Bounded-Length Smith–Waterman Alignment

Alexander Tiskin

Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom
http://warwick.ac.uk/alextiskin

Abstract

Given a fixed alignment scoring scheme, the bounded length (respectively, bounded total length) Smith–Waterman alignment problem on a pair of strings of lengths m, n, asks for the maximum alignment score across all substring pairs, such that the first substring's length (respectively, the sum of the two substrings' lengths) is above the given threshold w. The latter problem was introduced by Arslan and Eğecioğlu under the name "local alignment with length threshold". They proposed a dynamic programming algorithm solving the problem in time $O(mn^2)$, and also an approximation algorithm running in time $O(rmn)$, where r is a parameter controlling the accuracy of approximation. We show that both these problems can be solved exactly in time $O(mn)$, assuming a rational scoring scheme; furthermore, this solution can be used to obtain an exact algorithm for the normalised bounded total length Smith–Waterman alignment problem, running in time $O(mn \log n)$. Our algorithms rely on the techniques of fast window-substring alignment and implicit unit-Monge matrix searching, developed previously by the author and others.

2012 ACM Subject Classification Theory of computation → Pattern matching; Theory of computation → Divide and conquer; Theory of computation → Dynamic programming; Applied computing → Molecular sequence analysis; Applied computing → Bioinformatics

Keywords and phrases sequence alignment, local alignment, Smith–Waterman alignment, matrix searching

Digital Object Identifier 10.4230/LIPIcs.WABI.2019.16

1 The framework

In this section, we introduce briefly the framework developed by the author in [23, 22, 21, 24]. Definitions and results of this section have either appeared in these publications, or are their straightforward generalisations.

1.1 Preliminaries

For matrix and vector indices, we will use either integers, or half-integers (sometimes called odd half-integers):

$$\{\dots, -2, -1, 0, 1, 2, \dots\} \qquad \{\dots, -\tfrac{5}{2}, -\tfrac{3}{2}, -\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{3}{2}, \tfrac{5}{2}, \dots\}$$

For ease of reading, half-integer variables will be indicated by hats (e.g. $\hat{\imath}$, $\hat{\jmath}$). Ordinary variable names (e.g. i, j, with possible subscripts or superscripts), will normally denote integer variables, but can sometimes denote a variable that may be either integer, or half-integer.

It will be convenient to denote

$$i^- = i - \tfrac{1}{2} \qquad i^+ = i + \tfrac{1}{2}$$

for any integer or half-integer i. The set of all half-integers can now be written as

$$\{\dots, (-3)^+, (-2)^+, (-1)^+, 0^+, 1^+, 2^+, \dots\}$$

We denote integer and half-integer *intervals* by

$$[i : j] = \{i, i+1, \dots, j-1, j\} \qquad \langle i : j \rangle = \{i^+, i + \tfrac{3}{2}, \dots, j - \tfrac{3}{2}, j^-\}$$

19th International Workshop on Algorithms in Bioinformatics (WABI 2019).
Editors: Katharina T. Huber and Dan Gusfield; Article No. 16; pp. 16:1–16:12
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In both cases, the interval is defined by a pair of integer *endpoints*. We will use round parentheses $(i : j)$ to indicate that in the given context, either square $[i : j]$ or angle $\langle i : j\rangle$ brackets may be used.

For finite intervals $[i : j]$ and $\langle i : j\rangle$, we call the difference $j - i$ interval *length*. Note that for a given length n, an integer interval consists of $n + 1$ elements, and a half-integer interval of n elements.

We use the following notation for the Cartesian product of two integer or two half-integer intervals:

$$(i : i' \mid j : j') = (i : i') \times (j : j')$$

We will denote singleton Cartesian products by $(i, j) = (i \mid j)$.

We indicate the index range of a matrix by juxtaposition, e.g. matrix $A(i : i' \mid j : j')$ over an integer or half-integer index range. The same notation will be used for selecting subvectors and submatrices: for example, given matrix $A(0 : n \mid 0 : n)$, we denote by $A(i : i' \mid j : j')$ the submatrix defined by the given sub-intervals. When any of the indices i, i', j, j' coincide with the range boundary, they can be omitted, e.g. $A(i :|: j') = A(i : n \mid 0 : j')$, and $A(:| j : j') = A(0 : n \mid j : j')$. In particular, we write $A(i, j)$ for a single matrix element, $A(i |:)$ for a row, and $A(:| j)$ for a column. We will denote by $\sum A$ the sum of all elements in matrix A.

By default, vectors and matrices will be indexed by integers based at 0, or by half-integers based at $0^+ = \frac{1}{2}$. Where necessary, all our definitions and statements can easily be generalised to indexing over arbitrary integer or half-integer intervals.

Given a string, we distinguish between its contiguous *substrings*, and not necessarily contiguous *subsequences*. Special cases of a substring are *a prefix* and *a suffix* of a string. Unless indicated otherwise, an algorithm's input is a string a of length m, and a string b of length n; we assume $m \le n$. Normally, we will say that two characters α, β *match*, if $\alpha = \beta$, and *mismatch* otherwise. In addition to this, we introduce the *wildcard character* '?', which matches itself and all other characters.

▶ **Definition 1.** *Let $D\langle 0 : n_1 \mid 0 : n_2\rangle$ be a matrix. Its* dominance-sum matrix *(also called* distribution matrix*) $D^{\nearrow}[0 : n_1 \mid 0 : n_2]$ is defined by*

$$D^{\nearrow}[i, j] = \textstyle\sum D\langle i :|: j\rangle$$

for all $i \in [0 : n_1]$, $j \in [0 : n_2]$.

▶ **Definition 2.** *Let $A[0 : n_1 \mid 0 : n_2]$ be a matrix. Its* cross-difference matrix *$A^{\square}\langle 0 : n_1 \mid 0 : n_2\rangle$ (also called* density matrix*) is defined by*

$$A^{\square}\langle \hat\imath, \hat\jmath\rangle = A[\hat\imath^+, \hat\jmath^-] - A[\hat\imath^-, \hat\jmath^-] - A[\hat\imath^+, \hat\jmath^+] + A[\hat\imath^-, \hat\jmath^+]$$

for all $\hat\imath \in \langle 0 : n_1\rangle$, $\hat\jmath \in \langle 0 : n_2\rangle$.

▶ **Definition 3.** *Matrix A is a* Monge matrix*, if its cross-difference matrix $A^{\square}$ is nonnegative. Matrix A is an* anti-Monge matrix*, if its negative $-A$ is Monge.*

The structure of Monge matrices has been described by Burdyuk and Trofimov [6] and Bein and Pathak [4] (see also [8, 7]), and can be used to represent a Monge matrix implicitly by its cross-difference matrix and a pair of vectors.

▶ **Theorem 4** (Monge matrix canonical decomposition). *Let $A[0 : n_1 \mid 0 : n_2]$ be a Monge matrix. Let $D = A^{\square}$, $b = A[n_1 \mid :]$, $c = A[: \mid 0]$. We have*

$$A[i, j] = D^{\nearrow}[i, j] + b[j] + c[i] - b[0] \tag{1}$$

for all $i, j \in [0 : n_1 \mid 0 : n_2]$.

▶ **Definition 5.** *A* permutation matrix *is a zero-one matrix containing exactly one nonzero in every row and every column.*

▶ **Definition 6.** *Matrix A is called* unit-Monge*, if its cross-difference matrix $A^{\square}$ is a permutation matrix. Matrix A is called* unit-anti-Monge*, if its negative $-A$ is unit-Monge.*

An efficient row minima searching algorithm on a canonically represented unit-Monge matrix was given by Gawrychowski [10].

▶ **Theorem 7.** *Let A be a unit-Monge matrix, canonically represented by the permutation matrix $P = A^{\square}$ with n nonzeros, and by implicit vectors b, c, where each vector element can be queried in time $O(1)$. Given P, b, c, a data structure can be computed in time $O(n \log\log n)$ in the pointer machine model, and in time $O(n)$ in the unit-cost RAM model, such that the index of the leftmost minimum element in a row of A can be queried in time $O(1)$.*

1.2 LCS and semi-local LCS

▶ **Definition 8.** *Let a, b be strings. The* longest common subsequence (LCS) score *$lcs(a, b)$ is the length of the longest string that is a subsequence of both a and b. Given strings a, b, the* LCS problem *asks for the LCS score $lcs(a, b)$.*

The classical dynamic programming algorithm for the LCS problem [17, 25] runs in time $O(mn)$. The best known algorithms speed it up by a (model-dependent) polylogarithmic factor [16, 9, 5]. Similar speedups are possible for the algorithms presented in this paper; however, we will consider them outside the paper's scope, and will not discuss them any further.

Although global comparison (full string against full string) and fully-local comparison (all substrings against all substrings) are the two most common approaches to comparing strings, another important type of string comparison lies "in between".

▶ **Definition 9.** *Given strings a, b, the* semi-local LCS problem *asks for the LCS scores as follows:*

- *the whole a against every substring of b (*string-substring LCS*);*
- *every prefix of a against every suffix of b (*prefix-suffix LCS*);*
- *every suffix of a against every prefix of b (*suffix-prefix LCS*);*
- *every substring of a against the whole b (*substring-string LCS*).*

Note that this definition is symmetric with respect to exchanging the two strings. In many ways, our approach consists in exploiting to the full this and other symmetries of the LCS problem, such as the symmetry between the left and the right within each of the strings.

Some alternative terms for semi-local comparison, used especially in biological texts, are "end-free alignment" [14], [13, Subsection 11.6.4] or "semi-global alignment" [14], [15, Problem 6.24]. The string-substring (and its symmetric substring-string) component of semi-local string comparison is also called "fitting alignment" [15, Problem 6.23]. String-substring LCS is an important problem in its own right, closely related to approximate pattern matching, where a short fixed pattern string is compared to various substrings of a long text string.

Let $b^{pad}\langle -m : m + n\rangle = ?^m \underline{b} ?^m$.

▶ **Definition 10.** *The* semi-local LCS matrix *is defined as* $\mathsf{H}_{a,b}[i,j] = lcs(a, b^{pad}\langle i,j\rangle)$*, where* $i \in [-m : n]$*,* $j \in [0 : m+n]$*. The* string-substring LCS matrix *is defined as* $\mathsf{H}^{s\text{-}sub}_{a,b}[i,j] = \mathsf{H}_{a,b}[i,j] = lcs(a, b\langle i : j\rangle)$*, where* $i, j \in [0 : n]$*.*

In [23], we show that matrix $\mathsf{H}_{a,b}$ is unit-anti-Monge. By Theorem 4, it can be represented implicitly in compact form by a permutation matrix $\mathsf{P}_{a,b}$, that we call here *semi-local LCS kernel*, from which every element of $\mathsf{H}_{a,b}$ (and therefore also of its submatrix $\mathsf{H}^{s\text{-}sub}_{a,b}$) can be queried efficiently. In [23], we also give an algorithm computing this implicit representation in time $O(mn)$.

1.3 Alignment

The concept of LCS score is generalised by that of *alignment score* (see e.g. [14]). An *alignment* of strings a, b is obtained by putting a subsequence of a into one-to-one correspondence with a subsequence of b, respecting the index order. In contrast with an LCS, the two subsequences need not be identical. A pair of corresponding characters, one from a and the other from b, are said to be *aligned.* A character in one string that is not aligned against a character of the other string is said to be aligned against a *gap* in that string. An aligned character-character or character-gap pair is given a real-valued *score*:

- a pair of matching characters scores $w_+ \geq 0$;
- a pair of mismatching characters scores $w_0 < w_+$;
- a gap-character or character-gap pair scores $w_- \leq \frac{1}{2}w_0$; normally, also $w_- \leq 0$.

Negative scores are also called *penalties.* Any particular triple of character alignment scores (w_+, w_0, w_-) will be called a *scoring scheme.* A scoring scheme will be called *rational*, if all its components are rational numbers.

The intuition behind the score inequalities is as follows: aligning a matching pair of characters is always better than aligning a mismatching pair of characters, while the latter is at least as good as aligning each of the two characters against a gap.

▶ **Definition 11.** *Let* a*,* b *be strings. Assuming a fixed scoring scheme, the* alignment score $align(a, b)$ *is the maximum total score across all possible alignments of* a*,* b*. Given strings* a*,* b*, the* alignment problem *asks for their alignment score.*

In notation related to alignment under a general scoring scheme, we will use script typeface (e.g. $\mathcal{H}$), in order to distinguish it from the LCS scoring scheme, for which we keep using sans-serif typeface (e.g. H).

▶ **Definition 12.** *The* semi-local alignment problem *and its component subproblems (*string-substring alignment problem*, etc.) are defined analogously to Definition 9, replacing the LCS score by the alignment score. The* semi-local alignment matrix $\mathcal{H}_{a,b}$ *and the* string-substring alignment matrix $\mathcal{H}^{s\text{-}sub}_{a,b}$ *are defined analogously to Definition 10.*

The semi-local alignment problem can be reduced to the semi-local LCS problem by a couple of simple techniques, *regularisation* and *blow-up*, described (using slightly different terminology) in [22].

▶ **Definition 13.** *A scoring scheme* $(1, w_0, 0)$*, where* $0 \leq w_0 < 1$*, will be called* regular*.*

An arbitrary (finite) scoring scheme can be reduced to a regular one as follows (a similar method is used by Rice et al. [18]; see also [12, 15]). Let us consider first the global alignment of strings a, b, with an arbitrary scoring scheme (w_+, w_0, w_-). This scheme can be replaced by a scheme $(w_+ + 2x, w_0 + 2x, w_- + x)$ for any real x. Indeed, such a transformation increases the score of every global alignment by $(m+n)x$; therefore, the relative scores of different global alignments do not change. The desired regular scoring scheme can be obtained by taking $x = -w_-$, and then dividing each of the resulting scores by $w_+ - 2w_- > 0$:

$$(w_+, w_0, w_-) \mapsto \left(1, w_0^* = \tfrac{w_0 - 2w_-}{w_+ - 2w_-}, 0\right) \tag{2}$$

The resulting regular string alignment score h^* and the original alignment score h are related to each other by the following identities, defining regularisation and its reverse:

$$h^* = \tfrac{h-(m+n)w_-}{w_+ - 2w_-} \qquad h = h^* \cdot (w_+ - 2w_-) + (m+n) \cdot w_- \tag{3}$$

▶ **Definition 14.** *A* bistochastic matrix *is a matrix of nonnegative real elements, in which the sum of elements is exactly one in every row and every column.*

▶ **Definition 15.** *A Monge matrix A is called* regular*, if its cross-difference matrix $A^\square$ is bistochastic. An anti-Monge matrix A is called* regular*, if its negative $-A$ is regular.*

▶ **Theorem 16** (Alignment matrix canonical decomposition)**.** *Let a, b be strings. Assuming a fixed scoring scheme, the semi-local alignment matrix $\mathcal{H}_{a,b}$ is anti-Monge. Furthermore, for a regular scoring scheme, $\mathcal{H}_{a,b}$ is regular anti-Monge:*

$$\mathcal{H}_{a,b}[i,j] = j - i - \mathcal{S}_{a,b}^{\nearrow}[i,j]$$

for all $i \in [-m:n]$, $j \in [0:m+n]$, where $\mathcal{S}_{a,b} = -\mathcal{H}_{a,b}^\square$ is a bistochastic matrix.

Proof. The anti-Monge property of $\mathcal{H}_{a,b}$ is a straightforward generalisation of a similar property for semi-local LCS matrices.

Assume a regular scoring scheme $(1, w_0, 0)$. First, let us also assume it to be rational: $w_0 = \frac{\mu}{\nu}$. Then, the semi-local alignment problem on strings a, b can be reduced to the semi-local LCS problem by the following *blow-up* technique. We transform input strings a, b of lengths m, n into *blown-up* strings $\mathfrak{a}$, $\mathfrak{b}$ of lengths $\mathfrak{m} = \nu m$, $\mathfrak{n} = \nu n$. The transformation replaces every character γ by a substring $\$^\mu \gamma^{\nu-\mu}$ of length ν (here, $\$$ is a special guard character, not present in the original strings). We have

$$\mathcal{H}_{a,b}[i,j] = \tfrac{1}{\nu} \cdot \mathsf{H}_{\mathfrak{a},\mathfrak{b}}[\nu i, \nu j] \tag{4}$$

for all $i \in [-m:n]$, $j \in [0:m+n]$, where the alignment matrix $\mathcal{H}_{a,b}$ is defined by the given scoring scheme on the original strings a, b, and the LCS matrix $\mathsf{H}_{\mathfrak{a},\mathfrak{b}}$ by the LCS scoring scheme on the blown-up strings $\mathfrak{a}$, $\mathfrak{b}$.

Every element of $\mathcal{S}_{a,b}$ can now be obtained as a scaled sum of a $\nu \times \nu$ block of $\mathsf{P}_{\mathfrak{a},\mathfrak{b}}$:

$$\mathcal{S}_{a,b}\langle \hat{\imath}, \hat{\jmath} \rangle = \tfrac{1}{\nu} \sum \mathsf{P}_{\mathfrak{a},\mathfrak{b}} \langle \nu\hat{\imath}^- : \nu\hat{\imath}^+ \mid \nu\hat{\jmath}^- : \nu\hat{\jmath}^+ \rangle \tag{5}$$

for all $\hat{\imath} \in \langle -m:n \rangle$, $\hat{\jmath} \in \langle 0 : m+n \rangle$. By construction, matrix $\mathcal{S}_{a,b}$ is bistochastic.

A general regular scoring scheme can be approximated by a rational regular scheme to arbitrary precision. The property of matrix $\mathcal{S}_{a,b}$ to be bistochastic is preserved in the limit, therefore the theorem statement follows by compactness. ◀

▶ **Definition 17.** *Assuming a regular scoring scheme, the* semi-local alignment kernel *for strings a, b is the bistochastic matrix $\mathcal{S}_{a,b}\langle -m : n \mid 0 : m+n \rangle$, determined by Theorem 16. The* string-substring alignment kernel *is the submatrix $\mathcal{S}_{a,b}^{s.sub} = \mathcal{S}_{a,b}\langle 0:n \mid 0:n \rangle$*

The alignment matrix $\mathcal{H}_{a,b}$ can be represented implicitly by (the nonzeros of) the alignment kernel $\mathcal{S}_{a,b}$. However, in contrast with the LCS kernel, the alignment kernel may not be a permutation matrix. Sparsity of the alignment kernel is partially preserved for a rational scoring scheme.

▶ **Definition 18.** *Let $\nu > 0$ be a natural number. A bistochastic matrix will be called* ν-bistochastic*, if every its element is a multiple of* $1/\nu$*. A regular Monge matrix A will be called* ν-regular*, if $A^{\square}$ is ν-bistochastic. A regular anti-Monge matrix will be called* ν-regular*, if $-A$ is ν-regular.*

A permutation matrix is 1-bistochastic, and a unit-Monge matrix is 1-regular Monge. Note that in a ν-bistochastic matrix, there can be at most ν nonzeros in every row and every column.

▶ **Theorem 19.** *Let a, b be strings. Assuming a regular rational scoring scheme with denominator ν, the alignment kernel $\mathcal{S}_{a,b}$ is ν-bistochastic, and the alignment matrix $\mathcal{H}_{a,b}$ is ν-regular anti-Monge.*

Proof. Every element of $\mathcal{S}_{a,b}$ is the sum of a $\nu \times \nu$ block of the blown-up matrix $\frac{1}{\nu}\mathsf{P}_{\mathfrak{a},\mathfrak{b}}$ by (5), and is therefore a multiple of $1/\nu$. ◀

Theorem 19 implies that for $\nu = O(1)$, the alignment kernel $\mathcal{S}_{a,b}$ has $O(m+n)$ nonzeros. The described reduction also preserves the $O(mn)$ running time for computing the alignment kernel given a pair of strings, same as for the LCS kernel.

1.4 Window-substring alignment

Given a string and a fixed integer $w \geq 0$, we call any substring of length w a *w-window*.

▶ **Definition 20.** *Given strings a, b, and a window length w, the* window-substring alignment problem *asks for the alignment score of every w-window in string a against every substring in string b.*

In [23], we gave an efficient algorithm for the window-substring alignment problem.

▶ **Theorem 21.** *The window-substring alignment problem can be solved implicitly in time $O(mn)$, providing the window-substring alignment kernel $\mathcal{S}^{s.sub}_{a\langle k:l\rangle,b}$ for every $k, l \in [0:m]$, $l - k = w$.*

Proof. See [23]. Briefly, we compute a string-substring alignment kernel for a binary tree of specially defined *canonical substrings* of a against whole b. We then assemble these kernels into window-substring alignment kernels by a special procedure of implicit matrix multiplication, fully described in [24]. ◀

Definition 20 can be extended to other types of *length-constrained comparison*. In particular, we can consider a related symmetric problem, where the total length of strings a, b is fixed.

▶ **Definition 22.** *Given strings a, b, and a length parameter w, the* fixed total length alignment problem *asks for the alignment score of every substring in string a against every substring in string b, such that the sum of the two substrings' lengths is w.*

The fixed total length alignment problem can be solved by a technique similar to to the algorithm of Theorem 21. Observe that a window-substring alignment kernel corresponds to a horizontal rectangular strip of width w in the alignment grid, defined by the window in string a and the whole string b. To obtain an algorithm for the fixed total length alignment

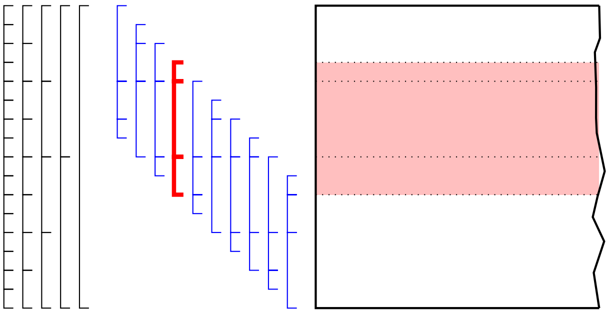

Figure 1 Window-substring alignment.

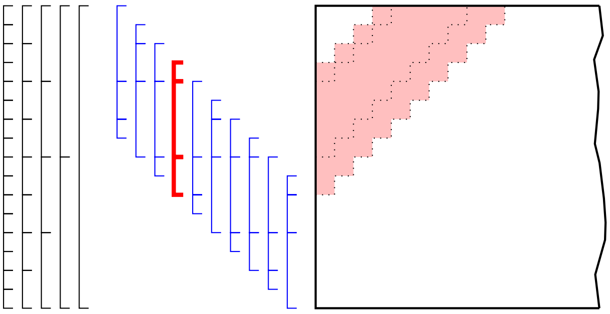

Figure 2 Fixed total length alignment.

problem with a rational scoring scheme, we replace these with an alternative type of kernel, corresponding to an anti-diagonal strip of rectilinear width w. Otherwise, the algorithm's structure is identical to the one of Theorem 21, and the running time remains $O(mn)$.

▶ **Example 23.** Figure 1 shows the computation of window-substring alignment by the algorithm of Theorem 21 on string a of length $m = 16$ with window size $w = 7$, against string b of arbitrary length n. The canonical substrings of a of lengths $1, 2, 4, 8, 16$ are shown in black, and 7-windows in blue. For each 7-window, the figure shows its decomposition into canonical substrings. For one of the 7-windows, highlighted in thick red, the corresponding area is outlined in the alignment dag.

Figure 2 shows the computation of fixed total length alignment on string a of length $m = 16$ with length parameter $w = 7$, against string b of arbitrary length n. Conventions are the same as in Figure 1.

1.5 Approximate pattern matching

Approximate pattern matching is a natural generalisation of classical (exact) pattern matching, allowing for some character differences between the pattern and a matching substring of the text. Given a *pattern string* p of length m and a *text string* t of length $n \geq m$, approximate pattern matching asks for all the substrings of the text that are close to the pattern, i.e. those that have sufficiently high alignment score (or, equivalently, sufficiently low edit distance) against the pattern. Such substrings of the text will be called *matching substrings*.

▶ **Definition 24.** *Given a pattern p and a text t, and assuming a fixed scoring scheme, the* complete approximate matching problem *is defined as follows. For every prefix $t\langle 0 : j\rangle$, the problem asks for the maximum alignment score of p against all possible choices of a suffix*

$t\langle i:j\rangle$ from this prefix:

$$h[j] = \max_{i\in[0:j]} \mathit{align}\big(p, t\langle i:j\rangle\big)$$

where $j \in [0:n]$.

The complete approximate matching problem corresponds to searching for column maxima in the string-substring alignment matrix $\mathcal{H}^{s.sub}_{p,t}$.

The complete approximate pattern matching problem can be solved by a classical dynamic programming algorithm due to Sellers [19] (see also [14]), running in time $O(mn)$.

Assuming a rational scoring scheme with constant denominator, an algorithm for complete approximate matching can also be obtained using alignment kernels.

▶ **Theorem 25.** *Let $A[0:n_1 \mid 0:n_2]$ be a ν-regular Monge matrix, where $\nu = O(1)$, canonically represented by the ν-bistochastic matrix $S = A^{\square}$ with $n \le \nu\cdot\min(n_1, n_2)$ nonzeros, and implicit vectors $b = A[n_1 \mid :]$, $c = A[: \mid 0]$, where random access to an element can be performed in time $O(1)$. Given S, b, c, the index of the leftmost minimum element in every row of A can be obtained in time $O(n \log\log n)$ in the pointer machine model, and in time $O(n)$ in the unit-cost RAM model.*

Proof. Straightforward generalisation of Theorem 7. ◀

▶ **Theorem 26.** *Assuming a rational scoring scheme with denominator $\nu = O(1)$, the complete approximate matching problem can be solved in time $O(mn)$.*

Proof. The algorithm runs in two phases.

Phase 1. We first regularise the scoring scheme by Equation (3). The alignment kernel $\mathcal{S}^*_{p,t}$ for the regular scheme is then obtained in time $O(mn)$.

Phase 2. Since complete approximate matching compares alignment scores across different string-substring pairs, we now reverse score regularisation via Equation (3). We then obtain column maxima of the (suitably scaled) string-substring submatrix $\mathcal{H}^{s.sub}_{p,t}$ by Theorem 25.

The total running time is dominated by Phase 1, and is therefore as claimed. ◀

2 Classical Smith–Waterman alignment

A classical method for variable-length local string comparison was given by Smith and Waterman [20]. Given a scoring scheme and a pair of input strings a, b, this method allows one to obtain a pair of substrings with the highest alignment score across all substring pairs in a, b. More generally, for each pair of prefixes of strings a, b, the method obtains their highest-scoring pair of suffixes.

▶ **Definition 27.** *Given a scoring scheme, the* Smith–Waterman (SW) alignment problem *on strings a, b is defined as follows. For every prefix $a\langle 0:l\rangle$ and every prefix $b\langle 0:j\rangle$, the problem asks for the maximum alignment score over all possible choices of a suffix $a\langle k:l\rangle$ and a suffix $b\langle i:j\rangle$ of the respective prefixes:*

$$h[l,j] = \max_{k\in[0:l], i\in[0:j]} \mathit{align}\big(a\langle k:l\rangle, b\langle i:j\rangle\big)$$

where $l \in [0:m]$, $j \in [0:n]$.

An efficient dynamic programming algorithm for SW-alignment was given by Smith and Waterman [20] and by Gotoh [11].

▶ **Algorithm 28** (SW-alignment).
Parameters: scoring scheme (w_+, w_0, w_-).
Input: strings a, b of length m, n, respectively.
Output: SW-alignment scores for a against b.
Description. We initialise

$$h[0,j] \leftarrow 0 \qquad h[l,0] \leftarrow 0$$

for all $l \in [0:m]$, $j \in [0:n]$. We then iterate through all indices $l \in [1:m]$ and $j \in [1:n]$. In each iteration, we assign

$$h[l,j] \leftarrow \max \begin{cases} h[l-1,j-1] + \begin{cases} w_+ & \text{if } a\langle l^-\rangle \text{ matches } b\langle j^-\rangle \\ w_0 & \text{otherwise} \end{cases} \\ h[l-1,j] + w_-;\ h[l,j-1] + w_-;\ 0 \end{cases}$$

▶ **Theorem 29.** *The SW-alignment problem can be solved in time $O(mn)$.*

Proof. In Algorithm 28, each iteration runs in constant time. The overall running time is $mn \cdot O(1) = O(mn)$. ◀

3 Bounded-length Smith–Waterman alignment

A commonly observed drawback in biological applications of SW-alignment is that it seeks to maximise the scores of substring pairs regardless of their lengths. However, if the substrings are too short, then their alignment may be less significant biologically than a slightly lower-scoring alignment of much longer substrings. To address this issue, various alternative definitions of local string comparison have been proposed, taking substring lengths into account; see [2] for a survey. Algorithms for such length-sensitive alignment problems are typically more computationally expensive than Algorithm 28 (SW-alignment); for this reason, approximation versions of such problems have also been considered [2].

The most straightforward approach to defining length-sensitive local alignment is to require that the aligned substrings satisfy a specific lower bound on their lengths, thus filtering out substrings that are too short.

▶ **Definition 30.** *Given strings a, b, an length threshold $w \geq 0$, and assuming a fixed scoring scheme, the* bounded length *(respectively, the* bounded total length*) SW-alignment problem is defined as follows. For every prefix $a\langle 0:l\rangle$ and every prefix $b\langle 0:j\rangle$, the problem asks for the maximum alignment score over all possible choices of a suffix $a\langle k:l\rangle$ and a suffix $b\langle i:j\rangle$ of the respective prefixes, such that $l-k \geq w$ (respectively, $l-k+j-i \geq w$):*

$$h_{len \geq w}[l,j] = \max_{\substack{k\in[0:l], i\in[0:j] \\ l-k\geq w}} \mathit{align}\big(a\langle k:l\rangle, b\langle i:j\rangle\big)$$

$$h_{tlen \geq w}[l,j] = \max_{\substack{k\in[0:l], i\in[0:j] \\ l-k+j-i\geq w}} \mathit{align}\big(a\langle k:l\rangle, b\langle i:j\rangle\big)$$

where $l \in [0:m]$, $j \in [0:n]$.

The bounded total length SW-alignment problem was introduced by Arslan and Eğecioğlu [1] (see also [2]) under the name *local alignment with length threshold (LAt)*. They proposed a dynamic programming algorithm solving the problem in time $O(mn^2)$. They also gave an approximation algorithm running in time $O(rmn)$, and returning a pair of substrings of total length at least $\left(1-\frac{1}{r}\right)w$, scoring no less than the exact solution to the original problem.

We now show that, assuming a rational scoring scheme, the bounded (total) length SW-alignment problem can be solved exactly by an efficient algorithm. In fact, our exact algorithm is asymptotically faster not only than the exact algorithm of [1], but also than their approximation algorithm. We first describe an algorithm for the bounded length alignment problem, and then adjust it to the bounded total length alignment problem.

▶ **Algorithm 31** (Bounded-length SW-alignment).

Parameters: a rational scoring scheme (w_+, w_0, w_-), where $w_- < 0$.

Input: strings a, b of length m, n, respectively; length threshold $w \geq 0$.

Output: Bounded length SW-alignment scores for a against b.

Description. The algorithm runs in three phases:

1. solving the window-substring alignment problem for every w-window in a against every substring in b;
2. solving the complete approximate matching problem for every w-window in a against b: for every w-window $a\langle k:l\rangle$ in a and every prefix $b\langle 0:j\rangle$ of b, we find the maximum alignment score of $a\langle k:l\rangle$ against all possible choices of a suffix $b\langle i:j\rangle$ from this prefix;
3. extending the complete approximate matching scores to optimal bounded-length SW-alignment scores by a dynamic programming procedure that generalises Algorithm 28 (SW-alignment).

We now describe each of these phases in more detail.

Phase 1. By Theorem 21, we obtain the window-substring alignment kernel $\mathcal{S}^{s.sub}_{a\langle k:l\rangle,b}$ for every k, l, $l-k=w$.

Phase 2. We now consider the window-substring alignment matrices $\mathcal{H}^{s.sub}_{a\langle k:l\rangle,b}$ for every k, l, $l-k=w$. Every such matrix is represented implicitly by the window-substring alignment kernel $\mathcal{S}^{s.sub}_{a\langle k:l\rangle,b}$, obtained in the first phase. By Theorem 26, for every w-window $a\langle k:l\rangle$ in a and every prefix $b\langle 0:j\rangle$ of b, we find a suffix $b\langle i:j\rangle$ of that prefix, that has the highest alignment score against $a\langle k:l\rangle$:

$$h_{len=w}[l,j] = \max_{i\in[0:j]} \mathcal{H}^{s.sub}_{a\langle k:l\rangle,b}[i,j]$$

Phase 3. We initialise

$$h_{len\geq w}[w,j] \leftarrow h_{len=w}[w,j] \qquad h_{len\geq w}[l,0] \leftarrow 0$$

for all $l \in [w:m]$, $j \in [0:n]$. We then iterate through all indices $l \in [w+1:m]$ and $j \in [1:n]$. In each iteration, we assign

$$h_{len\geq w}[l,j] \leftarrow \max \begin{cases} h_{len\geq w}[l-1,j-1] + \begin{cases} w_+ & \text{if } a\langle l^-\rangle \text{ matches } b\langle j^-\rangle \\ w_0 & \text{otherwise} \end{cases} \\ h_{len\geq w}[l-1,j] + w_-;\; h_{len\geq w}[l,j-1] + w_-;\; h_{len=w}[l,j] \end{cases}$$

▶ **Theorem 32.** *The bounded length SW-alignment problem can be solved in time $O(mn)$.*

Proof. In Algorithm 31, the running time of each of the three phases, and therefore the overall running time, is $O(mn)$. ◀

The bounded total length alignment problem can be solved by an algorithm structured similarly to Algorithm 31. In this algorithm Phase 1, instead of the window-substring alignment problem, solves the fixed total length alignment problem. Phase 2 obtains column maxima in the resulting implicit alignment matrices. Phase 3 remains unchanged. The algorithm's running time remains $O(mn)$.

4 Normalised bounded-length Smith–Waterman alignment

Arslan et al. [3] introduced another, more sophisticated approach to length-sensitive local alignment, by incorporating the substrings' lengths into the scoring function.

▶ **Definition 33.** *Let a, b be strings. Assuming a fixed scoring scheme, the strings'* normalised alignment score *is defined as their alignment score divided by their total length:*

$$nalign(a, b) = \frac{align(a, b)}{m + n}$$

Analogously to Definition 30, the normalised bounded total length SW-alignment problem was introduced by Arslan and Eğecioğlu [1] (see also [2]) under the name *normalised local alignment with length threshold (NLAt)*. They gave an approximation algorithm running in time $O(rmn \log n)$, assuming a rational scoring scheme, and returning a pair of substrings of total length at least $\left(1 - \frac{1}{r}\right)w$, scoring no less than the exact solution to the original problem. The algorithm uses the authors' approximation algorithm for ordinary (unnormalised) bounded total length SW-alignment as a subroutine; the only dependence on parameter r is within that subroutine. By replacing this subroutine with Algorithm 31, we obtain an exact algorithm for normalised bounded total length SW-alignment running in time $O(mn \log n)$, assuming a rational scoring scheme.

5 Conclusion and open problems

We have described an efficient algorithm for bounded-length alignment and bounded total length alignment; our algorithm can also be used as a subroutine for efficient normalised bounded total length alignment. The algorithm relies on a rather complex framework of semi-local string comparison, window-substring alignment and implicit matrix searching, developed previously by the author and others. It remains a challenge to simplify and streamline our algorithm to a point where it can be easily and efficiently implemented.

The algorithm relies on the scoring scheme to be rational, with the least common denominator of the character alignment scores considered to be a constant factor in the running time. It remains an open question whether the algorithm can be generalised to an arbitrary real-weighted scoring scheme.

References

1 Abdullah N. Arslan and Ömer Eğecioğlu. Dynamic Programming Based Approximation Algorithms for Sequence Alignment with Constraints. *INFORMS Journal on Computing*, 16(4):441–458, 2004.

2 Abdullah N. Arslan and Ömer Eğecioğlu. Dynamic and fractional programming-based approximation algorithms for sequence alignment with constraints. In *Handbook of Approximation Algorithms and Metaheuristics*, Chapman and Hall/CRC Computer and Information Science Series, chapter 76, pages 76–1–76–16. Chapman and Hall/CRC, 2007.

3 Abdullah N. Arslan, Ömer Eğecioğlu, and Pavel A. Pevzner. A new approach to sequence comparison: Normalized sequence alignment. *Bioinformatics*, 17(4):327–337, 2001.

4 Wolfgang W. Bein and Pramod K. Pathak. A characterization of the Monge property and its connection to statistics. *Demonstratio Mathematica*, 29(2):451–457, 1996.

5 Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science*, 409(3):486–496, 2008.

6 V. Y. Burdyuk and V. N. Trofimov. Generalization of the results of Gilmore and Gomory on the solution of the traveling salesman problem. *Engineering Cybernetics*, 14:12–18, 1976.

7 Rainer E. Burkard. Monge properties, discrete convexity and applications. *European Journal of Operational Research*, 176(1):1–14, 2007.

8 Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996.

9 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.

10 Paweł Gawrychowski. Faster algorithm for computing the edit distance between SLP-compressed strings. In *Lecture Notes in Computer Science*, volume 7608 of *Lecture Notes in Computer Science*, pages 229–236, 2012.

11 Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162(3):705–708, 1982.

12 D. Gusfield, K. Balasubramanian, and D. Naor. Parametric optimization of sequence alignment. *Algorithmica*, 12(4-5):312–326, 1994.

13 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, Cambridge, 1997.

14 Benjamin Jackson and Srinivas Aluru. Pairwise Sequence Alignment. In *Handbook of Computational Molecular Biology*, Chapman and Hall/CRC Computer and Information Science Series, chapter 1, pages 1–1–1–32. Chapman and Hall/CRC, 2010.

15 N. C. Jones and P. A. Pevzner. *An Introduction to Bioinformatics Algorithms.* Computational Molecular Biology. The MIT Press, 2004.

16 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.

17 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.

18 S. V. Rice, H. Bunke, and T. A. Nartker. Classes of Cost Functions for String Edit Distance. *Algorithmica*, 18(2):271–280, 1997.

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

20 T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

21 A. Tiskin. Faster subsequence recognition in compressed strings. *Journal of Mathematical Sciences*, 158(5):759–769, 2009.

22 Alexander Tiskin. Semi-local longest common subsequences in subquadratic time. *Journal of Discrete Algorithms*, 6(4):570–581, 2008.

23 Alexander Tiskin. Semi-local string comparison: Algorithmic techniques and applications. *Mathematics in Computer Science*, 1(4):571–603, 2008.

24 Alexander Tiskin. Fast Distance Multiplication of Unit-Monge Matrices. *Algorithmica*, 71(4):859–888, 2015.

25 Robert A. Wagner and Michael J. Fischer. The String-to-String Correction Problem. *Journal of the ACM*, 21(1):168–173, 1974.