Title

Optimal gap-affine alignment in O(s) space

Journal

Bioinformatics, 39(2)

ISSN

1367-4803

Authors

Marco-Sola, Santiago

Eizenga, Jordan M

Guarracino, Andrea

et al.

Publication Date

2023-02-03

DOI

10.1093/bioinformatics/btad074

Peer reviewed

Bioinformatics, 39(2), 2023, btad074
https://doi.org/10.1093/bioinformatics/btad074
Advance Access Publication Date: 7 February 2023
Original Paper

Sequence analysis

Optimal gap-affine alignment in *O*(*s*) space

Santiago Marco-Sola [1,2,*], Jordan M. Eizenga [3], Andrea Guarracino [4,5], Benedict Paten [3], Erik Garrison [5] and Miquel Moreto [1,6]

[1]Computer Sciences Department, Barcelona Supercomputing Center, Barcelona 08034, Spain, [2]Departament d'Arquitectura de Computadors i Sistemes Operatius, Universitat Autònoma de Barcelona, Barcelona 08193, Spain, [3]Genomics Institute, University of California Santa Cruz, Santa Cruz, CA 95064, USA, [4]Genomics Research Centre, Human Technopole, Milan 20157, Italy, [5]Department of Genetics, Genomics and Informatics, University of Tennessee Health Science Center, Memphis, TN 38163, USA and [6]Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, Barcelona 08034, Spain

*To whom correspondence should be addressed.
Associate Editor: Pier Luigi Martelli

Received on August 17, 2022; revised on January 2, 2023; editorial decision on January 30, 2023

Abstract

Motivation: Pairwise sequence alignment remains a fundamental problem in computational biology and bioinformatics. Recent advances in genomics and sequencing technologies demand faster and scalable algorithms that can cope with the ever-increasing sequence lengths. Classical pairwise alignment algorithms based on dynamic programming are strongly limited by quadratic requirements in time and memory. The recently proposed wavefront alignment algorithm (WFA) introduced an efficient algorithm to perform exact gap-affine alignment in $O(ns)$ time, where s is the optimal score and n is the sequence length. Notwithstanding these bounds, WFA's $O(s^2)$ memory requirements become computationally impractical for genome-scale alignments, leading to a need for further improvement.

Results: In this article, we present the bidirectional WFA algorithm, the first gap-affine algorithm capable of computing optimal alignments in $O(s)$ memory while retaining WFA's time complexity of $O(ns)$. As a result, this work improves the lowest known memory bound $O(n)$ to compute gap-affine alignments. In practice, our implementation never requires more than a few hundred MBs aligning noisy Oxford Nanopore Technologies reads up to 1 Mbp long while maintaining competitive execution times.

Availability and implementation: All code is publicly available at https://github.com/smarco/BiWFA-paper.

Contact: santiagomsola@gmail.com

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Pairwise sequence alignment provides a parsimonious transformation of one string into another. From this transformation, we can understand the relationship between pairs of sequences. Because similarities and differences between biosequences (DNA, RNA, protein) relate to variation in function and evolutionary history of living things, pairwise sequence alignment algorithms are a core part of many essential bioinformatics methods in read mapping (Li, 2013; Marco-Sola *et al.*, 2012), genome assembly (Koren *et al.*, 2017; Simpson *et al.*, 2009), variant calling (Garrison and Marth, 2012; McKenna *et al.*, 2010; Rodríguez-Martín *et al.*, 2017) and many others (Durbin *et al.*, 1998; Jones et al., 2004). Its importance has motivated the research and development of multiple solutions over the past 50 years.

Classical approaches to derive alignments involve the application of *dynamic programming* (DP) techniques. These methods require computing a matrix whose dimensions correspond to the lengths of the query q and target t sequences. Using DP recurrence relations, these methods compute the optimal alignment score for progressively longer prefixes of q and t, which correspond to the cells of the DP matrix. Thus, an optimal alignment can then be read out by tracing the recurrence back through the matrix.

Selecting a suitable alignment score function is essential to obtain biologically meaningful alignments, as it determines the characteristics of optimal alignments. In effect, the alignment score function encodes prior expectations about the probability of certain kinds of sequence differences. It has been observed that, in many contexts, insertions and deletions are non-uniformly distributed; they are infrequent but tend to be adjacent so that they form extended *gaps* with a long-tailed length distribution. This motivated the development of *gap-affine* models in which the penalty of starting a new gap is larger than that of extending a gap (Gotoh, 1982). Crucially, gap-affine penalties can be implemented efficiently using additional DP matrices.

Problematically, the efficiency of classical gap-affine DP-based methods is constrained by their quadratic requirements in time and

memory with respect to the lengths of the sequence pair. Consequently, multiple optimizations have been proposed over the years. Notable examples include bit-parallel techniques (Loving *et al.*, 2014), data-layout transformations to exploit SIMD instructions (Farrar, 2007; Rognes and Seeberg, 2000; Wozniak, 1997), difference encoding of the DP matrix (Suzuki and Kasahara, 2018), among other methods (Altschul *et al.*, 1990; Kiełbasa *et al.*, 2011; Xia *et al.*, 2021; Zhao *et al.*, 2013). Nonetheless, all these exact methods retain the quadratic requirements of the original DP algorithm and therefore struggle to scale when aligning long sequences.

In many cases, when two sequences are homologous, the majority of possible alignments are largely sub-optimal, having a substantially worse score than the optimal one. For this reason, heuristic methods are usually employed to find candidate alignment regions when the cost of exact algorithms becomes impractical. Most notable approaches use adaptive *band* methods (Suzuki and Kasahara, 2017) or pruning strategies [e.g. X-drop (Zhang *et al.*, 2000) and Z-drop (Li, 2018)] to avoid the computation of alignments extremely unlikely to be optimal. These heuristic methods have been implemented within many widely used tools (Altschul *et al.*, 1990; Li, 2018).

Recently, we proposed the wavefront alignment algorithm (WFA) (Marco-Sola *et al.*, 2021) to compute the exact alignment between two sequences using gap-affine penalties. WFA reformulates the alignment problem to compute the longest-possible alignments of increasing score until the optimal alignment is found. Notably, WFA takes advantage of homologous regions between sequences to accelerate alignment's computation. As a result, WFA computes optimal gap-affine alignments in $O(ns)$ time and $O(s^2)$ memory, where n is the sequence length and s the optimal alignment score. Being an exact algorithm, WFA provides the same guarantee for optimality as classical algorithms (Gotoh, 1982; Needleman and Wunsch, 1970; Smith and Waterman, 1981), but does away with the quadratic requirements in time.

WFA unlocked the path for optimal alignment methods capable of scaling to long sequences. Nevertheless, the $O(s^2)$ memory requirements quickly become the limiting factor when aligning sufficiently long or noisy sequences (Eizenga and Paten, 2022). As it happens, WFA's memory requirements can be impractical when aligning through large structural variations or highly divergent genome regions. Given that we use alignment to understand variation, these are some contexts in which optimal alignment could be most useful, but its memory requirements make it prohibitive.

To address this problem, this article presents the first gap-affine alignment algorithm to compute the optimal alignment in $O(ns)$ time and $O(s)$ memory (excluding the storage of the input sequences). Our method, the bidirectional WFA algorithm (BiWFA), computes the WFA alignment of two sequences in the forward and reverse direction until they meet. Using two wavefronts of $O(s)$ memory, we demonstrate how to find the optimal breakpoint of the alignment at score $\sim s/2$ and proceed recursively to solve the complete alignment in $O(ns)$ time. To our knowledge, this work improves the lowest known memory bound to compute gap-affine alignments $O(n)$ (Myers and Miller, 1988) to $O(s)$, while retaining the time complexity of the original WFA $O(ns)$. Furthermore, our experimental results demonstrate that the BiWFA delivers comparable, or even better, performance than the original WFA, outperforming other state-of-the-art tools while using a minimal amount of memory.

The rest of the article is structured as follows. Section 2 presents the definitions, algorithms and formal proofs supporting BiWFA. Section 3 shows the experimental evaluation of our method, comparing it against other state-of-the-art tools and libraries. Lastly, Section 4 presents a discussion on the BiWFA method and summarizes the contributions and impact of this work.

2 Materials and methods

2.1 Wavefront alignment algorithm

Let the query $q = q_0 q_1 \dots q_{n-1}$ and the text $t = t_0 t_1 \dots t_{m-1}$ be strings of length n and m, respectively. Likewise, let $v[i,j] = v_i v_{i+1} \dots v_j$ denote a substring of any string v from the ith to the jth character. We will use (x, o, e) to denote the gap-affine penalties. A mismatch costs x, and a gap of length l costs $o + l \cdot e$. We assume that $x > 0$ and $e > 0$, and further that all of the score parameters are constants.

Basically, WFA computes partial optimal alignments of increasing score until an alignment with score s reaches coordinate (n, m) of the DP matrix. In this way, the algorithm determines that s is the minimal alignment score. Moreover, it can derive the optimal alignment by tracing back the partial alignments that led to score s at (n, m).

Let $\mathcal{M}_{s,k}$, $\mathcal{X}_{s,k}$, $\mathcal{I}_{s,k}$ and $\mathcal{D}_{s,k}$ denote the offset within diagonal k in the DP-matrix to the farthest-reaching (f.r.) cell that has score s and ends with a match, mismatch, insertion or deletion, respectively. In general, we denote by *wavefront* the tuple of offsets for a given score $\mathcal{W}_s = (\mathcal{M}_s, \mathcal{X}_s, \mathcal{I}_s, \mathcal{D}_s)$. We refer to the four elements in this tuple as its *components*, and we associate a corresponding sentinel value to specify each component: $c \in \{M, X, I, D\}$.

In Marco-Sola *et al.* (2021), we proved that the f.r. points of $\mathcal{W}_s$ can be computed using previous wavefronts $\mathcal{W}_{s-o-e}$, $\mathcal{W}_{s-e}$ and $\mathcal{W}_{s-x}$, using Equation 1 where $LCP(v, w)$ is the length of longest common prefix between substrings v and w. The base case for this recursion is given by $\mathcal{X}_{0,0} = 0$.

$$\begin{aligned} \mathcal{I}_{s,k} &= \max\{\mathcal{M}_{s-o-e,k-1} + 1, \mathcal{I}_{s-e,k-1} + 1\} \\ \mathcal{D}_{s,k} &= \max\{\mathcal{M}_{s-o-e,k+1}, \mathcal{D}_{s-e,k+1}\} \\ \mathcal{X}_{s,k} &= \max\{\mathcal{M}_{s-x,k} + 1, \mathcal{I}_{s,k}, \mathcal{D}_{s,k}\} \\ \mathcal{M}_{s,k} &= \mathcal{X}_{s,k} + LCP(q[\mathcal{X}_{s,k} - k, n-1], t[\mathcal{X}_{s,k}, m-1]), \end{aligned} \tag{1}$$

Equation 1 shows that the computation of a given wavefront depends only on the previous $p = \max\{x, o + e\}$ wavefronts. We refer to p as the wavefront *scope* or, in other words, the maximum score increase between partial alignments. Moreover, note that $\mathcal{X}_{s,k}$ does not need to be explicitly stored as its values can be inferred using $\mathcal{M}_{s,k}$, $\mathcal{I}_{s,k}$ and $\mathcal{D}_{s,k}$.

In the worst case, WFA requires computing s wavefronts of increasing length, totalling $\sum_{i=0}^{s}(1 + 2i) = O(s^2)$ cells. Moreover, the *LCP* must be computed once for each cell. However, within a diagonal, the total number of offset increments cannot exceed the length of the sequences. Hence, WFA requires $O(ns)$ time and $O(s^2)$ memory in the worst case (Marco-Sola *et al.*, 2021). Since $s \leq pn$, the $O(ns)$ factor of the execution time, due to the *LCP*, dominates over the $O(s^2)$ factor in the worst case. However, in practice, the time is often closer to $O(s^2 + n)$. This is because spurious matches between high-entropy sequences are short in expectation. Accordingly, the *LCP* computations often finish after performing only a few character comparisons, except along the optimal alignment in which $O(n)$ comparisons are required.

2.2 Bidirectional wavefront alignment algorithm

The core idea of the BiWFA algorithm is to perform WFA simultaneously in both directions on the strings: from start to end (i.e. forward) and from end to start (i.e. reverse). Each direction will only retain p wavefronts in memory. This is insufficient to perform a full traceback. However, when they 'meet' in the middle, we can infer a breakpoint in the alignment that divides the optimal score roughly in half. Then, we can apply the same procedure on the two sides of the breakpoint recursively. We will show that this results in only a constant-factor slowdown. This technique was previously employed to a similar end with the Myers $O(ND)$ difference algorithm (Myers, 1986).

Figure 1 presents a graphical example of BiWFA computing a breakpoint in the optimal alignment between two sequences. The figure shows the DP cells computed by the forward and reverse wavefronts. Alignments in both directions progress until they overlap on cell $(4, 4)$ with score $8 + 8 = 16$ corresponding to the optimal alignment ($s_{opt} = 16$).

First, let us define the WFA equations for the forward and reverse alignment directions. The recursions for the forward direction are equivalent to those of the standard WFA presented above (Equation 1). However, to highlight the distinction, we will denote them $\overrightarrow{\mathcal{W}}_s = (\overrightarrow{\mathcal{I}}_s, \overrightarrow{\mathcal{D}}_s, \overrightarrow{\mathcal{X}}_s, \overrightarrow{\mathcal{M}}_s)$. The recursions for the reverse

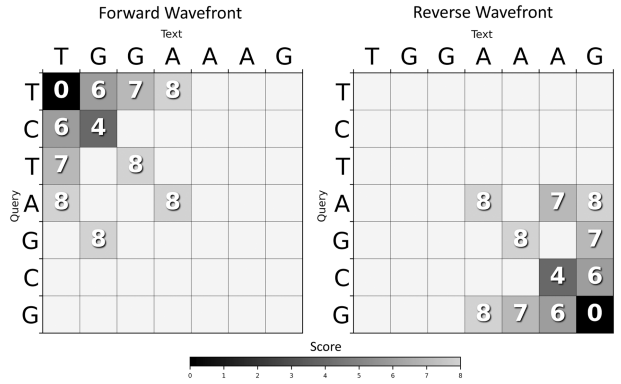

Fig. 1. Example of BiWFA aligning $q =$ "*TCTAGCG*" against $t =$ "*TGGAAAG*" under the penalties $(x = 4, o = 5, e = 1)$

direction are very similar (Equation 2), using $\overleftarrow{\mathcal{X}}_{0,m-n} = m$ as the base case and $LCS(v, w)$ to denote the length of the longest common suffix of v and w. Note that the same argument used in Marco-Sola *et al.* (2021) applies to the reverse recursions to prove that they are f.r. in the reverse direction.

$$\begin{aligned}
\overleftarrow{\mathcal{I}}_{s,k} &= \min\{\overleftarrow{\mathcal{M}}_{s-o-e,k+1} - 1, \overleftarrow{\mathcal{I}}_{s-e,k+1} - 1\} \\
\overleftarrow{\mathcal{D}}_{s,k} &= \min\{\overleftarrow{\mathcal{M}}_{s-o-e,k-1}, \overleftarrow{\mathcal{D}}_{s-e,k-1}\} \\
\overleftarrow{\mathcal{X}}_{s,k} &= \min\{\overleftarrow{\mathcal{M}}_{s-x,k} - 1, \overleftarrow{\mathcal{I}}_{s,k}, \overleftarrow{\mathcal{D}}_{s,k}\} \\
\overleftarrow{\mathcal{M}}_{s,k} &= \overleftarrow{\mathcal{X}}_{s,k} - LCS(q[0, \overleftarrow{\mathcal{X}}_{s,k} - k - 1], t[0, \overleftarrow{\mathcal{X}}_{s,k} - 1])
\end{aligned} \tag{2}$$

Algorithm 1 presents the BiWFA algorithm to compute a breakpoint in the optimal alignment at $\sim s/2$. Using forward and reverse wavefronts, the algorithm proceeds by alternatingly computing forward and reverse alignments (i.e. $\overrightarrow{\mathcal{W}}_1, \overleftarrow{\mathcal{W}}_1, \overrightarrow{\mathcal{W}}_2, \overleftarrow{\mathcal{W}}_2, \ldots$). To this end, BiWFA relies on the operators *WF_NEXT()* and *WF_EXTEND()* from the standard WFA [see Marco-Sola *et al.* (2021)] to compute successive wavefronts using Equaions 1 and 2. The process is halted after their offsets overlap to compute the position of a breakpoint in the optimal alignment. This algorithm iterates until it is guaranteed that the optimal breakpoint has been found. However, there are some technical details involving the detection of overlaps and the computation of the optimal breakpoint, which we cover in Sections 2.3 and 2.4.

2.3 Finding a score-balanced breakpoint in the optimal alignment

The first technical detail involved in finding an alignment breakpoint between the two directions is that it is often not possible to split an alignment into an equally scoring prefix and suffix. In general, two prefixes of the optimal alignment that differ by one character can have scores that differ by as much as p. Accordingly, we will demand a weaker notion of balance. If s_f and s_r are the forward and reverse scores, respectively, we will aim to have $|s_f - s_r| \leq p$.

The second technical detail is that the optimal score is not always the sum of the two scores. This occurs because the forward iteration incurs the gap open penalty o at the beginning of gaps, but the reverse incurs it at the end of gaps (or rather, at the beginning in the reverse direction). Thus, if the two directions meet in a gap, then we have $s_{opt} = s_f + s_r - o$ rather than $s_{opt} = s_f + s_r$, where s_{opt} is the optimal alignment score.

The final technical detail is that offsets of the two directions may not precisely meet. WFA proceeds by greedily taking matches in both directions. This makes it possible for the two directions to shoot past each other without actually meeting. It turns out that it is sufficient to detect that such an overshoot has occurred, as will be shown in Section 2.4.

In Algorithm 2, we reconcile these three difficulties. Without loss of generality, we assume that a forward wavefront $\overrightarrow{\mathcal{W}}_{s_f}$ has been computed (Algorithm 1), and we want to detect overlaps against the previously computed reverse wavefronts $\overleftarrow{\mathcal{W}}_{s_r \ldots s_r - p}$. First, if $\overrightarrow{\mathcal{W}}_{s_f}$ belongs to a score-balanced breakpoint (with $|s_f - s_r| \leq p$), it is sufficient to check for overlaps against $\overleftarrow{\mathcal{W}}_{s_r}$ and the previous $p - 1$ reverse wavefronts. Second, for every diagonal k in wavefront $\overrightarrow{\mathcal{W}}_{s_f}$, Algorithm 2 checks of overlaps in all wavefront components. This way, the algorithm keeps track of the overlap with the minimum score detected so far. Last, note that overlaps on $\mathcal{I}$ and $\mathcal{D}$ components account twice for the gap-open score o. Hence, the score from overlaps at indel components has to be decreased by o.

Algorithm 1: Compute optimal alignment breakpoint using BiWFA.

Input: q, t strings, c_0, c_f begin and end components
Output: s_b score, k_b diagonal, l_b offset, and c_b component of a breakpoint in the alignment $\mathcal{A}$ at $\sim s/2$ ($\mathcal{A}$ being the optimal alignment between q and t under $\{x, o, e\}$ penalties, starting at component c_0 and ending at component c_f).

Function BIWFA_BREAKPOINT(q, t, c_0, c_f) **begin**
  `// Initialise components` c_0, c_f `from` $\overrightarrow{\mathcal{M}}_0, \overleftarrow{\mathcal{M}}_0$
  `WF_INIT` $(\overrightarrow{\mathcal{M}}_0, c_0, 0)$; `WF_EXTEND` $(\overrightarrow{\mathcal{M}}_0, q, t)$
  `WF_INIT` $(\overleftarrow{\mathcal{M}}_0, c_f, 0)$; `WF_EXTEND` $(\overleftarrow{\mathcal{M}}_0, q, t)$
  `// Best breakpoint so far`
  $(s_b, k_b, l_b, c_b) \leftarrow$ `WF_OVERLAP` $(\overrightarrow{\mathcal{W}}_0, \overleftarrow{\mathcal{W}}_0)$
  `// Compute forward and reverse wavefronts`
  $(s_f, s_r) \leftarrow (0, 0)$
  while $s_f + s_r - o - p + 1 < s_b$ **do**
    `// Compute` $\overrightarrow{\mathcal{W}}_{s_f+1}$ `and find overlaps`
    $s_f \leftarrow s_f + 1$
    $\overrightarrow{\mathcal{W}}_{s_f} \leftarrow$ `WF_NEXT` $(\overrightarrow{\mathcal{W}}, s_f, q, t)$
    `WF_EXTEND` $(\overrightarrow{\mathcal{M}}_{s_f}, q, t)$
    if `ANTIDIAG` $(\overrightarrow{\mathcal{M}}_{s_f}) \geq$ `ANTIDIAG` $(\overleftarrow{\mathcal{M}}_{s_r})$ **then**
      $(s, k, l, c) \leftarrow$ `WF_OVERLAP` $(\overrightarrow{\mathcal{W}}_{s_f}, \overleftarrow{\mathcal{W}}_{s_r \ldots s_r - p})$
      if $s < s_b$ **then**
        $(s_b, k_b, l_b, c_b) \leftarrow (s, k, l, c)$
    `// Best breakpoint found?`
    if $s_f + s_r - o - p + 1 \geq s_b$ **then break ;**
    `// Compute` $\overleftarrow{\mathcal{W}}_{s_r+1}$ `and find overlaps`
    $s_r \leftarrow s_r + 1$
    $\overleftarrow{\mathcal{W}}_{s_r} \leftarrow$ `WF_NEXT` $(\overleftarrow{\mathcal{W}}, s_r, q, t)$
    `WF_EXTEND` $(\overleftarrow{\mathcal{M}}_{s_r}, q, t)$
    if `ANTIDIAG` $(\overrightarrow{\mathcal{M}}_{s_f}) \geq$ `ANTIDIAG` $(\overleftarrow{\mathcal{M}}_{s_r})$ **then**
      $(s, k, l, c) \leftarrow$ `WF_OVERLAP` $(\overleftarrow{\mathcal{W}}_{s_r}, \overrightarrow{\mathcal{W}}_{s_f \ldots s_f - p})$
      if $s < s_b$ **then**
        $(s_b, k_b, l_b, c_b) \leftarrow (s, k, l, c)$
  return (s_b, k_b, l_b, c_b)
end

In practice, Algorithm 1 can avoid most calls to *WF_OVERLAP()*. An efficient implementation can keep track of the farthest reached antidiagonal by each wavefront. If the most advanced antidiagonal reached by the forward and reverse wavefronts do not overlap (*ANTIDIAG()* on Algorithm 1), it follows that no offsets from any diagonal can overlap, rendering the call to *WF_OVERLAP()* unnecessary.

2.4 Correctness of the breakpoint detection

The correctness of the Algorithm 1 stems from the following lemma.

Lemma 2.1. *The optimal alignment score $s_{opt} \leq s$ if and only if there exist s_f, s_r and k such that $|s_f - s_r| \leq p$ and at least one of the following is true:*

1. $s_f + s_r = s$ *and* $\overrightarrow{\mathcal{M}}_{k,s_f} \geq \overleftarrow{\mathcal{M}}_{k,s_r}$
2. $s_f + s_r = s + o$ *and* $\overrightarrow{\mathcal{I}}_{k,s_f} \geq \overleftarrow{\mathcal{I}}_{k,s_r}$
3. $s_f + s_r = s + o$ *and* $\overrightarrow{\mathcal{D}}_{k,s_f} \geq \overleftarrow{\mathcal{D}}_{k,s_r}$

and further, $\overleftarrow{\mathcal{M}}_{k,s_r}$ *(resp.* $\overleftarrow{\mathcal{I}}_{k,s_r}$, $\overleftarrow{\mathcal{D}}_{k,s_r}$*) is included in the traceback of an alignment with score at most s if the first (resp. second, third) condition is true.*

Proof.
See Supplementary Material.

Algorithm 2: Detect overlaps and compute optimal breakpoint between forward and reverse wavefronts.

Input: $\overrightarrow{\mathcal{W}}_{s_f}$ last computed wavefront, $\overleftarrow{\mathcal{W}}_{s_r \ldots s_r - p}$ last p wavefronts in the opposite direction
Output: Breakpoint's s_b score, k_b diagonal, l_b offset, and c_b component of the overlap with least score

Function WF_OVERLAP($\overrightarrow{\mathcal{W}}_{s_f}, \overleftarrow{\mathcal{W}}_{s_r \ldots s_r - p}$) **begin**
  $(s_b, k_b, l_b, c_b) \leftarrow (\infty, none, none, none)$
  for *Diagonals* k **included in** $\overrightarrow{\mathcal{W}}_{s_f}$ **do**
    for $s \leftarrow s_r$ **to** $s_r - p$ **do**
      if $\overrightarrow{\mathcal{M}}_{k,s_f} \geq \overleftarrow{\mathcal{M}}_{k,s_r} \wedge s_f + s < s_b$ **then**
        $(s_b, k_b, l_b, c_b) \leftarrow (s_f + s, k, \overrightarrow{\mathcal{M}}_{k,s_f}, M)$
      if $\overrightarrow{\mathcal{I}}_{k,s_f} \geq \overleftarrow{\mathcal{I}}_{k,s_r} \wedge s_f + s - o < s_b$ **then**
        $(s_b, k_b, l_b, c_b) \leftarrow (s_f + s - o, k, \overrightarrow{\mathcal{I}}_{k,s_f}, I)$
      if $\overrightarrow{\mathcal{D}}_{k,s_f} \geq \overleftarrow{\mathcal{D}}_{k,s_r} \wedge s_f + s - o < s_b$ **then**
        $(s_b, k_b, l_b, c_b) \leftarrow (s_f + s - o, k, \overrightarrow{\mathcal{D}}_{k,s_f}, D)$
  return (s_b, k_b, l_b, c_b)
end

This lemma implies that the minimum value s for which the 'only if' condition holds is the optimal score. Moreover, if the first of the three conditions is found to hold for some values s_f and s_r, then $s_{opt} \leq s_f + s_r + o$. Therefore, Algorithm 1 is guaranteed to find part of a minimum-scoring alignment based on the following features:

- Algorithm 2 checks a window of p score values on each iteration.
- Algorithm 1 iterates through alternatingly increasing values of s_f and s_r, detecting breakpoints with scores of at least $s_f + s_r - o - p + 1$ in each iteration.
- After finding some s_f and s_r that satisfy the overlap condition, Algorithm 1 continues for additional iterations until it is no longer possible to find a lower score.

2.5 Combining breakpoints into an alignment

Algorithm 3 shows how to use BiWFA to recursively split alignments into smaller subproblems until the remaining alignment can be trivially solved.

Note that a breakpoint computed by BiWFA can be found on the I or D components. Thus, those alignments that connect with this breakpoint have to start or end at the given component. This way, Algorithm 3 considers the starting and ending component of each alignment, and forces the underlying WFAs to use different initial conditions depending on the alignment starting at the M ($\mathcal{X}_{0,0} = 0$), I ($\mathcal{I}_{0,0} = 0$) or D component ($\mathcal{D}_{0,0} = 0$). A similar argument applies to the ending conditions of each alignment ending at the M ($\mathcal{M}_{s,m-n} = m$), I ($\mathcal{I}_{s,m-n} = m$) or D component ($\mathcal{D}_{s,m-n} = m$).

2.6 BiWFA uses $O(s)$ space and $O((m+n)s)$ time

The memory complexity of Algorithm 1 is relatively simple to characterize. The range of diagonal values k increases by at most two every time s is incremented, and each forward and reverse search only needs to store the last p wavefronts. Thus, the memory use is proportional to the optimal alignment score, $O(s)$, excluding the storage of the input sequences. Also, note that the output alignment only requires storing the position (i, j) for the mismatches, insertions and deletions (matches can be inferred from the gaps). Concerning Algorithm 3, data structures are discarded before entering a recursive call. Therefore, the maximum memory use occurs in the outermost call, in which s is the optimal score of the full alignment.

Algorithm 3: BiWFA recursive computation of the optimal alignment in $O(s)$ space

Input: q, t strings, c_0, c_f begin and end components
Output: $\mathcal{A}$ optimal gap-affine alignment between q and t

Function BIWFA_ALIGN(q, t, c_0, c_f) **begin**
  `// Base cases`
  if $n = 0$ **then return** $D \times m$;
  if $m = 0$ **then return** $I \times n$;
  `// Find optimal breakpoint at` $\sim s/2$
  $(s, j, k, c) \leftarrow$`BIWFA_BREAKPOINT(`q, t, c_0, c_f`)`
  `// Align the first` $\mathcal{A}_0$ `and second` $\mathcal{A}_1$ `half`
  $i \leftarrow j - k$; `// Breakpoint at (i,j)`
  if $c \neq I$ **then** $i' = i - 1$ **else** $i' = i$
  if $c \neq D$ **then** $j' = j - 1$ **else** $j' = j$
  $\mathcal{A}_0 \leftarrow$`BIWFA_ALIGN(`$q_{0 \ldots i'}, t_{0 \ldots j'}, c_0, c$`)`
  $\mathcal{A}_1 \leftarrow$`BIWFA_ALIGN(`$q_{i+1 \ldots n-1}, t_{j+1 \ldots m-1}, c, c_f$`)`
  return $\mathcal{A}_0 + c + \mathcal{A}_1$
end

The time complexity is more complicated to analyze. Our proof follows similar arguments as those from Myers (1986).

Theorem 2.2. *BiWFA's time complexity is* $O((m+n)s)$*, being n and m the sequences' length and s the optimal alignment score.*

Proof.
Let $\ell = m + n$, and let $T(\ell, s)$ be BiWFA's execution time with score s. A call to BiWFA can result in two recursive calls. Let ℓ_f and ℓ_r be the combined length of the sequences in the two calls, and similarly let s_f and s_r be the two alignment scores. Following Lemma 2.1, we know that these variables obey the following inequalities:

- $\ell_f + \ell_r \leq \ell$
- $s_f + s_r \leq s$
- $|s_f - s_r| \leq p$

Because each direction of WFA executes in $O(s\ell)$ time (Marco-Sola *et al.*, 2021), we can choose a constant c_1 large enough that the following inequality holds for all $s > 3p$:

$$T(\ell, s) \leq c_1 s\ell + T(\ell_f, s_f) + T(\ell_r, s_r) \tag{3}$$

We can also choose a constant c_2 large enough that for all $s \leq 3p$

$$T(\ell, s) \leq c_2 \ell \tag{4}$$

This follows because the recursion depth depends only on s, which we have given an upper bound. Therefore, this term includes a bounded number of calls that all have linear dependence on ℓ.

Next, we argue that $T(\ell,s) \leq 3c_1 s\ell + c_2\ell$ by induction on s. The base cases for $s = 0,1,\ldots,3p$ follow trivially from the latter of the previous inequalities. Assume then that $s > 3p$ and the induction hypothesis holds for $0,1,\ldots,s-1$. Note that we then have $s_f, s_r \leq 2s/3$, else either $|s_f - s_r| > p$ or $s_f + s_r > s$. Thus,

$$\begin{aligned} T(\ell,s) &\leq c_1 s\ell + (3c_1(2s/3)\ell_f + c_2\ell_f) + (3c_1(2s/3)\ell_r + c_2\ell_r) \\ &\leq 3c_1 s\ell + c_2\ell \end{aligned} \quad (5)$$

This proves the claim.

3 Results

We implement the BiWFA algorithm described in this work in C. The code and the scripts required to reproduce the experimental results presented in this section are publicly available and can be found at https://github.com/smarco/BiWFA-paper. Moreover, the code has been integrated into the WFA2-lib alignment library (as ultralow memory mode) at https://github.com/smarco/WFA2-lib.

3.1 Experimental setup

We evaluate the performance of our BiWFA implementation compared to the state-of-the-art and other high-performance sequence alignment libraries. We select the original WFA (Marco-Sola *et al.*, 2021) (wfa-high) and its new low-memory modes (wfa-med and wfa-low) implemented in WFA2-lib (https://github.com/smarco/WFA2-lib). Also, we select the efficient wfalm (Eizenga and Paten, 2022) (wfalm) and its low-memory modes (wfalm-low and wfalm-rec). Moreover, we include the highly optimized KSW2-Z2 (ksw2_extz2_sse), from the KSW2 library (Li, 2018; Suzuki and Kasahara, 2018), as the best representative of DP-based methods due to its exceptional performance and widely usage within bioinformatics tools. In addition, we include the Edlib (Šošić and Šikić, 2017) and BitPal (Loving *et al.*, 2014) libraries, which implement bit-parallel alignment strategies for edit and non-unitary penalties (i.e. gap-linear), respectively. Although they solve a considerably easier problem (i.e. Edlib is restricted to edit-alignments and BitPal only computes the alignment score), and thus are not directly comparable, we included them in the evaluation to provide a performance upper bound.

We considered including other popular methods like those implemented in the Parasail (Daily *et al.*, 2015; Daily, 2016; Farrar, 2007; Wozniak, 1997), SeqAn (Rahn *et al.*, 2018) and Gaba (Suzuki and Kasahara, 2018) libraries. However, these libraries were not designed to align long and noisy sequences, and failed to complete the executions. Therefore these methods were discarded from the evaluation.

All the presented methods have been configured to generate global alignments. These algorithms are grouped in two categories: 'Gap-affine Exact' for exact algorithms that use gap-affine penalties (i.e. BiWFA, WFA and its low-memory modes, wfalm and its low-memory modes and KSW2-Z2), and 'Others' for methods that use simpler penalty models or can only compute the alignment score (i.e. Edlib and BitPal).

For the evaluation, we use simulated and real datasets. For the simulated datasets, we simulate several datasets of various sequence lengths (i.e. 100K, 500K, 1M and 2M bases) and different error rate (i.e. $e = 10\%$ and 20%) randomly generated. Regarding the evaluation with real datasets, we use a first set of sequences generated by the Human Pangenome Reference Consortium (Miga and Wang, 2021), consisting of long reads sequenced using Oxford Nanopore Technologies (ONT), PromethION platform, with an average error rate of 5–10%. The sequences are derived from the human cell line HG002, subset to chromosome 12 and restricted to those at least 10 kbp long, for a total number of 1312 sequence pairs of average length equal to 172 kbp (maximum ~306 kbp). In addition, we use a second dataset comprising ONT MinION reads from Bowden *et al.* (2019), with an average error rate of 5% and restricted to those at least 500 kbp long, for a total number of 48 sequence pairs of average length equal to 630 kbp (maximum ~1 Mbp).

All the executions are performed single-thread on a node running CentOS Linux (release 8.1.1911) equipped with an AMD EPYC 7742 CPU and 1 TB of DRAM (distributed in 16 dimms × 64 GiB @3200 MHz).

3.2 Evaluation on simulated data

Table 1 shows the performance results (i.e. execution time and memory) for the different methods using simulated datasets. Overall, the results show that BiWFA is faster and uses less memory than all other methods in the 'Gap-affine Exact' category. In particular, BiWFA requires $32-1000\times$ less memory than KSW2-Z2, while being $1.4-4.7\times$ faster. Compared to original WFA-based methods (i.e. WFA-high and wfalm), BiWFA uses $9-9620\times$ less memory, being up to $4.4\times$ faster. Similarly, BiWFA outperforms the other memory-efficient WFA-based methods (i.e. WFA-med, WFA-low, wfalm-low and wfalm-rec), reducing memory requirements down to $438\times$ while being $2.7-26.5\times$ faster. More importantly, most of the pairwise alignment methods evaluated fail to scale megabases-long sequences, requiring more memory than available in the node (i.e. 1TB). As opposed, BiWFA only requires a few

Table 1. Time and memory performance of pairwise alignment implementations on simulated data

	Time (s)								Memory (MBs)							
	10 Kbp		100 Kbp		1 Mbp		2 Mbp		10 Kbp		100 Kbp		1 Mbp		2 Mbp	
	10%	20%	10%	20%	10%	20%	10%	20%	10%	20%	10%	20%	10%	20%	10%	20%
edlib	0.4	0.6	2.5	4.5	17.9	35.3	35.4	69.1	4	4	5	5	13	13	22	23
bitpal	1.3	1.2	12.3	12.3	123.8	123.7	248.0	247.1	4	4	4	6	10	10	15	13
ksw2-extz2	9.8	9.9	96.7	97.5	n/a	n/a	n/a	n/a	193	196	19 081	19 083	n/a	n/a	n/a	n/a
WFA-high	**2.0**	**5.7**	28.5	84.1	312.9	n/a	n/a	n/a	128	313	8981	26 667	932 199	n/a	n/a	n/a
WFA-med	6.6	20.2	89.5	272.8	1922.1	3690.1	n/a	n/a	35	81	830	1620	42 464	24 874	n/a	n/a
WFA-low	8.0	24.1	101.4	301.1	4394.9	4857.0	7710.2	9813.1	25	60	554	884	25 321	12 539	52 551	26 067
wfalm	6.4	19.2	90.2	268.7	841.0	n/a	n/a	n/a	54	148	8968	26 575	898 770	n/a	n/a	n/a
wfalm-low	10.1	30.4	164.1	494.8	1525.1	4418.7	2990.1	8779.4	10	16	443	823	10 435	30 817	36 299	69 312
wfalm-rec	22.3	70.7	447.5	1402.6	5792.7	17 752.9	11 979.6	37 747.7	6	7	43	73	497	904	1064	1787
BiWFA	2.4	6.9	**20.8**	**61.0**	**218.3**	**680.4**	**466.9**	**1429.0**	**6**	**5**	**19**	**27**	**97**	**180**	**202**	**267**
BiWFA.score	1.1	3.1	10.2	30.2	112.2	355.8	245.4	750.2	3	4	16	23	97	186	204	256

Note: Execution time (in seconds) and memory (in MBs) required per 1M bases aligned, using different pairwise alignment implementations on simulated datasets. Executions that failed appear as 'n/a'. Best performing implementation in the 'Gap-affine Exact' category is marked in bold. Although Edlib and BitPal are not directly comparable to the other implementations, we included them in the comparison as a reference. Similarly, we include executions of BiWFA limited to compute the alignment score as 'BiWFA.score'.

hundred MBs of memory. Note that, computing the full alignment (BiWFA) requires a similar amount of memory as computing only the alignment score (BiWFA.score). Nonetheless, computing the alignment score is ∼2× faster than computing the full alignment.

For completeness, we present a comprehensive experimental evaluation on a wider range of sequence lengths (i.e. 100 bp, 1 Kbp, 10 Kbp, 100 Kbp, 1 Mbp and 2 Mbp) and error rates (0.1%, 1%, 5%, 10%, 20% and 40%) on Supplementary Material (Supplementary Tables S1 and S2). For short sequences (i.e. ≤1 Kbp), the results show that BiWFA delivers similar performance as the original WFA (1.25 – 2× slower) while reducing the memory requirements up to 60×. Our experiments indicate that BiWFA starts outperforming the original WFA when aligning sequences longer than ∼30 Kbp. Nevertheless, the exact performance breakpoint can vary depending on the error rate, implementation and processor specifics. Similarly, for smaller error rates (i.e. ≤1%), WFA-based methods largely outperform other approaches, being 2 – 3 orders of magnitude faster than other methods like KSW2-Z2, bitpal and edlib. In addition, for low error rates, memory-efficient WFA-based methods require a minimal amount of memory.

3.3 Evaluation on real data

Figure 2 shows the performance results obtained for all the evaluated algorithms in terms of execution time and consumed memory. BiWFA uses many times less memory than other methods. In particular, when aligning ultra-long ONT sequences (Fig. 2B), BiWFA requires between 68 – 93× less memory compared to wfalm and WFA low-memory modes. Furthermore, BiWFA uses 3.5× less memory compared to the efficient recursive mode from wfalm (most memory-efficient gap-affine algorithm to date).

At the same time, BiWFA proves to be one of the fastest implementations aligning long sequences. Using ultra-long sequences, our method is 25.7× faster than wfalm's recursive mode. Moreover, BiWFA's execution times are similar to those of BitPal (sometimes even faster, 1.1 – 1.28× faster on average) computing exact alignments (not just the score) under the gap-affine model.

For completeness, Supplementary Figure S1 shows experimental results limited to aligning sequences up to 10 Kbps. In this scenario, BiWFA demonstrates to be one of the fastest implementations, requiring less than 10 MB to execute.

4 Discussion

As long sequencing technologies improve and high-quality sequence assembly decreases in cost, we anticipate that the importance of pairwise alignment algorithm will continue to increase. To keep up with upcoming improvements in sequencing and genomics, pairwise alignment algorithms need to face crucial challenges in reducing execution time and memory consumption. In this work, we have presented the BiWFA, a gap-affine pairwise alignment algorithm that requires $O(ns)$ time and $O(s)$ space, being the first algorithm to improve the long standing space lower bound of $O(n)$. The BiWFA answers the pressing need for sequence alignment methods capable to scaling to genome-scale alignments and full pangenomes.

Most notably, BiWFA execution times are very similar, or even better, than those of the original WFA (despite BiWFA requiring 2954× and 607× less memory when aligning ultra-long MinION and PromethION sequences, respectively). This result can be better understood considering the memory inefficiencies that the original WFA experiences when using a large memory footprint. As the sequence's length and error increases, the original WFA uses a substantially larger memory footprint, putting a significant pressure on the memory hierarchy of the processor. Due to the pervasive memory inefficiencies of modern processors executing memory intensive applications, the original WFA's performance is severely deteriorated when aligning long sequence datasets (like those from Nanopore presented in the evaluation). In contrast, BiWFA relieves this memory pressure using a minimal memory footprint. As a

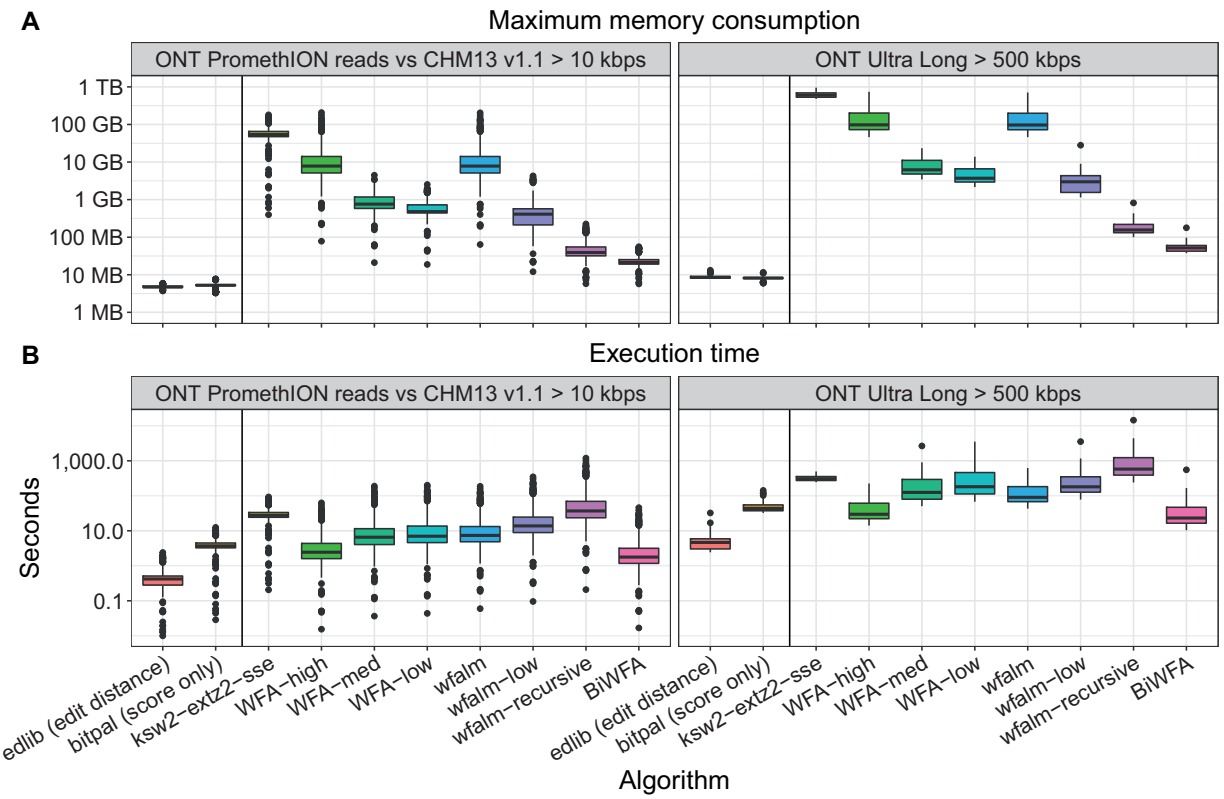

Fig. 2. Experimental results from the execution of BiWFA and other state-of-the-art implementations aligning long sequences. Figure shows (**A**) memory consumption and (**B**) execution time per sequence aligned. A vertical line on each panel separates algorithms that use simpler penalty models or can only compute the alignment score (i.e. edlib and bitpal) from those that compute the full gap-affine alignment

result, BiWFA is able to balance out the additional work induced by BiWFA's recursion, delivering a performance on-par with the original WFA.

We have presented the BiWFA using gap-affine penalties. Nevertheless, these very same ideas can be translated directly into other distances like edit, linear gap or piecewise gap-affine. Moreover, it can be easily extended to semi-global alignment (a.k.a. ends-free, glocal, extension or overlapped alignment) by modifying the initial conditions and termination criterion. At the same time, the BiWFA retains the strengths of the original WFA: no restrictions on the sequences' alphabet, preprocessing steps, nor prior estimation of the alignment error.

Due to the simplicity of the WFA's computational pattern, BiWFA's core functions can be easily vectorized to fully exploit the capabilities of modern SIMD multicore processors. Our implementation, relies on the automatic vectorization capabilities of modern compilers. As a result, the BiWFA implementation can exploit the SIMD capabilities of any processor supported by modern compilers, without rewriting any part of the source code.

Genomics and bioinformatics methods will continue to rely on sequence alignment as a core and critical component. BiWFA paves the way for the development of faster and more accurate tools that can scale with longer and noisy sequences using a minimal amount of memory. In this way, we expect BiWFA to enable efficient sequence alignment at genome-scale in years to come.

Acknowledgements

The authors thank Ragnar Groot Koerkamp and the anonymous reviewers for making useful suggestions and contributing to improving the manuscript.

Funding

This research was supported by the European Union Regional Development Fund within the framework of the ERDF Operational Program of Catalonia 2014-2020 with a grant of 50% of total cost eligible under the DRAC project [001-P-001723] and Lenovo-BSC Contract-Framework Contract (2022). It was also supported by the Ministerio de Ciencia e Innovacion MCIN AEI/10.13039/501100011033 and NextGenerationEU/PRTR under contracts PID2020-113614RB-C21, PID2019-107255GB-C21, and TED2021-132634A-I00, by the Generalitat de Catalunya GenCat-DIUiE (GRR) [contracts 2017-SGR-313, 2017-SGR-1328 and 2017-SGR-1414]. M.M. was partially supported by the Spanish Ministry of Economy, Industry and Competitiveness under Ramon y Cajal fellowship number RYC-2016-21104. S.M.-S. was supported by Juan de la Cierva fellowship grant IJC2020-045916-I funded by MCIN/AEI/10.13039/501100011033 and by 'European Union NextGenerationEU/PRTR'. B.P. and J.M.E. were supported, in part, by the United States National Institutes of Health [award numbers: R01HG010485, U01HG010961, OT2OD026682, OT3HL142481 and U24HG011853]. E.G. was supported by NIH/NIDA U01DA047638 and NSF PPoSS Award #2118709. A.G. acknowledges Dr. Nicole Soranzo's efforts to establish a pangenome research unit at the Human Technopole in Milan, Italy.

Conflict of Interest: none declared.

References

Altschul,S. *et al.* (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Bowden,R. *et al.* (2019) Sequencing of human genomes with nanopore technology. *Nat. Commun.*, **10**, 1869.

Daily,J. (2016) Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments. *BMC Bioinf.*, **17**, 1–11.

Daily,J. *et al.* (2015) A work stealing based approach for enabling scalable optimal sequence homology detection. *J. Parallel Distributed Comput.*, **79**, 132–142.

Durbin,R. *et al.* (1998) *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge.

Eizenga,J.M. and Paten,B. (2022) Improving the time and space complexity of the WFA algorithm and generalizing its scoring. *bioRxiv*.

Farrar,M. (2007) Striped Smith–Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*, **23**, 156–161.

Garrison,E. and Marth,G. (2012) Haplotype-based variant detection from short-read sequencing. *arXiv preprint arXiv:1207.3907*.

Gotoh,O. (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.*, **162**, 705–708.

Jones,N. *et al.* (2004) *An Introduction to Bioinformatics Algorithms*. MIT Press, Cambridge.

Kiełbasa,S. *et al.* (2011) Adaptive seeds tame genomic sequence comparison. *Genome Res.*, **21**, 487–493.

Koren,S. *et al.* (2017) CANU: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Res.*, **27**, 722–736.

Li,H. (2013) Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. *arXiv preprint arXiv:1303.3997*.

Li,H. (2018) Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, **34**, 3094–3100.

Loving,J. *et al.* (2014) BitPAl: a bit-parallel, general integer-scoring sequence alignment algorithm. *Bioinformatics*, **30**, 3166–3173.

Marco-Sola,S. *et al.* (2012) The gem mapper: fast, accurate and versatile alignment by filtration. *Nat. Methods*, **9**, 1185–1188.

Marco-Sola,S. *et al.* (2021) Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics*, **37**, 456–463.

McKenna,A. *et al.* (2010) The genome analysis toolkit: a mapreduce framework for analyzing next-generation DNA sequencing data. *Genome Res.*, **20**, 1297–1303.

Miga,K. H. and Wang,T. (2021) The need for a human pangenome reference sequence. *Annu. Rev. Genomics Hum. Genet.*, **22**, 81–102.

Myers,E. W. (1986) An $O(ND)$ difference algorithm and its variations. *Algorithmica*, **1**, 251–266.

Myers,E. W. and Miller,W. (1988) Optimal alignments in linear space. *Bioinformatics*, **4**, 11–17.

Needleman,S. B. and Wunsch,C. D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–453.

Rahn,R. *et al.* (2018) Generic accelerated sequence alignment in seqan using vectorization and multi-threading. *Bioinformatics*, **34**, 3437–3445.

Rodríguez-Martín,B. *et al.* (2017) Chimpipe: accurate detection of fusion genes and transcription-induced chimeras from RNA-seq data. *BMC Genomics*, **18**, 7–17.

Rognes,T. and Seeberg,E. (2000) Six-fold speed-up of Smith–Waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, **16**, 699–706.

Simpson,J. *et al.* (2009) ABYSS: a parallel assembler for short read sequence data. *Genome Res.*, **19**, 1117–1123.

Smith,T. F. and Waterman,M. S. (1981) Comparison of biosequences. *Adv. Appl. Math.*, **2**, 482–489.

Šošić,M. and Šikić,M. (2017) EDLIB: a C/C++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, **33**, 1394–1395.

Suzuki,H. and Kasahara,M. (2017) Acceleration of nucleotide semi-global alignment with adaptive banded dynamic programming. *bioRxiv*.

Suzuki,H. and Kasahara,M. (2018) Introducing difference recurrence relations for faster semi-global alignment of long sequences. *BMC Bioinformatics*, **19**, 33–47.

Wozniak,A. (1997) Using video-oriented instructions to speed up sequence comparison. *Bioinformatics*, **13**, 145–150.

Xia,Z. *et al.* (2021) A review of parallel implementations for the Smith–Waterman algorithm. In: *Interdisciplinary Sciences: Computational Life Sciences*, pp. 1–14.

Zhang,Z. *et al.* (2000) A greedy algorithm for aligning DNA sequences. *J. Comput. Biol.*, **7**, 203–214.

Zhao,M. *et al.* (2013) SSW library: an SIMD Smith–Waterman C/C++ library for use in genomic applications. *PLoS ONE*, **8**, e82138.