Linear-Space Subquadratic-Time String Alignment Algorithm for Arbitrary Scoring Matrices

Ryosuke Yamano
Division of Medical Data Informatics, Human Genome Center, Institute of Medical Science, The University of Tokyo, Japan
Department of Computer Science, Graduate School of Information Science and Technology, The University of Tokyo, Japan

Tetsuo Shibuya
Division of Medical Data Informatics, Human Genome Center, Institute of Medical Science, The University of Tokyo, Japan

Abstract

Theoretically, the fastest algorithm by Crochemore et al. for computing the alignment of two given strings of size n over a constant alphabet takes $O(n^2/\log n)$ time. The algorithm uses Lempel–Ziv parsing to divide the dynamic programming matrix into blocks and utilizes the repetitive structure. It is the only previously known subquadratic-time algorithm that can handle scoring matrices of arbitrary weights. However, this algorithm takes $O(n^2/\log n)$ space, and reducing the space while preserving the time complexity has been an open problem for more than 20 years. We present a solution to this issue by achieving an $O(n)$ space algorithm that maintains $O(n^2/\log n)$ time. The classical refinement by Hirschberg reduces the space complexity of the textbook $O(n^2)$ algorithm to $O(n)$ while preserving the quadratic time. However, applying this technique to the algorithm of Crochemore et al. has been considered challenging because their method requires $O(n^2/\log n)$ space even when computing only the alignment score. Our modification enables the application of Hirschberg's refinement, allowing traceback computation in $O(n)$ space while preserving the $O(n^2/\log n)$ overall time complexity. Our algorithm can be applied to both global and local string alignment problems.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases String alignment, dynamic programming, linear space algorithms

Digital Object Identifier 10.4230/LIPIcs.WABI.2025.21

Funding This work was supported by MEXT KAKENHI Grant Numbers 21H05052, 23H03345, and 23K18501.

1 Introduction

The technical breakthrough of DNA sequencing represented by next-generation sequencers allows for the analysis of large-scale genomes, which has opened new computational challenges. Because the problem size could potentially be large, only space-efficient algorithms could be applicable. Specifically, designing a time-efficient algorithm under the constraint of linear space complexity becomes a crucial challenge.

Aligning two biosequences to compute their similarity is a common problem in bioinformatics. Detecting highly similar regions is fundamental in many scenarios, such as mapping DNA references and examining protein structures.

Given two strings of length n each over an alphabet Σ, finding an alignment maximizing the total score of all its insertions, deletions, matches, and substitutions whose scores are defined by a given scoring matrix δ is called the global string alignment problem [11]. The textbook dynamic programming solution solves it in $O(n^2)$ time [5]. This algorithm can be implemented with $O(n)$ space if only the optimal alignment value is needed by keeping only

the alive rows. This feature enables the application of the classical refinement by Hirschberg to compute traceback in the same $O(n^2)$ time and $O(n)$ space [7]. The local version of the string alignment problem, which is finding the maximum alignment value over all pairs of substrings of the given two strings, can be solved in the same time and space complexity [12]. We can compute the traceback of local alignments by reducing the problem to the traceback of global alignments [8] [2].

One of the results of fine-grained complexity states that there are no $O(n^{2-\epsilon})$ time algorithms for any constant $\epsilon > 0$ assuming the Strong Exponential Time Hypothesis [1]. While strongly subquadratic algorithms are improbable, subpolynomial improvements over the textbook algorithm are known.

Masek and Peterson [10] proposed a global alignment algorithm taking $O(n^2/\log n)$ time, assuming that the alphabet is constant and the weights of the score matrix are all rational numbers. It is based on the "Four Russians" algorithm, which divides the dynamic programming table into uniform-sized blocks and uses table lookup.

Crochemore et al. [3] proposed an $O(n^2/\log n)$ time algorithm for both global and local alignment problems, assuming a constant alphabet. They applied a variable-sized block partition using Lempel-Ziv factorization and achieved speedup by only computing the borders of the blocks. It does not require the precomputation of lookup tables; therefore, their algorithm is general enough to support real number weights, which differs from the algorithm of Masek and Paterson. This was an important generalization because scoring weights tend to be real numbers. For example, the two major scoring matrices, PAM [4] and BLOSUM [6], are computed as log-odds ratios, which could be irrational. To the best of our knowledge, their algorithm is the only previously known subquadratic global and local alignment algorithm that can handle arbitrary scoring matrices.

However, their algorithm takes $O(n^2/\log n)$ space for arbitrary weights and $O(n^2/\log^2 n)$ space for rational weights, even when only the optimal alignment value is needed. Their paper concluded by explicitly posing an open problem: reducing the space requirement of their algorithm while preserving the subquadratic time complexity. A major challenge arises from the fact that data associated with each variable-sized partitioned block may be accessed at any time, as the algorithm recursively depends on previous blocks to compute the weights of paths within the block. Since we need to keep all the data, it has been believed that space reduction is challenging.

More than two decades later, we have finally resolved this fundamental theoretical question by reducing the space complexity to linear while retaining the same subquadratic time bound. Obtaining a linear space algorithm to compute the optimal alignment value enables the application of Hirschberg's refinement to compute the traceback with the same time and space complexity. This is the first string alignment algorithm that simultaneously achieves subquadratic time and linear space under arbitrary scoring matrices, representing a major step forward in the theory of space-efficient algorithms.

1.1 Our contributions

We assume a constant alphabet.

- We propose a novel algorithm based on the algorithm of Crochemore et al. that computes the optimal global alignment value in $O(n^2/\log n)$ time and $O(n)$ space.
- By applying Hirschberg's refinement to our algorithm mentioned above, we obtain an algorithm computing the optimal global alignment trace in the same time and space complexity. We prove that the overall $O(n^2/\log n)$ time complexity holds by solving the recurrence relation.

- With some modifications, we show that the described algorithms can be applied to compute the optimal local alignment value and trace in the same time and space complexity.

For simplicity, we discussed the case when the given strings have the same length n. In the case comparing strings of size m and n, the time complexity will be $O(mn/\log\min\{m,n\})$, whereas the space complexity will be $O(m+n)$.

1.2 Overview of our technique

Our approach begins by partitioning each input string of length n into $\sqrt{n}$ blocks, each of size $\sqrt{n}$. We then apply a slightly generalized version of the algorithm by Crochemore et al. to compute the alignment between corresponding block pairs. Since each such comparison takes $O((\sqrt{n})^2/\log\sqrt{n}) = O(n/\log n)$ time and space, comparing a single block pair fits within linear space.

To compute the full alignment, we iteratively process the boundary rows between blocks. As there are $(\sqrt{n})^2 = n$ block pairs to compare, the total time complexity remains $O(n^2/\log n)$. However, at any point, we only need to store a single active row and compute one block comparison at a time, both of which require $O(n)$ space. Therefore, the overall space complexity is reduced to $O(n)$.

2 Preliminaries

For a string T of length n, the i-th character of T is denoted as $T[i]$. For indices $1 \le i, j \le n$, $T[i..j]$ denotes the substring $T[i]T[i+1]\dots T[j]$ if $i \le j$ and the empty string otherwise. We omit the indices when $i = 1$ or $j = n$, so $T[..j] = T[1..j]$ and $T[i..] = T[i..n]$. A string P is a *prefix* of T when there exists $1 \le j \le n$ such that $P = T[..j]$, and a *suffix* of T when there exists $1 \le i \le n$ such that $P = T[i..]$. We denote the reversed string of T as T^{rev}. c^i denotes the string repeating c for i times. The concatenation of strings A and B is denoted as AB. Similarly, the concatenation of a character c and a string S is cS, and the concatenation of a string S and a character c is Sc.

2.1 Problem definitions

We follow the standard global alignment model described by Gusfield [5], which we briefly summarize below for completeness.

▶ **Definition 1** (Alignment). *Given strings A and B over the alphabet $\Sigma \notin \epsilon$, a* (global) alignment *is a pair of strings (A', B') over the extended alphabet $\Sigma \cup \{\epsilon\}$ such that*

- *A' and B' are of equal length,*
- *A' is obtained from A by inserting zero or more spaces (ϵ),*
- *B' is obtained from B by inserting zero or more spaces,*
- *no position simultaneously contains a space in A' and B'.*

Let l denote the equal length of A' and B'. Given a scoring matrix $\delta : ((\Sigma \cup \{\epsilon\})^2 \setminus (\epsilon, \epsilon)) \to \mathbb{R}$, the alignment value *is defined as $\sum_{i=1}^{l} \delta(A'[i], B'[i])$.*

An alignment represents a specific edit transformation (see Figure 1).

▶ **Definition 2** (The global alignment problem). *Given two strings A, B, and the scoring matrix δ, the* optimal global alignment value *is the maximum value of the alignment value over all alignments. It is also called* similarity. *The specific alignment maximizing the alignment value is the* optimal global alignment trace.

δ	ϵ	a	c	g	t
ϵ		−1	−1	−1	−1
a	−1	1	−1	−1	−1
c	−1	−1	1	−1	−1
g	−1	−1	−1	1	−1
t	−1	−1	−1	−1	1

A'	c	t	a	c	g	a	ϵ	g	a
B'	ϵ	a	a	c	g	a	c	g	a
edit	D	R	M	M	M	M	I	M	M
δ	−1	−1	1	1	1	1	−1	1	1

Figure 1 Example of the optimal global alignment when comparing A = ctacgaga and B = aacgacga with the scoring matrix δ. I, D, R, and M denote insertion, deletion, replacement, and match, respectively. The similarity is 3.

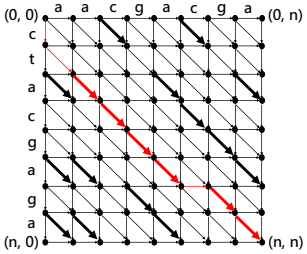

Figure 2 The alignment graph for strings A = ctacgaga and B = aacgacga with the scoring matrix δ shown in Figure 1. Edges in bold are assigned a weight of 1, while the remaining edges are assigned a weight of -1. The vertical edges correspond to the edit operation deletion, the horizontal edges correspond to insertion, and the diagonal edges correspond to replacement or match. The red path shows the optimal global alignment trace.

It might be the case that the whole two strings are not highly similar, but they have specific regions that are highly similar. In this case, the goal is to find such a pair of areas. It is called the *local alignment*, which we formally describe next.

▶ **Definition 3** (The local alignment problem). *Given two strings A, B, and the scoring matrix δ, the* optimal local alignment value *is the maximum similarity over all pairs of substrings from A and B. The* optimal local alignment trace *is the specific alignment of the pair of substrings that achieves the optimal local alignment value.*

▶ **Example 4.** In the case shown in Figure 1, we take the substring "acgaga" from A and "acgacga" from B to obtain the optimal local alignment value 5.

2.2 The alignment graph

Given strings A and B, the alignment graph is a grid graph that has $(|A|+1) \times (|B|+1)$ nodes, each labeled as (i, j) $(0 \leq i \leq |A|,\ 0 \leq j \leq |B|)$. It contains weighted directed edges from each node (i, j) to each node $(i, j+1), (i+1, j)$, and $(i+1, j+1)$, whose weights are $\delta(\epsilon, B[j+1]), \delta(A[i+1], \epsilon)$, and $\delta(A[i+1], B[j+1])$ respectively (see Figure 2).

2.3 The classical dynamic programming solution

The alignment problem can be regarded as computing the heaviest path in an alignment graph, which can be solved in quadratic time using dynamic programming [5] [12].

The global alignment problem. We define $V(i, j)$ as the similarity of $A[..i]$ and $B[..j]$. $V(i, j)$ is the optimal path weight from node $(0, 0)$ to node (i, j) in the alignment graph. The following recurrence holds.

$$V(i,j) = \max\{V(i,j-1) + \delta(\epsilon, B[j]), V(i-1,j) + \delta(A[i], \epsilon), \\ V(i-1,j-1) + \delta(A[i], B[j])\} \quad (1)$$

The local alignment problem. Given indices i and j, we define $L(i, j)$ as the maximum similarity over all pairs of suffixes of $A[..i]$ and $B[..j]$. The following recurrence holds.

$$L(i,j) = \max\{0, L(i,j-1) + \delta(\epsilon, B[j]), L(i-1,j) + \delta(A[i], \epsilon), \\ L(i-1,j-1) + \delta(A[i], B[j])\} \quad (2)$$

Note that we have a lower bound of 0 because we can take two empty suffixes. $L(i, j)$ is the optimal path weight that starts from any node and ends at node (i, j) in the alignment graph. The maximum value of $L(i, j)$ over all indices is the optimal local alignment value.

2.4 Hirschberg's refinement

Hirschberg's refinement is a divide-and-conquer technique that computes the longest common subsequence (LCS) in linear space [7]. However, it can also be used to compute the optimal global alignment trace [2] [8]. It uses an algorithm that computes the last row of the dynamic programming matrix V shown in the recurrence relation (1) as a subroutine to find an optimal division point. The technique can be regarded as gaining an algorithm that computes the traceback from an algorithm that computes only the score. They showed that the quadratic time and linear space bound of the subroutine can also be applied to the whole algorithm computing traceback, which can be summarized as follows.

▶ **Theorem 5** (Hirschberg's refinement [7]). *If there is an algorithm that computes $V(|A|, i)$ for $1 \leq i \leq |B|$, which is the last row of the dynamic programming matrix, in $O(|A||B|)$ time and $O(|A| + |B|)$ space, we gain an algorithm that computes the optimal global alignment trace of string A and B in the same $O(|A||B|)$ time and $O(|A| + |B|)$ space.*

2.5 Subquadratic string alignment algorithm by Crochemore et al.

Assuming a constant alphabet, Crochemore et al. [3] proposed an algorithm computing global and local alignments in $O(n^2/\log n)$ time and space, where n is the length of the two input strings. Their algorithm first partitions the given strings using LZ78 parsing [13]. LZ78 parses the input string into phrases, each consisting of the longest previously seen matching phrase followed by one extra character. For example, the string "aacgacg" will be divided into four phrases: "a", "ac", "g", and "acg". Since each phrase is distinct, the number of LZ78 parsed phrases is upper bounded by $O(n/\log n)$ [9].

LZ78 parsing of the input strings induces the partition of the alignment graph into variable-sized blocks. We denote these blocks as *LZ78 blocks*. The key idea of their algorithm is that they only compute the dynamic programming cells on the borders of the LZ78 blocks. Since there are $O(n/\log n)$ borders of length n, the total length is $O(n^2/\log n)$.

For each LZ78 block, they propagate the path weights from the top and left borders to the bottom and right borders, which they call "I/O propagation". They manage to propagate the path weights of the borders for each LZ78 block in linear time proportional to the length of the borders using a data structure whose size is $O(n^2/\log n)$. It can be constructed in the same $O(n^2/\log n)$ time. See Section 3 from the original paper [3] for further details. This can be summarized in the following theorem.

▶ **Theorem 6** (I/O propagation of LZ78 blocks [3]). *We are given the alignment graph comparing two strings of size n. For each LZ78 block, assuming that the optimal path weights from a fixed starting point to its top and left borders are known, the optimal path weights to its bottom and right borders can be computed in time linear in the border length, using a data structure of size $O(n^2/\log n)$. This data structure size is linear in the total size of all borders.*

By iteratively applying Theorem 6 to compute the path weights from node $(0, 0)$ to the borders of all LZ78 blocks, one obtains an $O(n^2/\log n)$ time and space algorithm for computing the optimal global alignment value.

When computing the optimal local alignment value, we need to compute the value L shown in the recurrence relation (2), where the starting point of the optimal path can be updated. The authors also managed to compute this in the same time and space by adding data structures that contain paths that start inside the LZ78 block. Additionally, without increasing the time and space complexities, the authors managed to compute and store the weights of optimal paths that start at each position in the top and left border and end inside the LZ78 block, as well as the optimal path that starts and ends within the block.

▶ **Theorem 7** (Local version of Theorem 6 [3]). *We are given the alignment graph comparing two strings of size n. For each LZ78 block, assume that the optimal path weights from the best starting point for each border position to its top and left borders are known. The optimal path weights to its bottom and right borders, each from their respective best starting point, and the maximum weight among all optimal paths terminating at internal positions within the block can all be computed in time linear in the border length, using a data structure of size $O(n^2/\log n)$. The size of this data structure is linear in the total size of all borders.*

By iteratively applying Theorem 7 to propagate the value L along the borders and compute the maximum L across all LZ78 blocks, one obtains an $O(n^2/\log n)$ time and space algorithm for computing the optimal local alignment value.

3 Computing the optimal global alignment value

We first partition the input strings into segments of size $\sqrt{n}$, which induces a uniform-sized partition of the alignment graph into blocks of size $\sqrt{n} \times \sqrt{n}$. We will denote these blocks as *segment blocks* to distinguish them from *LZ78 blocks*. Suppose there is an algorithm that takes the path weights to the top and left borders of a segment block as input and returns the path weight to its bottom and right borders in $O(n/\log n)$ time and space. We define this algorithm as *Segment Block Global I/O Propagation* in Definition 8. In that case, we can iteratively compute the borders of all segment blocks in a left-to-right, top-to-bottom order in $O(n^2/\log n)$ time since there are $\sqrt{n} \times \sqrt{n} = n$ segment blocks. The point is that we can free the internal space used for computing each segment block. We only need to keep the borders of a single active row of the segment blocks, which takes $O(n)$ space. Figure 3 illustrates the border update. With the $O(n/\log n)$ space usage for each computation of the segment block, the space complexity is reduced to $O(n)$. We formalize this in Theorem 9.

▶ **Definition 8** (Segment Block Global I/O Propagation). *Given a segment block of size $\sqrt{n} \times \sqrt{n}$ and the optimal path weights from the possibly external node $(0, 0)$ to its top and left borders, compute the optimal path weight from node $(0, 0)$ to its bottom and right borders.*

Figure 4 illustrates Segment Block Global I/O Propagation. It propagates the value V shown in the recurrence relation (1).

Algorithm 1 Computing the optimal global alignment value.

```
Function GlobalScore(A, B):
    Row[1] ← 0 /* Keep a single active row                    */
    for i ← 1 to n do
        Row[i + 1] ← Row[i] + δ(ϵ, B[i])
    end
    d ← ⌊√n⌋ /* Partition the alignment graph to d × d segment blocks */
    colStart ← 1
    for i ← 1 to d do
        colEnd ← colStart + ⌊n/d⌋
        if i ≤ (n mod d) then
            colEnd ← colEnd + 1
        end
        Col[1] ← Row[1]
        for i ← colStart + 1 to colEnd do
            Col[i − colStart + 1] ← Col[i − colStart] + δ(A[i − 1], ϵ)
        end
        rowStart ← 1
        for j ← 1 to d do
            rowEnd ← rowStart + ⌊n/d⌋
            if j ≤ (n mod d) then
                rowEnd ← rowEnd + 1
            end
            bottom, right ← SegBlockIO(top = Row[rowStart..rowEnd], left = Col)
            NextRow[rowStart..rowEnd] ← bottom
            Col ← right
            rowStart ← rowEnd
        end
        colStart ← colEnd
        Row ← NextRow
    end
    return Row
end
```

▶ **Theorem 9.** *An $O(n/\log n)$ time and space algorithm for Segment Block Global I/O Propagation implies an $O(n^2/\log n)$ time and $O(n)$ space algorithm computing the optimal global alignment value.*

The pseudo code is shown in Algorithm 1. *SegBlockIO* denotes the Segment Block Global I/O Propagation, taking the path weights to the top and left borders as input, and returning the path weights to the bottom and right borders. We denote the function returning the last row of the alignment graph as *GlobalScore*. The optimal global alignment value is the last element of the row.

We now show that an $O(n/\log n)$ time and space algorithm for Segment Block Global I/O Propagation can be obtained by applying Theorem 6.

▶ **Theorem 10.** *There is an $O(n/\log n)$ time and space algorithm for Segment Block Global I/O Propagation.*

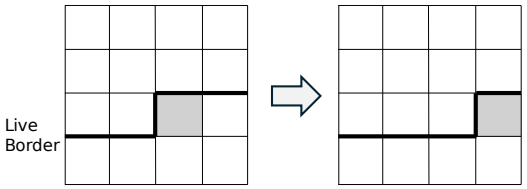

Figure 3 Computing the bottom and right borders of the gray segment block to update the live borders expressed as bold lines.

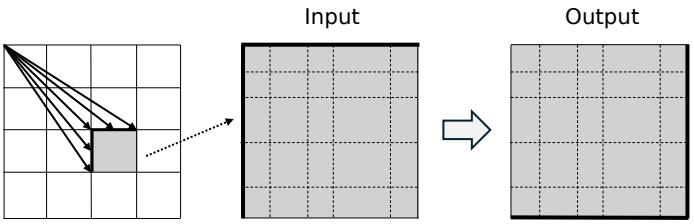

Figure 4 Illustration of Segment Block Global I/O Propagation. Here, it takes the path weights to the top and left borders of the gray segment block, and returns the path weights to its bottom and right borders. The computation is done by computing the borders of the LZ78 blocks inside the segment block, which are expressed as dotted lines.

Proof. The segment block is a subgraph of the alignment graph that compares two substrings of size $\sqrt{n}$. We first parse these two substrings using LZ78 parsing to induce the partition of the segment block into variable-sized LZ78 blocks. We then iteratively apply Theorem 6 to compute the borders of all LZ78 blocks in a left-to-right, top-to-bottom order. The total size of all borders is $O(\sqrt{n}^2/\log\sqrt{n}) = O(n/\log n)$, and the time and space usage are linear in this size. Therefore, the entire algorithm runs in $O(n/\log n)$ time and space. ◀

The algorithm for Segment Block Global I/O Propagation differs from the original algorithm of Crochemore et al. in that it can compute path weights from nodes that may lie outside the current block. This constitutes a slight generalization while using the same subroutine for propagating the borders of LZ78 blocks.

From Theorem 9 and Theorem 10, we gain the following theorem.

▶ **Theorem 11.** *Given two input strings of size n, the optimal global alignment value can be computed in $O(n^2/\log n)$ time and $O(n)$ space.*

We discussed the case where the input strings have the same length n. When the input strings have lengths m and n, then we partition both strings to segments of size $\sqrt{\min\{m,n\}}$, which induces the partition of the alignment graph into segment blocks of size $\sqrt{\min\{m,n\}} \times \sqrt{\min\{m,n\}}$, and we can apply the same argument. This will lead to the time complexity of $O(mn/\log\min\{m,n\})$, and the space complexity of $O(m+n)$.

4 Optimal global alignment trace recovery

Since our *GlobalScore* algorithm computes the last row of the dynamic programming matrix, we can use it as a subroutine for computing the optimal division point in Hirschberg's refinement, which immediately leads to an algorithm computing the optimal global alignment trace. The pseudo code is shown in Algorithm 2. Recall that the alignment is represented as a pair of space-inserted strings, as defined in Definition 1. The value $VV_1(t)$ corresponds to the

Algorithm 2 Computing the optimal global alignment trace.

Function `GlobalTrace(`A, B`)`**:**
  if $|B| = 0$ **then** `/* Solve trivial problem */`
    return $(A, \epsilon^{|A|})$
  else if $|A| = 1$ **then**`/* Solve trivial problem (brute force) */`
    $v \leftarrow \max_{1 \le j \le |B|}\{\delta(A[1], B[j]) - \delta(\epsilon, B[j])\}$
    $i \leftarrow \mathrm{argmax}_{1 \le j \le |B|}\{\delta(A[1], B[j]) - \delta(\epsilon, B[j])\}$
    if $v \ge \delta(A[1], \epsilon)$ **then**
      return $(\epsilon^{i-1} A \epsilon^{|B|-i}, B)$
    else
      return $(A\epsilon^{|B|}, \epsilon B)$
    end
  else `/* Find the optimal division point and solve recursively */`
    $i \leftarrow \lfloor |A|/2 \rfloor$
    $VV_1 \leftarrow$ `GlobalScore(`$A[..i]$, B`)`
    $VV_2 \leftarrow$ `GlobalScore(`$(A[i+1..])^{\mathrm{rev}}$, B^{rev}`)`
    $M \leftarrow \max_{0 \le j \le |B|}\{VV_1(j) + VV_2(|B| - j)\}$
    $k \leftarrow \min_j$ such that $VV_1(j) + VV_2(|B| - j) = M$
    $(C_1, C_2) \leftarrow$ `GlobalTrace(`$A[..i]$, $B[..k]$`)`
    $(D_1, D_2) \leftarrow$ `GlobalTrace(`$A[i+1..]$, $B[k+1..]$`)`
    return $(C_1 D_1, C_2 D_2)$
  end
end

optimal path weight from node $(0, 0)$ to node $(\lfloor |A|/2 \rfloor, t)$, and the value $VV_2(t)$ corresponds to the optimal path weight from node $(|A|, |B|)$ to node $(\lfloor |A|/2 \rfloor, t)$ in the alignment graph where $0 \le t \le |B|$.

We denote our algorithm computing the global alignment trace as *GlobalTrace*. Since our *GlobalScore*'s space usage is linear, we can apply Theorem 5 to obtain the linear space complexity of *GlobalTrace*. The problem remaining here is whether the subquadratic time complexity holds, which we prove next.

▶ **Theorem 12.** *Given input strings of size* $|A| = m$ *and* $|B| = n$*, the time complexity of* GlobalTrace *is* $O(mn/\log \min\{m, n\})$.

Proof. We denote the time bounds of *GlobalScore* and *GlobalTrace* as $GS(m, n)$ and $GT(m, n)$, respectively. We aim to prove that $GT(m, n)$ is upper bounded by $O(mn/\log \min\{m, n\})$. For simplicity, we assume m and n are both powers of two. This assumption does not affect the asymptotic time complexity for general m and n, since any string of length L can be padded to the smallest power of two that is at most $2L$, which only introduces a constant factor into the running time. The time bound will be linear when the size of either input string is constant; therefore, constants c_1, c_2 exist such that the following holds.

$$GT(m, n) \le c_1(m + n) + c_2 \qquad (\text{in case } \min\{m, n\} = O(1)). \tag{3}$$

Since *GlobalScore*'s time complexity is $O(mn/\log \min\{m, n\})$, constants c_3, c_4 exist such that the following inequality holds.

$$GS(m, n) \le c_3 mn/\log \min\{m, n\} + c_4. \tag{4}$$

GlobalTrace consists of two *GlobalScore* calls, the linear search of the optimal division point, and two recursive calls of itself. The linear search is proportional to the length of string B; thus, constants c_5, c_6 that time bound it by $c_5 n + c_6$ exist. To summarize, $GT(m, n)$ will be upper bounded by the following.

$$GT(m,n) \leq 2GS(m/2,n) + (c_5 n + c_6) + \max_{0 \leq n_l \leq n} \{GT(m/2, n_l) + GT(m/2, n - n_l)\}. \quad (5)$$

Constants $\alpha \geq 4$ and $0 < \beta < 1$ exist such that the following inequality holds (for example, we can take $\alpha = 32$ and $\beta = 0.99$).

$$\frac{3}{4} mn / \log \min\{m/2, n/2\} \leq \beta mn / \log \min\{m, n\} \qquad (\text{in case } \min\{m,n\} > \alpha). \quad (6)$$

We will show that the following inequality holds for all m, n by induction on m, where we define constants $d_1 = \frac{1}{1-\beta}(c_1 + c_2 + 2c_3 + 2c_4 + c_5 + c_6) \geq c_1$ and $d_2 = c_1 + c_2 \geq c_2$.

$$GT(m,n) \leq d_1 mn / \log \min\{m,n\} + d_2. \quad (7)$$

The induction hypothesis (7) holds for the base case $m \leq \alpha$ from inequality (3). Since inequality (3) holds for $\min\{m, n\} \leq \alpha$, we will only consider the case $\min\{m, n\} > \alpha$ hereafter. Now we assume that the induction hypothesis (7) holds for all $m' < m$ on $m > \alpha$. From inequality (4), the following inequality holds.

$$\begin{aligned} 2GS(m/2,n) &\leq c_3 mn / \log \min\{m/2, n\} + 2c_4 \\ &\leq 2c_3 mn / \log \min\{m,n\} + 2c_4. \end{aligned} \quad (8)$$

The following inequality holds from the induction hypothesis (7) and inequality (6).

$$\begin{aligned} &\max_{0 \leq n_l \leq n} \{GT(m/2, n_l) + GT(m/2, n - n_l)\} \\ &= \max_{0 \leq n_l \leq n/2} \{GT(m/2, n_l) + GT(m/2, n - n_l)\} \\ &\leq GT(m/2, n/2) + GT(m/2, n) \\ &\leq \frac{1}{4} d_1 mn / \log \min\{m/2, n/2\} + \frac{1}{2} d_1 mn / \log \min\{m/2, n\} + 2d_2 \\ &\leq \frac{3}{4} d_1 mn / \log \min\{m/2, n/2\} + 2d_2 \\ &\leq \beta d_1 mn / \log \min\{m,n\} + 2d_2. \end{aligned} \quad (9)$$

From inequalities (5) (8) (9), the following inequality holds.

$$\begin{aligned} GT(m,n) &\leq (\beta d_1 + 2c_3) mn / \log \min\{m,n\} + (c_5 n + d_2 + 2c_4 + c_6) + d_2 \\ &\leq (\beta d_1 + d_2 + 2c_3 + 2c_4 + c_5 + c_6) mn / \log \min\{m,n\} + d_2. \end{aligned} \quad (10)$$

Since we let $d_1 = \frac{1}{1-\beta}(c_1 + c_2 + 2c_3 + 2c_4 + c_5 + c_6)$ and $d_2 = c_1 + c_2$, the following holds.

$$\beta d_1 + d_2 + 2c_3 + 2c_4 + c_5 + c_6 \leq d_1. \quad (11)$$

Therefore, we obtain the induction hypothesis (7) for the current m, and the induction step is proved. ◀

Now that we have shown that *GlobalTrace* computes the global alignment trace in subquadratic time and linear space, we gain the following theorem.

▶ **Theorem 13.** *Given two input strings of size m and n, the optimal global alignment trace can be computed in $O(mn / \log \min\{m,n\})$ time and $O(m+n)$ space.*

5 Computing the optimal local alignment value

We follow the same framework we introduced for computing the optimal global alignment value. Specifically, we first partition the input strings into segments of size $\sqrt{n}$, inducing a uniform-sized partitioning of the alignment graph into segment blocks of size $\sqrt{n} \times \sqrt{n}$. The key difference here is that, instead of using *Segment Block Global I/O Propagation* to propagate the value V as defined in recurrence (1), we use *Segment Block Local I/O Propagation*, defined next, to propagate the value L defined in recurrence (2).

▶ **Definition 14** (Segment Block Local I/O Propagation). *Given a segment block of size $\sqrt{n} \times \sqrt{n}$, the optimal path weights from the best starting point for each border position to its top and left borders are provided as input. Using this information, compute the optimal path weights to the bottom and right borders, each from their respective best starting point, and the maximum weight among all optimal paths that end at internal positions within the segment block.*

Segment Block Local I/O Propagation runs in $O(n/\log n)$ time and space, identical to that of Segment Block Global I/O Propagation.

▶ **Theorem 15.** *There is an $O(n/\log n)$ time and space algorithm for Segment Block Local I/O Propagation.*

Proof. We begin by partitioning the segment block into variable-sized LZ78 blocks, as was done in Segment Block Global I/O Propagation, and then iteratively apply Theorem 7 to compute the borders of all LZ78 blocks in a left-to-right, top-to-bottom order. The optimal path weight ending inside each LZ78 block is also returned. The total size of all borders is $O(n/\log n)$, and the procedure runs in linear time and space with respect to this size. Since there are $O((\sqrt{n}/\log\sqrt{n})^2) = O(n/\log^2 n)$ LZ78 blocks in a segment block, the maximum path weight among them can be computed in additional $O(n/\log^2 n)$ time using only constant additional memory. Therefore, the overall time and space complexity is $O(n/\log n)$. ◀

We iteratively apply Segment Block Local I/O Propagation to compute the borders of all segment blocks in a left-to-right, top-to-bottom order. Since there are $\sqrt{n} \times \sqrt{n} = n$ segment blocks, the procedure runs in $O(n^2/\log n)$ time. Again, we only need to keep the borders of a single active row of the segment blocks, whose size is $O(n)$. Segment Block Local I/O Propagation also returns the optimal path weight ending inside each segment block. Since there are n segment blocks, the maximum among them, which is the optimal local alignment value, can be computed in additional $O(n)$ time using additional constant memory. With $O(n/\log n)$ internal space usage for each call of Segment Block Local I/O Propagation, the overall time complexity is $O(n^2/\log n)$ while the space complexity is reduced to $O(n)$. Therefore, we obtain the following theorem.

▶ **Theorem 16.** *Given two input strings of size n, the optimal local alignment value can be computed in $O(n^2/\log n)$ time and $O(n)$ space.*

When the input strings have lengths m and n, then we partition both strings into segments of size $\sqrt{\min\{m,n\}}$ and apply the same argument, which leads to an $O(mn/\log\min\{m,n\})$ time and $O(m+n)$ space algorithm.

6 Optimal local alignment trace recovery

We reduce the problem of optimal local alignment trace recovery to that of optimal global alignment trace recovery. When viewed as the problem of finding the optimal path in the alignment graph, the key difference from global alignment is that any node may serve as a start or end point. If the starting node (i_{begin}, j_{begin}) and the ending node (i_{end}, j_{end}) of the optimal path can be identified, the problem reduces to a global alignment problem between $A[i_{begin}..i_{end}]$ and $B[j_{begin}..j_{end}]$.

The ending node (i_{end}, j_{end}) is the position that maximizes the value L defined in the recurrence relation (2). Using Theorem 7, the maximum L inside each LZ78 block can be computed. To obtain the position, we need to add some data structures. Instead of storing only the value, we store it together with the corresponding position. Whenever we update the value with a larger candidate, we also update the associated position accordingly.

▶ **Theorem 17** (Extension of Theorem 7 to compute the maximizing position)**.** *In addition to Theorem 7, we can extend the result to also compute the position of the endpoint within each LZ78 block that maximizes the path weight, which corresponds to the index maximizing L within the block, with the same time and space complexity.*

Proof. We augment some of the internal data structures used in the original algorithm of Crochemore et al. with position information. To compute the weights of paths that terminate inside the LZ78 block, they introduce a vector E, which stores the optimal path weights starting at each node on the top and left borders and ending inside the block, and a scalar C, which stores the optimal path weight that both starts and ends within the block. By adding the weight of a path from outside the block (toward the top and left borders) to each entry in E, the algorithm considers all possible paths originating outside the block. Taking the maximum of these, along with C, yields the maximum path weight that ends within the block. For further details, see Section 6 of the original paper [3].

We modify the vector E and scalar C so that each stores not only the path weight but also the corresponding endpoint position. The additional memory required is the same as the size of E and C; thus, the space complexity remains unchanged. Whenever the maximum path weight ending within the block is updated, we also update the corresponding endpoint position, using the additional position information. Since the position is updated together with the value, this does not increase the time complexity. ◀

We can apply Theorem 17 to compute (i_{end}, j_{end}).

▶ **Theorem 18.** *Given two strings of size n, the index (i, j) maximizing the value $L(i, j)$ defined in recurrence (2) can be computed in $O(n^2/\log n)$ time and $O(n)$ space.*

Proof. We follow the same framework we used when computing the optimal local alignment value. We first partition the input strings into segments of size $\sqrt{n}$, which induces the uniform-sized partition of the alignment graph into segment blocks of size $\sqrt{n} \times \sqrt{n}$. We keep the borders of the single active row of segment blocks, which can be done in $O(n)$ space. For each segment block, we partition the corresponding substrings of length $\sqrt{n}$ using LZ78 parsing to induce a variable-sized partition into LZ78 blocks. We iteratively apply Theorem 17 to compute the borders of the LZ78 blocks while computing the maximum path weight and ending position that ends inside each LZ78 block. We can keep the best path weight and ending position among all LZ78 blocks using additional constant memory. The best position can be computed together with the optimal local alignment value, so the overall time and space complexity remain the same. ◀

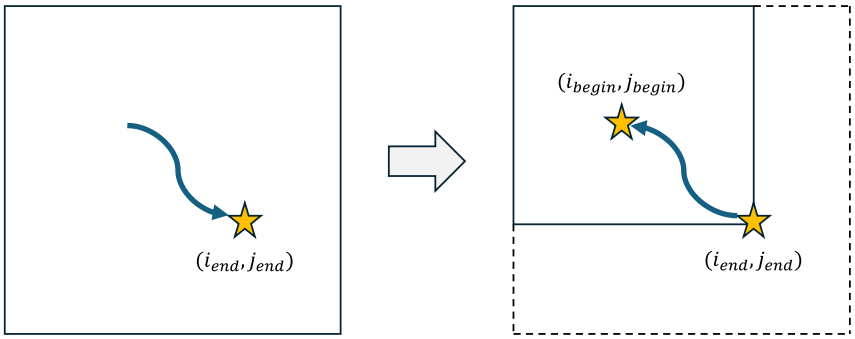

Figure 5 Illustration of computing the ending node (i_{end}, j_{end}) and the starting node (i_{begin}, j_{begin}) of the optimal local alignment path. The node (i_{end}, j_{end}) is obtained by computing the index maximizing the value L comparing the full strings A and B, while the node (i_{begin}, j_{begin}) is obtained by computing the index maximizing the value V comparing the reversed prefixes $(A[..i_{end}])^{\mathrm{rev}}$ and $(B[..j_{end}])^{\mathrm{rev}}$.

The starting node (i_{begin}, j_{begin}) can be identified as the index (i, j) that maximizes the similarity between the reversed substrings $(A[i..i_{end}])^{\mathrm{rev}}$ and $(B[j..j_{end}])^{\mathrm{rev}}$, as illustrated in Figure 5. This corresponds to the index that maximizes the value V defined in recurrence (1), where we compare $(A[..i_{end}])^{\mathrm{rev}}$ and $(B[..j_{end}])^{\mathrm{rev}}$ instead of the original strings A and B. Therefore, the following theorem can be applied to identify (i_{begin}, j_{begin}).

▶ **Theorem 19.** *Given two strings of size n, the index (i, j) maximizing the value $V(i, j)$ defined in recurrence (1) can be computed in $O(n^2/\log n)$ time and $O(n)$ space.*

Proof. When computing local alignments, it is necessary to track the starting points of the alignment paths. Crochemore et al. introduced a vector S, which stores the optimal path weights that end at each node on the bottom and right borders and start inside the LZ78 block, and a scalar C, which stores the optimal path weight that both starts and ends inside the block. They enable updating the starting points by taking the maximum between the weight of a path that starts outside the block and the weights stored in S and C. For further details, see Section 6 of the original paper [3].

In our case, however, we can eliminate the need for S and C, and simply omit the update of starting points, since the starting point is fixed. All other parts of the computation remain unchanged from Theorem 18. By keeping track of the maximum ending point, we obtain the index maximizing V instead of L in the same time and space complexity. ◀

Now we can reduce the problem of optimal local alignment trace recovery to that of optimal global alignment trace recovery.

▶ **Theorem 20.** *Given two strings A and B of size n, the optimal local alignment trace can be computed in $O(n^2/\log n)$ time and $O(n)$ space.*

Proof. First, we use Theorem 18 for the original strings A and B to identify the ending node (i_{end}, j_{end}). Next, we use Theorem 19 for reversed prefixes $(A[..i_{end}])^{\mathrm{rev}}$ and $(B[..j_{end}])^{\mathrm{rev}}$ to identify the starting node (i_{begin}, j_{begin}). Finally, we compute the optimal global alignment trace between the substrings $A[i_{begin}..i_{end}]$ and $B[j_{begin}..j_{end}]$ by Theorem 13, which corresponds to the optimal local alignment trace between A and B. All three procedures are performed in the same $O(n^2/\log n)$ time and $O(n)$ space. ◀

As in the previous sections, when the input strings have lengths m and n, the time complexity is $O(mn/\log\min\{m, n\})$ and the space complexity is $O(m + n)$.

7 Conclusion

We proposed an algorithm based on the algorithm by Crochemore et al. to obtain a linear space and subquadratic time algorithm for global and local alignment problems that can handle arbitrary scoring matrices, assuming that the given strings are over a constant alphabet. We partitioned the strings into segments of size $\sqrt{n}$ before the partition based on LZ78 parsing to enable the deletion of the internal data structure for blocks that are no longer referenced to obtain the space reduction. Additionally, we used Hirschberg's refinement to recover the traceback in the same time and space complexity.

References

1 Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 51–58, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2746539.2746612.

2 KUN-MAO CHAO, ROSS C. HARDISON, and WEBB MILLER. Recent developments in linear-space alignment methods: A survey. *Journal of Computational Biology*, 1(4):271–291, 1994. PMID: 8790471. doi:10.1089/cmb.1994.1.271.

3 Maxime Crochemore, Gad M. Landau, and Michal Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32(6):1654–1673, 2003. doi:10.1137/S0097539702402007.

4 Margaret Dayhoff, R Schwartz, and B Orcutt. 22 a model of evolutionary change in proteins. *Atlas of protein sequence and structure*, 5:345–352, 1978.

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

6 S Henikoff and J G Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992. doi:10.1073/pnas.89.22.10915.

7 D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, June 1975. doi:10.1145/360825.360861.

8 Xiaoqiu Huang and Webb Miller. A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics*, 12(3):337–357, 1991. doi:10.1016/0196-8858(91)90017-D.

9 A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976. doi:10.1109/TIT.1976.1055501.

10 William J. Masek and Michael S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980. doi:10.1016/0022-0000(80)90002-1.

11 Saul B. Needleman and Christian D. Wunsch. 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, 1970. doi:10.1016/0022-2836(70)90057-4.

12 T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981. doi:10.1016/0022-2836(81)90087-5.

13 J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978. doi:10.1109/TIT.1978.1055934.