 [12, 22, 25] to find all MEMs of length at least 50 between each read and the concatenation of all chromosomes in the T2T-CHM13 human genome reference [29, 36]. For each method, we measure how long it takes to sort the MEMs, join overlapping/touching anchors on the same diagonal, and then find the best semi-global chain. We call each method directly via its API.

[4] Run m64004 available at `https://s3-us-west-2.amazonaws.com/human-pangenomics/index.html?prefix=T2T/scratch/HG002/sequencing/hifi/`.

Algorithm 4 Partial implementation of the handling of the gap-gap, bigger T-gap case (Equation (2)), as part of Algorithm 1, and following Eppstein et al. [13]. The doubly-linked list $\overline{R}$ is a compact representation of the owner array R_t (Equation (9)) of elements $(d, a) \in \{\rightarrow, \searrow\} \times \mathcal{A}$ mapping each dividing line to the originating anchor $a_{R_t[q]}$. We omit the handling of intersection updates, and we omit the edge cases in `process_endpoint_case_3` where $\mathsf{end}(a_i)$ falls on top of an active diagonal or horizontal line.

Procedure `init_case_3`$(\mathcal{A})$
  $H \leftarrow \emptyset$; ▷ **predecessor data structure over** $\mathcal{H} = \{q_\mathrm{e} : ([t_\mathrm{s}..t_\mathrm{e}), [q_\mathrm{s}..q_\mathrm{e})]) \in \mathcal{A}\}$
  $D \leftarrow \emptyset$; ▷ **predecessor data structure over** $\mathcal{D} = \{\mathrm{diag}(a) : a \in \mathcal{A}\}$
  $H.\mathsf{insert}(0 \mapsto a_0)$; ▷ **base case for** $a_0 = a_{\text{start}}$
  $D.\mathsf{insert}(0 \mapsto a_0)$; ▷ **base case for** $a_0 = a_{\text{start}}$
  $\overline{R} \leftarrow \big(((\rightarrow, a_0) \mapsto a_0), ((\searrow, a_0) \mapsto \perp)\big)$;
  ▷ **doubly-linked list of dividing lines (Observation 15) representing** R_t**, where each line is mapped to anchor** $a_{R_t[q]}$ **of the corresponding region.**
  $Q \leftarrow \big\{t \mapsto \emptyset : t \in \{t^j_\mathrm{s} : j \in [1..n]\} \cup \{t^j_\mathrm{e} : j \in [1..n]\}\big\}$;
  ▷ **Intersection update queues before each** T**-coordinate in the input.**

Procedure `process_startpoint_case_3`(a_j)
  Process all intersection updates in $Q[\mathsf{rank}_T(t^j_\mathrm{s})]$;
  $a_\rightarrow \leftarrow H.\mathsf{pred}(q^j_\mathrm{s} + 1).\mathsf{value}$; ▷ **closest** $\rightarrow$ **at or above** q^j_s
  $a_\searrow \leftarrow D.\mathsf{succ}(\mathrm{diag}(a_j) - 1).\mathsf{value}$; ▷ **closest** $\searrow$ **at or above** $\mathrm{diag}(a_j)$
  if $q_s^{\rightarrow} < t^j_s - \mathrm{diag}(a_\searrow)$ **then** ▷ $(\rightarrow, a_\rightarrow)$ **is closer**
    $a_i \leftarrow \overline{R}.\mathsf{node}(\rightarrow, a_\rightarrow).\mathsf{value}$;
  else ▷ $(\searrow, a_\searrow)$ **is closer**
    $a_i \leftarrow \overline{R}.\mathsf{node}(\searrow, a_\searrow).\mathsf{value}$;
  return $q^j_\mathrm{s} + C[i] - t^i_\mathrm{e}$; ▷ **Equation (8)**

Procedure `process_endpoint_case_3`(a_i)
  Process all intersection updates in $Q[\mathsf{rank}_T(t^i_\mathrm{e})]$;
  $a_\rightarrow \leftarrow H.\mathsf{pred}(q^i_\mathrm{e}).\mathsf{value}$; ▷ **closest** $\rightarrow$ **above** q^i_e
  $a_\searrow \leftarrow D.\mathsf{succ}(\mathrm{diag}(a_i)).\mathsf{value}$; ▷ **closest** $\searrow$ **above** $\mathrm{diag}(a_i)$
  if $q_s^{\rightarrow} < t^i_e - \mathrm{diag}(a_\searrow)$ **then** ▷ $(\rightarrow, a_\rightarrow)$ **is closer**
    $a_k \leftarrow \overline{R}.\mathsf{node}(\rightarrow, a_\rightarrow).\mathsf{value}$;
    if $C[i] - t^i_e \le C[k] - t^k_e$ **then**
      $\overline{R}.\mathsf{node}(\rightarrow, a_\rightarrow).\mathsf{insert\_next}((\rightarrow, a_i) \mapsto a_i)$;
      $\overline{R}.\mathsf{node}(\rightarrow, a_i).\mathsf{insert\_next}((\searrow, a_i) \mapsto a_k)$;
  else ▷ $(\searrow, a_\searrow)$ **is closer**
    $a_k \leftarrow \overline{R}.\mathsf{node}(\searrow, a_\searrow).\mathsf{value}$;
    if $C[i] - t^i_e \le C[k] - t^k_e$ **then**
      $\overline{R}.\mathsf{node}(\searrow, a_\searrow).\mathsf{insert\_next}((\rightarrow, a_i) \mapsto a_i)$;
      $\overline{R}.\mathsf{node}(\rightarrow, a_i).\mathsf{insert\_next}((\searrow, a_i) \mapsto a_k)$;
  If $\overline{R}.\mathsf{node}(\rightarrow, a_i).\mathsf{prev}$ is a diagonal line, queue their intersection update at $Q[t]$ with t the closest T-coordinate after or including the intersection;
  If $\overline{R}.\mathsf{node}(\searrow, a_i).\mathsf{next}$ is a horizontal line, queue their intersection update at $Q[t]$ with t the closest T-coordinate after or including the intersection;

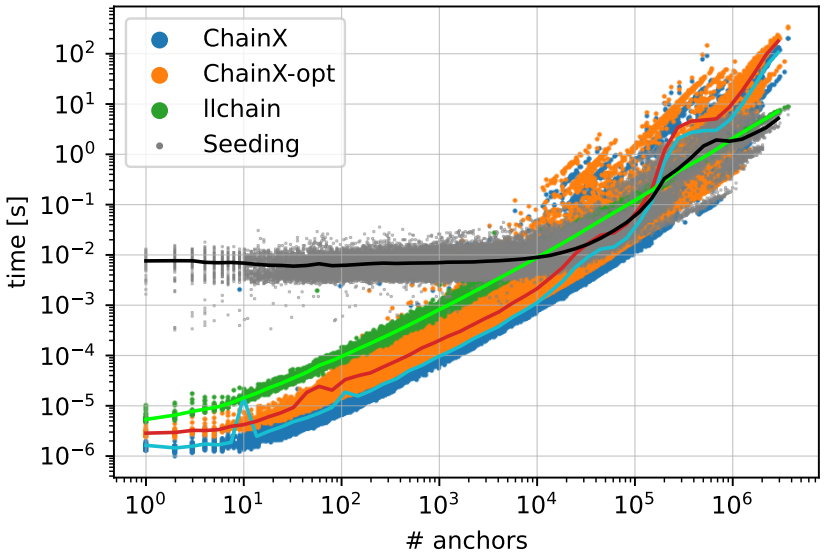

Figure 5 Time spent on seeding (grey, searching MEMs using MUMmer4) and chaining the resulting anchors using `ChainX`, `ChainX-opt`, and `llchain` for MEMs of length ≥ 50 between 100 000 HiFi reads and a human genome reference. The highlighted curves indicate the average running time after bucketing.

Results can be seen in Figure 5. `llchain` is around 10× slower than `ChainX` and 4× slower than `ChainX-opt` for 100 to 10 000 anchors, but in this case, the time needed for seeding dominates. When the number of anchors is large (above 100 000), `ChainX` and `ChainX-opt` have a high variance in their running time, and likely slow down when there are many matches in repetitive regions. In these cases, the running time grows roughly quadratically, and the slow instances dominate the overall running time. On the other hand, `llchain` has a very consistent running time and, as predicted by the theory, has a complexity roughly linear in the number of anchors.

Overall, seeding takes on average 0.052s, `ChainX` 0.19s, `ChainX-opt` 0.33s, `llchain` 0.059s. Thus, `llchain` is only slightly slower than the seeding on average, whereas the largest cases hurt the other methods, making them over 3× slower.

We further note that our implementation is not yet optimized, and in particular that `std::map` is known to be quite slow. A B-tree based implementation will likely be significantly faster.

6 Conclusion

Our new algorithm for anchored edit distance combines many existing ideas and runs in $O(n \log \log n)$ time and $O(n)$ space. Unlike the original chaining algorithm with this time complexity [13, 9], it achieves $O(n)$ space rather than e.g. $O(|Q| + |T|)$ space, which might be of independent interest.

Future work could include extending our method to the gap-cost of A*PA [15], as well as a generalization that handles inversions. Furthermore, it should be possible to adapt the method to allow for arbitrary fragment scores, affine gap cost penalties, and inexact matches. Lastly, the implementation could be optimized to use a more efficient predecessor structure than `std::map` and it might be possible to remove the linked list, as it is implicitly represented by the predecessor structures on horizontal and diagonal lines.

References

1 Mohamed Ibrahim Abouelhoda and Enno Ohlebusch. CHAINER: Software for comparing genomes. In *Proceedings of the 12th International Conference on Intelligent Systems for Molecular Biology+ 3rd European Conference on Computational Biology*. Citeseer, 2004. Short paper. URL: https://web.archive.org/web/20250708030638/https://www.iscb.org/cms_addon/conferences/ismbeccb2004/short%20papers/19.pdf.

2 Mohamed Ibrahim Abouelhoda and Enno Ohlebusch. Chaining algorithms for multiple genome comparison. *J. Discrete Algorithms*, 3(2-4):321–341, 2005. doi:10.1016/J.JDA.2004.08.011.

3 Brenda S. Baker and Raffaele Giancarlo. Sparse dynamic programming for longest common subsequence from fragments. *J. Algorithms*, 42(2):231–254, 2002. doi:10.1006/JAGM.2002.1214.

4 Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, 1972. doi:10.1007/bf00288683.

5 Gary Benson, Avivit Levy, S. Maimoni, D. Noifeld, and B. Riva Shalom. LCSk: A refined similarity measure. *Theor. Comput. Sci.*, 638:11–26, 2016. doi:10.1016/J.TCS.2015.11.026.

6 Gary Benson, Avivit Levy, and B. Riva Shalom. Longest common subsequence in k length substrings. In *Similarity Search and Applications 2013*, page 257–265. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-41062-8_26.

7 Elena Bushmanova, Dmitry Antipov, Alla L. Lapidus, Vladimir Suvorov, and Andrey D. Prjibelski. rnaQUAST: a quality assessment tool for *de novo* transcriptome assemblies. *Bioinform.*, 32(14):2210–2212, 2016. doi:10.1093/BIOINFORMATICS/BTW218.

8 Patrick S. G. Chain, Stefan Kurtz, Enno Ohlebusch, and Tom Slezak. An applications-focused review of comparative genomics tools: Capabilities, limitations and future challenges. *Briefings Bioinform.*, 4(2):105–123, 2003. doi:10.1093/BIB/4.2.105.

9 Kun-Mao Chao and Webb Miller. Linear-space algorithms that build local alignments from fragments. *Algorithmica*, 13(1/2):106–134, 1995. doi:10.1007/BF01188583.

10 Kun-Mao Chao, Jinghui Zhang, James Ostell, and Webb Miller. A local alignment tool for very long DNA sequences. *Comput. Appl. Biosci.*, 11(2):147–153, 1995. doi:10.1093/BIOINFORMATICS/11.2.147.

11 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. doi:10.1007/978-3-540-77974-2.

12 Arthur L. Delcher, Simon Kasif, Robert D. Fleischmann, Jeremy Peterson, Owen White, and Steven L. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, January 1999. doi:10.1093/nar/27.11.2369.

13 David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming I: linear cost functions. *J. ACM*, 39(3):519–545, 1992. doi:10.1145/146637.146650.

14 David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe F. Italiano. Sparse dynamic programming II: convex and concave cost functions. *J. ACM*, 39(3):546–567, 1992. doi:10.1145/146637.146656.

15 Ragnar Groot Koerkamp and Pesho Ivanov. Exact global alignment using A* with chaining seed heuristic and match pruning. *Bioinform.*, 40(1), 2024. doi:10.1093/BIOINFORMATICS/BTAE032.

16 Leo J. Guibas and Robert Sedgewick. A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, pages 8–21, 1978. doi:10.1109/SFCS.1978.3.

17 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/CBO9780511574931.

18 Yijie Han. Deterministic sorting in $O(n \log\log n)$ time and linear space. In John H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 602–608. ACM, 2002. doi:10.1145/509907.509993.

19 Robert S. Harris. *Improved pairwise alignment of genomic DNA*. PhD thesis, Pennsylvania State University, December 2007. URL: https://www.bx.psu.edu/~rsharris/rsharris_phd_thesis_2007.pdf.

20 Chirag Jain, Daniel Gibney, and Sharma V. Thankachan. Algorithms for colinear chaining with overlaps and gap costs. *J. Comput. Biol.*, 29(11):1237–1251, 2022. doi:10.1089/CMB.2022.0266.

21 Donald B. Johnson. A priority queue in which initialization and queue operations take $O(\log \log D)$ time. *Mathematical Systems Theory*, 15(1):295–309, December 1981. doi:10.1007/bf01786986.

22 Stefan Kurtz, Adam Phillippy, Arthur L. Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L. Salzberg. Versatile and open software for comparing large genomes. *Genome biology*, 5(2):R12, 2004. doi:10.1186/gb-2004-5-2-r12.

23 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinform.*, 34(18):3094–3100, 2018. doi:10.1093/BIOINFORMATICS/BTY191.

24 Veli Mäkinen and Kristoffer Sahlin. Chaining with overlaps revisited. In Inge Li Gørtz and Oren Weimann, editors, *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, Copenhagen, Denmark, June 17-19, 2020*, LIPIcs, pages 25:1–25:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.CPM.2020.25.

25 Guillaume Marçais, Arthur L. Delcher, Adam M. Phillippy, Rachel Coston, Steven L. Salzberg, and Aleksey Zimin. Mummer4: A fast and versatile genome alignment system. *PLOS Computational Biology*, 14(1):e1005944, January 2018. doi:10.1371/journal.pcbi.1005944.

26 Alla Mikheenko, Andrey D. Prjibelski, Vladislav Saveliev, Dmitry Antipov, and Alexey A. Gurevich. Versatile genome assembly evaluation with QUAST-LG. *Bioinform.*, 34(13):i142–i150, 2018. doi:10.1093/BIOINFORMATICS/BTY266.

27 Gene Myers. An $O(ND)$ difference algorithm and its variations. *Algorithmica*, 1(1–4):251–266, Nov 1986. doi:10.1007/bf01840446.

28 Gene Myers and Webb Miller. Chaining multiple-alignment fragments in sub-quadratic time. In Kenneth L. Clarkson, editor, *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22–24 January 1995. San Francisco, California, USA*, pages 38–47. ACM/SIAM, 1995. URL: http://dl.acm.org/citation.cfm?id=313651.313661.

29 Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, Andrey V. Bzikadze, Alla Mikheenko, Mitchell R. Vollger, Nicolas Altemose, Lev Uralsky, Ariel Gershman, Sergey Aganezov, Savannah J. Hoyt, Mark Diekhans, Glennis A. Logsdon, Michael Alonge, Stylianos E. Antonarakis, Matthew Borchers, Gerard G. Bouffard, Shelise Y. Brooks, Gina V. Caldas, Nae-Chyun Chen, Haoyu Cheng, Chen-Shan Chin, William Chow, Leonardo G. de Lima, Philip C. Dishuck, Richard Durbin, Tatiana Dvorkina, Ian T. Fiddes, Giulio Formenti, Robert S. Fulton, Arkarachai Fungtammasan, Erik Garrison, Patrick G. S. Grady, Tina A. Graves-Lindsay, Ira M. Hall, Nancy F. Hansen, Gabrielle A. Hartley, Marina Haukness, Kerstin Howe, Michael W. Hunkapiller, Chirag Jain, Miten Jain, Erich D. Jarvis, Peter Kerpedjiev, Melanie Kirsche, Mikhail Kolmogorov, Jonas Korlach, Milinn Kremitzki, Heng Li, Valerie V. Maduro, Tobias Marschall, Ann M. McCartney, Jennifer McDaniel, Danny E. Miller, James C. Mullikin, Gene Myers, Nathan D. Olson, Benedict Paten, Paul Peluso, Pavel A. Pevzner, David Porubsky, Tamara Potapova, Evgeny I. Rogaev, Jeffrey A. Rosenfeld, Steven L. Salzberg, Valerie A. Schneider, Fritz J. Sedlazeck, Kishwar Shafin, Colin J. Shew, Alaina Shumate, Ying Sims, Arian F. A. Smit, Daniela C. Soto, Ivan Sović, Jessica M. Storer, Aaron Streets, Beth A. Sullivan, Françoise Thibaud-Nissen, James Torrance, Justin Wagner, Brian P. Walenz, Aaron Wenger, Jonathan M. D. Wood, Chunlin Xiao, Stephanie M. Yan, Alice C. Young, Samantha Zarate, Urvashi Surti, Rajiv C. McCoy, Megan Y. Dennis, Ivan A. Alexandrov, Jennifer L. Gerton, Rachel J. O'Neill, Winston Timp, Justin M. Zook, Michael C. Schatz, Evan E. Eichler, Karen H. Miga, and Adam M. Phillippy. The complete sequence of a human genome. *Science*, 376(6588):44–53, Apr 2022. doi:10.1126/science.abj6987.

30 Enno Ohlebusch. *Bioinformatics Algorithms: Sequence Analysis, Genome Rearrangements, and Phylogenetic Reconstruction*. Oldenbusch Verlag, 2013.

31 Enno Ohlebusch and Mohamed I Abouelhoda. Chaining algorithms and applications in comparative genomics. *Handbook of Computational Molecular Biology*, pages 15–1, 2006.

32 Christian Otto, Steve Hoffmann, Jan Gorodkin, and Peter F. Stadler. Fast local fragment chaining using sum-of-pair gap costs. *Algorithms Mol. Biol.*, 6:4, 2011. doi:10.1186/1748-7188-6-4.

33 Filip Pavetić, Ivan Katanić, Gustav Matula, Goran Žužić, and Mile Šikić. Fast and simple algorithms for computing both LCS_k and LCS_{k+}. *arXiv*, 2017. doi:10.48550/ARXIV.1705.07279.

34 Filip Pavetić, Goran Žužić, and Mile Šikić. $LCSk{+}{+}$: Practical similarity metric for long strings. *arXiv*, 2014. doi:10.48550/ARXIV.1407.2407.

35 Jingwen Ren and Mark J. P. Chaisson. lra: A long read aligner for sequences and contigs. *PLoS Comput. Biol.*, 17(6), 2021. doi:10.1371/JOURNAL.PCBI.1009078.

36 Arang Rhie, Sergey Nurk, Monika Cechova, Savannah J. Hoyt, Dylan J. Taylor, Nicolas Altemose, Paul W. Hook, Sergey Koren, Mikko Rautiainen, Ivan A. Alexandrov, Jamie Allen, Mobin Asri, Andrey V. Bzikadze, Nae-Chyun Chen, Chen-Shan Chin, Mark Diekhans, Paul Flicek, Giulio Formenti, Arkarachai Fungtammasan, Carlos Garcia Giron, Erik Garrison, Ariel Gershman, Jennifer L. Gerton, Patrick G. S. Grady, Andrea Guarracino, Leanne Haggerty, Reza Halabian, Nancy F. Hansen, Robert Harris, Gabrielle A. Hartley, William T. Harvey, Marina Haukness, Jakob Heinz, Thibaut Hourlier, Robert M. Hubley, Sarah E. Hunt, Stephen Hwang, Miten Jain, Rupesh K. Kesharwani, Alexandra P. Lewis, Heng Li, Glennis A. Logsdon, Julian K. Lucas, Wojciech Makalowski, Christopher Markovic, Fergal J. Martin, Ann M. Mc Cartney, Rajiv C. McCoy, Jennifer McDaniel, Brandy M. McNulty, Paul Medvedev, Alla Mikheenko, Katherine M. Munson, Terence D. Murphy, Hugh E. Olsen, Nathan D. Olson, Luis F. Paulin, David Porubsky, Tamara Potapova, Fedor Ryabov, Steven L. Salzberg, Michael E. G. Sauria, Fritz J. Sedlazeck, Kishwar Shafin, Valery A. Shepelev, Alaina Shumate, Jessica M. Storer, Likhitha Surapaneni, Angela M. Taravella Oill, Françoise Thibaud-Nissen, Winston Timp, Marta Tomaszkiewicz, Mitchell R. Vollger, Brian P. Walenz, Allison C. Watwood, Matthias H. Weissensteiner, Aaron M. Wenger, Melissa A. Wilson, Samantha Zarate, Yiming Zhu, Justin M. Zook, Evan E. Eichler, Rachel J. O'Neill, Michael C. Schatz, Karen H. Miga, Kateryna D. Makova, and Adam M. Phillippy. The complete sequence of a human Y chromosome. *Nature*, 621(7978):344–354, August 2023. doi:10.1038/s41586-023-06457-y.

37 Nicola Rizzo, Manuel Cáceres, and Veli Mäkinen. Practical colinear chaining on sequences revisited. In Jing Tang, Xin Lai, Zhipeng Cai, Wei Peng, and Yanjie Wei, editors, *Bioinformatics Research and Applications - 21st International Symposium, ISBRA 2025, Helsinki, Finland, August 3-5, 2025, Proceedings, Part II*, volume 15757 of *Lecture Notes in Computer Science*, pages 203–216. Springer, 2025. doi:10.1007/978-981-95-0695-8_17.

38 Nicola Rizzo, Manuel Cáceres, and Veli Mäkinen. Practical colinear chaining on sequences revisited. *CoRR*, abs/2506.11750, 2025. arXiv:2506.11750, doi:10.48550/ARXIV.2506.11750.

39 Kristoffer Sahlin, Thomas Baudeau, Bastien Cazaux, and Camille Marchet. A survey of mapping algorithms in the long-reads era. *Genome Biology*, 24(1):133, 2023.

40 Kristoffer Sahlin and Veli Mäkinen. Accurate spliced alignment of long RNA sequencing reads. *Bioinform.*, 37(24):4643–4651, 2021. doi:10.1093/BIOINFORMATICS/BTAB540.

41 P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, June 1977. doi:10.1016/0020-0190(77)90031-x.

42 W John Wilbur and David J Lipman. Rapid similarity searches of nucleic acid and protein data banks. *Proceedings of the National Academy of Sciences*, 80(3):726–730, 1983.

43 W John Wilbur and David J Lipman. The context dependent comparison of biological sequences. *SIAM Journal on Applied Mathematics*, 44(3):557–567, 1984.

44 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\theta(n)$. *Information Processing Letters*, 17(2):81–84, August 1983. doi:10.1016/0020-0190(83)90075-3.

45 Zheng Zhang, Balaji Raghavachari, Ross C. Hardison, and Webb Miller. Chaining multiple-alignment blocks. *J. Comput. Biol.*, 1(3):217–226, 1994. doi:10.1089/CMB.1994.1.217.