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

Santiago Marco-Sola [1,2]*, Jordan M. Eizenga [3], Andrea Guarracino [4], Benedict Paten [3], Erik Garrison [5], 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, Viale Rita Levi-Montalcini 1, Milan, 20157, Italy.
[5]Department of Genetics, Genomics and Informatics, University of Tennessee Health Science Center, Memphis, TN 38163, USA.
[6]Departament d'Arquitectura de Computadors, Universitat Politècnica de Catalunya, Barcelona, 08034, Spain.

*To whom correspondence should be addressed.

Abstract

Motivation: Pairwise sequence alignment remains a fundamental problem in computational biology and bioinformatics. Recent advances in genomics and sequencing technologies demand for 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 (WFA) algorithm introduced an efficient algorithm to perform exact 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 paper, we present the bidirectional WFA algorithm (BiWFA), the first gap-affine algorithm capable of computing optimal alignments in $O(s)$ memory while retaining the WFA's time complexity of $O(ns)$. In practice, our implementation never requires more than 183 MB aligning long and noisy sequences up to 1 Mbp long, while maintaining competitive execution times.
Availability: All code is publicly available at https://github.com/smarco/BiWFA-paper
Contact: santiagomsola@gmail.com

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 (Simpson *et al.*, 2009; Koren *et al.*, 2017), 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 involved the application of *dynamic programming* (DP) techniques. These methods often 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 DP-based methods is constrained by their quadratic requirements in time and memory with respect the lengths of the sequence pair. Consequently, multiple variations have been proposed over the years (Sellers, 1980; Ukkonen, 1985; Navarro, 2001). Notable optimizations include bit-parallel techniques (Myers, 1986, 1999; Loving *et al.*, 2014), data-layout transformations to exploit SIMD instructions (Rognes and Seeberg, 2000; Farrar, 2007; Wozniak, 1997), difference encoding of the DP matrix (Suzuki and Kasahara, 2018), among other methods (Ukkonen, 1985; 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 algorithm 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 (WFA) algorithm (Marco-Sola *et al.*, 2021) to compute the exact alignment between two sequences using gap-affine penalties. The WFA algorithm reformulates the alignment problem to compute the longest-possible alignments of increasing score until the optimal alignment is found. Notably, the WFA algorithm takes advantage of homologous regions between sequences to accelerate alignment's computation. As a result, the WFA algorithm 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, it provides the same guarantee for optimality as classical algorithms (Needleman and Wunsch, 1970; Smith and Waterman, 1981; Gotoh, 1982), but it does away with the quadratic requirements in time.

The WFA algorithm 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 variation or highly divergent genome regions. Given that we use alignment to understand variation, these are some of the contexts in which optimal alignment could be most useful, but its memory requirements makes it prohibitive.

To address this problem, this paper presents the first gap-affine alignment algorithm to compute the optimal alignment in $O(ns)$ time and $O(s)$ memory. Our method, the bidirectional WFA (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 algorithm $O(ns)$. Furthermore, our experimental results demonstrate that the BiWFA delivers comparable, or even better, performance than the original WFA algorithm, outperforming other state-of-the-art tools while using a minimal amount of memory.

The rest of the paper is structured as follows. Section 2 presents the definitions, algorithms, and formal proofs supporting the 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 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 i-th to the j-th character. We will use $\{x, o, e\}$ to denote to the affine-gap 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, the WFA algorithm 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), the authors prove 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 Eq. 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 $\mathcal{X}_{0,0} = 0$.

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

In order to compute the next wavefront, Eq. 1 shows that it is only necessary to have access to the previous $p = \max\{x, o + e\}$ wavefronts. We refer to p as the *scope*. Also, 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, the WFA algorithm 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, the 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 run 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 finishes after performing only a few character comparisons. Except in the case of the optimal alignment where $O(n)$ comparisons are required.

2.2 Bidirectional wavefront alignment algorithm

The core idea of the BiWFA algorithm is to perform the WFA algorithm simultaneously in both directions on the strings: from start to end, and from end to start. 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 then 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 has previously been employed to a similar end with the Myers $O(ND)$ difference algorithm (Myers, 1986).

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 (Eq. 1). However, to highlight the distinction, we will denote them by $\overrightarrow{\mathcal{W}}_s = (\overrightarrow{\mathcal{I}}_s, \overrightarrow{\mathcal{D}}_s, \overrightarrow{\mathcal{X}}_s, \overrightarrow{\mathcal{M}}_s)$. The recursions for the reverse direction are very similar (Eq. 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.

Algorithm 1: Compute optimal alignment breakpoint using the BiWFA algorithm.

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 $\sim s/2$ (being $\mathcal{A}$ 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 $\overrightarrow{\mathcal{M}}_0, \overleftarrow{\mathcal{M}}_0$
  $(s_f, s_r) \leftarrow (0, 0)$
  WF_INIT $(\overrightarrow{\mathcal{M}}_0, c_0, 0)$
  WF_INIT $(\overleftarrow{\mathcal{M}}_0, c_f, 0)$
  // Best breakpoint so far
  WF_EXTEND $(\overrightarrow{\mathcal{M}}_0, q, t)$
  WF_EXTEND $(\overleftarrow{\mathcal{M}}_0, q, t)$
  $(s_b, k_b, l_b, c_b) \leftarrow$WF_OVERLAP $(\overrightarrow{\mathcal{W}}_0, \overleftarrow{\mathcal{W}}_0)$
  while $s_f + s_r - o < s_b$ **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 $(\overrightarrow{\mathcal{M}}_{s_f}, q, t)$
    $(s, k, l, c) \leftarrow$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 $s_f + s_r - o \geq s_b$ **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 $(\overleftarrow{\mathcal{M}}_{s_r}, q, t))$
    $(s, k, l, c) \leftarrow$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

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

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, the BiWFA algorithm relies on the operators *WF_NEXT()* and *WF_EXTEND()* from the standard WFA (see Marco-Sola *et al.* (2021)) to compute successive wavefronts using Eqs. 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 the below (Sections 2.3 and 2.4).

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**
        $(s_b, k_b, l_b, c_b) \leftarrow (s, k, \overrightarrow{\mathcal{M}}_{k,s_f}, M)$
      if $\overrightarrow{\mathcal{I}}_{k,s_f} \geq \overleftarrow{\mathcal{I}}_{k,s_r} \wedge s_f + s - o < s_b$ **then**
        $(s_b, k_b, l_b, c_b) \leftarrow (s, k, \overrightarrow{\mathcal{I}}_{k,s_f}, I)$
      if $\overrightarrow{\mathcal{D}}_{k,s_f} \geq \overleftarrow{\mathcal{D}}_{k,s_r} \wedge s_f + s - o < s_b$ **then**
        $(s_b, k_b, l_b, c_b) \leftarrow (s, k, \overrightarrow{\mathcal{D}}_{k,s_f}, D)$
  return (s_b, k_b, l_b, c_b)
end

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.

2.4 Correctness of the breakpoint detection

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

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

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.

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, the Algorithm 1 is guaranteed to find part of a minimum-scoring alignment based on the following features:

- Algorithm 1 iterates through alternatingly incrementally increasing values of s_f and s_r.
- Algorithm 1 continues for o additional iterations after finding some s_f and s_r that satisfy the overlap condition.
- Algorithm 2 checks a window of p score values on each iteration.

2.5 Combining breakpoints into an alignment

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

Note that a breakpoint computed by the 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 WFA algorithms to use different initial condition 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 2 every time s is incremented. Thus, the memory use is proportional to the optimal alignment score, $O(s)$. Further, data structures are discarded before entering a recursive call, so the maximum memory use occurs in the outermost call, in which s is the optimal score of the full alignment.

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 $O((m+n)s)$.*

Proof. Let $\ell = m + n$, and let $T(\ell, s)$ be BiWFA's run 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 runs 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.

We will 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,

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

This proves the claim.

3 Results

We implemented the BiWFA algorithm described in this work in C. The code is publicly available at https://github.com/smarco/BiWFA-paper together with the scripts required to reproduced the experimental results presented in this section.

3.1 Experimental setup

We evaluate the performance of our BiWFA implementation compared to other high-performance sequence alignment libraries. We selected the original WFA (Marco-Sola *et al.*, 2021) and its new low-memory modes (i.e., medium and low) implemented in WFA2-lib (https://github.com/smarco/WFA2-lib). Also, we selected the efficient wfalm (Eizenga and Paten, 2022) with its low-memory modes (i.e., low-mem and recursive). Moreover, we included the highly optimized KSW2-Z2 (ksw2_extz2_sse) from the KSW2 library (Suzuki and Kasahara, 2018;

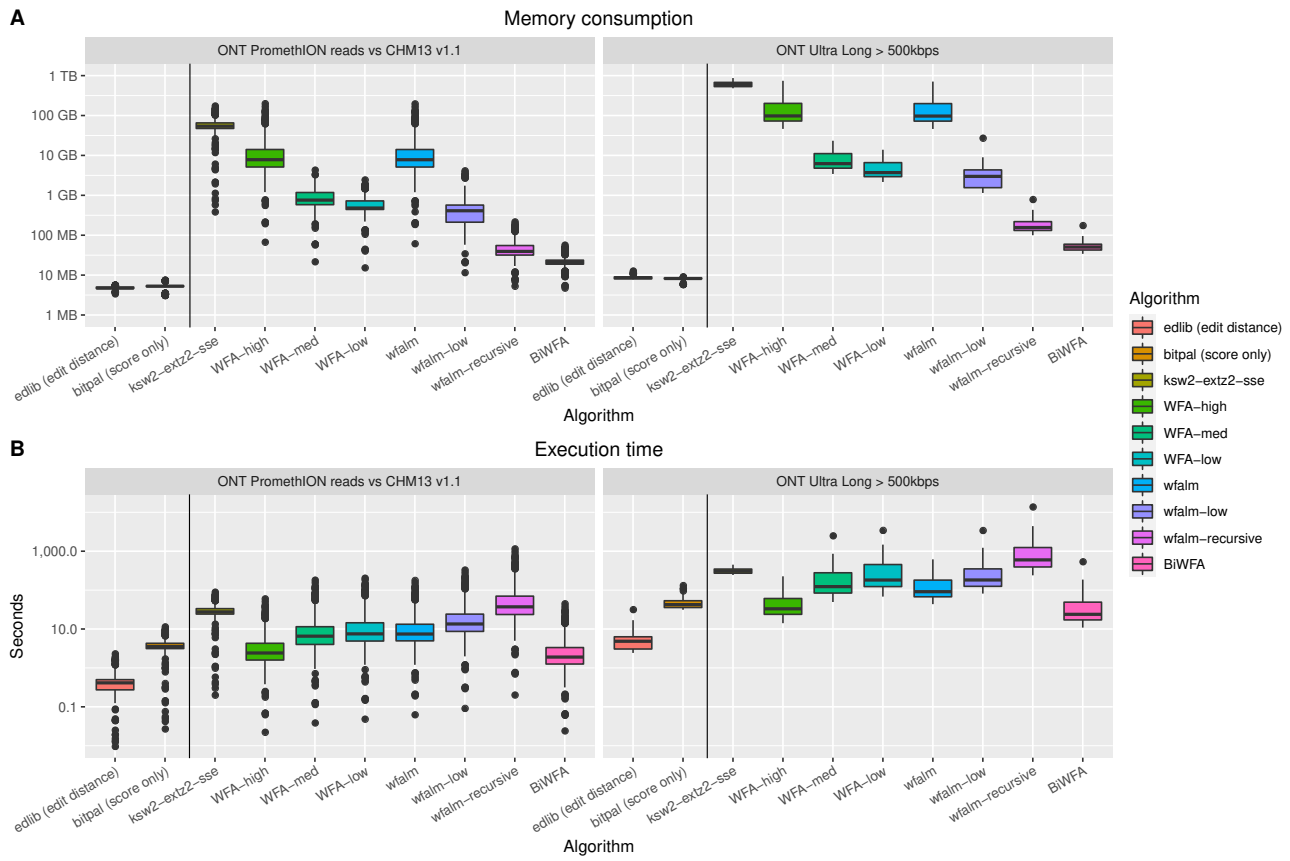

Fig. 1: Experimental results from the application of BiWFA and other state-of-the-art libraries and tools on long sequences. Each plot shows the different algorithms versus the memory consumption (A) and execution time (B). We differentiate algorithms that compute edit distance or only the score (left of vertical line) from those that compute the full alignment, CIGAR string included (right of vertical line). In the ONT Ultra Long dataset, not all alignments have been completed, as the tools ran out of memory (OOM) for some of the 48 sequence pairs. In particular, we obtained 10 OOMs with ksw2-extz2-sse, and a single OOM for both WFA-high and wfalm.

Li, 2018) as the best representative of DP-based methods. Additionally, we included 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 score), and thus are not directly comparable, we included them in the comparison to provide a performance upper bound. 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 used two real datasets. The first set of sequences was generated by the Human Pangenome Reference Consortium (Miga and Wang, 2021) and consists of long reads sequenced using Oxford Nanopore Technologies (ONT), PromethION platform. 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 kbps. The second dataset comprises ONT MinION reads from Bowden *et al.* (2019), restricted to those at least 500 kbp long, for a total number of 48 sequence pairs of average length equal to 630 kbps.

All the executions were performed single thread on a node running CentOS Linux equipped with an AMD EPYC 7742 CPU and 1TB of RAM.

3.2 Evaluation

Figure 1 shows the performance results obtained for all the evaluated algorithms in terms of execution time and memory consumed. BiWFA uses many times less memory than other methods. In particular, when aligning ultra long ONT sequences (Figure 1B), BiWFA requires between $68-93\times$ less memory compared to wfalm and WFA low-memory modes. Furthermore, BiWFA uses $3.5\times$ 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 tools in aligning long sequences. Using ultra long sequences, our method is $23.5\times$ faster than wfalm's recursive mode. Moreover, BiWFA's execution times are similar to those of BitPal (sometimes even faster, $1-1.2\times$ faster on average) computing exact alignments (not just the score) under the gap-affine model.

Most notably, BiWFA execution times are very similar, or even better, than those of the original WFA (despite the BiWFA requiring $2958\times$ and $604\times$ 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 this evaluation). In contrast, the BiWFA relieves this memory pressure using a minimal memory footprint. As a result, the BiWFA is able to balance out the additional work induced by BiWFA's recursion, delivering a performance on-par with the original WFA.

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 algorithm, a gap-affine pairwise alignment algorithm that runs in $O(ns)$ 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.

We have presented the BiWFA algorithm using affine gap scoring model. Nevertheless, these very same ideas can be translated directly into other distances like edit, linear gap, or piecewise affine gap. 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 algorithm retains the strengths of the original WFA algorithm: 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 and parallelized to fully exploit the capabilities of modern SIMD multicore processors. Our implementation, relies on the automatic vectorization capabilities of modern compilers and the open multi-processing API (OpenMP) to implement these features. As a result, the BiWFA implementation can exploit the SIMD and parallel capabilities of any platform and processor supported by modern compilers, without rewriting any part of the source code.

As a consequence of the minimal memory footprint required by the BiWFA algorithm, our implementation relieves the pressure on the memory system, reducing the memory penalties and slowdowns associated with classical alignment algorithms. Thus, our algorithm can exploit the benefits of the cache memory hierarchy and outperform the original WFA when it becomes limited by the memory bandwidth.

Genomics and bioinformatics methods will continue to rely on sequence alignment as a core and critical component. The BiWFA algorithm 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 the BiWFA to enable efficient sequence alignment at genome-scale in years to come.

Funding

This research was supported by the 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]. It was also supported by the Ministerio de Ciencia e Innovacion MCIN AEI/10.13039/501100011033 under contracts PID2020-113614RB-C21 and PID2019-107255GB-C21, 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. 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.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. A.G. acknowledges Dr. Nicole Soranzo's efforts to establish a pangenome research unit at the Human Technopole in Milan, Italy.

References

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, **215**(3), 403–410.

Bowden, R., Davies, R. W., Heger, A., Pagnamenta, A. T., de Cesare, M., Oikkonen, L. E., Parkes, D., Freeman, C., Dhalla, F., Patel, S. Y., Popitsch, N., Ip, C. L. C., Roberts, H. E., Salatino, S., Lockstone, H., Lunter, G., Taylor, J. C., Buck, D., Simpson, M. A., and Donnelly, P. (2019). Sequencing of human genomes with nanopore technology. *Nature Communications*, **10**(1).

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press.

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**(2), 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. *Journal of Molecular Biology*, **162**(3), 705–708.

Jones, N. C., Pevzner, P. A., and Pevzner, P. (2004). *An introduction to bioinformatics algorithms*. MIT press.

Koren, S., Walenz, B. P., Berlin, K., Miller, J. R., Bergman, N. H., and Phillippy, A. M. (2017). Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Research*, **27**(5), 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**(18), 3094–3100.

Loving, J., Hernandez, Y., and Benson, G. (2014). BitPAl: a bit-parallel, general integer-scoring sequence alignment algorithm. *Bioinformatics*, **30**(22), 3166–3173.

Marco-Sola, S., Sammeth, M., Guigó, R., and Ribeca, P. (2012). The gem mapper: fast, accurate and versatile alignment by filtration. *Nature methods*, **9**(12), 1185–1188.

Marco-Sola, S., Moure, J. C., Moreto, M., and Espinosa, A. (2021). Fast gap-affine pairwise alignment using the wavefront algorithm. *Bioinformatics*, **37**(4), 456–463.

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., *et al.* (2010). The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data. *Genome Research*, **20**(9), 1297–1303.

Miga, K. H. and Wang, T. (2021). The need for a human pangenome reference sequence. *Annual Review of Genomics and Human Genetics*, **22**(1), 81–102.

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

Myers, E. W. and Miller, W. (1988). Optimal alignments in linear space. *Bioinformatics*, **4**(1), 11–17.

Myers, G. (1999). A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM*, **46**(3), 395–415.

Navarro, G. (2001). A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, **33**(1), 31–88.

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. *Journal of Molecular Biology*, **48**(3), 443–453.

Rodríguez-Martín, B., Palumbo, E., Marco-Sola, S., Griebel, T., Ribeca, P., Alonso, G., Rastrojo, A., Aguado, B., Guigó, R., and Djebali, S. (2017). Chimpipe: accurate detection of fusion genes and transcription-induced chimeras from rna-seq data. *BMC genomics*, **18**(1), 1–17.

Rognes, T. and Seeberg, E. (2000). Six-fold speed-up of smith–waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, **16**(8), 699–706.

Sellers, P. H. (1980). The theory and computation of evolutionary distances: pattern recognition. *Journal of Algorithms*, **1**(4), 359–373.

Simpson, J. T., Wong, K., Jackman, S. D., Schein, J. E., Jones, S. J., and Birol, I. (2009). Abyss: a parallel assembler for short read sequence data. *Genome Research*, **19**(6), 1117–1123.

Smith, T. F. and Waterman, M. S. (1981). Comparison of biosequences. *Advances in Applied Mathematics*, **2**(4), 482–489.

Šošić, M. and Šikić, M. (2017). Edlib: a C/C++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, **33**(9), 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**(1), 33–47.

Ukkonen, E. (1985). Finding approximate patterns in strings. *Journal of Algorithms*, **6**(1), 132–137.

Wozniak, A. (1997). Using video-oriented instructions to speed up sequence comparison. *Bioinformatics*, **13**(2), 145–150.

Zhang, Z., Schwartz, S., Wagner, L., and Miller, W. (2000). A greedy algorithm for aligning dna sequences. *Journal of Computational biology*, **7**(1-2), 203–214.

Zhao, M., Lee, W.-P., Garrison, E. P., and Marth, G. T. (2013). Ssw library: an simd smith-waterman c/c++ library for use in genomic applications. *PloS one*, **8**(12).