Fast and exact gap-affine partial order alignment with POASTA

van Dijk, Lucas R.; Manson , Abigail L.; Earl, Ashlee M.; Garimella, Kiran V.; Abeel, Thomas

DOI
10.1093/bioinformatics/btae757

Publication date
2025

Document Version
Final published version

Published in
Bioinformatics

Bioinformatics, 2025, **41(1)**, btae757
https://doi.org/10.1093/bioinformatics/btae757
Advance Access Publication Date: 3 January 2025
Original Paper

OXFORD

Genome analysis

Fast and exact gap-affine partial order alignment with POASTA

Lucas R. van Dijk [1,2,*], Abigail L. Manson [1], Ashlee M. Earl [1], Kiran V Garimella [3], Thomas Abeel [1,2]

[1]Infectious Disease and Microbiome Program, Broad Institute of MIT and Harvard, Cambridge, MA 02142, United States
[2]Delft Bioinformatics Lab, TU Delft, 2628 XE Delft, The Netherlands
[3]Data Sciences Platform, Broad Institute of MIT and Harvard, Cambridge, MA 02142, United States

*Corresponding author. Delft Bioinformatics Lab, TU Delft, Van Mourik Broekmanweg 6, Zuid-Holland, 2628 XE Delft, The Netherlands.
E-mail: lvandijk@broadinstitute.org

Associate Editor: Peter Robinson

Abstract

Motivation: Partial order alignment is a widely used method for computing multiple sequence alignments, with applications in genome assembly and pangenomics, among many others. Current algorithms to compute the optimal, gap-affine partial order alignment do not scale well to larger graphs and sequences. While heuristic approaches exist, they do not guarantee optimal alignment and sacrifice alignment accuracy.

Results: We present POASTA, a new optimal algorithm for partial order alignment that exploits long stretches of matching sequence between the graph and a query. We benchmarked POASTA against the state-of-the-art on several diverse bacterial gene datasets and demonstrated an average speed-up of 4.1× and up to 9.8×, using less memory. POASTA's memory scaling characteristics enabled the construction of much larger POA graphs than previously possible, as demonstrated by megabase-length alignments of 342 *Mycobacterium tuberculosis* sequences.

Availability and implementation: POASTA is available on Github at https://github.com/broadinstitute/poasta.

1 Introduction

Multiple sequence alignments (MSAs) are central to computational biology. MSAs have many applications, including computing genetic distances, which can serve as a basis for a phylogeny; determining consensus sequences, e.g. to perform read error correction; and identifying allele frequencies, e.g. for sequence motif identification.

Computing the optimal MSA with the "sum of all pairs" (SP) score is an NP-complete problem (Wang and Jiang 1994). These classical exact algorithms have a runtime exponentially related to the number of sequences and are thus intractable for even modest-sized datasets. Instead, nearly all popular MSA tools, including MAFFT (Katoh and Standley 2013) and MUSCLE (Edgar 2004), compute the MSA progressively: first, an alignment between two sequences is computed, then additional sequences are added one by one until all sequences have been aligned. The runtime of these approaches is linear in the number of sequences instead of exponential. While MSAs computed this way do not necessarily find the globally optimal solution for the SP objective, they are still highly useful approximations to otherwise intractable alignment problems.

Partial order alignment (POA) is a well-known progressive MSA approach that pioneered using a graph to represent an MSA rather than a sequence profile (Lee *et al.* 2002). This improved the ability to represent indels, leading to higher-quality alignments. Since POA is a progressive MSA algorithm, the optimal SP score is not guaranteed for the entire MSA. However, POA does guarantee that each individual sequence-to-graph alignment is optimal.

POA is relevant to many applications, including *de novo* genome assembly (e.g. read error correction and consensus generation) (Chin *et al.* 2013, Loman *et al.* 2015, Vaser *et al.* 2017), RNA isoform inference (Lee 2003), structural variant (SV) characterization (Chaisson *et al.* 2019), and variant phasing (Holt *et al.* 2023).

POA is also essential to two recent human pangenome graph construction pipelines (Garrison *et al.* 2024, Hickey *et al.* 2024). These pipelines are pushing the limits of POA, as aligning long stretches of homologous sequence among input genomes requires substantial computing and memory resources. For example, consider the gap-affine alignment of a 500 kbp sequence to a graph with 500k character-labeled nodes. Conventional POA approaches have a runtime and memory complexity of $O(|V|m)$, i.e. a product of the number of nodes in a POA graph $|V|$ and the sequence length m. This example would, therefore, require about 3 TB of RAM (assuming 32-bit integers for storing alignment costs in three alignment state matrices).

Several tools, including SPOA (Vaser *et al.* 2017) and abPOA (Gao *et al.* 2021), have been developed to address the need for faster and more memory-efficient POA algorithms. The current state-of-the-art, SPOA, is a reimplementation of the original algorithm, which accelerates computing the dynamic programming (DP) matrix by using single-instruction-

Received: 31 May 2024; **Revised:** 19 November 2024; **Editorial Decision:** 21 December 2024; **Accepted:** 2 January 2025

multiple-data (SIMD) instructions available on modern CPUs. While faster, SPOA still computes the full DP matrix and thus does not ameliorate demands on memory usage. abPOA additionally improves performance by applying an adaptive banding strategy to partially compute the DP matrix. However, this sacrifices the guarantee of finding the optimal sequence-to-graph alignment.

Here, we present POASTA: a fast, memory-efficient, and optimal POA algorithm that computes many fewer alignment states than SPOA, thus enabling the construction of much larger POA graphs (Fig. 1). It is built on top of the A* algorithm (Hart *et al.* 1968), with a new POA-specific heuristic. Inspired by the recently published wavefront algorithm for pairwise alignment (Marco-Sola *et al.* 2021), it also exploits exact matches between a query sequence and the graph. We additionally introduce a novel superbubble-informed (Onodera *et al.* 2013) technique for pruning the number of computed alignment states without sacrificing alignment optimality. We benchmarked POASTA against SPOA (Vaser *et al.* 2017) on diverse sets of bacterial housekeeping genes extracted from RefSeq and demonstrated its increased performance. Additionally, we constructed megabase-length alignments of 342 *Mycobacterium tuberculosis* sequences, demonstrating its reduced memory usage and highlighting POASTA's ability to align much longer sequences than previously possible.

2 Materials and methods

POA algorithms compute an MSA by iteratively computing the alignment of a query to a directed acyclic graph (DAG) representing the MSA from the previous iteration (Lee *et al.* 2002). Instead of the original DP formulation (Supplementary Methods; Supplementary Fig. S1a), POASTA's algorithm is based on an "alignment graph" (Supplementary Fig. S1b; not to be confused with the POA graph), enabling the use of common graph traversal algorithms such as the A* algorithm to compute alignments (Hart *et al.* 1968, Rautiainen and Marschall 2017, Ivanov *et al.* 2020, Jain *et al.* 2020). POASTA further accelerates alignment using three novel techniques: (1) a cheap-to-compute, POA-specific heuristic for the A* algorithm (Fig. 2a), (2) a depth-first search (DFS) component, greedily aligning exact matches between the query and the graph (Fig. 2b); and (3) a method to detect and prune alignment states that are not part of the optimal solution, informed by the POA graph topology (Fig. 2c). Together, they substantially reduce the number of computed alignment states (Supplementary Fig. S2).

2.1 Definitions and notation

To describe the algorithm in detail, we will use the following notation. A POA graph $G = (V, E)$ is a character-labeled DAG, where nodes $v \in V$ represent the symbols in the input sequences, each labeled with a character from an alphabet Σ. Edges $(u, v) \in E$ connect nodes that are adjacent in at least one input sequence. We additionally assume the POA graph has a special start node ν with outgoing edges to all nodes with no other incoming edges and a special termination node τ with incoming edges from all nodes with no other outgoing edges.

The optimal alignment of a query sequence $Q = q_1 q_2 \ldots q_m$ (of length m) to G is the alignment of Q to a path $\pi = \nu v_1 v_2 \ldots v_n \tau$, spelling a sequence R that minimizes the alignment cost C (Supplementary Fig. S1a). Commonly used cost models are linear gap penalties and gap-affine penalties. In the former, each gap position is weighted equally, and the alignment cost is defined as $C = N_m \Delta_m + N_x \Delta_x + N_g \Delta_g$, where N_m represents the number of matches, N_x is the number of mismatches, and N_g is the total length of gaps. The cost of each alignment operation is represented by Δ_m, Δ_x, and Δ_g, representing the cost of a match, mismatch, and a gap, respectively. In the case of gap-affine penalties, opening a new gap has a different (typically higher) cost than extending an existing gap. The total cost is defined as $C = N_m \Delta_m + N_x \Delta_x + N_o \Delta_o + N_g \Delta_e$, with N_o the number of distinct gaps and Δ_o the cost of opening a new gap, and Δ_e the cost of extending a gap (Durbin *et al.* 1998). POASTA supports both the gap-linear and the gap-affine cost models,

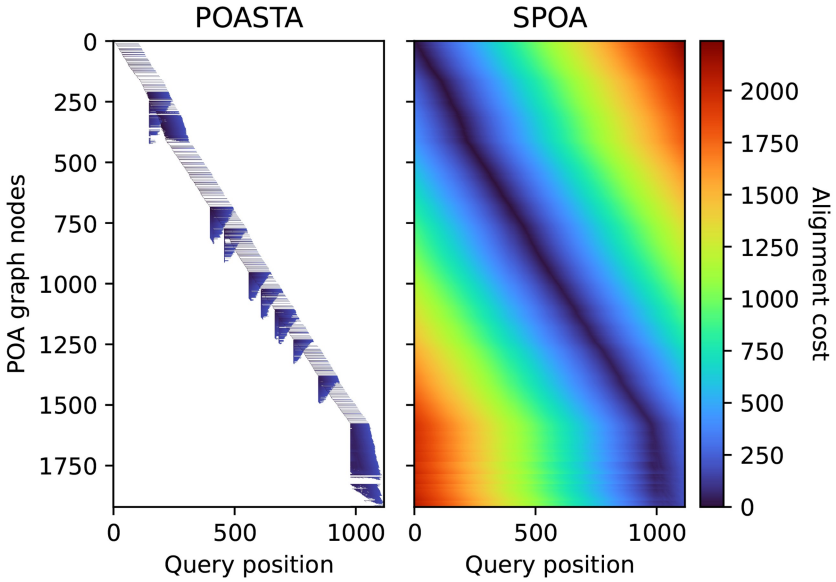

Figure 1. Representation of the dynamic programming matrix to compute the global alignment of a *nusA* gene sequence (*x*-axis) to a POA graph constructed from 50 other *nusA* gene sequences (*y*-axis). Each pixel represents a computed alignment state, and the color represents the alignment cost of that state. White pixels represent uncomputed states.

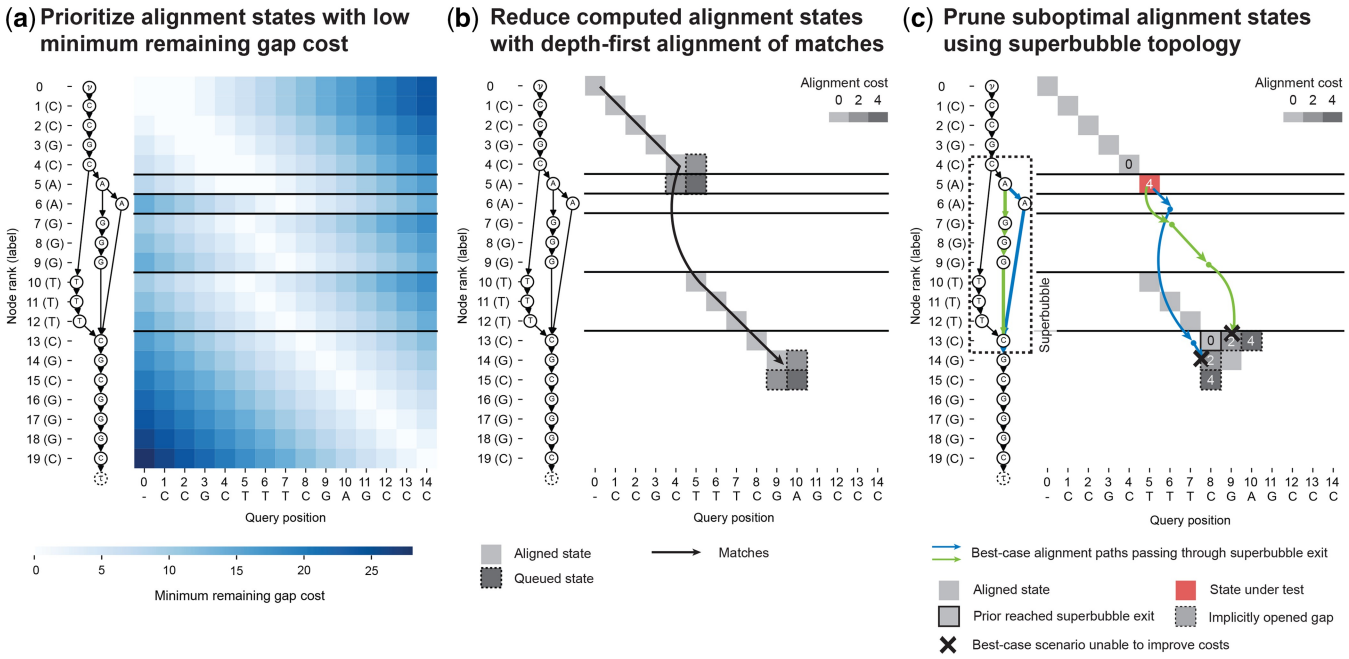

Figure 2. POASTA is based on the A* algorithm and accelerates alignment through three algorithmic innovations: (a) A novel heuristic for POA that prioritizes alignment states with a low minimum remaining gap cost (light-colored squares); i.e. states where the unaligned query sequence length is similar to the path lengths to the POA graph end node τ. (b) Reducing the number of computed alignment states by combining the A* algorithm with a depth-first search component, greedily aligning matches between the query and a path in the graph (black arrow). Adjacent insertion and deletion states are only queued when encountering a mismatch (squares with dashed borders). (c) Using knowledge about *superbubble* topology to prune states not part of the optimal solution. POASTA checks whether the best-case alignment paths (blue and green arrows) from a state under test (red square) can improve over the costs of implicitly opened gaps from prior reached bubble exits (bordered squares). All examples use the linear gap cost model with $\Delta_m = 0, \Delta_x = 4, \Delta_g = 2$.

though it constrains Δ_m to be zero and all other costs $\Delta_x, \Delta_o, \Delta_g, \Delta_e$ to be ≥ 0. Additionally, in case of the gap-affine model, it requires that the gap open cost Δ_o is greater than the gap extension cost Δ_e. For clarity, we focus on the gap-linear cost model; the use of POASTA with the gap-affine cost is explained in the Supplemental Methods.

The alignment graph $G^A = (V^A, E^A)$ is a product of the POA graph and the query sequence, and paths through it represent possible alignments between them. Nodes $\langle v, i \rangle \in V^A = (V \times \{0, 1, \ldots, m\})$ represent "alignment states" with a cursor pointing to a node v in the POA graph and a cursor to a query position i (Supplementary Fig. S1b). Edges in the alignment graph correspond to different alignment operations, such as (mis)match, insertion, or deletion, and are weighted with the respective alignment cost. Edges connect alignment states where either one (indel) or both of the cursors have moved ((mis)match), and the construction of edges is further detailed in the Supplementary Methods. The lowest-cost path in the alignment graph from $\langle \nu, 0 \rangle$ to alignment termination state $\langle \tau, m \rangle$ is equivalent to the optimal alignment of Q to G.

2.2 Optimal alignment with A* using a minimum remaining gap cost heuristic

To compute the lowest-cost path in the alignment graph, i.e. the optimal alignment, POASTA uses the A* algorithm (Hart *et al.* 1968). For POASTA, we adapted the widely used gap cost heuristic for pairwise alignment to POA (Fig. 2a) (Ukkonen 1985, Hadlock 1988). This heuristic is "admissible", i.e. a lower bound on the true remaining cost, thus guaranteeing that A* finds the lowest-cost path. The intuition behind the heuristic is to prioritize alignment states in which the length of the unaligned query sequence is similar to the path lengths to the end node τ.

To compute heuristic $h\langle v, i \rangle$, POASTA scans the POA graph before alignment starts and stores the shortest and longest path length to the end node τ for all nodes in the graph, denoted as $d^{\min}_{v,\tau}$ and $d^{\max}_{v,\tau}$. This can be computed in $O(V+E)$ time by visiting the nodes in reverse topological order. POASTA compares these path lengths to the length of the unaligned query sequence $l_r = m - i$ and infers the minimum number of indel edges to traverse from $\langle v, i \rangle$ to the alignment termination $\langle \tau, m \rangle$ state as follows:

Definition 1 (Minimum number of indel edges).

$$N_g^{\min} = \begin{cases} l_r - (d^{\max}_{v,\tau} - 1) & \text{if } d^{\max}_{v,\tau} - 1 < l_r \\ (d^{\min}_{v,\tau} - 1) - l_r & \text{if } d^{\min}_{v,\tau} - 1 > l_r \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

We subtract one from $d^{\min}_{v,\tau}$ and $d^{\max}_{v,\tau}$ to exclude the edge toward τ.

Proof. See Supplemental Methods. □

Combining the computed minimum number of indel edges to traverse with the alignment cost model, e.g. the linear gap cost model, enables us to compute the heuristic.

Definition 2 (Minimum remaining gap cost heuristic).

$$h\langle v,i\rangle = N_g^{\min}\Delta_g \tag{2}$$

Lemma 1 (Admissibility). *$h\langle v,i\rangle$ is admissible.*

Proof. The true remaining alignment cost, using linear gap penalties and assuming a match cost Δ_m of zero, is defined as $C_r = N_x\Delta_x + N_g\Delta_g$, where N_x and N_g represent the number of remaining mismatches and the total remaining gap length, respectively, Δ_x the mismatch cost, and Δ_g the gap cost.
Using Definition 1, we infer that $N_g \geq N_g^{\min}$. Since the mismatch cost $\Delta_x \geq 0$, we note that the $N_x\Delta_x \geq 0$, and thus observe that

$$N_x\Delta_x + N_g\Delta_g \geq N_g^{\min}\Delta_g \Rightarrow C_r \geq h\langle v,i\rangle.$$

$h\langle v,i\rangle$ is thus a lower bound on the true remaining alignment cost. □

2.3 Depth-first alignment of exact matches between query and graph

To further speed-up alignment and reduce the number of computed alignment states, POASTA greedily aligns exact matches between the query and graph (Fig. 2b). This is possible because POASTA requires that the alignment cost for a match is zero and all other alignment costs be ≥ 0. Traversing a match edge $\langle u,i\rangle \rightarrow \langle v,i+1\rangle$ will always be the optimal choice if the latter state has not been visited yet since match edges have zero cost and all other paths (requiring indels) will have higher or equal cost (Ivanov *et al.* 2020, Marco-Sola *et al.* 2021). This implies that in the presence of an unvisited match, we can ignore insertion edge $\langle u,i\rangle \rightarrow \langle u,i+1\rangle$ and deletion edge $\langle u,i\rangle \rightarrow \langle v,i\rangle$.

To implement this, POASTA combines the regular A* algorithm with a depth-first search (DFS) component. When a state $\langle u,i\rangle$ is popped from the A* queue, we initiate a DFS from this state. We assess whether a successor state $\langle v,i+1\rangle \quad v : (u,v) \in E$ is a match; if it is, we push it on the stack to be processed in the next DFS iteration; when there is a mismatch, we append it to the A* queue. In the latter case, we no longer can ignore the insertion and deletion edges, so we additionally queue insertion state $\langle u,i+1\rangle$, and deletion state $\langle v,i\rangle$. Note that, just like regular DFS, a state is removed from the stack after all its successors (matches or mismatches) have been explored. Thus, using DFS enables greedily aligning long stretches of exact matches, even in the presence of branches in the graph.

2.4 Pruning alignment states not part of the optimal solution

When POASTA's depth-first alignment finds a long stretch of matching sequence, the corresponding path through the POA graph might traverse a "superbubble" (Onodera *et al.* 2013). A superbubble (s, t) is a substructure in the POA graph with specific topological features (Supplementary Fig. S3): it is acyclic; it has a single entrance s and a single exit t; all paths leaving s should end in t; and no path from "outside" the superbubble can have an endpoint inside the bubble. The set of nodes U on paths from s to t is called the "interior" of a bubble, which can be empty. In a POA graph, superbubbles represent the alleles present at particular loci in the MSA.

POASTA exploits the fact that all paths through a superbubble have a common endpoint, its exit t. If an alignment state $\langle t,p\rangle$ is reached during alignment with a particular cost $C_{\langle t,p\rangle}$, POASTA can detect whether another yet-to-visit state $\langle v,i\rangle : v \in U \cup \{s\}$ that is part of the same superbubble, can improve over this cost. This is especially effective when combined with the depth-first greedy alignment described above; if a bubble exit is reached at a low cost because of a long stretch of matching sequence, we can often prune alignment states on alternative paths through the bubble because they cannot improve over the already-found path.

To quickly retrieve topological information about superbubbles, POASTA constructs a *superbubble index* before alignment. For every node in the POA graph, it stores the superbubbles in which it is contained, along with the shortest and longest path length to the corresponding superbubble exit (Fig. 3a). For example, the red node (node 5) in the example shown in Fig. 3b has two paths to the superbubble exit (node 13): one path with length 2 (blue) and one path with length 4 (green). POASTA identifies superbubbles using the $O(V+E)$ algorithm described by Gärtner *et al.* (2018). The shortest path lengths can be computed using a backward breadth-first search (BFS), and the longest path lengths can be computed by recursively visiting nodes in postorder, both $O(V+E)$ operations.

To test if a state $\langle v,i\rangle$ should be pruned, POASTA first uses the superbubble index to infer the range of states $\langle t,j^{\min}\rangle \ldots \langle t,j^{\max}\rangle$ reachable from $\langle v,i\rangle$ assuming the best-case scenario of traversing zero-cost match edges (Fig. 3c). For example, when aligning a query `CCGCTTTCGAGCCC` to the graph in Fig. 3b, POASTA will initially find a long stretch of matches between the query and a path in the graph, traversing the superbubble (4, 13) (Fig. 3d; grey squares). In a following iteration, it tests alignment state $\langle 5,5\rangle$, where node 5 is part of the same superbubble (4, 13), which is reached with an alignment cost 4 (Fig. 3d; red square). It looks up the path lengths to the superbubble exit $d_{5,13}^{\min} = 2$ and $d_{5,13}^{\max} = 4$ and infers that we can reach $\langle 13,7\rangle$ and $\langle 13,9\rangle$ from $\langle 5,5\rangle$ with the same alignment cost of four (Fig. 3d; blue and green arrows and dotted squares).

POASTA can now compare this best-case alignment cost, when reached from a state $\langle v,i\rangle$, to the alignment costs of states that reached the superbubble exit prior, or an "implicitly opened gap" from those. Implicitly opened gap costs are upper bounds on the cost for yet-to-visit alignment states and are computed on the fly when testing to prune a state (Fig. 3e). For example, the green path in Fig. 3d could reach alignment state $\langle 13,9\rangle$ with an alignment cost of four. However, alignment state $\langle 13,9\rangle$ is also reachable from the prior reached bubble exit $\langle 13,8\rangle$, by opening an insertion and reaching it with a lower cost of two (Fig. 3e). Similarly, the blue path in Fig. 3d could reach alignment state $\langle 13,7\rangle$ with an alignment cost of four. This state has not yet been reached and is also not reachable by opening a gap from a previously reached exit. However, suppose we extend the blue path, assuming additional traversal of zero-cost match edges. In that case, we reach an alignment state $\langle 14,8\rangle$ which is reachable from a previously reached exit by opening a deletion. The opened deletion would reach $\langle 14,8\rangle$ with a cost of two, lower than the cost of four when reached through the blue path. Since both best-case scenarios from $\langle 5,5\rangle$ would result in

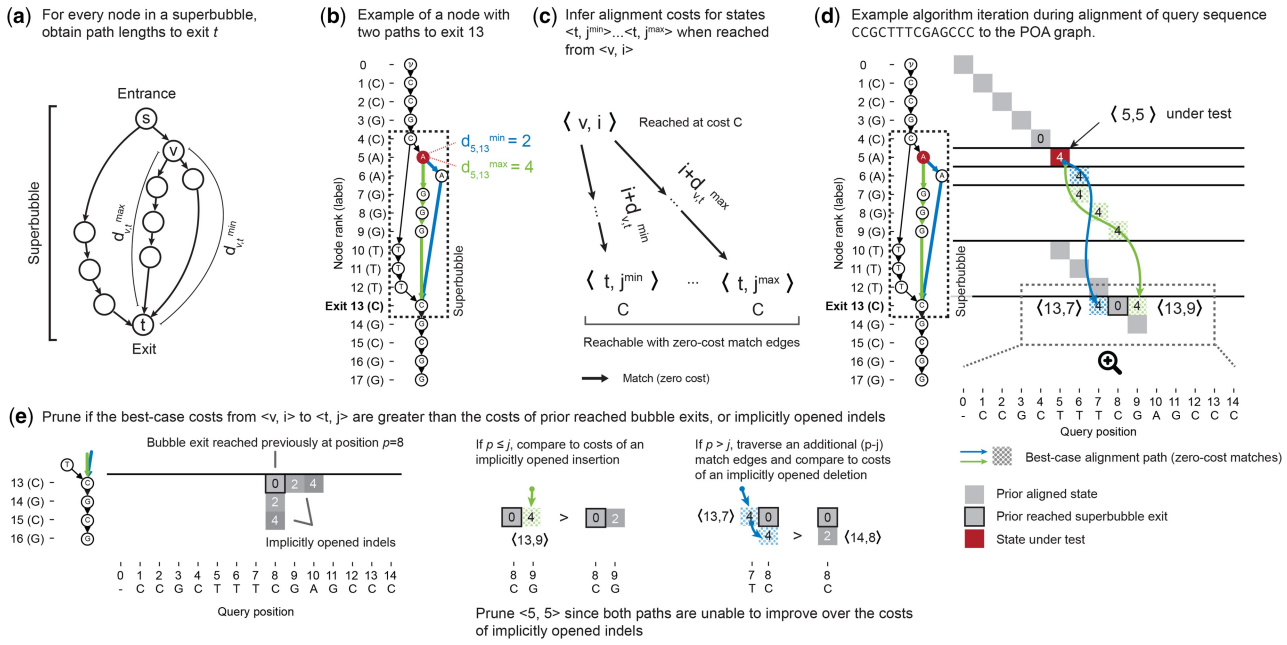

Figure 3. POASTA detects and prunes alignment states that are not part of the optimal solution. (a) A superbubble with entrance s and exit t. For every node v in the superbubble, POASTA stores the minimum and maximum path length, $d^{\text{min}}_{v,t}$, and $d^{\text{max}}_{v,t}$, to exit t. (b) An example POA graph with a superbubble (dashed rectangle) and the path lengths from the highlighted node 5 (red) to the superbubble exit (green and blue paths). (c) The path lengths to the superbubble exit are used during alignment to infer the range of states reachable with zero-cost match edges (black arrows) from another state $\langle v, i \rangle$. (d) An example aligning the query CCGCTTTCGAGCCC to the graph in (b). Grey squares: states aligned in a prior iteration. Red square: state under test. Blue and green arrows and dotted squares: best-case alignment paths from the state under test to the superbubble exit. (e) POASTA compares best-case alignment costs from $\langle v, i \rangle$ (blue and green dotted squares) to implicitly opened indels from prior reached bubble exits (grey squares). Implicitly opened indels act as an upper bound for the alignment cost of yet-to-visit states. Examples use the linear gap cost model with $\Delta_m = 0, \Delta_x = 4, \Delta_g = 2$.

higher alignment costs compared to opened indels from a prior reached exit, POASTA infers $\langle 5, 5 \rangle$ will not be part of the optimal solution and prunes it from further consideration.

In the example discussed above, the bubble exit was only reached once. Bubble exits, however, can be reached multiple times during alignment (with varying alignment costs). All previously reached positions should be considered when testing whether a state can be pruned (Supplementary Fig. S4). The Supplemental Methods further detail how POASTA prunes alignment states when the bubble exit has been reached multiple times.

3 Results

3.1 Benchmarking using bacterial housekeeping genes

To compare POASTA's speed and memory usage to the current state-of-the-art, we generated multiple benchmark datasets from bacterial housekeeping genes (*dnaG*, *nusA*, *pgk*, *pyrG*, and *rpoB*). These genes are present in nearly all bacteria and are commonly used to create bacterial phylogenies, requiring MSA (Wu and Eisen 2008). We downloaded all 40 188 RefSeq-complete genomes representing the breadth of bacterial diversity and extracted genes of interest using the accompanying gene annotations. Gene sequences were deduplicated and coarsely clustered using single-linkage hierarchical clustering. This resulted in multiple genus-spanning clusters. For each gene family, we selected one or more clusters as benchmark datasets, choosing clusters with at least 100 sequences and varying pairwise ANI (Supplemental Methods). The 13 selected benchmark sets each contained 140–2385 gene sequences, with mean sequence lengths of 1–4 kbp and pairwise ANIs of 82%–97% (Supplemental Table S1).

3.2 POASTA constructs MSAs 4× faster than other optimal methods

We assessed POASTA's runtime and memory compared to SPOA, the only other POA algorithm that guarantees optimal POAs (Vaser *et al.* 2017). We did not benchmark against general sequence-to-graph aligners such as Astarix (Ivanov *et al.* 2020), GWFA (Zhang *et al.* 2022), PaSGAL (Jain *et al.* 2019), and GraphAligner (Rautiainen and Marschall 2020) since these are unable to compute MSAs and we would be unable to compare total runtime. We ran POASTA and SPOA to compute the full MSA of the 13 selected datasets and recorded their total runtime and memory usage.

For 12 of 13 datasets, POASTA computed the complete MSA faster than SPOA, achieving an average speed-up of 4.1×. The highest speed-up was 9.8× (Fig. 4a). The one instance where SPOA was faster corresponded to the gene set with the lowest pairwise ANI (82.6%). POASTA's strongest relative performance was in settings with ANIs of 90%–100% and sequences longer than 1500 bp (Fig. 4b and c). Furthermore, SPOA required, on average, 2.6× more memory than POASTA (Fig. 4d–f).

We also compared POASTA's runtime and memory to abPOA, a popular tool for POA that does not guarantee optimal alignment (Gao *et al.* 2021). As expected, due to its adaptive banding strategy, abPOA is faster than POASTA (3.5×; Supplementary Fig. S5a). Surprisingly, abPOA used more memory than POASTA across nearly all benchmark sets (Supplementary Fig. S5b), as it allocates memory for the entire matrix, even though it only computes a fraction of it.

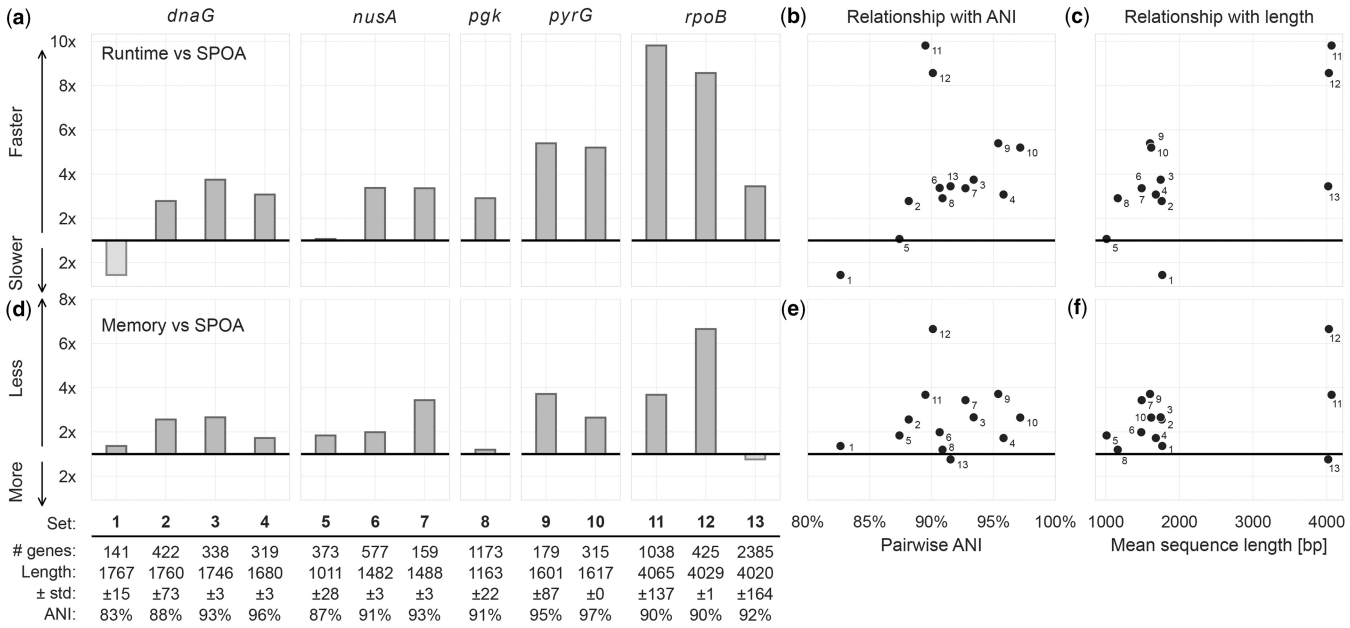

Figure 4. POASTA creates multiple sequence alignments of sequences from five bacterial housekeeping genes, in all but one case faster than SPOA and with less memory. (a) Relative runtime of POASTA compared to SPOA for each set of gene sequences. (b) The relationship between pairwise ANI of each gene sequence set and POASTA's relative runtime. (c) The relationship between mean sequence length and POASTA's relative runtime. (d) Relative memory usage of POASTA compared to SPOA for each set of gene sequences. (e) The relationship between pairwise ANI of each sequence set and POASTA's relative memory usage. (f) The relationship between the mean sequence length of each sequence set and POASTA's relative memory usage.

For our dataset, we found that abPOA found the optimal alignment the vast majority (99.8%) of the time (Supplemental Methods). However, our test dataset had few large indels and adaptive banding strategies are known to miss the optimal alignment more frequently in the presence of indels larger than the band size (Suzuki and Kasahara 2017). For many cases where the optimal alignment was missed in our test dataset, abPOA produced erroneous alignments that started or ended at unexpected nodes. This resulted in alignment costs that were lower than the global optimum reported by SPOA and POASTA, which should be impossible (see Supplemental Methods).

3.3 POASTA enables the construction of megabase-length POA graphs

To further test POASTA's limits, we benchmarked its ability to align datasets with average sequence lengths of approximately 250 kbp, 500 kbp, and 1 Mbp. We extracted subsequences from all 370 RefSeq-complete whole genome assemblies of *M. tuberculosis*, covering a broad range of the species' diversity (including representatives from all known lineages; Mash-estimated average pairwise ANI of 99.3%; Ondov *et al.* 2016). *M. tuberculosis* has relatively little large-scale structural variation, including few large inversions or genes translocating to different locations, which POA cannot model and align accurately. After orienting genomes such that each started with the gene *dnaA*, we truncated at specific shared genes to achieve sequences of the desired length (Supplemental Methods). For the 250 kbp, 500 kbp, and 1 Mbp benchmarks, we truncated at the genes *trmB*, *thiE*, and *gltA2*, respectively. Since POA expects sequences to be colinear, we also excluded 28 genomes with more than 15% ($\geq$ 660 kbp) of its complete genome inverted with respect to the canonical reference H37Rv (Supplemental Methods).

POASTA successfully computed MSAs for the 250 kbp, 500 kbp, and 1 Mbp benchmark sets with manageable runtimes and memory (Table 1). None of these alignments could be completed with SPOA or abPOA, which required more memory than the 240 GB available in the Google Cloud VM used for benchmarking (Supplemental Methods). The estimated memory requirements for the 250, 500, and 1000 kbp benchmarks would be 0.95, 3.5, and 13 TB, respectively (assuming 32-bit integers for storing scores).

We assessed computed alignments at a known drug resistance locus to validate that the MSA correctly captured known variation. In *M. tuberculosis*, the S450L change in the *rpoB* gene is one of the most common rifampicin resistance-causing mutations (Munir *et al.* 2019, Jamieson *et al.* 2014). We first characterized codons representing the 450th amino acid of *rpoB* using just the reference genomes and accompanying gene annotations. We obtained each codon using the start position of the *rpoB* gene to compute the reference locus representing the 450th amino acid of *rpoB*. In our set of genomes, we similarly observed that the S450L mutation is the most common allele present other than the reference or wild-type allele (Table 2; 103 genomes have the S450L mutation). To check if the observed codons were correctly aligned in the POA graph, we extracted a small subgraph surrounding the 450th amino acid of *rpoB* in H37Rv (Fig. 5). While this subgraph was obtained using H37Rv coordinates, all

Table 1. POASTA runtime and peak memory usage for three benchmark sets comprising 342 *M. tuberculosis* sequences of approximately 250, 500, and 1000 kbp.

Sequence set (kbp)	Runtime (h)	Max. memory (GB)
250	5.3	63.8
500	24	120
1000	69	231

codons listed in Table 2 were also represented as different paths in the graph, and the graph edge counts, indicating the number of genomes sharing that edge, matched the codon counts obtained through gene annotations. POASTA thus correctly captured known variation at this locus while the alignments were computed unaware of genes.

4 Discussion

In this work, we introduced POASTA, an optimal POA algorithm supporting gap-affine penalties with increased performance. These improvements are achieved using three algorithmic innovations: a minimum remaining gap cost heuristic for A*, depth-first greedy alignment of matches, and pruning states not part of the optimal solution using superbubble topology. In benchmarking on short sequences (1–4 kbp), POASTA was, on average, 4.1× faster than the current state-of-the-art SPOA (Vaser *et al.* 2017) and used 2.6× less memory. On longer sequences (250–1000 kbp), POASTA generated alignments with manageable runtime and memory, while SPOA failed.

POASTA includes several algorithmic innovations inspired by recent advances in pairwise and graph alignment. For example, POASTA takes inspiration from the recently published wavefront algorithm (WFA), a fast algorithm for pairwise alignment (Marco-Sola *et al.* 2021). WFA similarly exploits exact matches between sequences and rapidly computes alignments by only considering the furthest-reaching points on DP matrix diagonals. However, their DP matrix diagonal formulation does not directly apply to graph alignment. In contrast to pairwise alignment, a stretch of exact matches between the query and the graph may span multiple diagonals in the DP matrix because of branches in the graph, complicating the definition of furthest-reaching points. While others have introduced variants of the WFA for graphs (Zhang *et al.* 2022, Holt *et al.* 2023), none support the gap-affine scoring model, which is preferred because it gives more biologically relevant alignments Durbin *et al.* (1998). As an alternative to processing only the furthest-reaching points on a diagonal, POASTA uses its knowledge of graph topology, as stored in its superbubble index, to detect and prune alignment states that are not part of the optimal solution, thus speeding up alignment.

POASTA additionally takes inspiration from the recent read-to-graph aligner Astarix. Like POASTA, Astarix uses the A* algorithm for alignment, though with a different heuristic (Ivanov *et al.* 2020, 2022). The benefit of our minimum remaining gap cost A* heuristic is the simplicity of the required computation. All preprocessing can be done in $O(V+E)$ time, and all the necessary data are stored in $O(V)$ additional memory. The fast computation of the heuristic is important because the POA graph is updated at each iteration. Combined, these innovations can substantially reduce the number of computed alignment states, speeding up the construction of the complete MSA and enabling MSAs for longer sequences than was previously possible.

POASTA did not improve over SPOA in every scenario—it performed less well than SPOA in settings with high sequence diversity, where there are fewer stretches of exact matches for POASTA to exploit. In this situation, POASTA must explore more mismatch and indel states, increasing computation time. Though POASTA still computes fewer alignment states than SPOA, its runtime can become longer because the A* algorithm is less predictable and CPU cache-efficient than computing the full DP matrix row-by-row in a contiguous block of memory. Despite POASTA's higher compute time "per alignment state" compared to SPOA, the reduction in computed alignment states is often large enough to gain a net decrease in total runtime. To further develop our understanding of POASTA's performance characteristics, future work could include determining tight upper bounds on its runtime complexity, e.g. by adapting the arguments of Myers' $O(nd)$

Table 2. Diversity of codons across 342 *M. tuberculosis* genomes representing the 450th amino acid in the *rpoB* gene.[a]

Codon	Amino acid	Count
TCG (reference)	S	232
TTG	L	103
TTT	F	2
TNG	–	2
GCG	A	1
TGG	W	1
TYG	–	1

[a] In three genomes, there was uncertainty about the second base in the triplet indicated by IUPAC code N (any base) or Y (C or T).

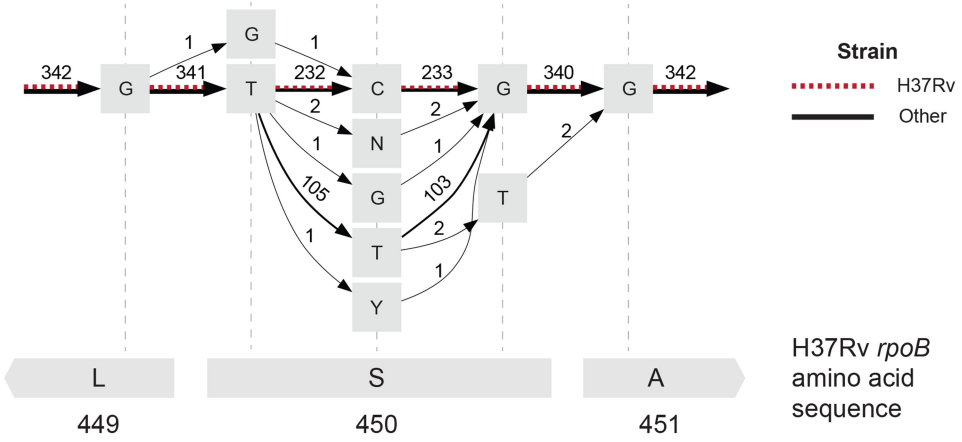

Figure 5. The POA subgraph surrounding the 450th amino acid in *M. tuberculosis* H37Rv *rpoB* (red dashed edges) captures extensive allelic diversity in other references (black edges). Grey squares represent nodes in the POA graph labeled with a base or an IUPAC code representing uncertainty about the base at that site (N: any base, Y: C or T). Edges are labeled with the number of genomes that share that edge. The bottom grey rectangles represent the H37Rv amino acid sequence.

algorithm for pairwise alignment to the sequence-to-graph alignment problem (Myers 1986).

We envision several future improvements to the POASTA algorithm. POASTA could be expanded to support dual gap-affine penalties, enabling computing improved alignments in the presence of large indels (Sedlazeck *et al.* 2018). Bi-directed variants of the A* algorithm, where the search for the shortest path is started from both the start and the end, could substantially improve POASTA's runtime with respect to sequence diversity (de Champeaux 1983). A more informative A* heuristic, e.g. the recently published seed-heuristic (Ivanov *et al.* 2022) or one inspired by A*PA2 (Groot Koerkamp 2024), could speed-up alignment by improving estimates of the remaining alignment cost, improving the prioritization of alignment states to visit. Other strategies could be to utilize GPUs since massively parallel versions of A* exist (Zhou and Zeng 2015). Finally, we could combine the superbubble index with the Gwfa algorithm (Zhang *et al.* 2022) to link diagonals across nodes and increase power to prune suboptimal alignment states.

5 Conclusions

We present POASTA, a novel optimal algorithm for POA. Through several algorithmic innovations, POASTA computed the complete MSA faster than existing tools in diverse bacterial gene sequence sets. It further enabled the creation of much longer MSAs, as demonstrated by successfully constructing MSAs from *M. tuberculosis* sequence sets with average sequence lengths of up to 1 Mbp. The algorithms and ideas presented here will accelerate the development of scalable pangenome construction and analysis tools that will drive the coming era of genome analysis.

Acknowledgements

We would like to thank Fabio Cunial and Ryan Lorig-Roach for their helpful discussions and their reviews of early versions of the article.

Supplementary data

Supplementary data are available at *Bioinformatics* online.

Conflict of interest: None declared.

Funding

This project was funded in part with federal funds from the National Institute of Allergy and Infectious Diseases, National Institutes of Health, Department of Health and Human Services, under Grant Number U19AI110818 to the Broad Institute.

Data availability

POASTA is written in Rust and available under the BSD-3-clause license at https://github.com/broadinstitute/poasta (DOI: 10.5281/zenodo.11153323). POASTA is available as both a standalone utility and a Rust crate that can be included as part of other software packages. The benchmark suite is written in Rust and Python and is available under the same license at https://github.com/broadinstitute/poa-bench. The data underlying this article are included in the benchmark suite repository (DOI: 10.5281/zenodo.11153368).

References

Chaisson MJP, Sanders AD, Zhao X *et al.* Multi-platform discovery of haplotype-resolved structural variation in human genomes. *Nat Commun* 2019;**10**:1784. https://doi.org/10.1038/s41467-018-08148-z

Chin C-S, Alexander DH, Marks P *et al.* Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nat Methods* 2013;**10**:563–9. https://doi.org/10.1038/nmeth.2474

de Champeaux D. Bidirectional heuristic search again. *J ACM* 1983;**30**:22–32. https://doi.org/10.1145/322358.322360

Durbin R, Eddy S, Krogh A *et al. Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, 1st edn. Cambridge, UK: Cambridge University Press, 1998. https://doi.org/10.1017/CBO9780511790492

Edgar RC. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res* 2004;**32**:1792–7. https://doi.org/10.1093/nar/gkh340

Gao Y, Liu Y, Ma Y *et al.* abPOA: an SIMD-based C library for fast partial order alignment using adaptive band. *Bioinformatics* 2021;**37**:2209–11. https://doi.org/10.1093/bioinformatics/btaa963

Garrison E, Guarracino A, Heumos S. *et al.* Building pangenome graphs. *Nat Methods* 2024;**21**:2008–12. https://doi.org/10.1038/s41592-024-02430-3

Groot Koerkamp R. A*PA2: Up to 19× Faster Exact Global Alignment. In: 24th International Workshop on Algorithms in Bioinformatics (WABI 2024). Leibniz International Proceedings in Informatics (LIPIcs), Vol. 312, pp. **17**:1–25, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. https://doi.org/10.4230/LIPIcs.WABI.2024.17

Gärtner F, Müller L, Stadler PF. Superbubbles revisited. *Algorithms Mol Biol* 2018;**13**:16. https://doi.org/10.1186/s13015-018-0134-3

Hadlock F. An efficient algorithm for pattern detection and classification. In: *Proceedings of the 1st International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems—Volume 2, IEA/AIE'88, New York, NY, USA, June 1988*. New York City, NY, USA: Association for Computing Machinery. 1988, 645–53. https://dl.acm.org/doi/10.1145/55674.55676

Hart PE, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans Syst Sci Cyber* 1968;**4**:100–7. https://doi.org/10.1109/TSSC.1968.300136

Hickey G, Monlong J, Ebler J, *et al.* Human Pangenome Reference Consortium. Pangenome graph construction from genome alignments with Minigraph-Cactus. *Nat Biotechnol* 2024;**42**:663–73. https://doi.org/10.1038/s41587-023-01793-w

Holt JM *et al.* HiPhase: jointly phasing small, structural, and tandem repeat variants from HiFi sequencing, *Bioinformatics* 2024;**40**:btae042. https://doi.org/10.1093/bioinformatics/btae042

Ivanov P *et al.* AStarix: fast and optimal sequence-to-graph alignment. In: Schwartz R (ed.), *Research in Computational Molecular Biology, Lecture Notes in Computer Science*. Cham: Springer International Publishing, 2020, 104–19. https://doi.org/10.1007/978-3-030-45257-5_7

Ivanov P, Bichsel B, Vechev M. Fast and optimal sequence-to-graph alignment guided by seeds. In: Pe'er I (ed), *Research in Computational Molecular Biology, Lecture Notes in Computer Science*. Cham: Springer International Publishing, 2022, 306–25. https://doi.org/10.1007/978-3-031-04749-7_22

Jain C *et al.* Accelerating sequence alignment to graphs. In: *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rio de Janeiro, Brazil, May 2019*. New York City, NY, USA: IEEE, 2019, 451–61. https://doi.org/10.1109/IPDPS.2019.00055

Jain C, Zhang H, Gao Y *et al.* On the complexity of sequence-to-graph alignment. *J Comput Biol* 2020;**27**:640–54. https://doi.org/10.1089/cmb.2019.0066

Jamieson FB, Guthrie JL, Neemuchwala A *et al.* Profiling of rpoB mutations and MICs for rifampin and rifabutin in *Mycobacterium tuberculosis*. *J Clin Microbiol* 2014;**52**:2157–62. https://doi.org/10.1128/jcm.00691-14

Katoh K, Standley DM. MAFFT multiple sequence alignment software version 7: improvements in performance and usability. *Mol Biol Evol* 2013;**30**:772–80. https://doi.org/10.1093/molbev/mst010

Lee C. Generating consensus sequences from partial order multiple sequence alignment graphs. *Bioinformatics* 2003;**19**:999–1008. https://doi.org/10.1093/bioinformatics/btg109

Lee C, Grasso C, Sharlow MF. Multiple sequence alignment using partial order graphs. *Bioinformatics* 2002;**18**:452–64. https://doi.org/10.1093/bioinformatics/18.3452

Loman NJ, Quick J, Simpson JT. A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nat Methods* 2015;**12**:733–5. https://doi.org/10.1038/nmeth.3444

Marco-Sola S, Moure JC, Moreto M *et al.* Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics* 2021;**37**:456–63. https://doi.org/10.1093/bioinformatics/btaa777

Munir A, Kumar N, Ramalingam SB *et al.* Identification and characterization of genetic determinants of isoniazid and rifampicin resistance in *Mycobacterium tuberculosis* in Southern India. *Sci Rep* 2019;**9**:10283. https://doi.org/10.1038/s41598-019-46756-x

Myers EW. An O(ND) difference algorithm and its variations. *Algorithmica* 1986;**1**:251–66. https://doi.org/10.1007/BF01840446

Ondov BD, Treangen TJ, Melsted P *et al.* Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biol* 2016;**17**:132. https://doi.org/10.1186/s13059-016-0997-x

Onodera T, Sadakane K, Shibuya T. Detecting superbubbles in assembly graphs. In: *Algorithms in Bioinformatics: 13th International Workshop*. New York City, NY, USA: Springer, 2013, 338–48. https://doi.org/10.1007/978-3-642-40453-5_26

Rautiainen M, Marschall T. Aligning sequences to general graphs in O (V + mE) time. bioRxiv. https://www.biorxiv.org/content/10.1101/216127v1, November 2017, preprint: not peer reviewed.

Rautiainen M, Marschall T. GraphAligner: rapid and versatile sequence-to-graph alignment. *Genome Biol* 2020;**21**:253. https://doi.org/10.1186/s13059-020-02157-2

Sedlazeck FJ, Rescheneder P, Smolka M *et al.* Accurate detection of complex structural variations using single-molecule sequencing. *Nat Methods* 2018;**15**:461–8. https://doi.org/10.1038/s41592-018-0001-7

Suzuki H, Kasahara M. Acceleration of nucleotide semi-global alignment with adaptive banded dynamic programming. bioRxiv, http://biorxiv.org/lookup/doi/10.1101/130633, April 2017, preprint: not peer reviewed.

Ukkonen E. Finding approximate patterns in strings. *J Algorithms* 1985;**6**:132–7. https://doi.org/10.1016/0196-6774(85)90023-9

Vaser R, Sović I, Nagarajan N *et al.* Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Res* 2017;**27**:737–46. https://doi.org/10.1101/gr.214270.116.

Wang L, Jiang T. On the complexity of multiple sequence alignment. *J Comput Biol* 1994;**1**:337–48. https://doi.org/10.1089/cmb.1994.1.337

Wu M, Eisen JA. A simple, fast, and accurate method of phylogenomic inference. *Genome Biol* 2008;**9**:R151. https://doi.org/10.1186/gb-2008-9-10-r151

Zhang H *et al.* Fast sequence to graph alignment using the graph wavefront algorithm. June 2022. http://arxiv.org/abs/2206.13574, preprint: not peer reviewed

Zhou Y, Zeng J. Massively parallel a search on a GPU. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 29. Washington, DC, USA: AAAI Press, 2015. https://ojs.aaai.org/index.php/AAAI/article/view/9367

Bioinformatics, 2025, 41, 1–9
https://doi.org/10.1093/bioinformatics/btae757
Original Paper